

CS 228: Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 1, January 10, 2017

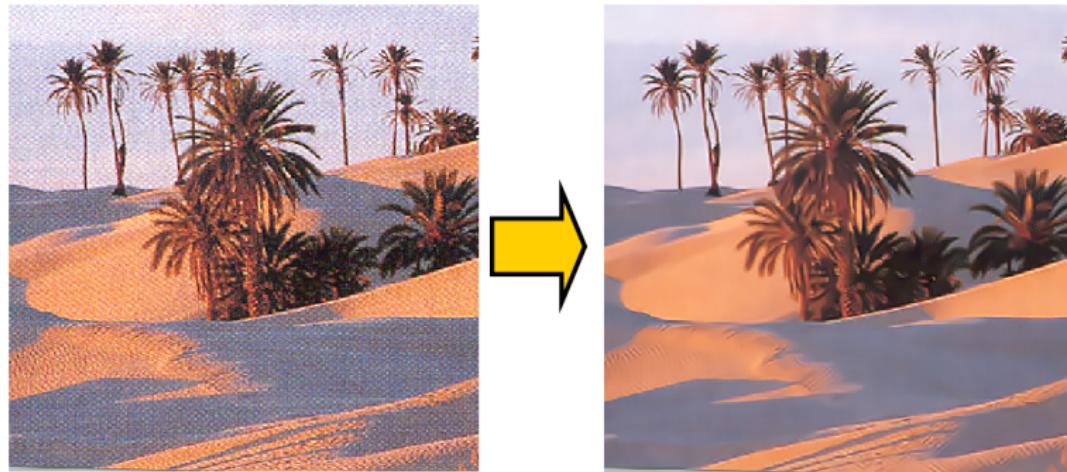
One of the **most exciting advances** in Artificial Intelligence
(machine learning, signal processing, coding, control, ...) in the last
decades

Can we model **large-scale, complex** systems as a collection of
simpler, locally interacting subsystems?

Key idea

- ① **Represent** the world as a collection of random variables X_1, \dots, X_n with joint distribution $p(X_1, \dots, X_n)$
- ② **Learn** the distribution from data
- ③ Perform “**inference**” (compute conditional distributions $p(X_i | X_1 = x_1, \dots, X_m = x_m)$)

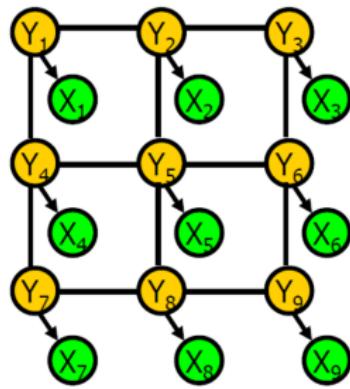
Applications: Image Denoising



Applications: Image Denoising



Markov Random Field

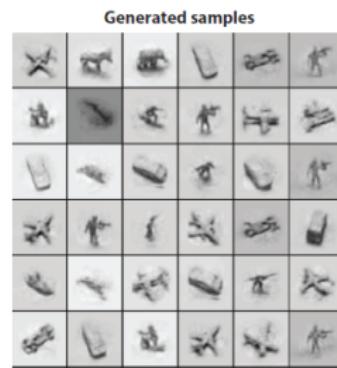


X_i : noisy pixels
 Y_i : “true” pixels

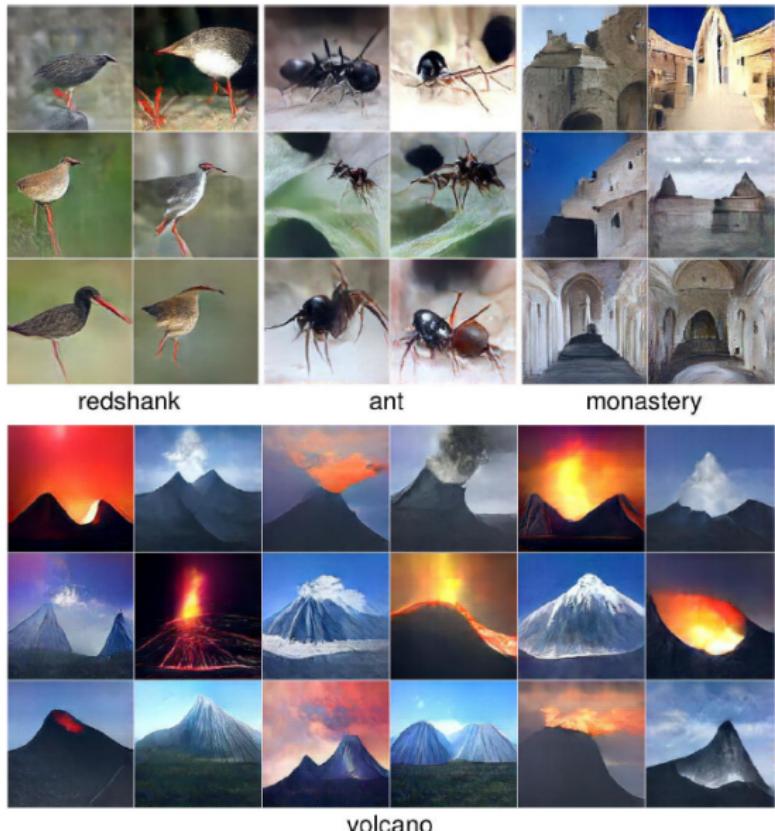
Applications: Generative Models of Images



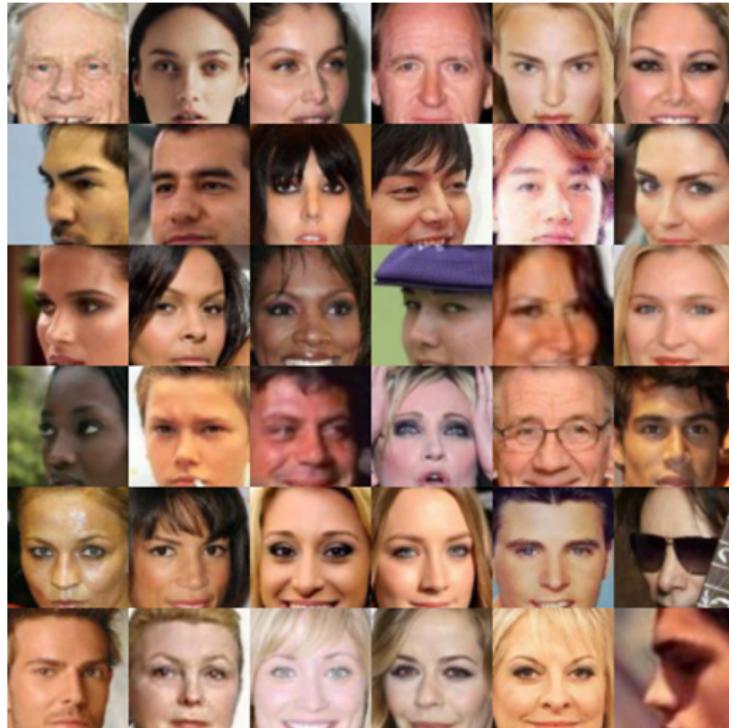
Learn a probability distribution over images



Applications: Generative Models of Images



Applications: Inpainting

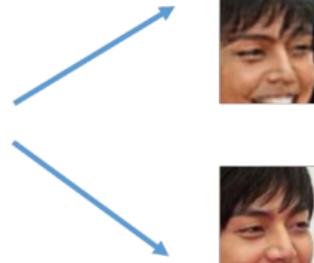
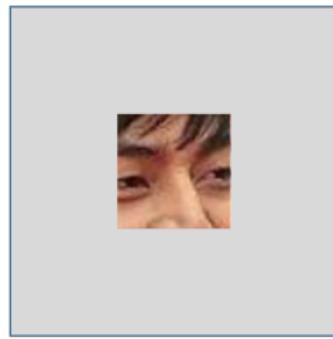


$$\sim p(x)$$

200k photos of celebrities

Applications: Inpainting

Conditioned on the central patch, what is the most likely way to complete the image?

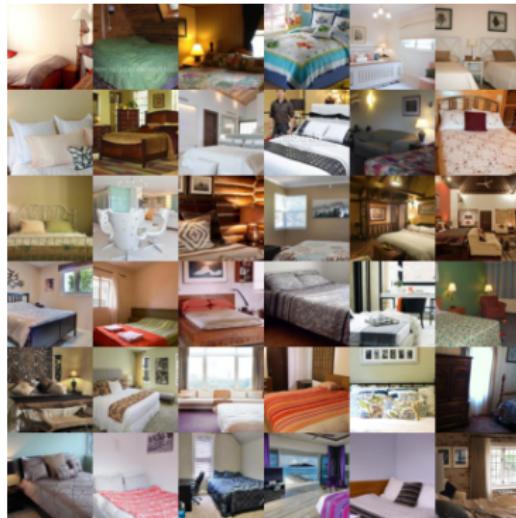


True/sample?

True/sample?

$P(\text{Full image} \mid \text{central patch})$

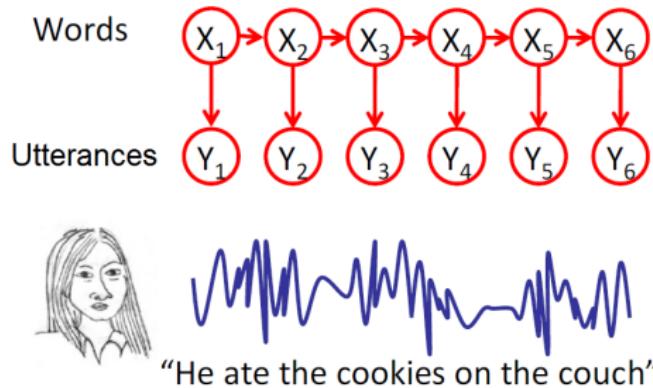
Applications: Inpainting



Applications: Inpainting

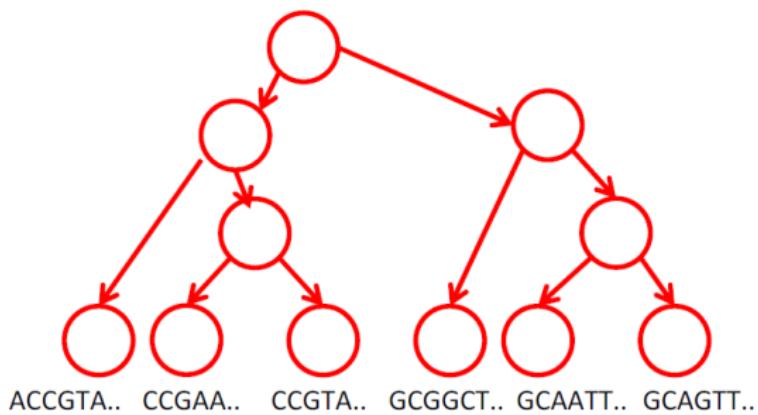


Applications: Speech Recognition



- Infer spoken words from audio signals
- Hidden Markov Models

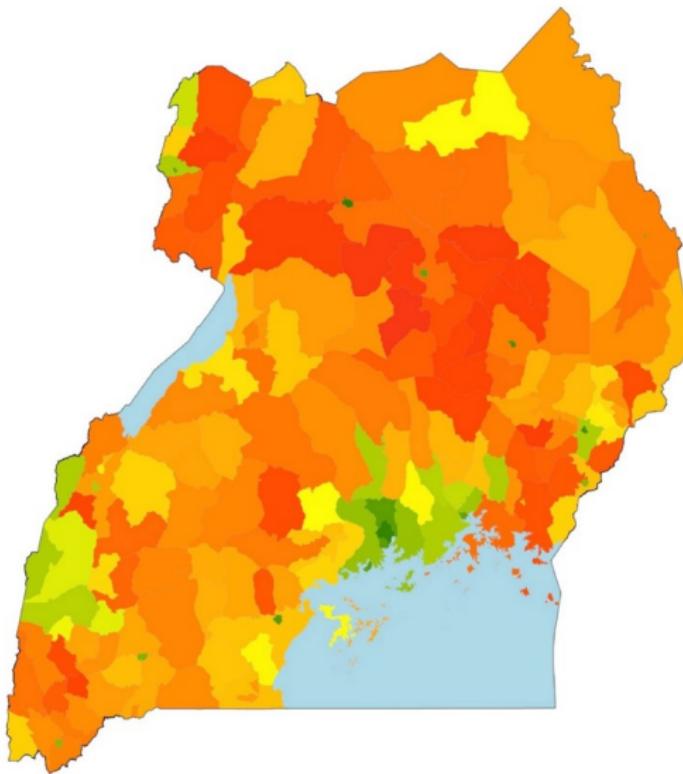
Applications: Evolutionary Biology



- Reconstruct a phylogenetic tree from DNA sequences of current species

Applications: Poverty Mapping

Uganda, estimated daily per capita expenditure (2012-2015)

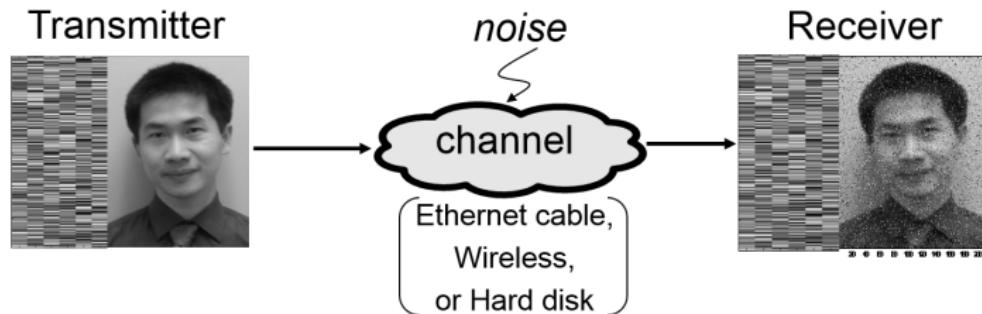


Applications: Ecology



- Learn models of bird migration using data from the *eBird* citizen science project

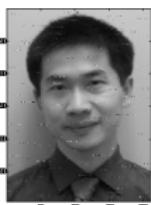
Applications: Error correcting codes



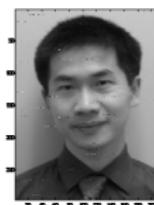
Iterative message passing decoding



Iteration 1



Iteration 5



Iteration 15



Iteration 16

Applications: Many more!

- Robot Localization and Mapping
- Neuroscience (fMRI data)
- Topic Models
- Identifying gene regulatory networks
- Distributed Control
- Graphical Games
- Collaborative Filtering
- Crowdsourcing
- ...

Key challenges

- ① **Represent** the world as a collection of random variables X_1, \dots, X_n with joint distribution $p(X_1, \dots, X_n)$
 - How does one *compactly describe* this joint distribution?
- ② **Learn** the distribution from data
 - How much data do we need?
 - How much computation does it take?
- ③ Perform “**inference**” (compute conditional distributions $p(X_i | X_1 = x_1, \dots, X_m = x_m)$)

Syllabus overview

- We will study Representation, Inference & Learning
- First in the simplest case
 - Only discrete variables
 - Fully observed models
 - Exact inference & learning
- Then generalize
 - Partially observed data during learning (hidden variables)
 - *Approximate* inference & learning
- Learn about algorithms, theory & applications

Learning objectives

At the end of the course, you should be able to:

- ① Characterize main types of graphical models, and associate graphical properties to statistical ones
- ② Implement common probabilistic inference and learning algorithms.
Analyze their runtime and evaluate the results
- ③ Tackle real-world problems with the appropriate models and algorithms

Logistics: class

- **Class webpage:**
 - <http://cs.stanford.edu/~ermon/cs228/>
 - Sign up on Piazza
 - Sign up on GradeScope
- **Book:** *Probabilistic Graphical Models: Principles and Techniques* by Daphne Koller and Nir Friedman, MIT Press (2009)
 - Readings for each lecture posted to course website.
 - Lecture notes
- **Teaching Assistants:** Aditya Grover, Alex Bishara, Ethan Chan, Kratarth Goel, Xiaocheng Li, Bo Wang
- **Office hours:** See calendar on website

Logistics: prerequisites & grading

- **Prerequisites:**
 - Probability and statistics
 - Algorithms (e.g., dynamic programming, graphs, complexity)
 - Programming
 - Basic optimization concepts
- **Grading:** problem sets (70%) + final exam (30%) + participation (3%)
 - 5 assignments. Both theory and programming.
 - **Important:** See collaboration policy on class webpage
- Solutions to the theoretical questions require formal proofs.
- For the programming assignments, I recommend Python. Take substantial time. Builds up skills needed for later lectures.

Review of probability: outcomes

Reference: Chapter 2 and Appendix A

- Consider a random experiment, e.g., a coin flip, roll of a die, shot of an arrow. An **outcome space** specifies the possible outcomes that we would like to reason about, e.g.

$$\Omega = \{ \text{}, \text{} \} \quad \text{Coin toss}$$

$$\Omega = \{ \text{}, \text{}, \text{}, \text{}, \text{}, \text{} \} \quad \text{Die toss}$$

- We specify a **probability** $p(\omega)$ for each outcome ω such that

$$p(\omega) \geq 0, \quad \sum_{\omega \in \Omega} p(\omega) = 1$$

E.g., $p(\text{}) = .6$

$p(\text{}) = .4$

Review of probability: events

- An **event** is a subset of the outcome space, e.g.

$$E = \{ \text{}, \text{}, \text{} \} \quad \text{Even die tosses}$$

$$O = \{ \text{}, \text{}, \text{} \} \quad \text{Odd die tosses}$$

- The **probability** of an event is given by the sum of the probabilities of the outcomes it contains,

$$p(E) = \sum_{\omega \in E} p(\omega)$$

E.g., $p(E) = p(\text{}) + p(\text{}) + p(\text{})$
 $= 1/2, \text{ if fair die}$

Example

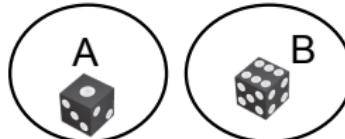
- Imagine an experiment in which one rolls a **fair** six-sided die:
 $\Omega = \{1, 2, 3, 4, 5, 6\}$
- Consider the events $C_1 = \{1, 2\}$ and $C_2 = \{2\}$
- Then $P(C_1) = 1/3$, $P(C_2) = 1/6$
- Also, $P(C_1 \cap C_2) = 1/6$, $P(C_1 \cup C_2) = 1/3$

Independence of events

- Two events A and B are **independent** if

$$p(A \cap B) = p(A)p(B)$$

- Assuming fair die, are these two events independent?



No! $p(A \cap B) = 0, \quad p(A)p(B) = \left(\frac{1}{6}\right)^2$

- Now suppose our outcome space had two different die:

$$\Omega = \{ \text{die 1}, \text{die 2}, \dots, \text{die 36} \} \quad 2 \text{ die tosses}$$

$6^2 = 36$ outcomes

and the probability distribution is such that each die is independent,

$$p(\text{die 1}, \text{die 2}) = p(\text{die 1}) p(\text{die 2})$$

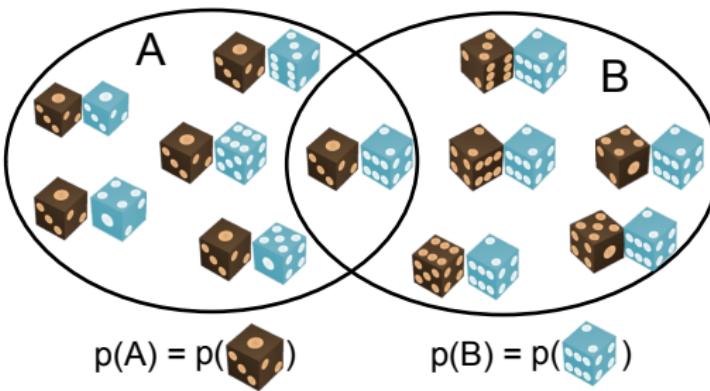
$$p(\text{die 3}, \text{die 4}) = p(\text{die 3}) p(\text{die 4})$$

Independence of events

- Two events A and B are **independent** if

$$p(A \cap B) = p(A)p(B)$$

- Are these two events independent?

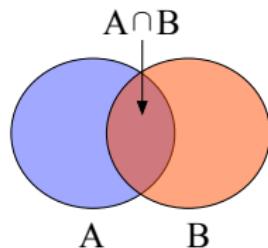


Yes!

$$p(A \cap B) = p(\text{1 brown and blue die})$$

$$p(A)p(B) = p(\text{1 brown die}) p(\text{1 blue die})$$

Conditional probability



- Let A, B be events, $p(B) > 0$.

$$p(A | B) = \frac{p(A \cap B)}{p(B)}$$

- Claim 1: $\sum_{\omega \in \Omega} p(\omega | S) = 1$
- Claim 2: If A and B are independent, then $p(A | B) = p(A)$

Two important rules

- ① **Chain rule** Let S_1, \dots, S_n be events, $p(S_i) > 0$.

$$p(S_1 \cap S_2 \cap \dots \cap S_n) = p(S_1)p(S_2 | S_1) \cdots p(S_n | S_1 \cap \dots \cap S_{n-1})$$

- ② **Bayes' rule** Let S_1, S_2 be events, $p(S_1) > 0$ and $p(S_2) > 0$.

$$p(S_1 | S_2) = \frac{p(S_1 \cap S_2)}{p(S_2)} = \frac{p(S_2 | S_1)p(S_1)}{p(S_2)}$$

Discrete random variables

- Events are cumbersome to work with
- A **random variable** X is a mapping $X : \Omega \rightarrow D$
 - D is some set (e.g., the integers)
 - Induces a partition of all outcomes Ω
- For some $x \in D$, we say

$$p(X = x) = p(\{\omega \in \Omega : X(\omega) = x\})$$

“probability that variable X assumes state x ”

- Notation: $\text{Val}(X) = \text{set } D \text{ of all values assumed by } X$
(will interchangeably call these the “values” or “states” of variable X)

Examples

- Bernoulli distribution: (biased) coin flip
 - $D = \{Heads, Tails\}$
 - Specify $P(X = Heads) = p$. Then $P(X = Tails) = 1 - p$.
 - Write: $X \sim Ber(p)$
- Multinomial distribution: (biased) m -sided dice
 - $D = \{1, \dots, m\}$
 - Specify $P(Y = i) = p_i$, such that $\sum p_i = 1$
 - Write: $Y \sim Mult(p_1, \dots, p_m)$

Multivariate distributions

- Instead of one random variable, have random *vector*

$$\mathbf{X}(\omega) = [X_1(\omega), \dots, X_n(\omega)]$$

- $X_i = x_i$ is an event. The **joint distribution**

$$p(X_1 = x_1, \dots, X_n = x_n)$$

is simply defined as $p(X_1 = x_1 \cap \dots \cap X_n = x_n)$

- We will often write $p(x_1, \dots, x_n)$ instead of $p(X_1 = x_1, \dots, X_n = x_n)$

Example of joint distribution

- Student's Intelligence I . $\text{Val}(I) = \{i_0 \text{ (high)}, i_1 \text{ (very high)}\}$
- Exam's Difficulty D . $\text{Val}(D) = \{d_0 \text{ (easy)}, d_1 \text{ (hard)}\}$
- Student's Grade G . $\text{Val}(G) = \{g_1 \text{ (A)}, g_2 \text{ (B)}, g_3 \text{ (C)}\}$

I	D	G	Prob.
i^0	d^0	g^1	0.126
i^0	d^0	g^2	0.168
i^0	d^0	g^3	0.126
i^0	d^1	g^1	0.009
i^0	d^1	g^2	0.045
i^0	d^1	g^3	0.126
i^1	d^0	g^1	0.252
i^1	d^0	g^2	0.0224
i^1	d^0	g^3	0.0056
i^1	d^1	g^1	0.06
i^1	d^1	g^2	0.036
i^1	d^1	g^3	0.024

How many parameters do we need to specify the joint distribution?

$$2 * 2 * 3 - 1$$

Example of joint distribution

- Suppose X_1, \dots, X_n are binary (Bernoulli) random variables, i.e., $\text{Val}(X_i) = \{0, 1\}$.
- How many parameters to specify the joint distribution $Pr(x_1, \dots, x_n)$?

$$2^n - 1$$

Working with random variables

Conditioning, chain rule, Bayes' rule, etc. **all apply**

- For example, the **conditional distribution**

$$p(X_1 | X_2 = x_2) = \frac{p(X_1, X_2 = x_2)}{p(X_2 = x_2)}.$$

This notation means

$$p(X_1 = x_1 | X_2 = x_2) = \frac{p(X_1 = x_1, X_2 = x_2)}{p(X_2 = x_2)} \quad \forall x_1 \in \text{Val}(X_1)$$

- Two random variables are **independent**, $X_1 \perp X_2$, if

$$p(X_1 = x_1, X_2 = x_2) = p(X_1 = x_1)p(X_2 = x_2)$$

for all values $x_1 \in \text{Val}(X_1)$ and $x_2 \in \text{Val}(X_2)$.

Inference: Marginalization

- Suppose X and Y are random variables with distribution $p(X, Y)$
 X : Intelligence, $\text{Val}(X) = \{\text{"Very High"}, \text{"High"}\}$
 Y : Grade, $\text{Val}(Y) = \{\text{"a"}, \text{"b"}\}$
- Joint distribution specified by:

		X	
		vh	h
Y	a	0.7	0.15
	b	0.1	0.05

- $p(Y = a) = ? = 0.85$
- More generally, suppose we have a joint $p(X_1, \dots, X_n)$. Then,

$$p(X_i = x_i) = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} p(x_1, \dots, x_n)$$

- If all the X_i are binary, how many terms in the sum? 2^{n-1}

Inference: conditioning

- Suppose X and Y are random variables with distribution $p(X, Y)$

X : Intelligence, $\text{Val}(X) = \{\text{"Very High"}, \text{"High"}\}$

Y : Grade, $\text{Val}(Y) = \{\text{"a"}, \text{"b"}\}$

		X	
		vh	h
Y	a	0.7	0.15
	b	0.1	0.05

- Can compute the conditional probability

$$\begin{aligned} p(Y = a | X = vh) &= \frac{p(Y = a, X = vh)}{p(X = vh)} \\ &= \frac{p(Y = a, X = vh)}{p(Y = a, X = vh) + p(Y = b, X = vh)} \\ &= \frac{0.7}{0.7 + 0.1} = 0.875. \end{aligned}$$

Roadmap

Recall the paradigm:

- ① **Represent** the world as a collection of random variables X_1, \dots, X_n with joint distribution $p(X_1, \dots, X_n)$
- ② (**Learn** the distribution from data)
- ③ Perform “**inference**” (compute conditional distributions $p(X_i | X_1 = x_1, \dots, X_m = x_m)$)

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 10, February 9, 2017

Today's lecture

- Sampling algorithms
 - A very powerful approach to approximate inference

Recap: junction trees

- The Junction Tree algorithms generalize Variable Elimination to the **simultaneous execution** of probability queries
- The algorithms take the form of message passing on a graph called a junction tree, whose nodes are clusters (sets of variables)
- The complexity of the algorithms scales (exponentially) with the **size of the largest clique**

Why Approximate Inference?

- We've seen that inference is NP-hard.
 - Computationally intractable. No hope!
- A tractable class: When the treewidth of the graphical model is small (< 25).
 - Many real world problems have high treewidth.
- In many applications, approximations are sufficient.
 - $P(X_i = x_i | E = e) = 0.29292$
 - Suppose approximate inference yields $\hat{P}(X_i = x_i | E = e) = 0.3$
 - Might be good enough for many applications!

Approximate marginal inference

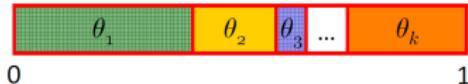
- Two main families of approximate inference algorithms:
 - ① Variational algorithms (e.g., mean-field, loopy BP)
 - ② Monte-Carlo sampling methods (e.g., importance sampling, MCMC)
- The basic idea is to approximate an (intractable, high treewidth) probability distribution using
 - ① Variational: a family of simpler distributions (e.g., corresponding to graphs with low treewidth)
 - ② Monte Carlo: a small number of states that are “representative” of the entire probability distribution

What we will cover

- Sampling fundamentals
- Monte Carlo techniques
 - Rejection Sampling
 - Likelihood Weighting
 - Importance sampling
- Markov Chain Monte Carlo techniques
 - Metropolis-Hastings
 - Gibbs sampling

What is a sample and how to generate one?

- Given a set of variables $X = \{X_1, \dots, X_n\}$, a sample $x = (x_1, \dots, x_n)$ is an **assignment to all variables** (also called an instantiation or a state)
- How to randomly generate a sample/state according to probabilities assigned by $P(x)$?
- Algorithm to draw a sample from a *univariate* distribution $P(X)$. A sample is just an assignment to X . Domain of $X = \{a^0, \dots, a^{k-1}\}$
 - Divide a real line $[0, 1]$ into k intervals such that the width θ_j of the j -th interval is equal to $P(X = a^j)$
 - Draw a random number $r \in [0, 1]$
 - Determine the region j in which r lies. Output a^j

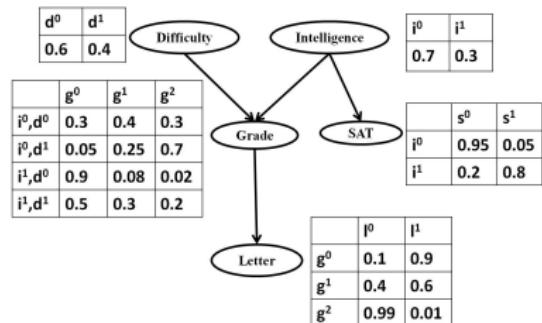


- Example

- Domain of $X_i = \{a_i^0, a_i^1, a_i^2, a_i^3\}$; $P(X_i) = (0.3, 0.25, 0.27, 0.18)$
- Random numbers:
 - $r=0.2929$. $X_i = ?$,
 - $r=0.5209$. $X_i = ?$.

Sampling from a Bayesian network (Forward Sampling)

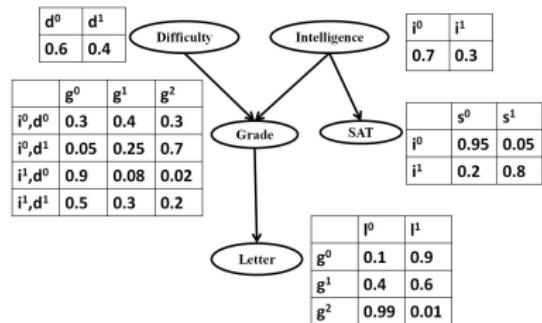
- BN is a *generative* model: Sample variables one by one in a topological order (parents of a node before the node)



- Sample **Difficulty** from $P(D)$.
 $r = 0.723$. $D = ?$
- Sample **Intelligence** from $P(I)$.
 $r = 0.34$. $I = ?$.

Sampling from a Bayesian network (Forward Sampling)

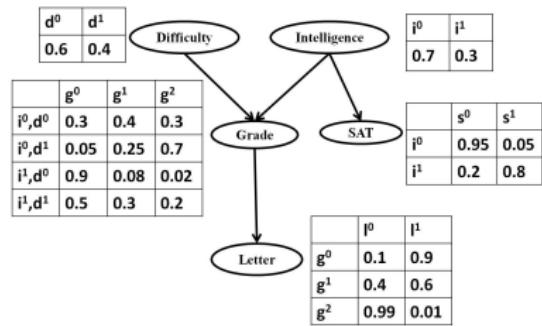
- Sample variables one by one in a topological order (parents of a node before the node)



- Sample **Difficulty** from $P(D)$.
 $r = 0.723$. $D = d^1$
- Sample **Intelligence** from $P(I)$.
 $r = 0.349$. $I = i^0$.
- Sample **Grade** from $P(G|i^0, d^1)$.
 $r = 0.281$, $G = ?$.

Sampling from a Bayesian network (Forward Sampling)

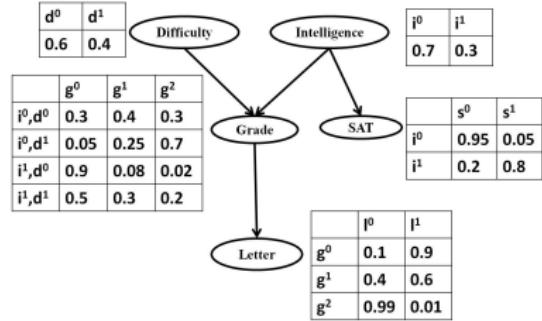
- Sample variables one by one in a topological order (parents of a node before the node)



- Sample **Difficulty** from $P(D)$.
 $r = 0.723$. $D = d^1$
- Sample **Intelligence** from $P(I)$.
 $r = 0.349$. $I = i^0$.
- Sample **Grade** from $P(G|d^1, i^0)$.
 $r = 0.281$, $G = g^1$.
- Sample **SAT** from $P(S|i^0)$.
 $r = 0.992$, $S = ?$.

Sampling from a Bayesian network (Forward Sampling)

- Sample variables one by one in a topological order (parents of a node before the node)



Sample =
 $(d^1, i^0, g^1, s^1, l^0)$

- Sample **Difficulty** from $P(D)$.
 $r = 0.723$. $D = d^1$
- Sample **Intelligence** from $P(I)$.
 $r = 0.349$. $I = i^0$.
- Sample **Grade** from $P(G|i^0, d^1)$.
 $r = 0.281$, $G = g^1$.
- Sample **SAT** from $P(S|i^0)$.
 $r = 0.992$, $S = s^1$.
- Sample **Letter** from $P(L|g^1)$.
 $r = 0.034$, $L = l^0$.

Main idea in Monte Carlo Estimation

- ① Express the quantity of interest as the expected value of a random variable.

$$E_{x \sim P}[g(x)] = \sum_x g(x)P(x)$$

- For example, write a marginal probability as

$$\begin{aligned} P[X_2 = 0] &= \sum_{x_1, x_3, \dots, x_n} P(x_1, 0, \dots, x_n) = \\ &\sum_{x_1, x_2, x_3, \dots, x_n} P(x_1, \dots, x_n) 1[x_2 = 0] = E_P[g(x)] \text{ where} \\ &g(x_1, \dots, x_n) = 1[x_2 = 0] \text{ is an indicator function} \end{aligned}$$

- ② Generate samples from the distribution P with respect to which the expectation was taken.
- ③ Estimate the expected value from the samples using:

$$\hat{g}(x_1, \dots, x_T) \triangleq \frac{1}{T} \sum_{t=1}^T g(x^t)$$

where x^1, \dots, x^T are independent samples from P . Note: \hat{g} is a random variable. Why?

Properties of the Monte Carlo Estimate

- **Unbiased:**

$$E_P[\hat{g}] = E_P[g(x)]$$

- **Convergence:** By law of large numbers

$$\hat{g} = \frac{1}{T} \sum_{t=1}^T g(x^t) \rightarrow E_P[g(x)] \text{ for } T \rightarrow \infty$$

- **Variance:**

$$V_P[\hat{g}] = V_P \left[\frac{1}{T} \sum_{t=1}^T g(x^t) \right] = \frac{V_P[g(x)]}{T}$$

Thus, variance of the estimator can be reduced by increasing the number of samples. We have no control over the numerator when P is given. How quickly does the estimate converge to the true expectation?

Concentration

- There are two general results we can use, depending on whether we care about additive or multiplicative error
 - ① Hoeffding bound says that:

$$P_{x_i \sim P(x)}[E_P[g(x)] - \epsilon \leq \hat{g}(x_1, \dots, x_T) \leq E_P[g(x)] + \epsilon] \geq 1 - 2e^{-2T\epsilon^2}$$

- ② Chernoff bound:

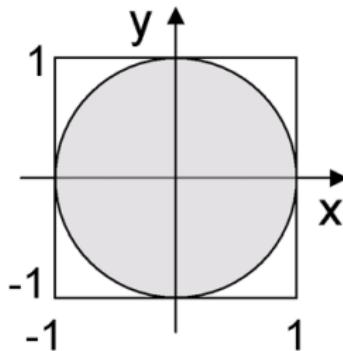
$$P_{x_i \sim P(x)}[E_P[g(x)](1-\epsilon) \leq \hat{g}(x_1, \dots, x_T) \leq E_P[g(x)](1+\epsilon)] \geq 1 - 2e^{-2T\frac{\epsilon^2}{3}E_P[g(x)]}$$

- **Error probability decreases exponentially fast in the number of samples T**
 - Example. $\epsilon = 0.2$, $T = 20$ samples, then $1 - 2e^{-2T\epsilon^2} \approx 0.6$
 - Example. $\epsilon = 0.2$, $T = 30$ samples, then $1 - 2e^{-2T\epsilon^2} \approx 0.82$
 - Example. $\epsilon = 0.2$, $T = 40$ samples, then $1 - 2e^{-2T\epsilon^2} \approx 0.92$
- *Estimating single-variable marginals for a BN is easy: just forward sample!* Approximate, but can bound error with high probability. What about computing conditional queries such as $p(X = x | E = e)$?

What we will cover

- Sampling fundamentals
- Monte Carlo techniques
 - Rejection Sampling
 - Likelihood Weighting
 - Importance sampling
- Markov Chain Monte Carlo techniques
 - Metropolis-Hastings
 - Gibbs sampling

Rejection Sampling: geometric idea



- Suppose you want to sample points uniformly within the circle
- You have access to a uniform random generator in $[-1, 1]$
- Sample $x \sim \mathcal{U}[-1, 1]$
- Sample $y \sim \mathcal{U}[-1, 1]$
- If $x^2 + y^2 \leq 1$, accept the sample. Otherwise reject it and try again.

Rejection Sampling

- Express $P(E = e)$ as an expectation:

$$P(E = e) = \sum_x \delta_e(x) P(x) = E_P[\delta_e(x)]$$

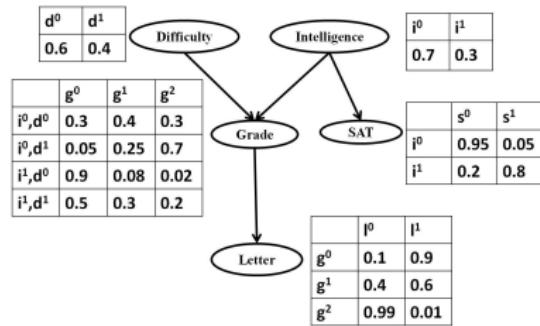
where $\delta_e(x)$ is an indicator function which is 1 if x is consistent with the evidence $E = e$ and 0 otherwise.

- Generate samples from the Bayesian network.
- Monte Carlo estimate $\hat{g}(x_1, \dots, x_T) = \frac{1}{T} \sum_{t=1}^T g(x^t)$:

$$\hat{P}(E = e) = \frac{\text{Number of samples that have } E = e}{\text{Total number of samples}}$$

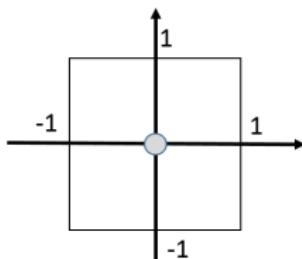
- Issues: If $P(E = e)$ is very small (e.g., 10^{-55}), nearly all samples will be rejected.
- Note: even if $P(E = e)$ is extremely small, $p(X = x | E = e) = p(X = x, E = e)/p(E = e)$ can be large.

Rejection Sampling: Example



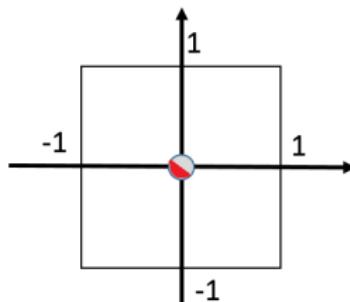
- Let the evidence be $e = (i^0, g^1, s^1, l^0)$
- Probability of evidence
 $= 0.00475$
- On an average, you will need approximately $1/0.00475 \approx 210$ samples to get **one** sample consistent with the evidence
 $(Intelligence, Grade, SAT, Letter) = e$
- Imagine how many samples will be needed if $P(E = e)$ is very small!

Rejection Sampling: failure case



- Suppose you want to sample points uniformly within the circle
- You have access to a uniform random generator in $[-1, 1]$
- Sample $x \sim \mathcal{U}[-1, 1]$, sample $y \sim \mathcal{U}[-1, 1]$
- If (x, y) is in the circle, accept the sample. Otherwise reject it and try again.
- Can be extremely inefficient if the circle is small

Rejection Sampling: failure case



- Suppose you want to sample points uniformly within the circle
- You have access to a uniform random generator in $[-1, 1]$
- Sample $x \sim \mathcal{U}[-1, 1]$, sample $y \sim \mathcal{U}[-1, 1]$
- If (x, y) is in the circle, accept the sample. Otherwise reject it and try again.
- Can be extremely inefficient if the circle is small
- A conditional probability is like the ratio between the red vs. gray circle areas. Can we sample directly inside the gray circle?

Importance Sampling

- Idea: evidence variables are fixed, so let's just sample over non-evidence ones
- Idea: use a **proposal distribution** over non-evidence variables

$Q(Z = X \setminus E)$ that we **can efficiently sample from** and such that
 $P(Z = z, E = e) > 0 \Rightarrow Q(Z = z) > 0$. Express $P(E = e)$ as follows:

$$\begin{aligned} P(E = e) &= \sum_z P(Z = z, E = e) \\ &= \sum_z P(Z = z, E = e) \frac{Q(Z = z)}{Q(Z = z)} \\ &= E_Q \left[\frac{P(Z = z, E = e)}{Q(Z = z)} \right] = E_Q[w(z)] \end{aligned}$$

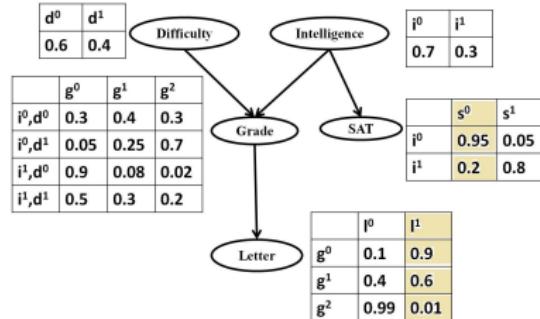
- Generate samples from Q and estimate using $P(E = e)$ using the following Monte Carlo estimate:

$$\hat{P}(E = e) = \frac{1}{T} \sum_{t=1}^T \frac{P(Z = z^t, E = e)}{Q(Z = z^t)} = \frac{1}{T} \sum_{t=1}^T w(z^t)$$

where (z^1, \dots, z^T) are sampled from Q .

Importance Sampling: Example

- Let $L = I^1, S = s^0$ be the evidence
- Consider a **uniform proposal distribution** Q defined over (D, I, G) . Suppose we generate some samples $z^t = (d^t, i^t, g^t)$ from Q .
- Using the previous formula, $\hat{P}(E = e) = \frac{1}{T} \sum_{t=1}^T w(z^t) = \text{Average of } \{P(\text{sample}, \text{evidence})/Q(\text{sample})\}$



- $\text{sample} = (d^0, i^0, g^0)$,
 $P(\text{sample}, \text{evidence}) = 0.6 \times 0.7 \times 0.3 \times 0.9 \times 0.95$,
 $Q(\text{sample}) = 0.5 \times 0.5 \times 0.333$
- $\text{sample} = (d^1, i^0, g^0)$,
 $P(\text{sample}, \text{evidence}) = 0.4 \times 0.7 \times 0.05 \times 0.9 \times 0.95$,
 $Q(\text{sample}) = 0.5 \times 0.5 \times 0.333$
- and so on

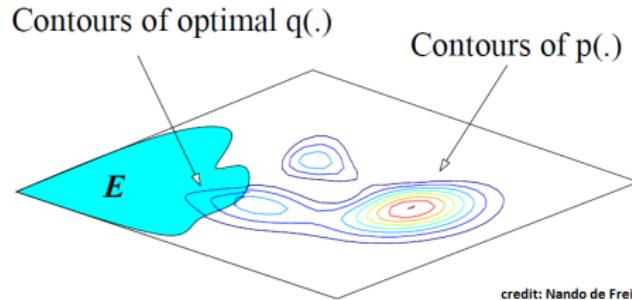
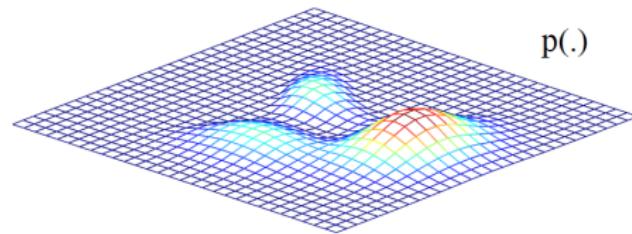
Importance sampling: Issues

- For optimum performance, the proposal distribution Q should be as close as possible to $P(Z|E = e)$.
 - When $Q = P(Z|E = e)$, the weight of every sample is $P(E = e)$!

$$\begin{aligned} w(z^t) &= \frac{P(Z = z^t, E = e)}{Q(Z = z^t)} &= \frac{P(Z = z^t, E = e)}{P(Z = z^t | E = e)} \\ &= \frac{P(Z = z^t, E = e)P(E = e)}{P(Z = z^t, E = e)} \\ &= P(E = e) \end{aligned}$$

- Weight does not depend on z^t
- One sample would be sufficient!
- However, sampling from $P(Z|E = e)$ is NP-hard.
- Need a proposal that is tractable (easy to sample from), and “close” to $P(Z|E = e)$. Picking a good proposal is an art.

Importance sampling and rare events



Likelihood weighting

- How can we make Q close to $P(Z|E = e)$?
- **Likelihood weighting:** a special kind of Importance sampling in which Q is given by doing forward sampling on the network obtained by **clamping evidence** in the original net.

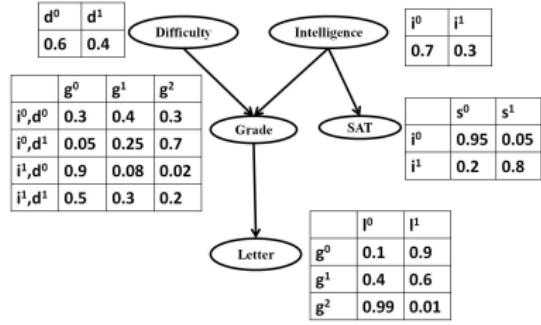
$$Q(x) = \prod_{v \notin E} P(x_v | x_{pa(v)})$$

- Note: pays attention to evidence in ancestors only, thus somewhere “in between” prior and posterior distribution
- $P(\text{sample}, \text{evidence})/Q(\text{sample})$ can be efficiently computed:

$$w(x) = \frac{\prod_{v \in \{Z \cup E\}} P(x_v | x_{pa(v)})}{\prod_{v \notin E} P(x_v | x_{pa(v)})} = \prod_{v \in E} P(x_v | x_{pa(v)})$$

Likelihood weighting

- **Likelihood weighting:** a special kind of Importance sampling in which Q is given by the network obtained by **clamping evidence** in the original net.
- Evidence = (g^0, s^0)

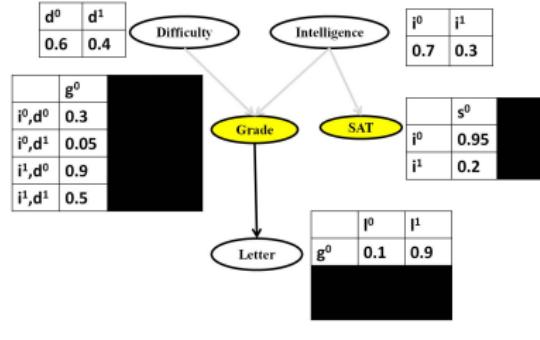


- The ratio equals the product of the corresponding CPT values at the evidence nodes. The remaining values cancel out.
- Let the sample = (d^0, i^0, l^1) .

$$\frac{P(\text{sample}, \text{evidence})}{Q(\text{sample})} = 0.3 \times 0.95$$

Likelihood weighting

- **Likelihood weighting:** a special kind of Importance sampling in which Q is given by the network obtained by **clamping evidence** in the original net.
- Evidence = (g^0, s^0)



- The ratio equals the product of the corresponding CPT values at the evidence nodes. The remaining values cancel out.
- Let the sample = (d^0, i^0, i^1) .

$$\frac{P(\text{sample}, \text{evidence})}{Q(\text{sample})} = 0.3 \times 0.95$$

Importance sampling: Issues

- For optimal performance, the proposal distribution Q should be as close as possible to $P(X|E = e)$.
 - When $Q = P(X|E = e)$, the weight of every sample is $P(E = e)!$ However, achieving this is NP-hard.
- Likelihood weighting performs poorly when evidence is at the leaves and is unlikely.
- In particular, designing a good proposal distribution is an art rather than a science!

Normalized Importance sampling

- (Un-normalized) IS is not suitable for estimating $P(X_i = x_i | E = e)$.
- One option: Estimate the numerator and denominator by IS.

$$\hat{P}(X_i = x_i | E = e) = \frac{\hat{P}(X_i = x_i, E = e)}{\hat{P}(E = e)}$$

- This ratio estimate can be inaccurate because errors in the numerator and denominator may be cumulative.
 - For example, if the numerator is an under-estimate and the denominator is an over-estimate.
- Other times, we only know P up to a constant factor, i.e. we have $P = \frac{1}{Z} P'$ for unknown Z (MRF case)
- How to fix this? **Use: Normalized importance sampling.**

Normalized Importance sampling: Theory

- Partition the variables into evidence E and non-evidence Z
- Given an indicator function $\delta_{x_i}(z)$ (which is 1 if z is consistent with $X_i = x_i$ and 0 otherwise), we can write $P(X_i = x_i | E = e)$ as:

$$P(X_i = x_i | E = e) = \frac{P(X_i = x_i, E = e)}{P(E = e)} = \frac{\sum_z \delta_{x_i}(z) P(Z = z, E = e)}{\sum_z P(Z = z, E = e)}$$

- Now we can use the same Q and **same samples** from it to estimate both the numerator and the denominator.

$$\hat{P}(X_i = x_i | E = e) = \frac{\frac{1}{T} \sum_{t=1}^T \delta_{x_i}(z^t) w(z^t)}{\frac{1}{T} \sum_{t=1}^T w(z^t)}$$

Normalized Importance sampling: Properties

- Normalized IS:

$$\hat{P}(X_i = x_i | E = e) = \frac{\sum_{t=1}^T \delta_{x_i}(z^t) w(z^t)}{\sum_{t=1}^T w(z^t)}$$

- Biased Estimator. If $T = 1$, z^1 is a sample from Q

$$E_Q[\hat{P}(X_i = x_i | E = e)] = E_Q\left[\frac{\delta_{x_i}(z^1) w(z^1)}{w(z^1)}\right] = E_Q[\delta_{x_i}(z^1)]$$

- Asymptotically Unbiased (numerator and denominator are both unbiased):

$$\lim_{T \rightarrow \infty} E_Q[\hat{P}(X_i = x_i | E = e)] = P(X_i = x_i | E = e)$$

Summary

- Importance sampling
 - Generate samples from a proposal distribution
 - Performance depends on how close the proposal is to the posterior distribution
- Next: Markov chain Monte Carlo (MCMC) sampling
 - Attempts to generate samples from the posterior distribution by creating a Markov chain whose stationary distribution equals the posterior distribution
 - Works for undirected models

Markov chain Monte Carlo

Lecture 11

Stefano Ermon
Stanford

Slides adapted from Eric Xing, Qirong Ho (CMU), and David Sontag (NYU)

Announcements

- Please complete online course evaluation. Today is the last day.

Monte Carlo Estimation

- Express the quantity of interest as the expected value of a random variable.

$$E_P[g(x)] = \sum_x g(x)P(x)$$

- For example, write a marginal probability as
 $P[X_1 = x_1] = \sum_x p(x)1[X_1 = x_1] = E_P[g(x)]$ where
 $g(x) = 1[X_1 = x_1]$
- Generate samples from the distribution P with respect to which the expectation was taken.
- Estimate the expected value from the samples using:

$$\hat{g}(x_1, \dots, x_T) = \frac{1}{T} \sum_{t=1}^T g(x^t)$$

where x^1, \dots, x^T are independent samples from P .

Simple Rejection Sampling

- Express $P(E = e)$ as an expectation:

$$\begin{aligned} P(E = e) &= \sum_x \delta_e(x) P(x) \\ &= E_P[\delta_e(x)] \end{aligned}$$

where $\delta_e(x)$ is an indicator function which is 1 if x is consistent with the evidence $E = e$ and 0 otherwise.

- Generate samples from the Bayesian network.
- Monte Carlo estimate $\hat{g}(x_1, \dots, x_T) = \frac{1}{T} \sum_{t=1}^T g(x^t)$:

$$\hat{P}(E = e) = \frac{\text{Number of samples that have } E = e}{\text{Total number of samples}}$$

- Issues: If $P(E = e)$ is very small (e.g., 10^{-55}), all samples will be rejected.

Importance sampling

- Idea: use a **proposal distribution** over non-evidence variables $Q(Z = X \setminus E)$ that we **can efficiently sample from** and such that $P(Z = z, E = e) > 0 \Rightarrow Q(Z = z) > 0$. Express $P(E = e)$ as follows:

$$\begin{aligned} P(E = e) &= \sum_z P(Z = z, E = e) \\ &= \sum_z P(Z = z, E = e) \frac{Q(Z = z)}{Q(Z = z)} \\ &= E_Q \left[\frac{P(Z = z, E = e)}{Q(Z = z)} \right] = E_Q[w(z)] \end{aligned}$$

- Generate samples from Q and estimate using $P(E = e)$ using the following Monte Carlo estimate:

$$\hat{P}(E = e) = \frac{1}{T} \sum_{t=1}^T \frac{P(Z = z^t, E = e)}{Q(Z = z^t)} = \frac{1}{T} \sum_{t=1}^T w(z^t)$$

where (z^1, \dots, z^T) are sampled from Q .

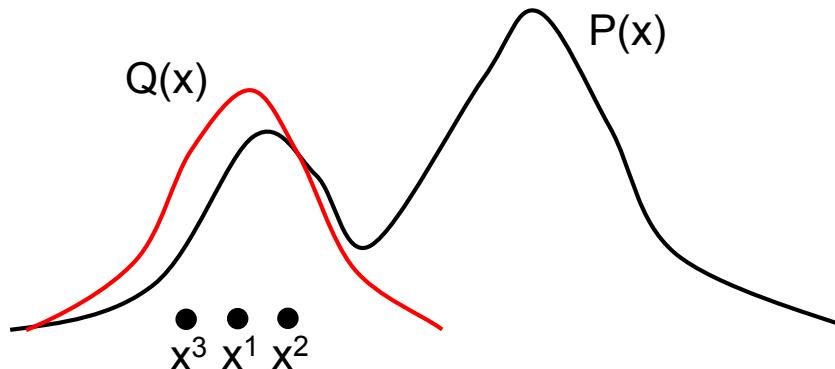
Limitations of Monte Carlo

- Forward (unconditional) sampling
 - Hard to get rare events in high-dimensional spaces
 - Infeasible for MRFs, unless we know the normalizer Z
- Importance sampling
 - Do not work well if the proposal $Q(x)$ is very different from $P(x)$
 - Yet constructing a $Q(x)$ similar to $P(x)$ can be difficult
 - Making a good proposal usually requires knowledge of the analytic form of $P(x)$ – but if we had that, we wouldn't even need to sample!
- Intuition: instead of a fixed proposal $Q(x)$, what if we could use an **adaptive** proposal?

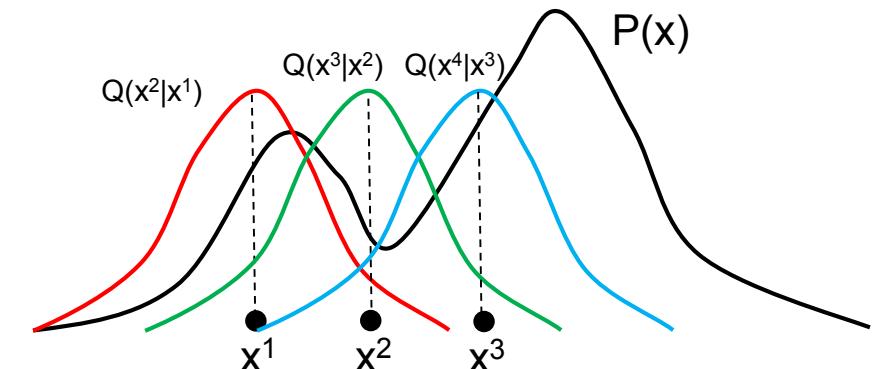
Markov Chain Monte Carlo

- MCMC algorithms feature adaptive proposals
 - Instead of $Q(x')$, they use $Q(x'|x)$ where x' is the new state being sampled, and x is the previous sample
 - As x changes, $Q(x'|x)$ can also change (as a function of x')

Importance sampling with
a (bad) proposal $Q(x)$



MCMC with adaptive
proposal $Q(x'|x)$



Metropolis-Hastings

- Let's see how MCMC works in practice
 - Later, we'll look at the theoretical aspects
- Metropolis-Hastings algorithm
 - Draws a sample x' from $Q(x'|x)$, where x is the previous sample
 - The new sample x' is **accepted** or **rejected** with some probability $A(x'|x)$
 - This acceptance probability is
$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$
 - $A(x'|x)$ is like a ratio of importance sampling weights
 - $P(x')/Q(x'|x)$ is the importance weight for x' , $P(x)/Q(x|x')$ is the importance weight for x
 - We divide the importance weight for x' by that of x
 - Notice that we only need to compute $P(x')/P(x)$ rather than $P(x')$ or $P(x)$ separately
 - $A(x'|x)$ ensures that, after sufficiently many draws, our samples will come from the true distribution $P(x)$ – we shall learn why later in this lecture

The MH Algorithm

1. Initialize starting state $x^{(0)}$, set $t = 0$
2. Burn-in: while samples have “not converged”

- $x = x^{(t)}$
- $t = t + 1,$
- sample $x^* \sim Q(x^*|x)$ // draw from proposal
- sample $u \sim \text{Uniform}(0,1)$ // draw acceptance threshold
- - if $u < A(x^*|x) = \min\left(1, \frac{P(x^*)Q(x|x^*)}{P(x)Q(x^*|x)}\right)$
- $x^{(t)} = x^*$ // transition
- - else
- $x^{(t)} = x$ // stay in current state

Function
Draw sample ($x(t)$)

- Take samples from $P(x)$: Reset $t=0$, for $t = 1:N$
 - $x(t+1) \leftarrow \text{Draw sample } (x(t))$
- (Monte Carlo Estimation using the samples)

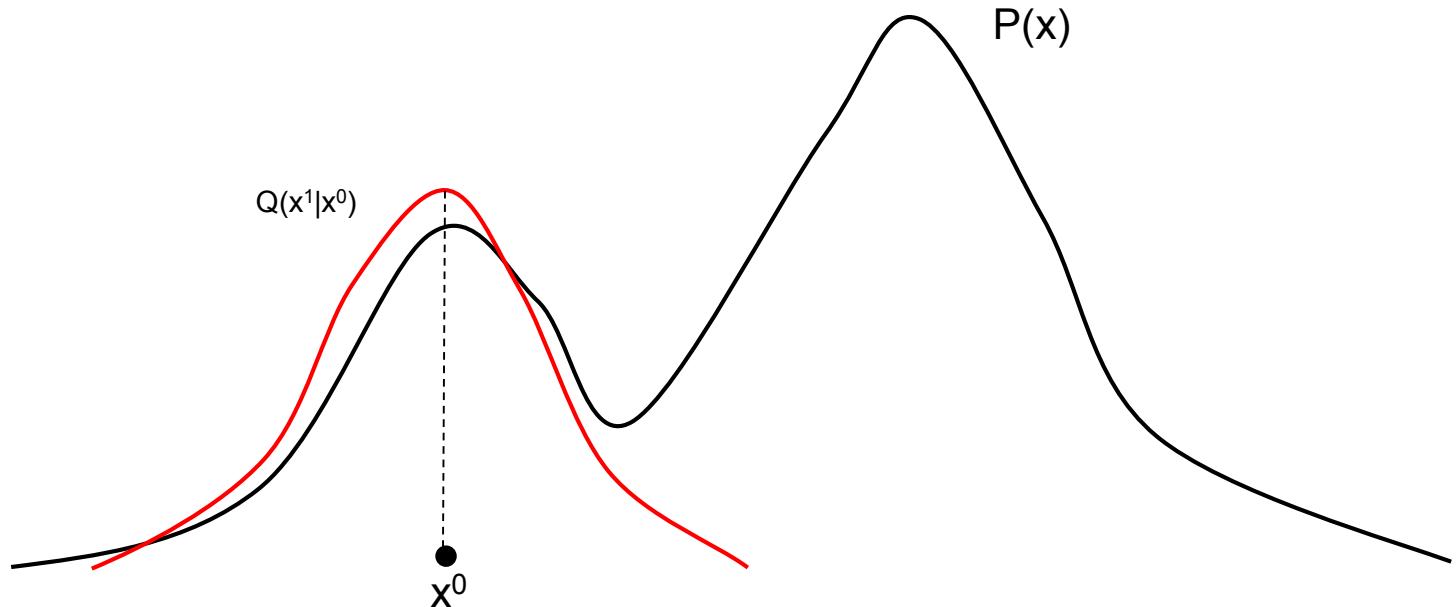
$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$

...

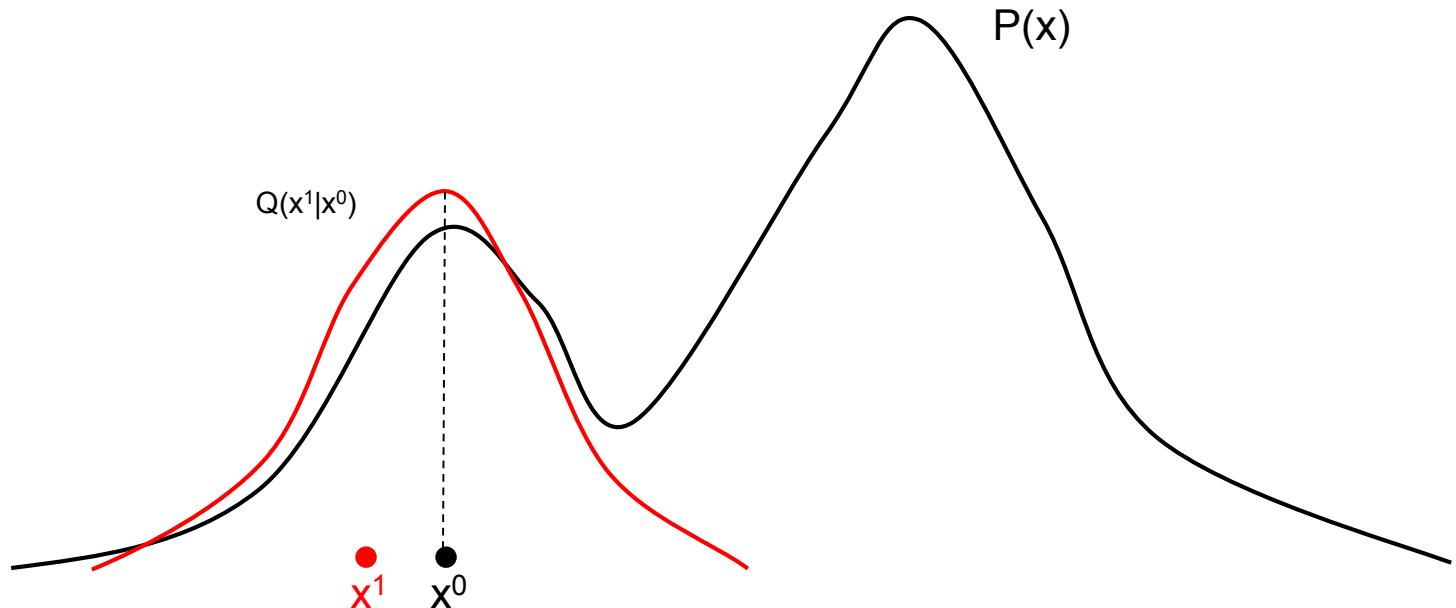


$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$
Draw, accept x^1

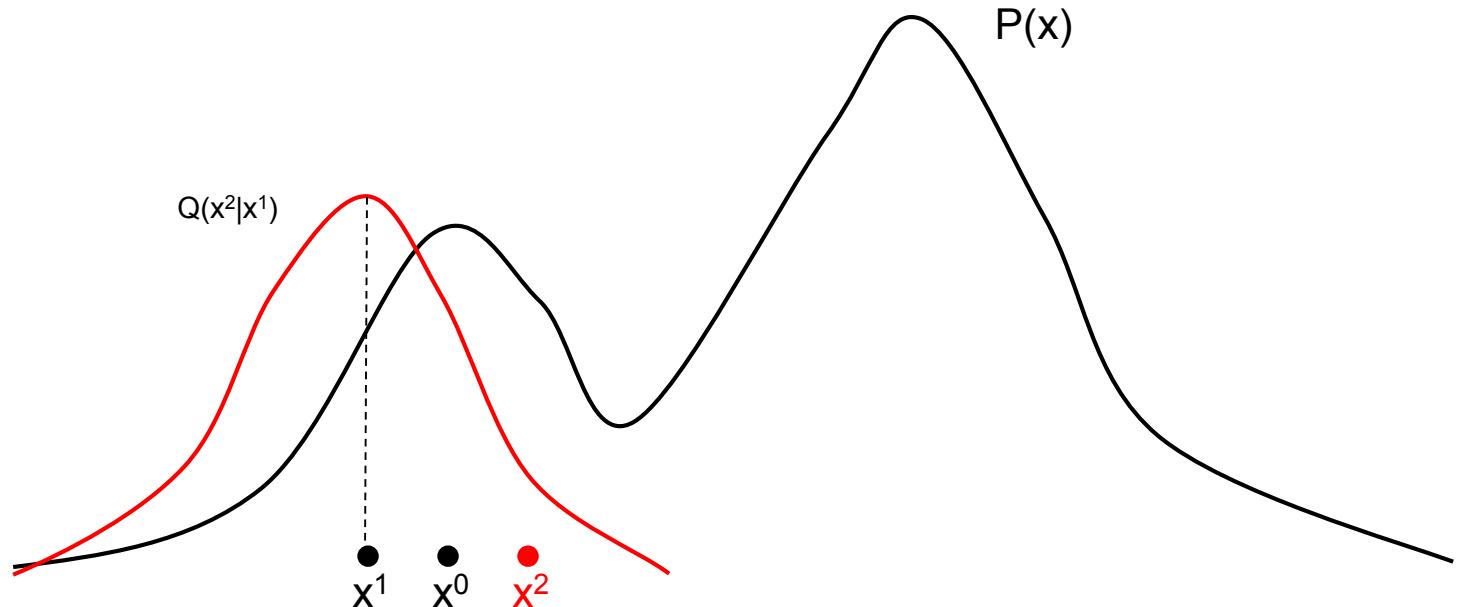


$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$
 Draw, accept x^1
 Draw, accept x^2



$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

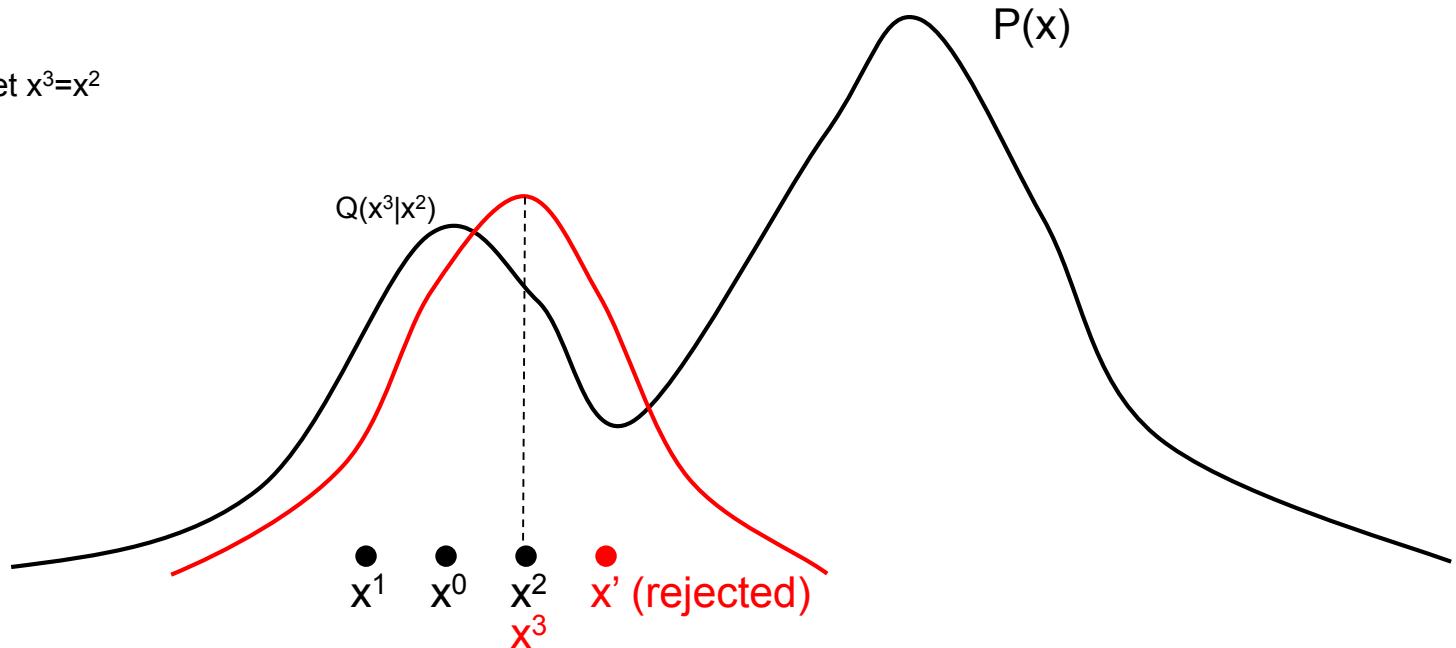
- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$

Draw, accept x^1

Draw, accept x^2

Draw but reject; set $x^3=x^2$



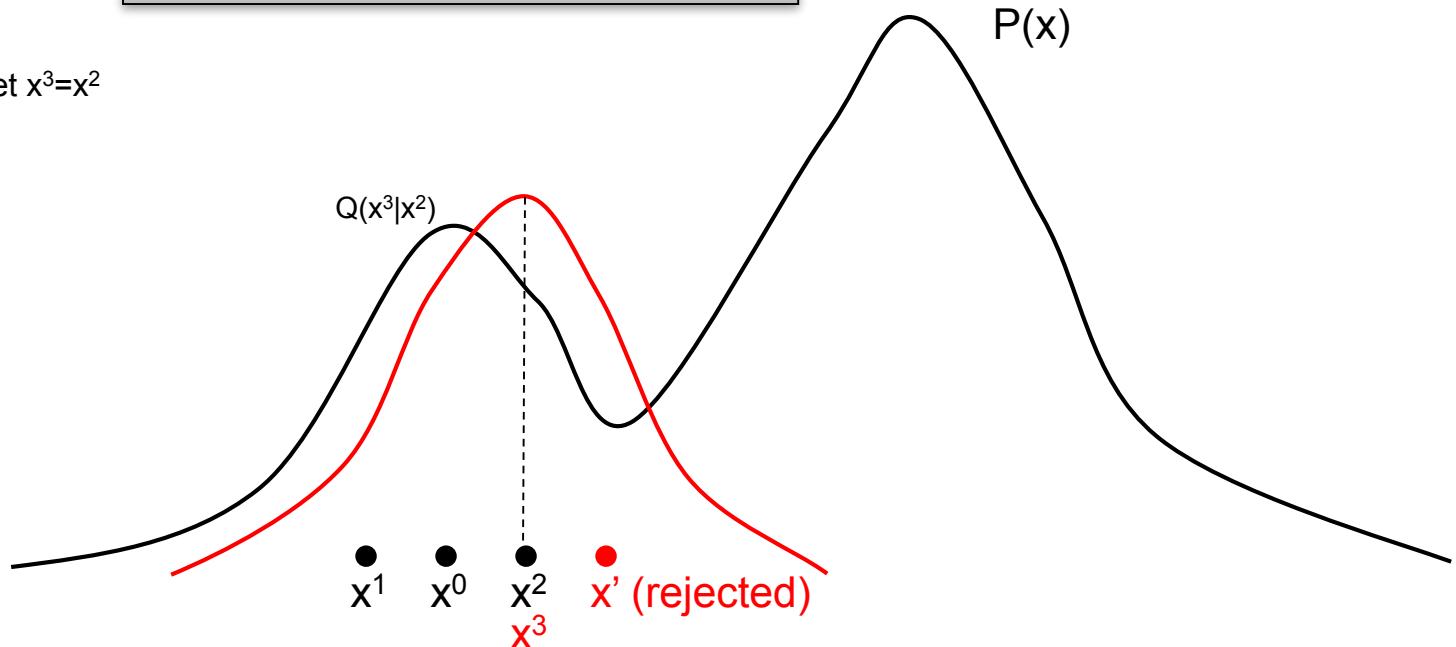
$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$
 Draw, accept x^1
 Draw, accept x^2
 Draw but reject; set $x^3=x^2$

We reject because $P(x')/P(x^2)$ is very small,
hence $A(x'|x^2)$ is close to zero!



$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

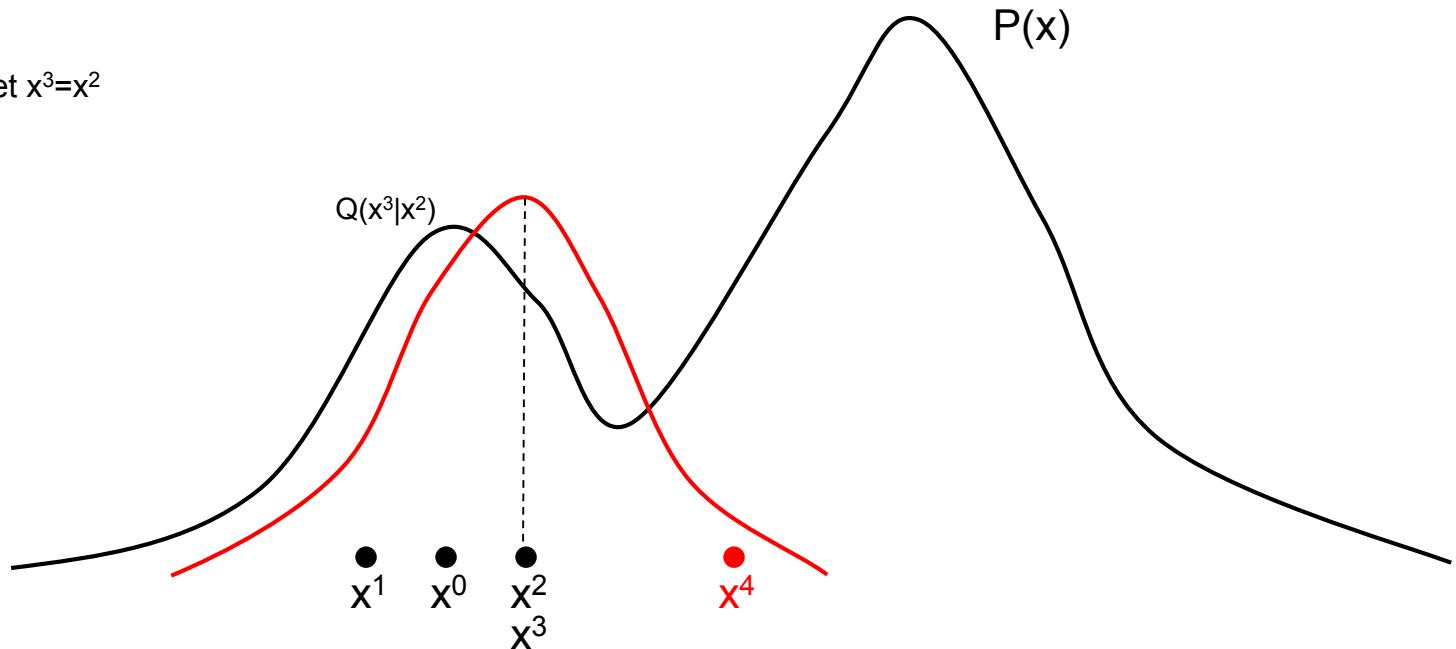
Initialize $x^{(0)}$

Draw, accept x^1

Draw, accept x^2

Draw but reject; set $x^3=x^2$

Draw, accept x^4



$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$

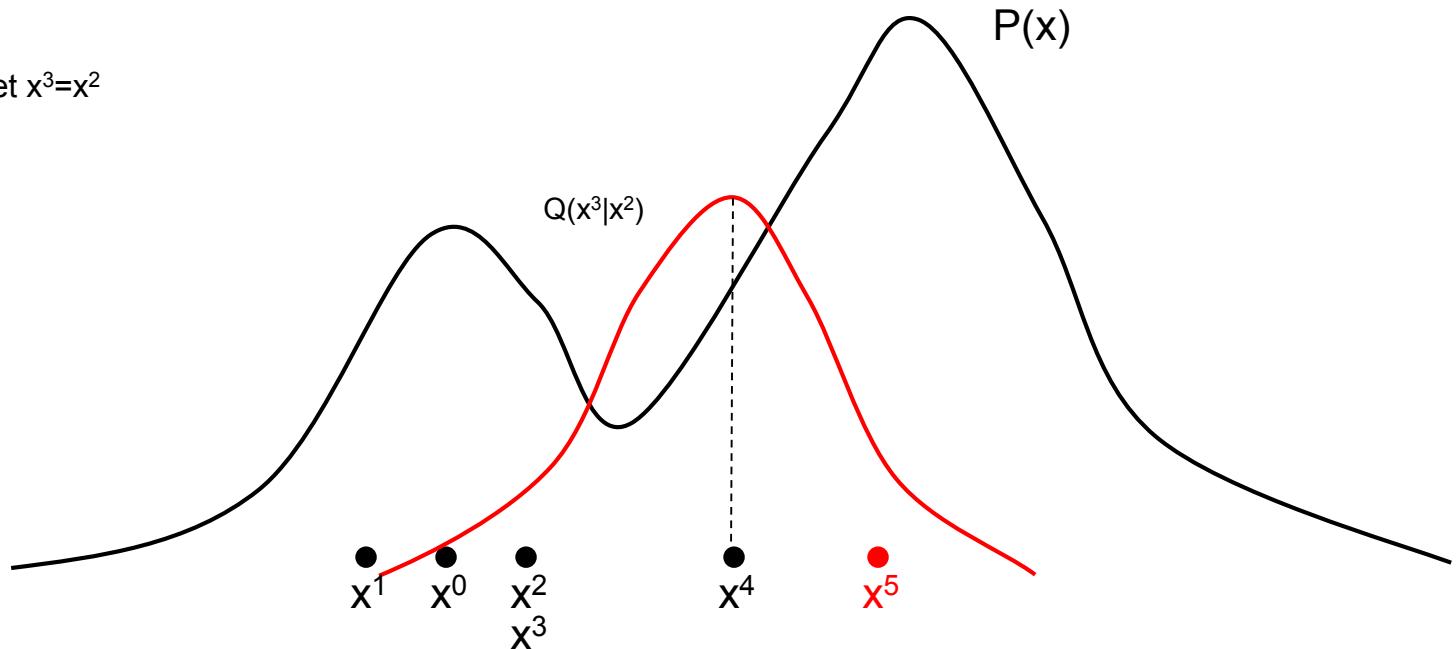
Draw, accept x^1

Draw, accept x^2

Draw but reject; set $x^3=x^2$

Draw, accept x^4

Draw, accept x^5



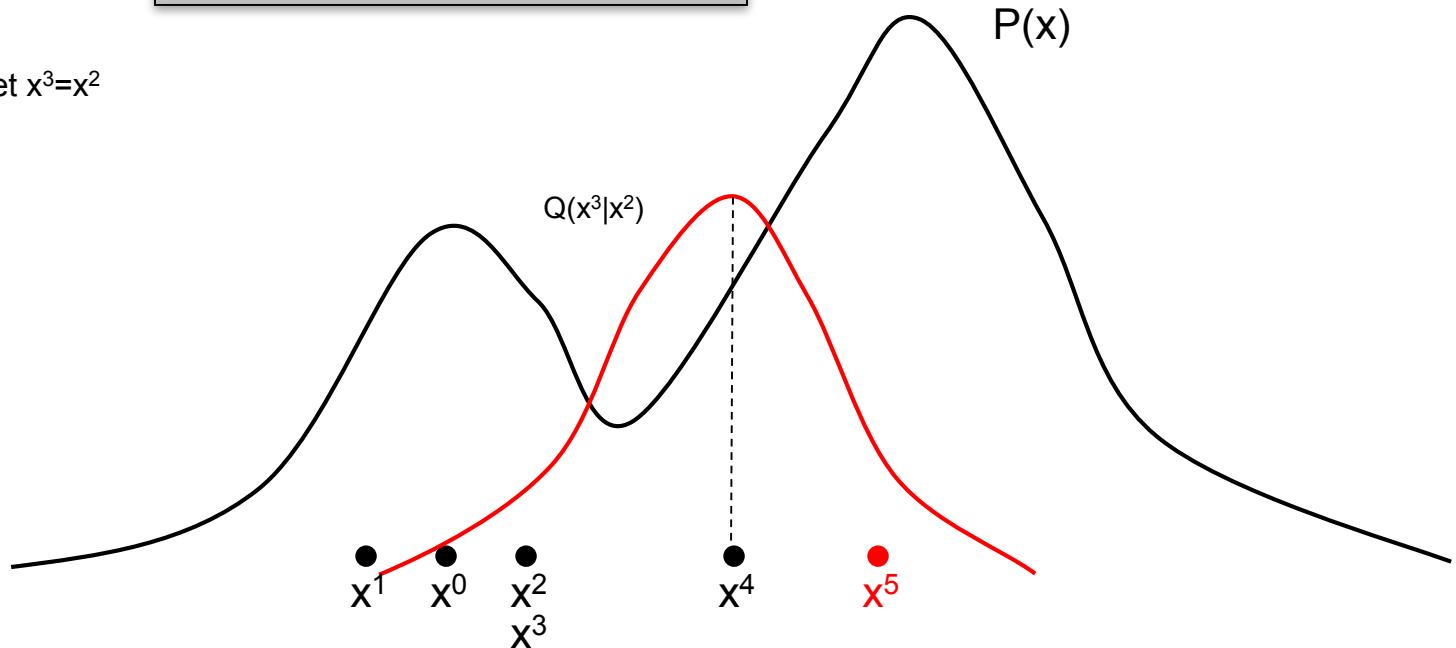
$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

The MH Algorithm

- Example:
 - Let $Q(x'|x)$ be a Gaussian centered on x
 - We're trying to sample from a bimodal distribution $P(x)$

Initialize $x^{(0)}$
 Draw, accept x^1
 Draw, accept x^2
 Draw but reject; set $x^3=x^2$
 Draw, accept x^4
 Draw, accept x^5

The adaptive proposal $Q(x'|x)$ allows us to sample both modes of $P(x)$!



Theoretical aspects of MCMC

- The MH algorithm has a “burn-in” period
 - Why do we throw away samples from burn-in?
- Why are the MH samples guaranteed to be from $P(x)$?
 - The proposal $Q(x'|x)$ keeps changing with the value of x ; how do we know the samples will eventually come from $P(x)$?
 - Has to do with the connection between Markov chains & MCMC
 - We will return to this later
- What are good, general-purpose, proposal distributions?

Gibbs Sampling

- Gibbs Sampling is an MCMC algorithm that samples each random variable of a graphical model, one at a time
 - GS is a special case of the MH algorithm
- Gibbs Sampling algorithms...
 - Are fairly easy to derive for many graphical models
 - Have reasonable computation and memory requirements, because they sample one random variable at a time

Gibbs Sampling

- The GS algorithm:
 1. Suppose the graphical model contains variables x_1, \dots, x_n
 2. Initialize starting values for x_1, \dots, x_n
 3. Do until convergence:
 1. Pick an ordering of the n variables (can be fixed or random)
 2. For each variable x_i in order:
 1. Sample $x \sim P(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$, i.e. the conditional distribution of x_i given the current values of all other variables
 2. Update $x_i \leftarrow x$
-
- When we update x_i , we immediately use its new value for sampling other variables x_j

Markov Blankets

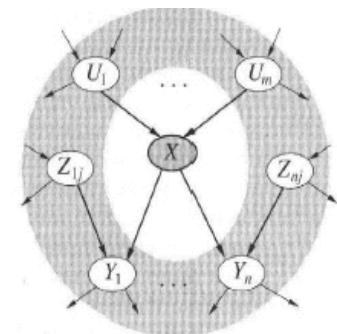
- The conditional $P(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ looks intimidating, but recall Markov Blankets:

- Let $MB(x_i)$ be the Markov Blanket of x_i , then

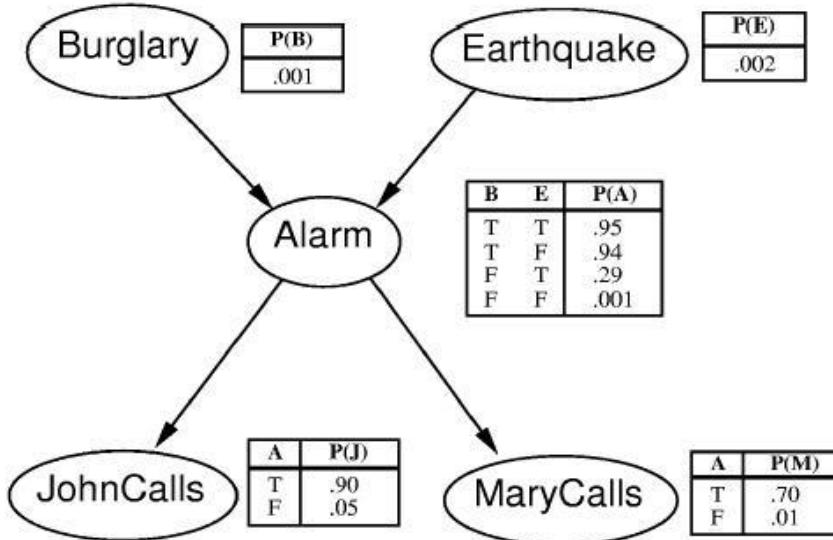
$$P(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | MB(x_i))$$

- For a BN, the Markov Blanket of x_i is the set containing its parents, children, and co-parents

- For an MRF, the Markov Blanket of x_i is its immediate neighbors



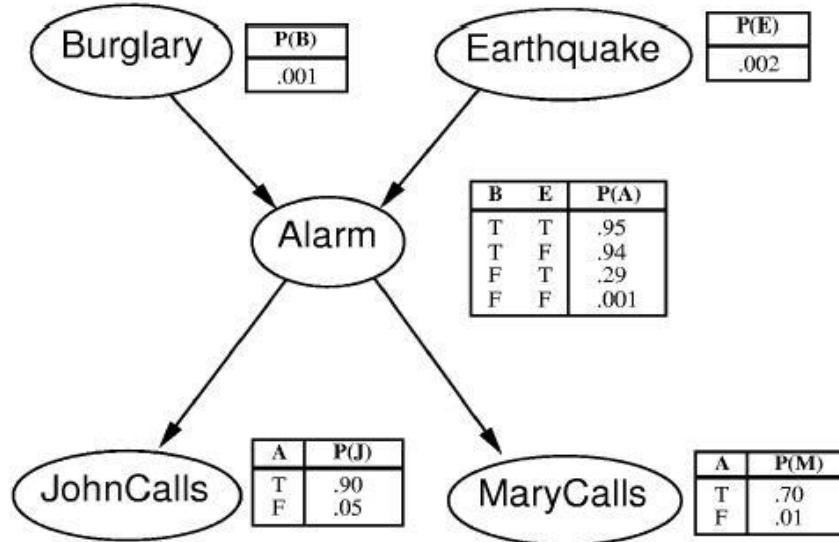
Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1					
2					
3					
4					

- Consider the alarm network
 - Assume we sample variables in the order B,E,A,J,M
 - Initialize all variables at t = 0 to False

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F				
2					
3					
4					

- Sampling $P(B|A,E)$ at $t = 1$: Using Bayes Rule,

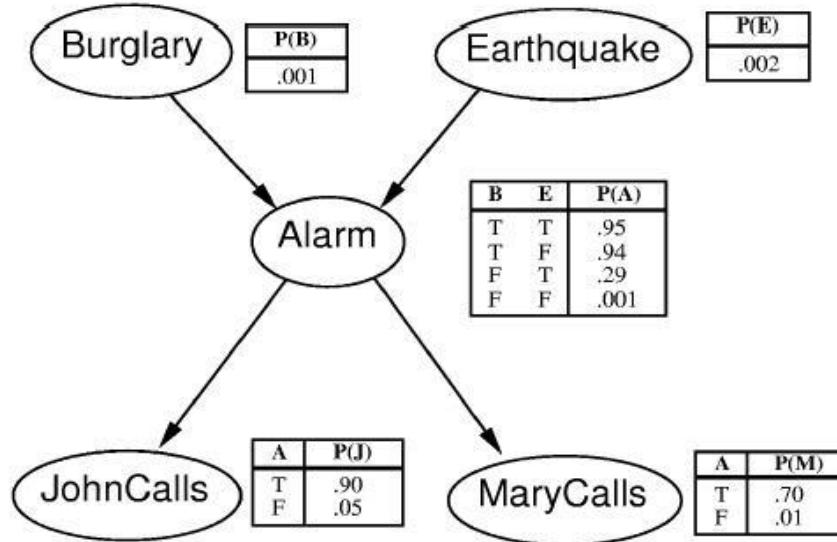
$$P(B | A, E) \propto P(A | B, E)P(B)$$

- $A=\text{false}$, $E=\text{false}$, so we compute:

$$P(B = T | A = F, E = F) \propto (0.06)(0.001) = 0.00006$$

$$P(B = F | A = F, E = F) \propto (0.999)(0.999) = 0.9980$$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T			
2					
3					
4					

- Sampling $P(E|A,B)$: Using Bayes Rule,

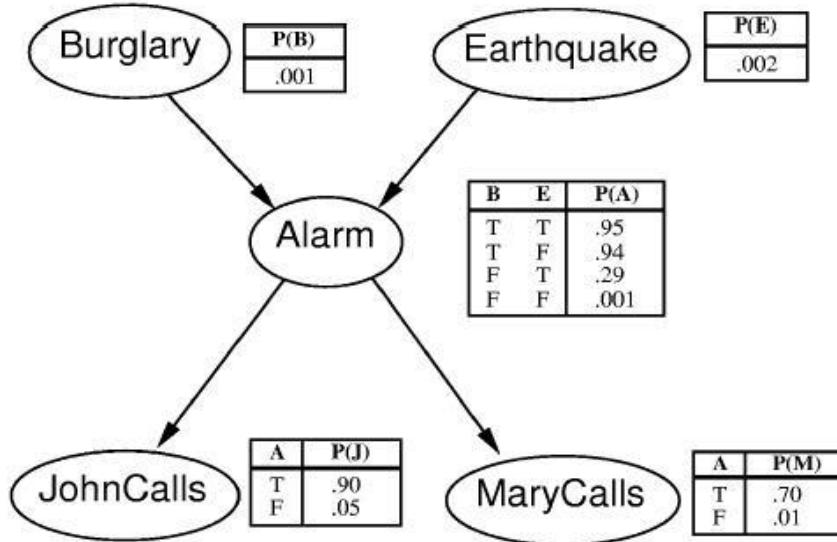
$$P(E | A, B) \propto P(A | B, E)P(E)$$

- $(A,B) = (F,F)$, so we compute the following,

$$P(E = T | A = F, B = F) \propto (0.71)(0.02) = 0.0142$$

$$P(E = F | A = F, B = F) \propto (0.999)(0.998) = 0.9970$$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F		
2					
3					
4					

- Sampling $P(A|B,E,J,M)$: Using Bayes Rule,

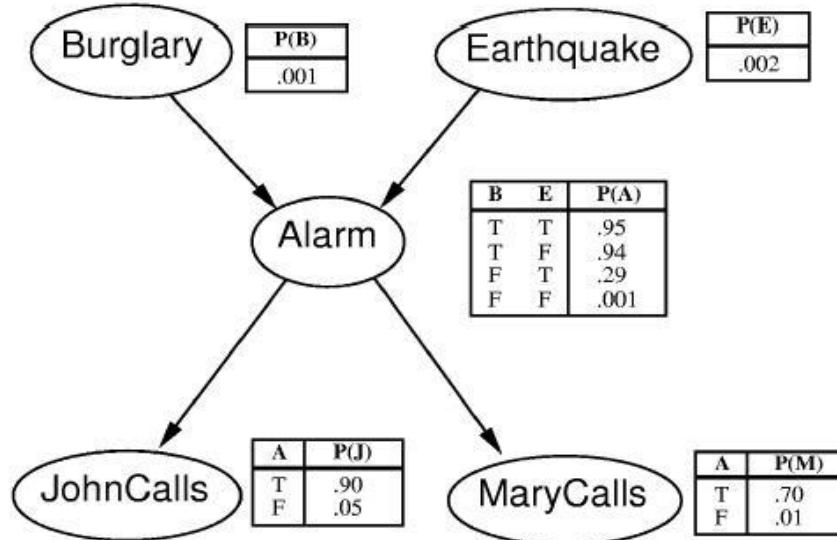
$$P(A | B, E, J, M) \propto P(J | A)P(M | A)P(A | B, E)$$

- $(B, E, J, M) = (F, T, F, F)$, so we compute:

$$P(A = T | B = F, E = T, J = F, M = F) \propto (0.1)(0.3)(0.29) = 0.0087$$

$$P(A = F | B = F, E = T, J = F, M = F) \propto (0.95)(0.99)(0.71) = 0.6678$$

Gibbs Sampling: An Example



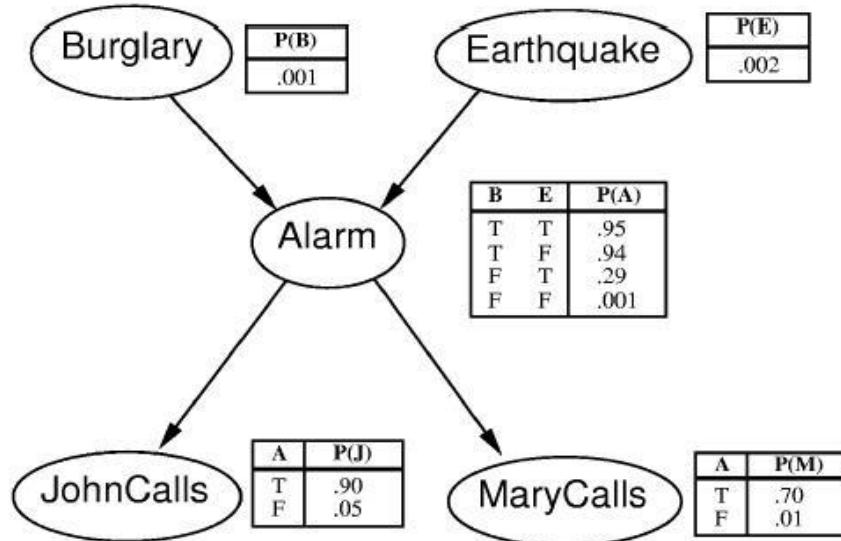
t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	
2					
3					
4					

- Sampling $P(J|A)$: No need to apply Bayes Rule
- $A = F$, so we compute the following, and sample

$$P(J = T | A = F) \propto 0.05$$

$$P(J = F | A = F) \propto 0.95$$

Gibbs Sampling: An Example



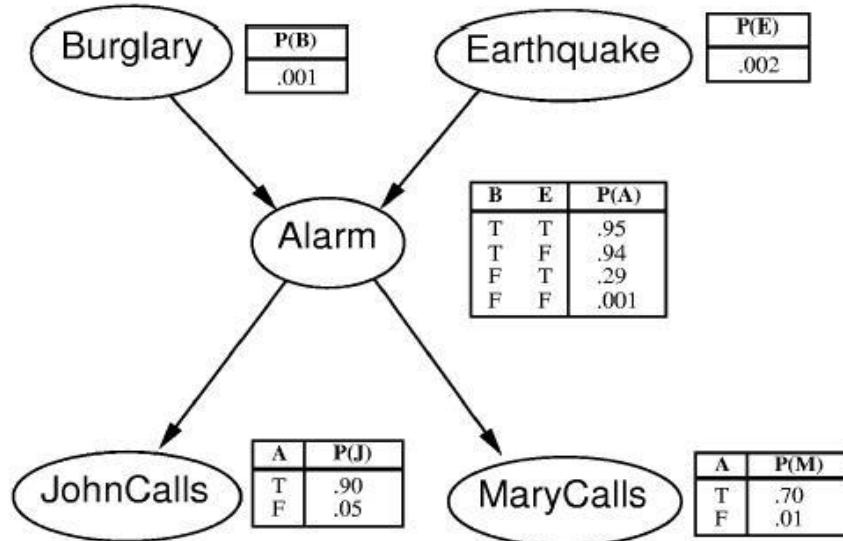
t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	F
2					
3					
4					

- Sampling $P(M|A)$: No need to apply Bayes Rule
- $A = F$, so we compute the following, and sample

$$P(M = T | A = F) \propto 0.01$$

$$P(M = F | A = F) \propto 0.99$$

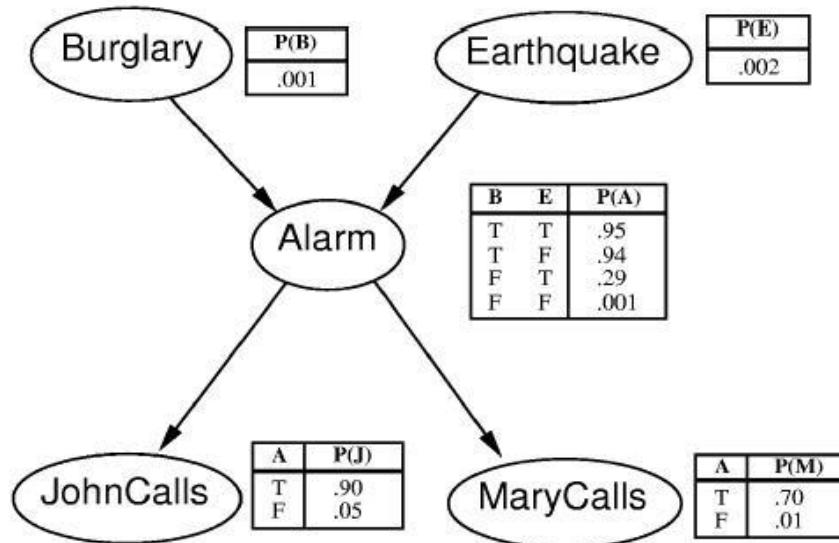
Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	F
2	F	T	T	T	T
3					
4					

- Now $t = 2$, and we repeat the procedure to sample new values of $B, E, A, J, M \dots$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	F
2	F	T	T	T	T
3	T	F	T	F	T
4	T	F	T	F	F

- Now $t = 2$, and we repeat the procedure to sample new values of $B, E, A, J, M \dots$
- And similarly for $t = 3, 4$, etc.

Gibbs Sampling is a special case of MH

- The GS proposal distribution is

$$Q(x'_i, \mathbf{x}_{-i} | x_i, \mathbf{x}_{-i}) = P(x'_i | \mathbf{x}_{-i})$$

(\mathbf{x}_{-i} denotes all variables except x_i)

- Applying Metropolis-Hastings with this proposal, we obtain:

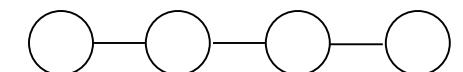
$$\begin{aligned} A(x'_i, \mathbf{x}_{-i} | x_i, \mathbf{x}_{-i}) &= \min \left(1, \frac{P(x'_i, \mathbf{x}_{-i}) Q(x_i, \mathbf{x}_{-i} | x'_i, \mathbf{x}_{-i})}{P(x_i, \mathbf{x}_{-i}) Q(x'_i, \mathbf{x}_{-i} | x_i, \mathbf{x}_{-i})} \right) \\ &= \min \left(1, \frac{P(x'_i, \mathbf{x}_{-i}) P(x_i | \mathbf{x}_{-i})}{P(x_i, \mathbf{x}_{-i}) P(x'_i | \mathbf{x}_{-i})} \right) = \min \left(1, \frac{P(x'_i | \mathbf{x}_{-i}) P(\mathbf{x}_{-i}) P(x_i | \mathbf{x}_{-i})}{P(x_i | \mathbf{x}_{-i}) P(\mathbf{x}_{-i}) P(x'_i | \mathbf{x}_{-i})} \right) \\ &= \min(1, 1) = 1 \end{aligned}$$

GS is simply MH with a proposal that is always accepted!

Markov Chains

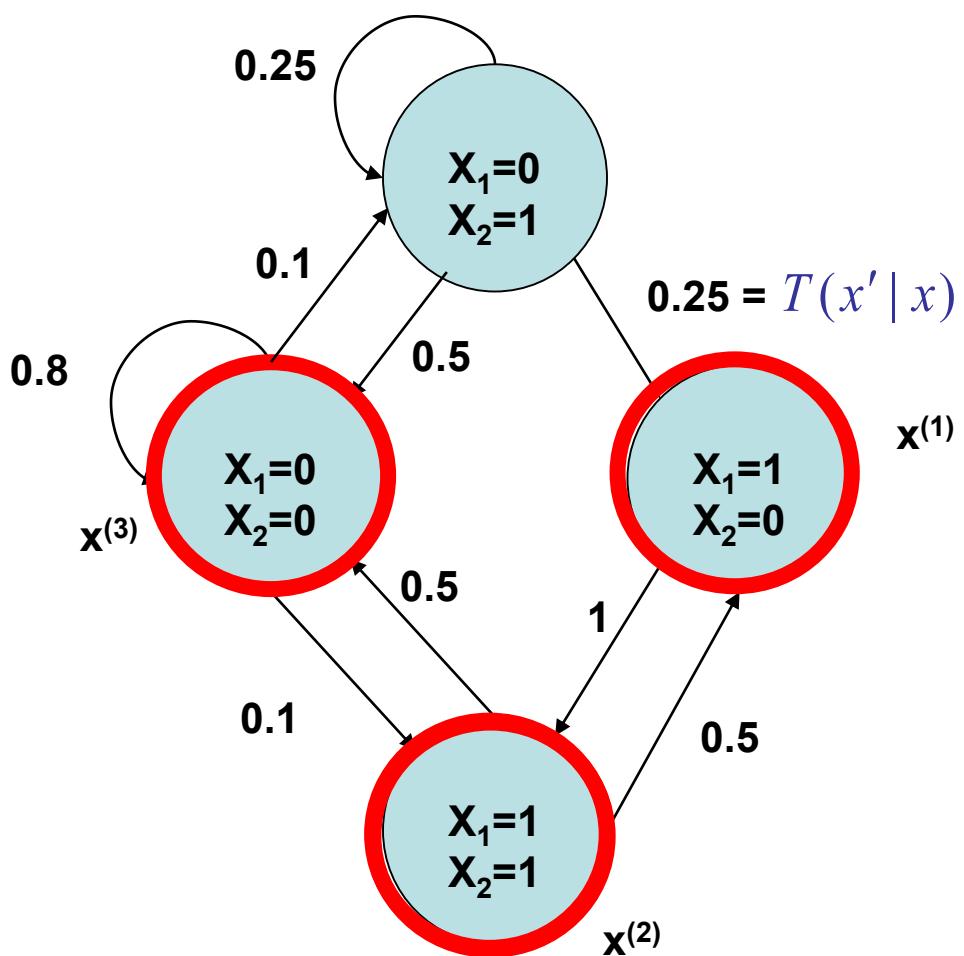
- A Markov Chain is a sequence of random variables $x^{(1)}, x^{(2)}, \dots, x^{(n)}$ with the Markov Property

$$P(x^{(n)} = x | x^{(1)}, \dots, x^{(n-1)}) = P(x^{(n)} = x | x^{(n-1)})$$

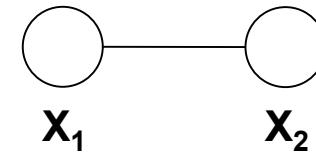


- $P(x^{(n)} = x | x^{(n-1)})$ is known as the transition kernel
- The next state depends only on the preceding state – recall HMMs!
- Note: the r.v.s $x^{(i)}$ can be vectors
 - We define $x^{(t)}$ to be the t-th sample of all variables in a graphical model
 - $X^{(t)}$ represents the entire state of the graphical model at time t
- We study homogeneous Markov Chains, in which the transition kernel $P(x^{(t)} = x | x^{(t-1)})$ is fixed with time
 - To emphasize this, we will call the kernel $T(x' | x)$, where x is the previous state and x' is the next state

Markov Chains



MRF



$$0.25 = T(x' | x) = T(x' = (1,0) | x = (0,1))$$

Markov Chain Sampling = simulating the dynamics of a Markov Chain

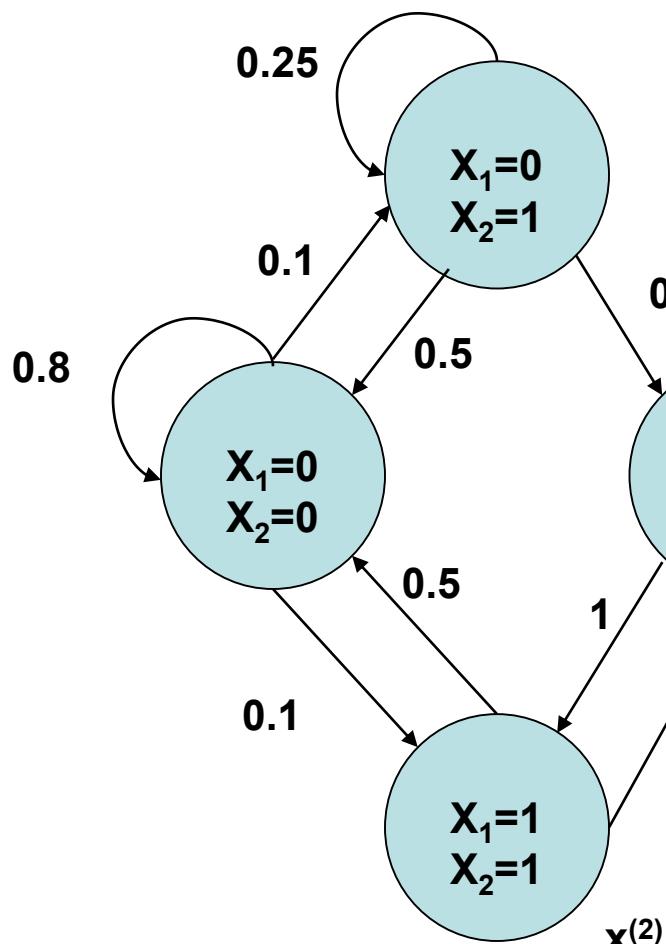
Initialize the simulation in one state (or randomly) $x^{(0)}$

Markov Chain Concepts

- To understand MCs, we need to define a few concepts:
 - Probability distributions over states: $\pi^{(t)}(x)$ is a distribution over the state of the system x , at time t
 - When dealing with MCs, we don't think of the system as being in one state, but as having a distribution over states
 - For graphical models, remember that x represents all variables
 - Transitions: recall that states transition from $x^{(t)}$ to $x^{(t+1)}$ according to the transition kernel $T(x' | x)$. We can also transition entire distributions:
$$\pi^{(t+1)}(x') = \sum_x \pi^{(t)}(x)T(x' | x)$$
 - At time t , state x has probability mass $\pi^{(t)}(x)$. The transition probability redistributes this mass to other states x' .
 - **Stationary distributions:** $\pi(x)$ is stationary if it does not change under the transition kernel:

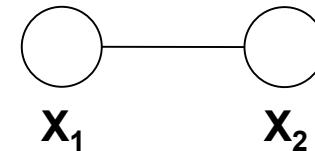
$$\pi(x') = \sum_x \pi(x)T(x' | x) \quad \text{for all } x'$$

Markov Chains

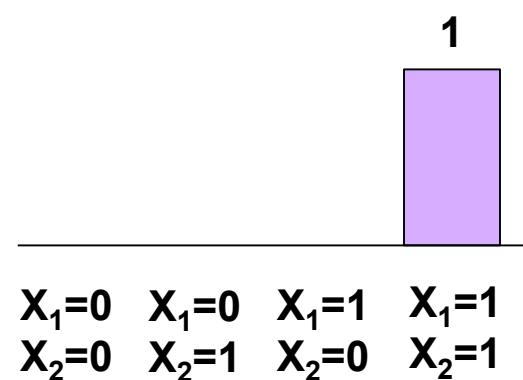


$$0.25 = T(x' | x) = T(x' = (1,0) | x = (0,1))$$

MRF

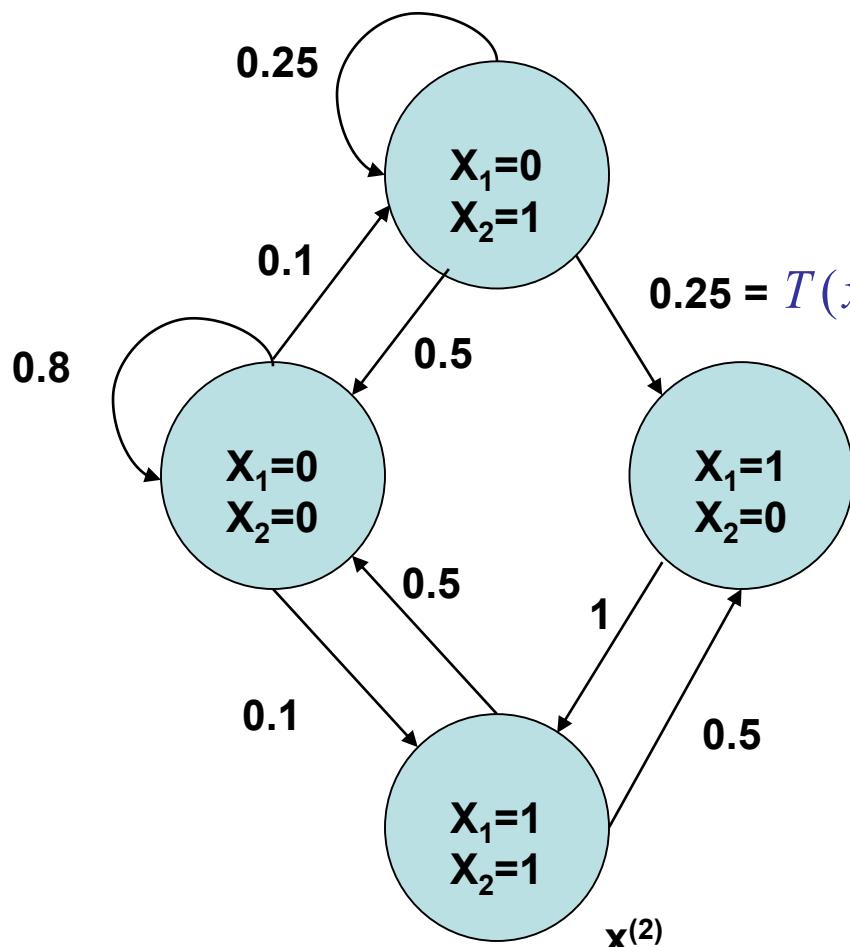


$$\pi^{(0)}(x)$$



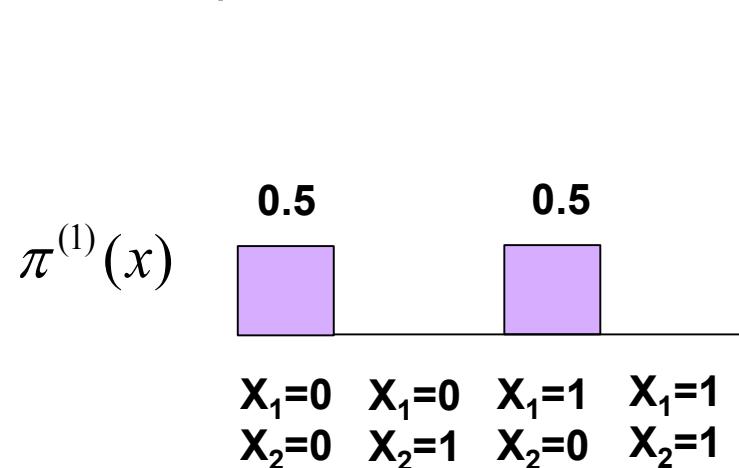
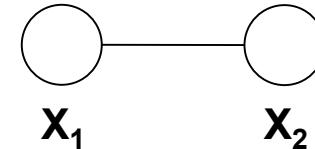
Initialize the simulation in one state $x^{(0)}$

Markov Chains

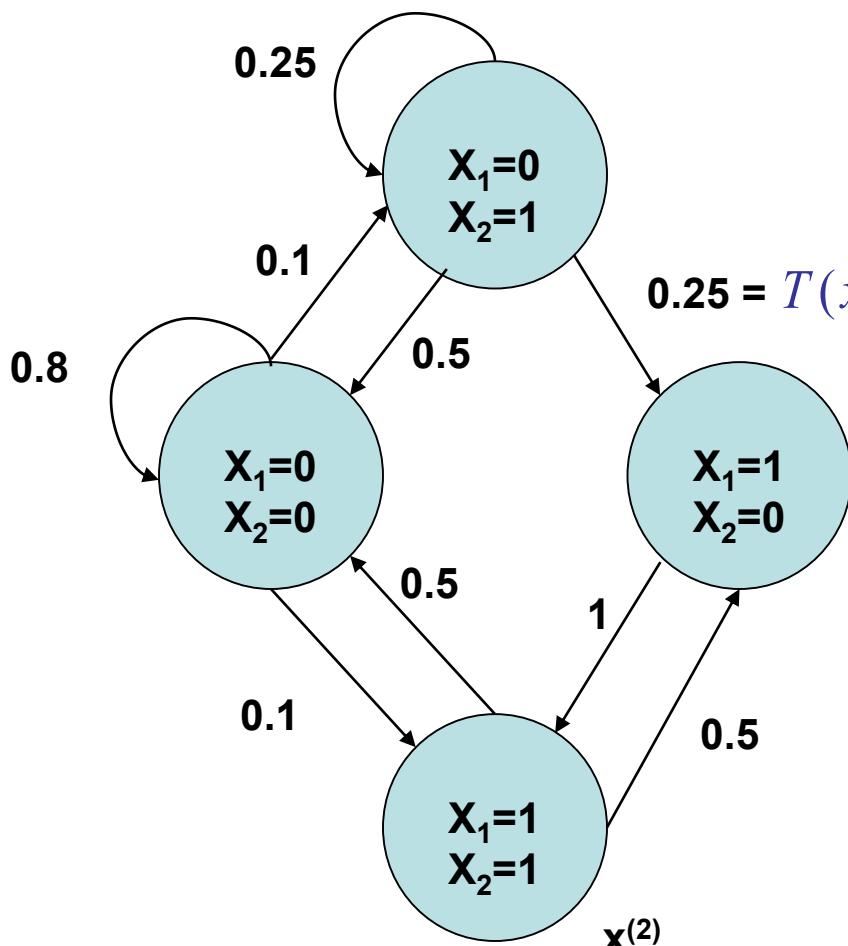


Initialize the simulation in one state $x^{(0)}$

MRF

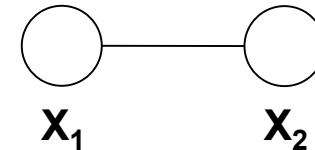


Markov Chains

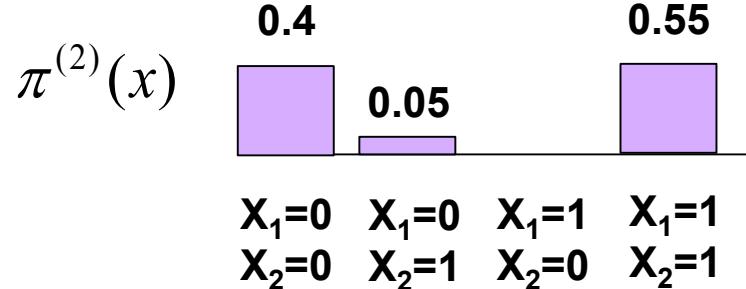


Initialize the simulation in one state $x^{(0)}$

MRF



$$0.25 = T(x' | x) = T(x' = (1,0) | x = (0,1))$$

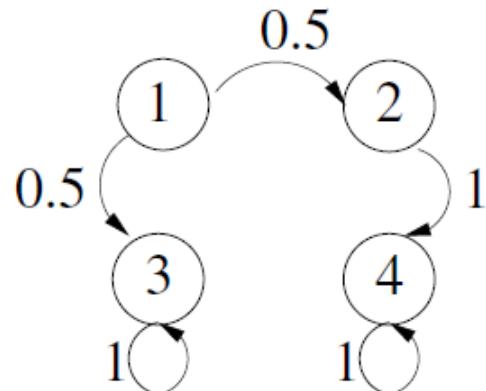


stationary if it does not change

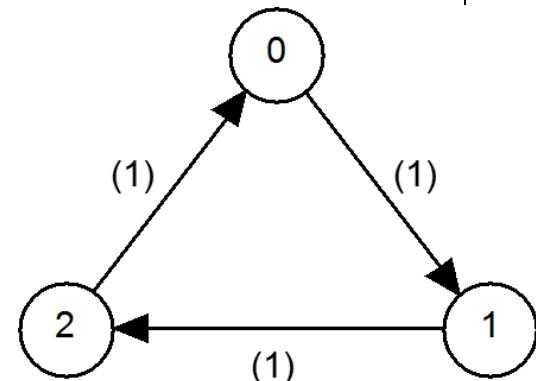
Markov Chain Concepts

- Stationary distributions are of great importance in MCMC. To understand them, we need to define some notions:
 - **Irreducible**: an MC is irreducible if you can get from any state x to any other state x' with probability > 0 in a finite number of steps
 - i.e. there are no unreachable parts of the state space
 - This property depends on the transition kernel!
 - **Aperiodic**: an MC is aperiodic if you can return to any state i at any time
 - If there exists n such that for all $n' \geq n$, $\Pr(x^{(n')} = i | x^{(0)} = i) > 0$
 - **Ergodic (or regular)**: an MC is ergodic if it is irreducible and aperiodic
- Ergodicity is important: it implies you can reach the stationary distribution $\pi_{st}(x)$, no matter the initial distribution $\pi^{(0)}(x)$
 - All good MCMC algorithms must satisfy ergodicity, so that you can't initialize in a way that will never converge

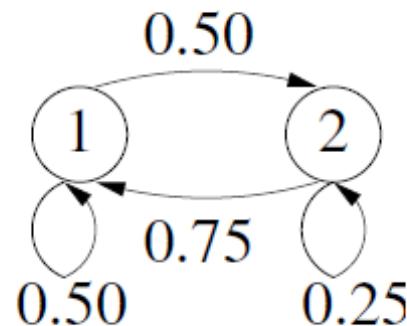
Examples



Reducible.
Limiting distribution depends
on initial condition



**Irreducible, periodic (each state
visited every 3 iterations)**
Limiting distribution does not exist



Irreducible, aperiodic.
Unique limiting distribution
 $P(x) = [0.6, 0.4]$

Markov Chain Concepts

- **Reversible (detailed balance):** an MC is reversible if there exists a distribution $\pi(x)$ such that the detailed balance condition is satisfied:

$$\pi(x')T(x|x') = \pi(x)T(x'|x)$$

- Probability of $x' \rightarrow x$ is the same as $x \rightarrow x'$
- $\pi(x)$ is a stationary distribution of the MC! Proof:

$$\pi(x')T(x|x') = \pi(x)T(x'|x)$$

$$\sum_x \pi(x')T(x|x') = \sum_x \pi(x)T(x'|x)$$

$$\pi(x')\sum_x T(x|x') = \sum_x \pi(x)T(x'|x)$$

$$\pi(x') = \sum_x \pi(x)T(x'|x)$$

- The last line is the definition of a stationary distribution!

Why does Metropolis-Hastings work?

- Recall that we draw a sample x' according to $Q(x'|x)$, and then accept/reject according to $A(x'|x)$.

- In other words, the transition kernel is

$$T(x'|x) = Q(x'|x)A(x'|x)$$

- We can prove that MH is reversible:

- Recall that

$$A(x'|x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

- Notice this implies the following:

$$\text{if } A(x'|x) < 1 \text{ then } \frac{P(x)Q(x'|x)}{P(x')Q(x|x')} > 1 \text{ and thus } A(x|x') = 1$$

Why does Metropolis-Hastings work?

$$\text{if } A(x' | x) < 1 \quad \text{then} \quad \frac{P(x)Q(x' | x)}{P(x')Q(x | x')} > 1 \quad \text{and thus} \quad A(x | x') = 1$$

- Now suppose $A(x' | x) < 1$ and $A(x | x') = 1$. We have

$$A(x' | x) = \frac{P(x')Q(x | x')}{P(x)Q(x' | x)}$$

$$P(x)Q(x' | x)A(x' | x) = P(x')Q(x | x')$$

$$P(x)Q(x' | x)A(x' | x) = P(x')Q(x | x')A(x | x')$$

$$P(x)T(x' | x) = P(x')T(x | x')$$

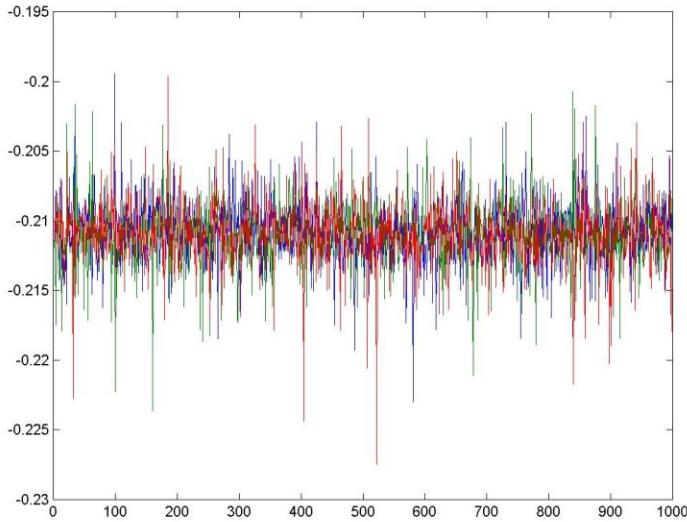
- The last line is exactly the **detailed balance condition**
 - In other words, the MH algorithm leads to a stationary distribution $P(x)$
 - Recall we defined $P(x)$ to be the true distribution of x
 - If ergodic (irreducible & aperiodic), MH algorithm eventually converges to the true distribution!

Why does Metropolis-Hastings work?

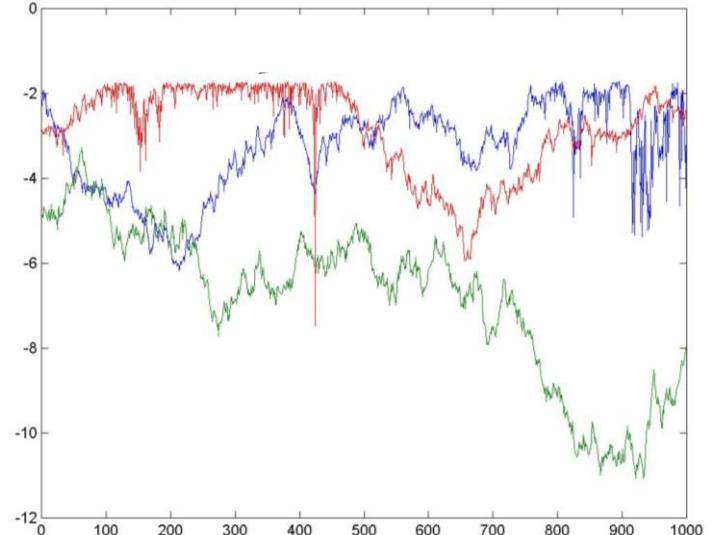
- Theorem: If a Markov chain is **regular (ergodic)** and satisfies **detailed balance** with respect to $p(x)$, then $p(x)$ is its unique stationary distribution. The chain converges to the stationary distribution regardless of where it begins.
- If the BN has no zeros, it is easy to verify that Gibbs sampling satisfies aperiodicity and is irreducible, and thus is regular
- The *mixing time*, or how long it takes to **reach** something close the stationary distribution, can be very long

Sample Values vs Time

Well-mixed chains



Poorly-mixed chains



- Monitor convergence by plotting samples (of r.v.s) from multiple MH runs (chains)
 - If the chains are well-mixed (left), they are probably converged
 - If the chains are poorly-mixed (right), we should continue burn-in

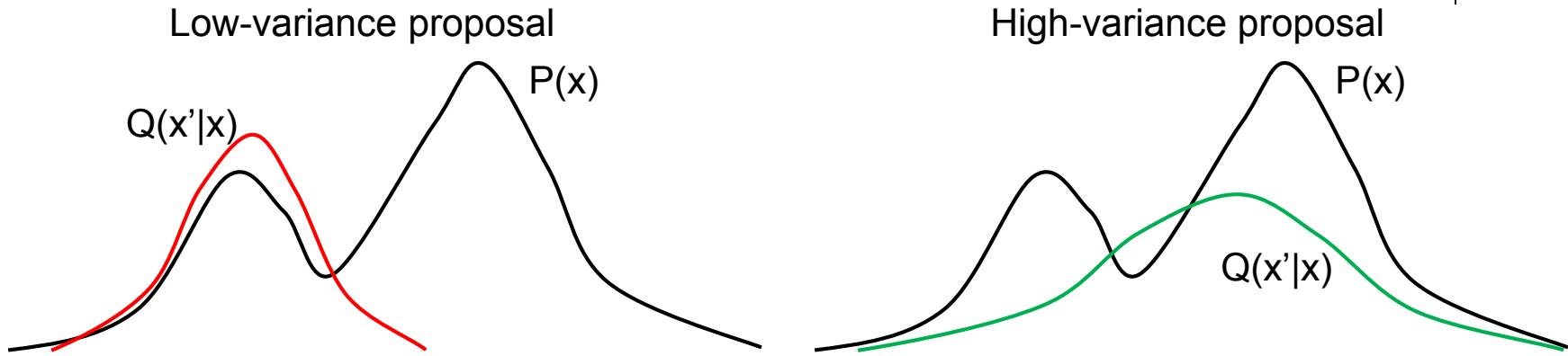
Caveats

- Can in principle sample from **any distribution**, including posterior $P(Z|E)$.
- Only requires evaluating **ratios of probabilities**. Can be used for undirected models
- Although MH eventually converges to the true distribution $P(x)$, we have no guarantees as to when this will occur
 - The burn-in period represents the un-converged part of the Markov Chain – that's why we throw those samples away!
 - Knowing when to halt burn-in is an art.

Practical Aspects of MCMC

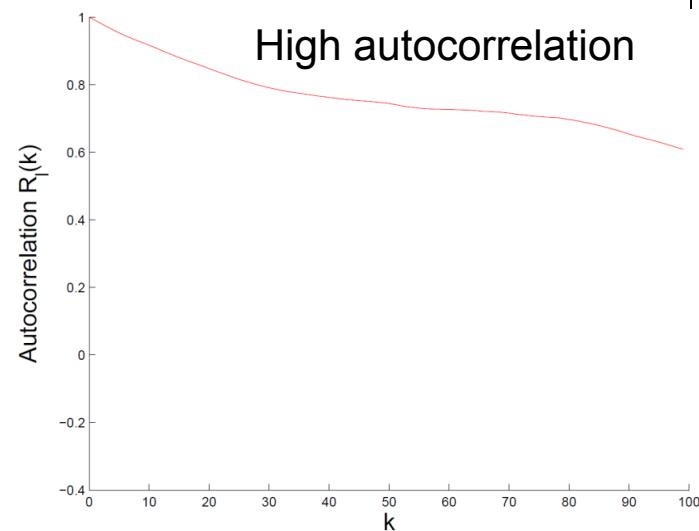
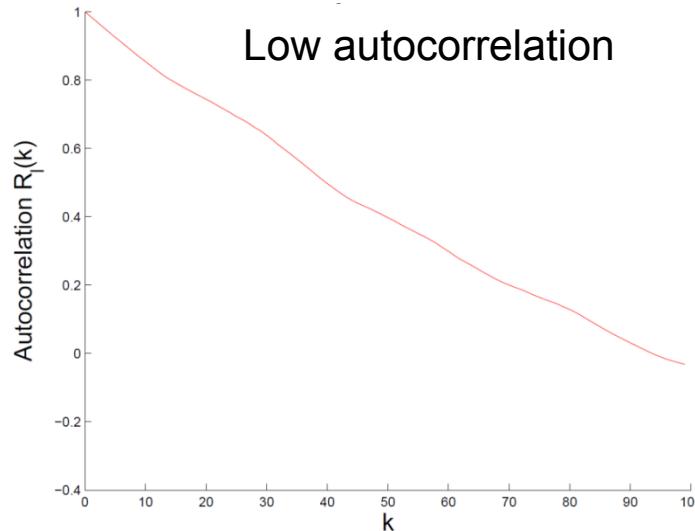
- How do we know if our proposal $Q(x'|x)$ is any good?
 - Monitor the acceptance rate
 - Plot the autocorrelation function
- How do we know when to stop burn-in?
 - Plot the sample values vs time
 - Plot the log-likelihood vs time

Acceptance Rate



- Choosing the proposal $Q(x'|x)$ is a tradeoff:
 - “Narrow”, low-variance proposals have high acceptance, but take many iterations to explore $P(x)$ fully because the proposed x' are too close
 - “Wide”, high-variance proposals have the potential to explore much of $P(x)$, but many proposals are rejected which slows down the sampler
- A good $Q(x'|x)$ proposes distant samples x' with a sufficiently high acceptance rate

Autocorrelation function



- MCMC chains always show autocorrelation (AC)
 - AC means that adjacent samples in time are highly correlated
- We quantify AC with the **autocorrelation function** of an r.v. x :

$$R_x(k) = \frac{\sum_{t=1}^{n-k} (x_t - \bar{x})(x_{t+k} - \bar{x})}{\sum_{t=1}^{n-k} (x_t - \bar{x})^2}$$

Summary

- Markov Chain Monte Carlo methods use adaptive proposals $Q(x'|x)$ to sample from the true distribution $P(x)$
- Metropolis-Hastings allows you to specify any proposal $Q(x'|x)$
 - But choosing a good $Q(x'|x)$ requires care
- Gibbs sampling sets the proposal $Q(x'|x)$ to the conditional distribution $P(x'|x)$
 - Acceptance rate always 1!
 - But remember that high acceptance usually entails slow exploration
 - In fact, there are better MCMC algorithms for certain models
- Knowing when to halt burn-in is an art

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 12, February 16, 2017

Today's lecture

- ① MAP inference
- ② Simulated Annealing
- ③ Max-product
- ④ Linear programming relaxations for MAP inference

Recap: MAP inference

- Recall the MAP inference task,

$$\arg \max_{\mathbf{x}} p(\mathbf{x}), \quad p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

(we assume any evidence has been subsumed into the potentials)

- Since the normalization term is simply a constant, this is equivalent to

$$\arg \max_{\mathbf{x}} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

(called the *max-product* inference task)

- Furthermore, since \log is monotonic, letting $\theta_c(\mathbf{x}_c) = \log \phi_c(\mathbf{x}_c)$, we have that this is equivalent to

$$\arg \max_{\mathbf{x}} \sum_{c \in C} \theta_c(\mathbf{x}_c)$$

(called *max-sum*)

Simulated Annealing

- Metropolis-Hastings can be used to sample from $p(x) \propto \exp(\sum_{c \in C} \theta_c(x_c))$
- Can also sample from $p_T(x) \propto \exp(\frac{1}{T} \sum_{c \in C} \theta_c(x_c))$. T is a parameter called *temperature*.
- For $T \rightarrow \infty$, $p_T(x)$ is close to uniform. Easy to sample from!
- For $T \rightarrow 0$, $p_T(x)$ “concentrates” on $\arg \max_x \sum_{c \in C} \theta_c(x_c)$. Hard to sample from!
- Simulated Annealing idea: start running MH with high temperature T , then slowly decrease T (anneal) as you run MH. Note: temperature affects the acceptance probability.
- If “cooling” is sufficiently slow, it is guaranteed to find the mode (theoretical guarantee is often impractical).

Simulated Annealing

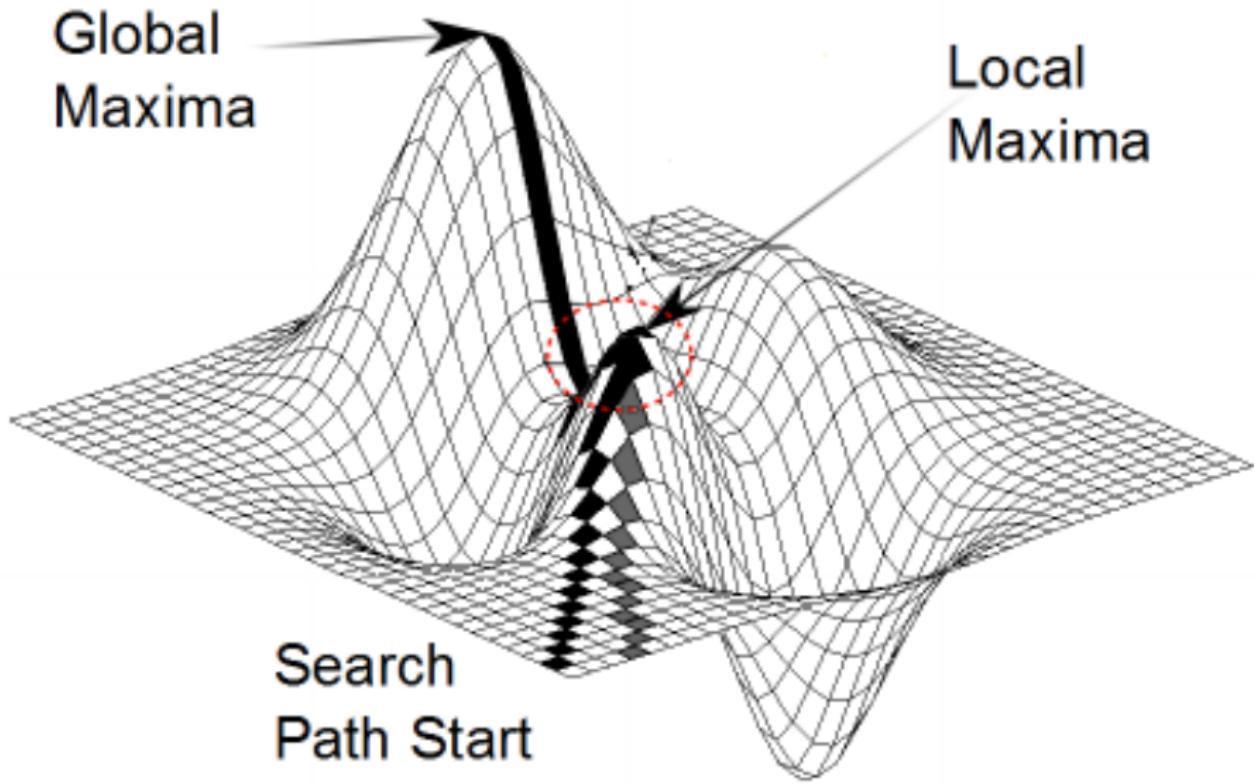
Use MH to sample from $p_T(x) \propto \exp(\frac{1}{T} \sum_{c \in C} \theta_c(x_c))$

- Pick an initial state x . Set the temperature T to a high value
- Repeat
 - Propose a new state x' : randomly pick a variable, randomly pick a new value
 - Accept with probability

$$\begin{aligned} A(x'|x) &= \min \left\{ 1, \frac{p_T(x')Q(x|x')}{Q(x'|x)p_T(x)} \right\} = \min \left\{ 1, \frac{\exp(\frac{1}{T} \sum_{c \in C} \theta_c(x'_c))}{\exp(\frac{1}{T} \sum_{c \in C} \theta_c(x_c))} \right\} \\ &= \min \left\{ 1, \exp \left(\frac{1}{T} \left(\sum_{c \in C} \theta_c(x'_c) - \sum_{c \in C} \theta_c(x_c) \right) \right) \right\} \end{aligned}$$

- Slowly decrease the temperature T

Visualization



Structured prediction

- **Standard Machine Learning:**

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

inputs \mathcal{X} are complex, output y is a real number (classification, regression, etc.)

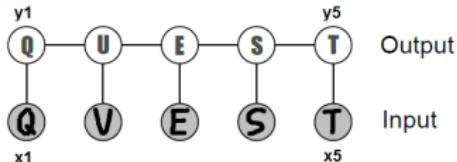
- **Structured Output Learning:**

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

inputs \mathcal{X} are complex , outputs \mathcal{Y} are also complex (structured) objects (tag sequences, parse trees, segmentations, etc.)

Handwriting Recognition Example

- input space \mathcal{X} = 5-letter word images , $\mathbf{x} = (x_1, \dots, x_5)$, $x_j \in \{0, 1\}^{300 \times 80}$
- output space \mathcal{Y} = ASCII translation , $\mathbf{y} = (y_1, \dots, y_5)$, $y_j \in \{A, \dots, Z\}$



- Potentials: $\phi_1(x_1, y_1), \dots, \phi_5(x_5, y_5)$, and $\phi_{12}(y_1, y_2), \dots, \phi_{45}(y_4, y_5)$

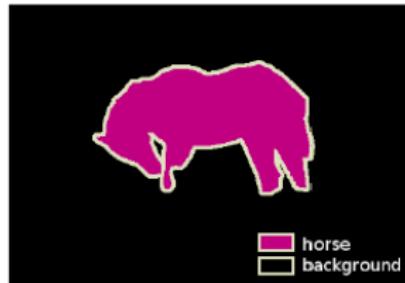
$$\begin{aligned} f(\mathbf{x}) &= \arg \max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y} | \mathbf{x}) \\ &= \arg \max_{\mathbf{y} \in \mathcal{Y}} \phi_1(x_1, y_1) \cdots \phi_5(x_5, y_5) \cdot \phi_{12}(y_1, y_2) \cdots \phi_{45}(y_4, y_5) / Z(x) \end{aligned}$$

- $\phi_1(x_1, y_1)$: letter y_1 should look like the image x_1 (could be the score assigned by a convnet)
- $\phi_{12}(y_1, y_2)$: pair QU is more likely than QV

Computer Vision Example



input: images



output: segmentation masks

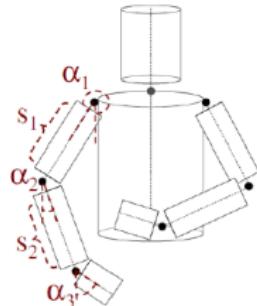
- input space $\mathcal{X} = \{\text{color images}\} = [0, 255]^{3 \cdot M \cdot N}$
- output space $\mathcal{Y} = \{\text{segmentations}\} = \{0, 1\}^{M \times N}$
- (structured output) prediction function: $f : \mathcal{X} \rightarrow \mathcal{Y}$

$$f(x) = \arg \max_{y \in \mathcal{Y}} P(y|x)$$

Computer Vision Example



input: image



body model



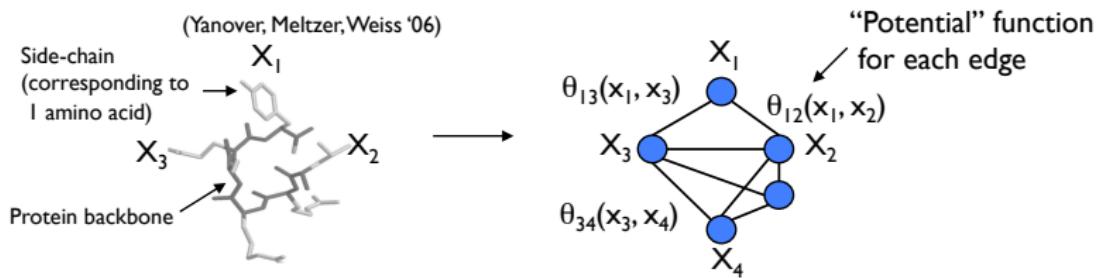
output: model fit

- input space $\mathcal{X} = \{\text{color images}\} = [0, 255]^{3 \cdot M \cdot N}$
- output space $\mathcal{Y} = \{\text{positions of body parts}\}$
- (structured output) prediction function: $f : \mathcal{X} \rightarrow \mathcal{Y}$

$$f(x) = \arg \max_{y \in \mathcal{Y}} P(y|x)$$

Motivating application: protein side-chain placement

- Find “minimum energy” conformation of amino acid side-chains along a fixed backbone:



- Orientations of the side-chains are represented by discretized angles (rotamers)
- Orientations nearby amino acids are energetically coupled (attractive and repulsive forces)

Distributivity

- Max distributes over multiplication in the same way that summation does

$$a \cdot b + a \cdot c = a(b + c)$$

$$\max(a \cdot b, a \cdot c) = a \cdot \max(b, c)$$

- Hence, a max can be slid along a product just as a summation can be

$$\max_{x_1} \max_{x_2} \max_{x_3} f(x_1)g(x_2)h(x_2, x_3) = \max_{x_1} f(x_1) \max_{x_2} g(x_2) \max_{x_3} h(x_2, x_3)$$

- Max is also *commutative*: $\max(a, b) = \max(b, a)$ and *associative*: $\max(\max(a, b), c) = \max(a, \max(b, c))$
- It forms a commutative semi-ring with multiplication

Semi-rings

- Compare the sum-product problem with the max-product:

$$\text{sum-product} \quad \sum_{\mathbf{x}} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

$$\text{max-product} \quad \max_{\mathbf{x}} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

- Can exchange operators $(+, *)$ for $(\max, *)$ and, because both are semirings satisfying associativity and commutativity, everything works!
- We get “max-product variable elimination” and “max-product belief propagation”, junction tree, etc.

Simple example

- Suppose we have a simple chain, $A - B - C - D$, and we want to find the MAP assignment,

$$\max_{a,b,c,d} \phi_{AB}(a, b)\phi_{BC}(b, c)\phi_{CD}(c, d)$$

- Just as we did before, we can push the maximizations inside to obtain:

$$\max_{a,b} \phi_{AB}(a, b) \max_c \phi_{BC}(b, c) \max_d \phi_{CD}(c, d)$$

or, equivalently (more stable numerically),

$$\max_{a,b} \left(\log \phi_{AB}(a, b) + \max_c \left(\log \phi_{BC}(b, c) + \max_d \log \phi_{CD}(c, d) \right) \right)$$

- See book on how to find the actual maximizing assignment (need to keep back pointers in the DP algorithm)

Linear (Integer) programs

- A linear program is an optimization problem of the form

$$\min \mathbf{c} \cdot \mathbf{x}$$

subject to $A\mathbf{x} \leq \mathbf{b}$.

- Linear objective function, with linear constraints on the variables. Can be solved in polynomial time.
- Note: linear equality constraints are also OK: just add inequalities $a\mathbf{x} \leq b$ and $a\mathbf{x} \geq b$ to get $a\mathbf{x} = b$.
- An **integer** linear program is an optimization problem of the form

$$\min \mathbf{c} \cdot \mathbf{x}$$

subject to $A\mathbf{x} \leq \mathbf{b}$ and integrality constraints $\mathbf{x} \in \{0, 1\}^N$.

- Generally NP-hard to solve.

MAP as an integer linear program (ILP)

- For simplicity, let's consider MAP in pairwise MRFs
- MAP as a discrete optimization problem is

$$\arg \max_{\mathbf{x}} \sum_{i \in V} \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j).$$

- To turn this into an integer linear program, we introduce indicator variables
 - $\mu_i(x_i)$, one for each $i \in V$ and state x_i
 - $\mu_{ij}(x_i, x_j)$, one for each edge $ij \in E$ and pair of states x_i, x_j
- The objective function is then

$$\max_{\boldsymbol{\mu}} \sum_{i \in V} \sum_{x_i} \theta_i(x_i) \mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) \mu_{ij}(x_i, x_j)$$

- What is the dimension of $\boldsymbol{\mu}$, if binary variables?

What are the constraints?

- Force every “cluster” of variables to choose a local assignment:

$$\mu_i(x_i) \in \{0, 1\} \quad \forall i \in V, x_i$$

$$\sum_{x_i} \mu_i(x_i) = 1 \quad \forall i \in V$$

$$\mu_{ij}(x_i, x_j) \in \{0, 1\} \quad \forall ij \in E, x_i, x_j$$

$$\sum_{x_i, x_j} \mu_{ij}(x_i, x_j) = 1 \quad \forall ij \in E$$

- Enforce that these local assignments are consistent:

$$\mu_i(x_i) = \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i$$

$$\mu_j(x_j) = \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j$$

MAP as an integer linear program (ILP)

$$\text{MAP}(\theta) = \max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i) \mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) \mu_{ij}(x_i, x_j)$$

subject to:

$$\mu_i(x_i) \in \{0, 1\} \quad \forall i \in V, x_i$$

$$\sum_{x_i} \mu_i(x_i) = 1 \quad \forall i \in V$$

$$\mu_{ij}(x_i, x_j) \in \{0, 1\} \quad \forall ij \in E, x_i, x_j$$

$$\sum_{x_i, x_j} \mu_{ij}(x_i, x_j) = 1 \quad \forall ij \in E$$

$$\mu_i(x_i) = \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i$$

$$\mu_j(x_j) = \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j$$

- Many extremely good off-the-shelf solvers, such as CPLEX and Gurobi

Linear programming relaxation for MAP

Integer linear program was:

$$\text{MAP}(\theta) = \max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i) \mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) \mu_{ij}(x_i, x_j)$$

subject to

$$\begin{aligned}\mu_i(x_i) &\in \{0, 1\} \quad \forall i \in V, x_i \\ \sum_{x_i} \mu_i(x_i) &= 1 \quad \forall i \in V \\ \mu_{ij}(x_i, x_j) &\in \{0, 1\} \quad \forall ij \in E, x_i, x_j \\ \sum_{x_i, x_j} \mu_{ij}(x_i, x_j) &= 1 \quad \forall ij \in E \\ \mu_i(x_i) &= \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i \\ \mu_j(x_j) &= \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j\end{aligned}$$

Relax integrality constraints, allowing the variables to be **between** 0 and 1:

$$\mu_i(x_i) \in \{0, 1\} \text{ relaxed to } \mu_i(x_i) \in [0, 1]$$

Linear programming relaxation for MAP

Linear programming relaxation is:

$$\text{LP}(\theta) = \max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i) \mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) \mu_{ij}(x_i, x_j)$$

$$\mu_i(x_i) \in [0, 1] \quad \forall i \in V, x_i$$

$$\sum_{x_i} \mu_i(x_i) = 1 \quad \forall i \in V$$

$$\mu_{ij}(x_i, x_j) \in [0, 1] \quad \forall ij \in E, x_i, x_j$$

$$\sum_{x_i, x_j} \mu_{ij}(x_i, x_j) = 1 \quad \forall ij \in E$$

$$\mu_i(x_i) = \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i$$

$$\mu_j(x_j) = \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j$$

- Linear programs can be solved **efficiently!**
- Since the LP relaxation maximizes over a **larger** set of solutions, its value can only be *higher*

$$\text{MAP}(\theta) \leq \text{LP}(\theta)$$

- LP relaxation is **tight** ($\text{MAP}(\theta) = \text{LP}(\theta)$) for tree-structured MRFs!

Other approaches to solve MAP

- Local search
 - Start from an arbitrary assignment (e.g., random). Iterate:
 - Choose a variable. Change a new state for this variable to maximize the value of the resulting assignment
- Branch-and-bound
 - Exhaustive search over space of assignments, pruning branches that can be provably shown not to contain a MAP assignment
 - Can use the LP relaxation or its dual to obtain upper bounds
 - Lower bound obtained from value of any assignment found

Summary

- ① **Representation.** How does one compactly describe a joint distribution?
- ② **Inference.** How to solve probabilistic inference queries?
 - ① Exact
 - ② (Variational)
 - ③ Sampling
- ③ Next: **learning.** How to infer the distribution from data?

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 13, February 21, 2017

Roadmap for the course

- **Representation:** How do we specify distributions that satisfy particular independence properties? ✓
- **Inference:** How can we exploit independence properties for efficient computation? ✓
- **Learning:** How can we acquire a model from data?

Today: learning undirected graphical models

- ① Learning MRFs
 - a. Log-linear models
 - b. Maximum likelihood estimation
- ② Getting around complexity of inference
 - a. Using approximate inference within learning
 - b. Pseudo-likelihood
- ③ Conditional random fields

How to acquire a model?

- Lets assume that the domain is governed by some underlying distribution p^*
- We are given a dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ of m samples from p^*
- The standard assumption is that the data instances are **independent and identically distributed (IID)**
- We are also given a family of models \mathcal{M} , and our task is to learn some “good” model $\hat{\mathcal{M}} \in \mathcal{M}$ (i.e., in this family) that defines a distribution $p_{\hat{\mathcal{M}}}$
 - For example, all Bayes nets with a given graph structure, for all possible choices of the CPD tables
- We can learn model parameters for a fixed structure, or both the structure and model parameters
- Minimizing KL divergence $\mathbf{D}(p^* || p_{\hat{\mathcal{M}}})$ is equivalent to *maximizing the expected log-likelihood*

$$\mathbf{E}_{\mathbf{x} \sim p^*} [\log \hat{p}(\mathbf{x})]$$

Maximum likelihood

- Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x} \sim p^*} [\log \hat{p}(\mathbf{x})]$$

with the *empirical log-likelihood* (Monte Carlo estimate):

$$\mathbf{E}_{\mathcal{D}} [\log \hat{p}(\mathbf{x})] = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log \hat{p}(\mathbf{x})$$

- **Maximum likelihood learning** is then:

$$\max_{\hat{\mathcal{M}}} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log \hat{p}(\mathbf{x})$$

- Equivalently, maximize likelihood of the data
 $\hat{p}(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}) = \prod_{\mathbf{x} \in \mathcal{D}} \hat{p}(\mathbf{x})$. Note: the $\frac{1}{|\mathcal{D}|}$ scaling is irrelevant.

Recall: ML estimation in Bayesian networks

- Maximum likelihood estimation: $\max_{\theta} \ell(\theta; \mathcal{D})$, where

$$\begin{aligned}\ell(\theta; \mathcal{D}) = \log p(\mathcal{D}; \theta) &= \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) \\ &= \sum_{i=1}^n \sum_{\mathbf{x}_{pa(i)}} \sum_{x_i} N_{x_i, \mathbf{x}_{pa(i)}} \log \theta_{x_i | \mathbf{x}_{pa(i)}}\end{aligned}$$

where $N_{x_i, \mathbf{x}_{pa(i)}}$ is the number of times that the (partial) assignment $x_i, \mathbf{x}_{pa(i)}$ is observed in the training data

- In Bayesian networks, we have the closed form ML solution:

$$\theta_{x_i | \mathbf{x}_{pa(i)}}^{ML} = \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{\sum_{\hat{x}_i} N_{\hat{x}_i, \mathbf{x}_{pa(i)}}}$$

- We were able to estimate each CPD independently because the objective **decomposes** by variable and parent assignment

Undirected graphical models

- Let's start with an MRF model:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{c \in C} \theta_c(\mathbf{x}_c) - \ln Z \right)$$

$$Z = \sum_{\mathbf{x}} \prod_{c \in C} \phi_c(\mathbf{x}_c) = \sum_{\mathbf{x}} \exp \left(\sum_{c \in C} \theta_c(\mathbf{x}_c) \right)$$

- If we fix the graph structure (the cliques C), can we learn the parameters $\theta_c(\mathbf{x}_c) = \log \phi_c(\mathbf{x}_c)$ from data?
- Define a family of probability distributions parameterized by θ :

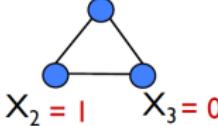
$$p(\mathbf{x}; \theta) = \frac{1}{Z(\theta)} \prod_{c \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{c \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right)$$

$$= \exp \left(\sum_{c \in C} \sum_{\bar{\mathbf{x}}_c} \theta_c(\bar{\mathbf{x}}_c) \mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c) - \ln Z(\theta) \right) = \exp \left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta) \right)$$

- Note: the normalization constant $Z(\theta)$ depends on the parameters (unlike BN)

Overcomplete representation: pairwise MRF example

$$f(x) = \begin{bmatrix} 1[x_1=0] \\ 1[x_1=1] \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{\quad} \text{Assignment for } X_1 \\ \xleftarrow{\quad} \text{Assignment for } X_2 \\ \xleftarrow{\quad} \text{Assignment for } X_3 \\ \xleftarrow{\quad} \text{Edge assignment for } X_1X_3 \\ \xleftarrow{\quad} \text{Edge assignment for } X_1X_2 \\ \xleftarrow{\quad} \text{Edge assignment for } X_2X_3 \end{array}$$

$X_1 = 0$

 $X_2 = 1$ $X_3 = 0$

For a discrete-variable MRF, $f(\mathbf{x})$ is simply the concatenation of indicator functions:

$$p(\mathbf{x}; \theta) = \exp \left(\sum_{\mathbf{c} \in C} \sum_{\bar{\mathbf{x}}_{\mathbf{c}}} \theta_{\mathbf{c}}(\bar{\mathbf{x}}_{\mathbf{c}}) 1(\mathbf{x}_{\mathbf{c}} = \bar{\mathbf{x}}_{\mathbf{c}}) - \ln Z(\theta) \right) = \exp \left(\theta^T f(\mathbf{x}) - \ln Z(\theta) \right)$$

ML parameter estimation for Markov networks

- Let the class of models we consider be

$$p(\mathbf{x}; \theta) = \exp \left(\sum_{\mathbf{c} \in C} \sum_{\bar{\mathbf{x}}_{\mathbf{c}}} \theta_c(\bar{\mathbf{x}}_{\mathbf{c}}) \mathbf{1}(\mathbf{x}_{\mathbf{c}} = \bar{\mathbf{x}}_{\mathbf{c}}) - \ln Z(\theta) \right) = \exp \left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta) \right)$$

- Suppose we have some i.i.d. data \mathcal{D} and we want to estimate the parameters θ via maximum likelihood

$$\begin{aligned}\theta^{ML} &= \arg \max_{\theta} \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \arg \max_{\theta} \log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) \\ &= \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) \\ &= \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \left(\theta^T \mathbf{f}(\mathbf{x}) - \log Z(\theta) \right) \\ &= \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{c} \in C} \sum_{\bar{\mathbf{x}}_{\mathbf{c}}} \theta_c(\bar{\mathbf{x}}_{\mathbf{c}}) \mathbf{1}(\mathbf{x}_{\mathbf{c}} = \bar{\mathbf{x}}_{\mathbf{c}}) - |\mathcal{D}| \log Z(\theta)\end{aligned}$$

- The global normalization constant $Z(\theta)$ kills decomposability. Cannot decompose the objective into a sum over terms for each potential
- Solving for the parameters becomes much more complicated

Log-likelihood for log-linear models

- ML estimation problem

$$\theta^{ML} = \arg \max_{\theta} \theta^T \left(\sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - |\mathcal{D}| \log Z(\theta)$$

- The first term is linear in θ (easy)
- The second term is also a function of θ :

$$\log Z(\theta) = \log \sum_{\mathbf{x}} \exp \left(\theta^T \mathbf{f}(\mathbf{x}) \right)$$

- $\log Z(\theta)$ does not decompose
 - No closed form solution; even *computing* likelihood requires inference
 - Computing the partition function is hard (exponential in treewidth)

Log-likelihood for log-linear models

- The difficult part is

$$\log Z(\theta) = \log \sum_{\mathbf{x}} \exp (\theta^T \mathbf{f}(\mathbf{x}))$$

- We want to understand the dependence on θ :

$$\begin{aligned}\frac{\partial \log Z(\theta)}{\partial \theta_i} &= \frac{1}{Z(\theta)} \frac{\partial Z(\theta)}{\partial \theta_i} = \frac{1}{Z(\theta)} \frac{\partial \sum_{\mathbf{x}} \exp (\theta^T \mathbf{f}(\mathbf{x}))}{\partial \theta_i} \\ &= \frac{1}{Z(\theta)} \sum_{\mathbf{x}} \frac{\partial \exp (\theta^T \mathbf{f}(\mathbf{x}))}{\partial \theta_i} = \frac{1}{Z(\theta)} \sum_{\mathbf{x}} f_i(\mathbf{x}) \exp (\theta^T \mathbf{f}(\mathbf{x})) \\ &= \sum_{\mathbf{x}} f_i(\mathbf{x}) \frac{\exp (\theta^T \mathbf{f}(\mathbf{x}))}{Z(\theta)} = \sum_{\mathbf{x}} f_i(\mathbf{x}) p(\mathbf{x}; \theta) = E_{p(\mathbf{x}; \theta)}[f_i]\end{aligned}$$

so the full gradient is

$$\nabla_{\theta} \log Z(\theta) = \mathbb{E}_{p(\mathbf{x}; \theta)}[\mathbf{f}(\mathbf{x})]$$

- Thus, the gradient of the log-partition function can be computed by *inference* (computing marginals with respect to the current parameters θ)

Log-likelihood for log-linear models

- The difficult part is

$$\log Z(\theta) = \log \sum_{\mathbf{x}} \exp (\theta^T \mathbf{f}(\mathbf{x}))$$

- We want to understand the dependence on θ . We showed that the full gradient is

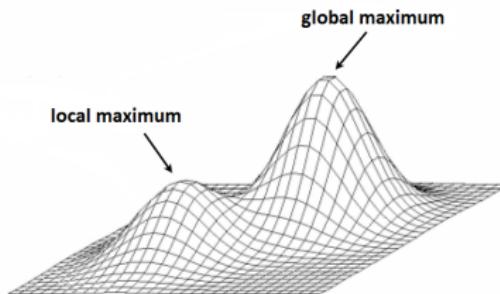
$$\nabla_{\theta} \log Z(\theta) = \sum_{\mathbf{x}} \mathbf{f}(\mathbf{x}) p(\mathbf{x}; \theta) = \mathbb{E}_{p(\mathbf{x}; \theta)} [\mathbf{f}(\mathbf{x})]$$

- Similarly, you can show that 2nd derivative of the log-partition function gives the second-order moments, i.e.

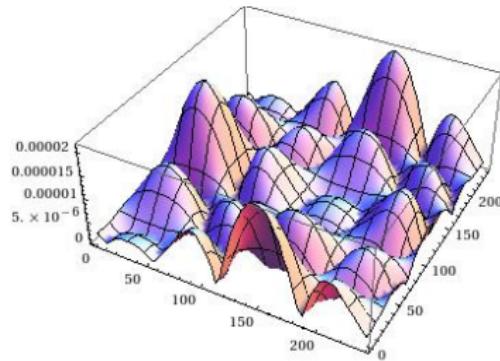
$$\left(\nabla^2 \log Z(\theta) \right)_{ij} = \mathbb{E}_{p(\mathbf{x}; \theta)} [f^i(\mathbf{x}) f^j(\mathbf{x})] - \mathbb{E}_{p(\mathbf{x}; \theta)} [f^i(\mathbf{x})] \mathbb{E}_{p(\mathbf{x}; \theta)} [f^j(\mathbf{x})] = (\text{cov}[\mathbf{f}(\mathbf{x})])_{ij}$$

- Since covariance matrices are always positive semi-definite, this proves that $\log Z(\theta)$ is convex (so $-\log Z(\theta)$ is concave)

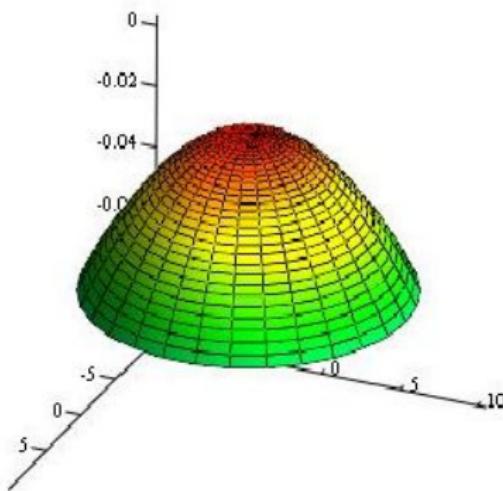
Non-concave optimization problem



- Difficult to optimize. Can get stuck in a local optimum.



Concave optimization problem



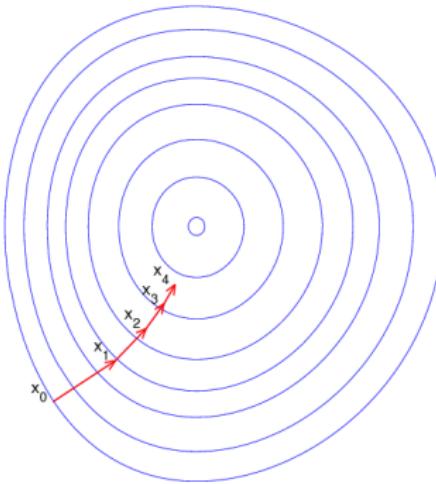
- Easy to optimize
- Local maximizers are global maximizers

Gradient Ascent

- Simplest optimization approach. To maximize $f(x)$, iterate

$$x_{t+1} = x_t + \eta_t \nabla f(x_t)$$

where η_t is a (small) stepsize.



Solving the maximum likelihood problem in MRFs

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \theta^T \left(\sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - |\mathcal{D}| \log Z(\theta) \\ &\propto \theta^T \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \log Z(\theta)\end{aligned}$$

- The objective is concave in θ . No local optima, gradient ascent will not get stuck!
 - Initialize θ_0
 - Repeat until convergence
 - Evaluate $\nabla_\theta \ell(\theta; \mathcal{D})|_{\theta=\theta_t} = \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \nabla_\theta \log Z(\theta)|_{\theta=\theta_t}$
 - $\theta_{t+1} = \theta_t + \eta_t \nabla_\theta \ell(\theta; \mathcal{D})|_{\theta=\theta_t}$

Solving the maximum likelihood problem in MRFs

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \theta^T \left(\sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - |\mathcal{D}| \log Z(\theta) \\ &\propto \theta^T \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \log Z(\theta)\end{aligned}$$

- Let's study some properties of the ML solution:

$$\begin{aligned}\nabla_\theta \ell(\theta; \mathcal{D}) &= \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \nabla_\theta \log Z(\theta) \\ &= \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \mathbb{E}_{p(\mathbf{x}; \theta)} [\mathbf{f}(\mathbf{x})]\end{aligned}$$

The gradient of the log-likelihood

$$\nabla_{\theta} \ell(\theta; \mathcal{D}) = \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \mathbb{E}_{p(\mathbf{x}; \theta)}[\mathbf{f}(\mathbf{x})]$$

- Difference of expectations: empirical (data) vs. model!
- Consider the earlier pairwise MRF example. Let's look at one component:

$$\left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c) \right) - \mathbb{E}_{p(\mathbf{x}; \theta)}[\mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c)]$$

Setting the gradient to zero, the model marginals for ML solution equal the empirical marginals!

- **Moment matching:** property of maximum likelihood learning (in exponential families)

Gradient ascent requires repeated marginal inference,
which in many models is **hard**!

How can we get around the complexity of inference during learning?

Learning with approximate inference

- Recall the original learning objective

$$\ell(\theta; \mathcal{D}) = \theta^T \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \log Z(\theta)$$

- All we need for learning (i.e., to compute derivatives of $\ell(\theta; \mathcal{D})$) are **marginals** of the distribution
- Use any of the approximate inference techniques (e.g., Gibbs sampling)
- Algorithm:
 - Initialize θ^0
 - Repeat until convergence:
 - Estimate $\nabla_{\theta} \ell(\theta; \mathcal{D}) = \left(\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{f}(\mathbf{x}) \right) - \mathbb{E}_{p(\mathbf{x}; \theta^t)}[\mathbf{f}(\mathbf{x})]$
 - Update in the direction of the gradient: $\theta^{t+1} = \theta^t + \alpha \nabla_{\theta} \ell(\theta; \mathcal{D})$

Summary

- Learning MRFs is hard (even with complete data)
- Likelihood does not decompose nicely
- Needs inference to evaluate likelihood and gradient
- Usually, need to use approximate inference (sampling)

Pseudo-likelihood

- Alternatively, can we come up with a *different* objective function (i.e., a different *estimator*) which succeeds at learning while avoiding inference altogether?
- Pseudo-likelihood method (Besag 1971) yields an exact solution if the data is generated by a model in our model family $p(\mathbf{x}; \theta^*)$ and $|\mathcal{D}| \rightarrow \infty$ (i.e., it is **consistent**)
- Note that, via the chain rule,

$$p(\mathbf{x}; \theta) = \prod_i p(x_i | x_1, \dots, x_{i-1}; \theta)$$

- We consider the following approximation:

$$p(\mathbf{x}; \theta) \approx \prod_i p(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n; \theta) = \prod_i p(x_i | x_{-i}; \theta)$$

where we have added conditioning over additional variables

Pseudo-likelihood

- The pseudo-likelihood method replaces the likelihood,

$$\ell(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \log p(\mathcal{D}; \theta) = \frac{1}{|\mathcal{D}|} \sum_{m=1}^{|\mathcal{D}|} \log p(\mathbf{x}^m; \theta)$$

with the following approximation:

$$\ell_{PL}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{m=1}^{|\mathcal{D}|} \sum_{i=1}^n \log p(x_i^m | x_{N(i)}^m; \theta)$$

(we replaced x_{-i} with $x_{N(i)}$, i 's Markov blanket)

- For example, suppose we have a pairwise MRF. Then,

$$p(x_i^m | x_{N(i)}^m; \theta) = \frac{1}{Z(x_{N(i)}^m; \theta)} e^{\sum_{j \in N(i)} \theta_{ij}(x_i^m, x_j^m)}, \quad Z(x_{N(i)}^m; \theta) = \sum_{\hat{x}_i} e^{\sum_{j \in N(i)} \theta_{ij}(\hat{x}_i, x_j^m)}$$

Pseudo-likelihood

$$\ell_{PL}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{m=1}^{|\mathcal{D}|} \sum_{i=1}^n \log p(x_i^m | x_{N(i)}^m; \theta)$$

- This objective only involves summation over x_i and is tractable
- Has many small partition functions (one for each variable and each setting of its neighbors) instead of one big one
- It is still concave in θ and thus easy to optimize. Let θ^{PL} be the maximizer.
- **Theorem:** Assuming the data is drawn from a MRF with parameters θ^* , can show that as the number of data points gets large, $\theta^{PL} \rightarrow \theta^*$
- Intuition: pseudolikelihood tries to match all conditional distributions to the data. If it succeeds, then a Gibbs sampler run on the model distribution will have the same invariant distribution as a Gibbs sampler run on the true data distribution.

How about conditional models?

Conditional random fields

- Recall a CRF is a Markov network on variables $\mathbf{X} \cup \mathbf{Y}$, which specifies the conditional distribution

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in C} \phi_c(\mathbf{x}, \mathbf{y}_c)$$

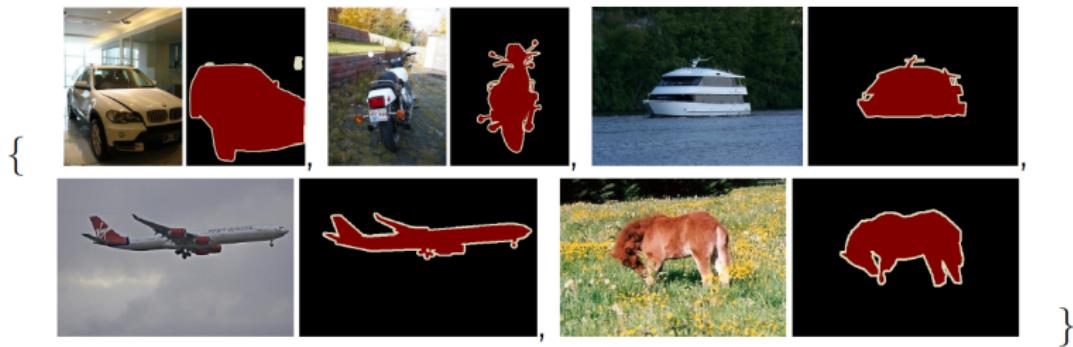
with partition function

$$Z(\mathbf{x}) = \sum_{\hat{\mathbf{y}}} \prod_{c \in C} \phi_c(\mathbf{x}, \hat{\mathbf{y}}_c).$$

- The feature functions now depend on \mathbf{x} in addition to \mathbf{y}
- For each potential c , a vector-valued **feature function** $\mathbf{f}_c(\mathbf{x}, \mathbf{y}_c) \in \mathbb{R}^d$
- Then, $\phi_c(\mathbf{x}, \mathbf{y}_c; \theta) = \exp(\theta_c \cdot \mathbf{f}_c(\mathbf{x}, \mathbf{y}_c))$

Learning with conditional random fields

- Given: parametric model (family) $P(\mathbf{y} | \mathbf{x}, \theta) = \frac{1}{Z(\mathbf{x}; \theta)} \prod_{c \in C} \exp(\theta_c \cdot \mathbf{f}_c(\mathbf{x}, \mathbf{y}_c))$
- Not given: parameter vector θ
- Given: example pairs $\{(x_1, y_1), \dots, (x_M, y_M)\} \subseteq \mathcal{X} \times \mathcal{Y}$, typical inputs with “the right” outputs for them.
- Goal: find a “good” θ that captures the relationship between x and y .



Optimizing conditional log-likelihood

- Training set $\mathcal{D} = \{(\mathbf{x}^1, \mathbf{y}^1), \dots, (\mathbf{x}^M, \mathbf{y}^M)\}$ of pairs (e.g., sentence and tags).
- Objective: conditional log-likelihood $\sum_{i=1}^M \log P(\mathbf{y}^i | \mathbf{x}^i, \theta)$
- $P(\mathbf{y}^i | \mathbf{x}^i, \theta) = \frac{1}{Z(\mathbf{x}^i; \theta)} \prod_{c \in C} \exp(\theta_c \cdot \mathbf{f}_c(\mathbf{x}^i, \mathbf{y}_c^i))$
- Same as learning with MRFs, except that we have a different partition function for each data point

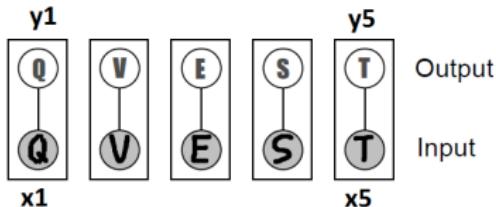
$$\theta^{ML} = \arg \max_{\theta} \theta \cdot \left(\sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{D}} \sum_c \mathbf{f}_c(\mathbf{x}^i, \mathbf{y}_c^i) \right) - \sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{D}} \log Z(\mathbf{x}^i; \theta)$$

- Good news: conditional log-likelihood is still a concave function. Can do gradient ascent as before
- Bad news: computing gradient requires now requires **one inference per training data point** $(\mathbf{x}^i, \mathbf{y}^i)$:

- $\nabla_{\theta} \sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{D}} \log Z(\mathbf{x}^i; \theta) = \sum_{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{D}} \nabla_{\theta} \log Z(\mathbf{x}^i; \theta)$

Handwriting Recognition Example

- input space $\mathcal{X} = 5\text{-letter word images}, x = (x_1, \dots, x_5), x_j \in \{0, 1\}^{300 \times 80}$
- output space $\mathcal{Y} = \text{ASCII translation}, y = (y_1, \dots, y_5), y_j \in \{A, \dots, Z\}$



- feature functions based on unary terms
 $f(x, y) = (\psi_1(x_1, y_1), \dots, \psi_5(x_5, y_5))$. Each feature measures how close image x_i is to the typical way of drawing letter y_i .
- Advantage: Inference is easy. We can find each y_i independently, check $5 \times 26 = 130$ values.
- Problem: only local information, we cannot correct errors.

Handwriting Recognition Example

- input space $\mathcal{X} = 5\text{-letter word images}$, $x = (x_1, \dots, x_5)$, $x_j \in \{0, 1\}^{300 \times 80}$
- output space $\mathcal{Y} = \text{ASCII translation}$, $y = (y_1, \dots, y_5)$, $y_j \in \{A, \dots, Z\}$
- Global feature function:

$$\varphi(x, y) = \begin{cases} (0, \dots, 0, \overset{y\text{th pos.}}{\widehat{\Phi(x)}}, 0, \dots, 0) & \text{if } y \in \mathcal{D} \text{ dictionary,} \\ (0, \dots, 0, 0, 0, \dots, 0) & \text{otherwise.} \end{cases}$$

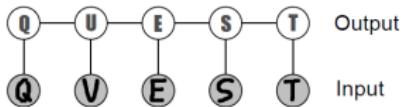


- Advantage: access to global information, e.g. can check if the word exists in a dictionary.
- Problem: Inference is expensive! Have to check $26^5 = 11881376$ values.
Need more training data

Handwriting Recognition Example

- input space $\mathcal{X} = 5\text{-letter word images}, x = (x_1, \dots, x_5), x_j \in \{0, 1\}^{300 \times 80}$
- output space $\mathcal{Y} = \text{ASCII translation}, y = (y_1, \dots, y_5), y_j \in \{A, \dots, Z\}$
- Feature functions based on unary and binary terms

$$\varphi(x, y) = (\varphi_1(y_1, x), \varphi_2(y_2, x), \dots, \varphi_5(y_5, x), \\ \varphi_{1,2}(y_1, y_2), \dots, \varphi_{4,5}(y_4, y_5))$$



- Compromise: Inference is easy, it's a tree.
- Compromise: can take into account some local interactions, e.g., *QUE* is more likely than *QVE*

Summary

- Learning MRF/CRF is hard even in the fully observed case
- The problematic part is the global normalization $Z(\theta)$
- Maximum likelihood learning requires inference
- Other objectives are possible: pseudolikelihood, classification error, etc.

Mixture Models & EM algorithm

CS228 Probabilistic Graphical Models

Stefano Ermon

Slides adapted from David Sontag, Carlos Guestrin, Dan Klein, Luke Zettlemoyer, Dan Weld, Vibhav Gogate, and Andrew Moore

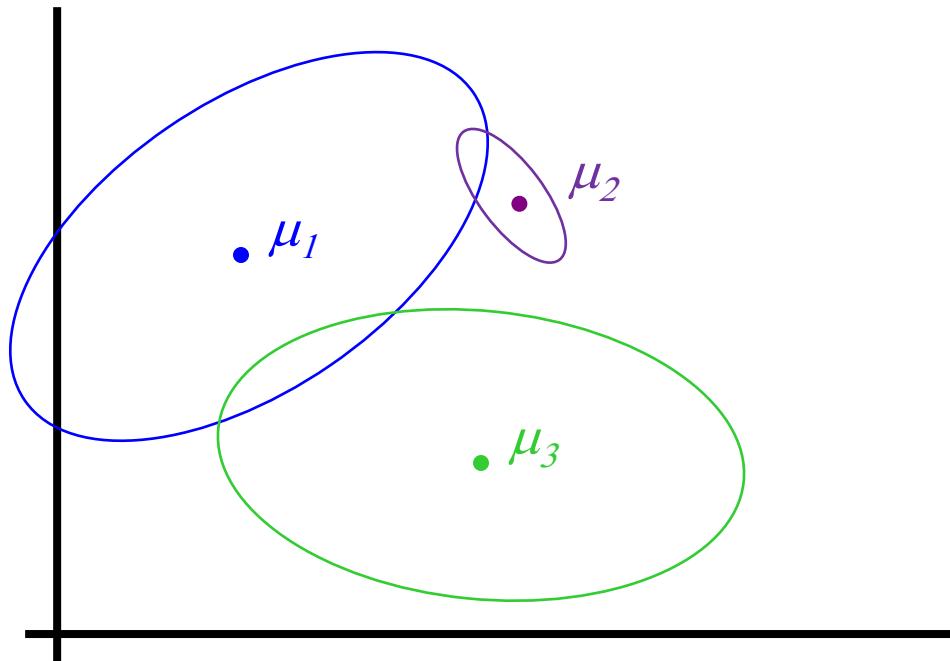
Mixture of Gaussians model

- $P(Y)$: There are k components
- $P(X|Y)$: Each component generates data from a **multivariate Gaussian** with mean μ_i and covariance matrix Σ_i

Each data point is sampled from a **generative process**:

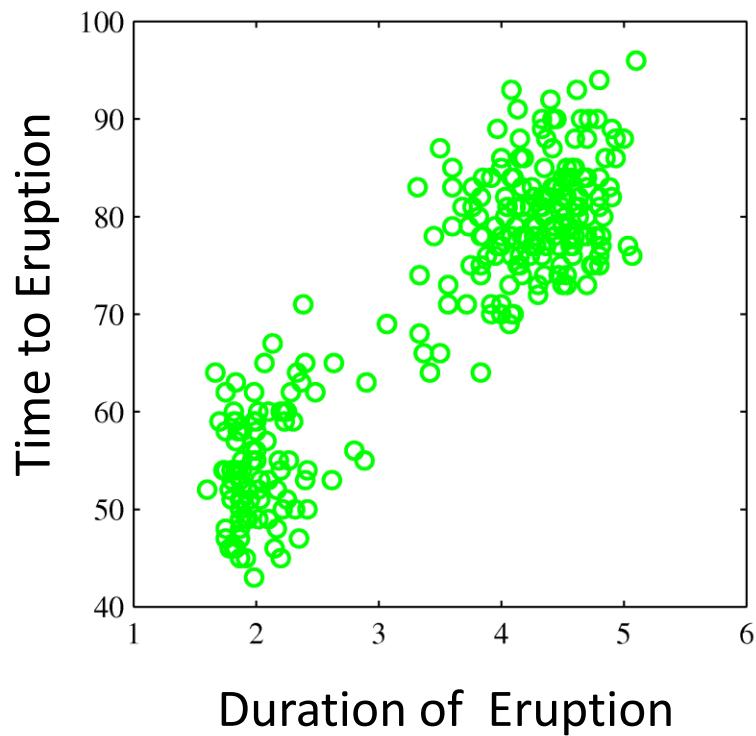
1. Choose component i with probability $P(y=i)$
2. Generate datapoint $\sim N(\mu_i, \Sigma_i)$

Gaussian mixture model
(GMM)



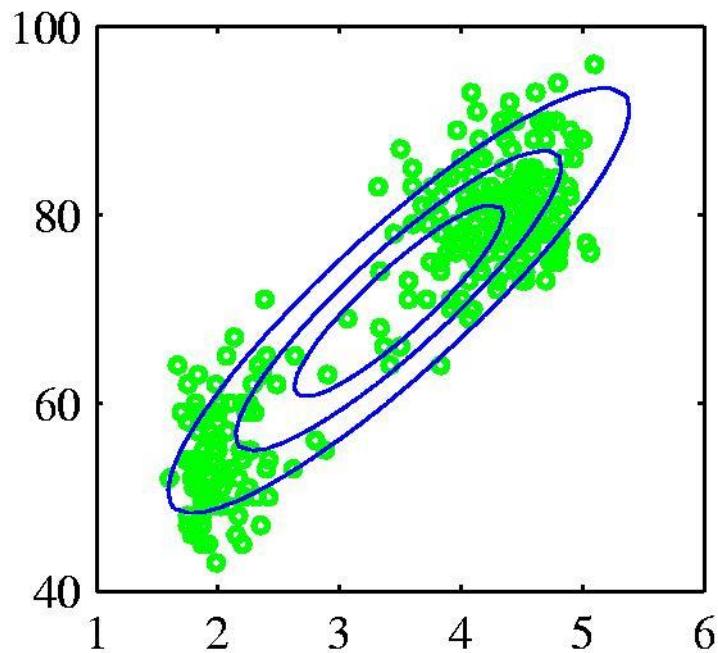
Mixtures of Gaussians (1)

Old Faithful Data Set

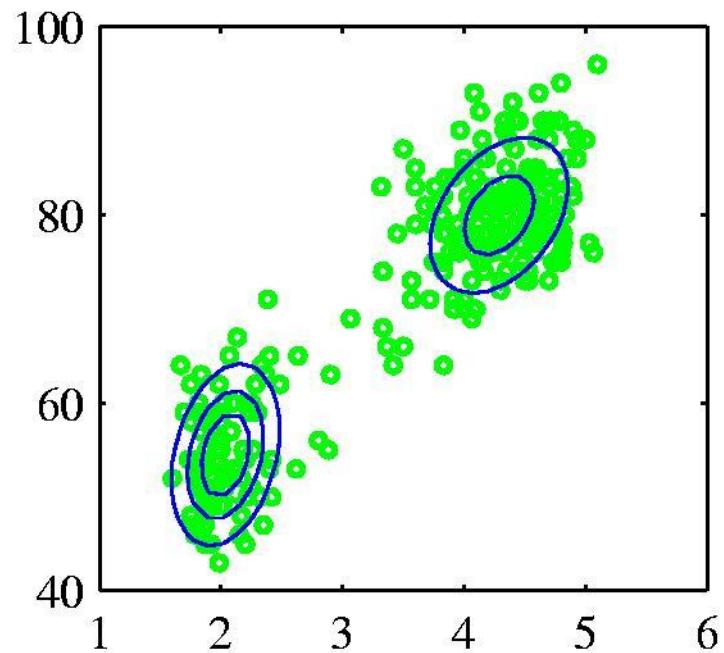


Mixtures of Gaussians (1)

Old Faithful Data Set



Single Gaussian



Mixture of two Gaussians

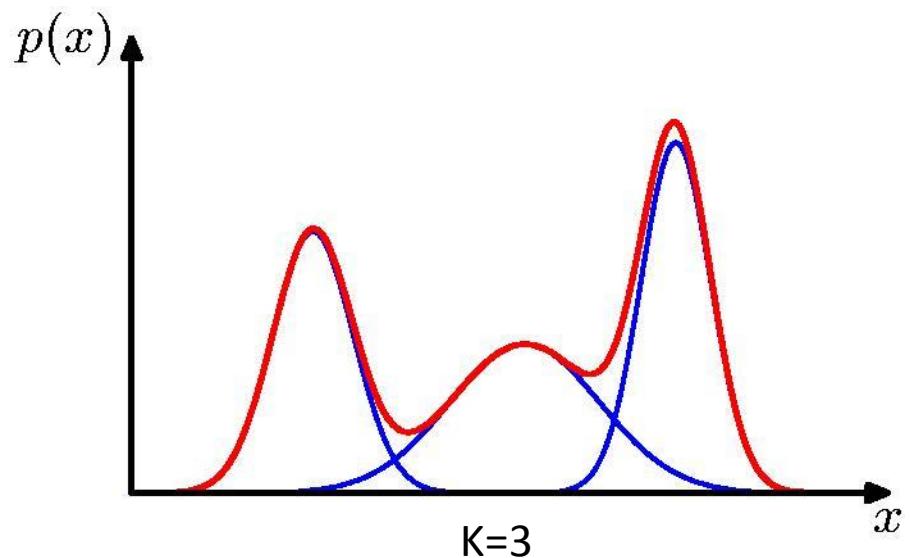
Mixtures of Gaussians (2)

Combine simple models into a complex model:

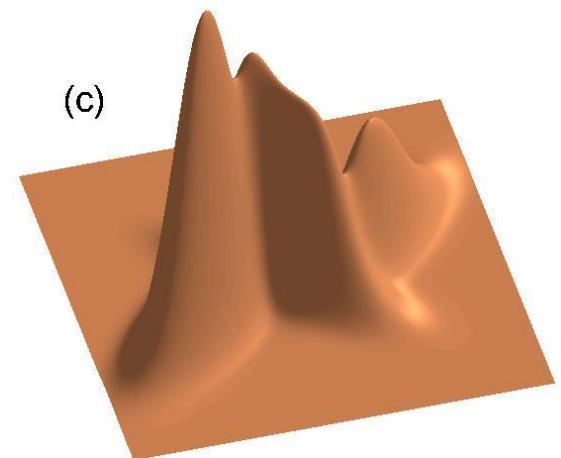
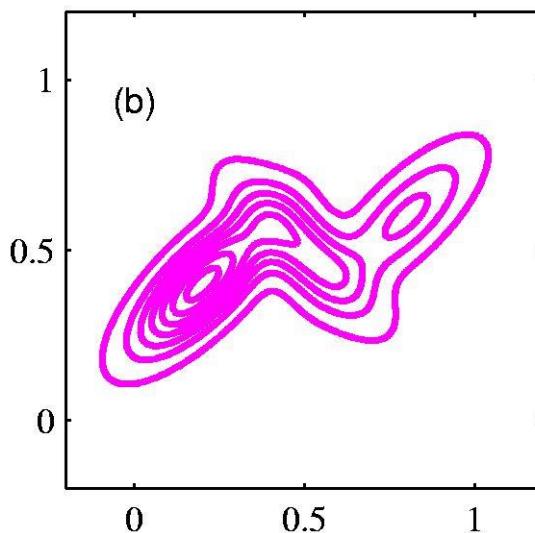
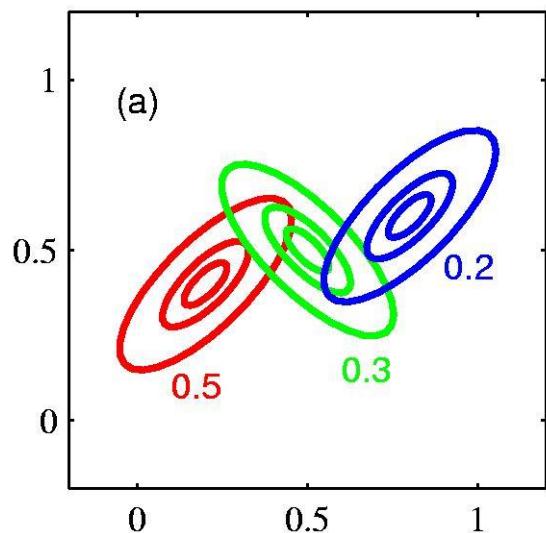
$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

↑
Component
Mixing coefficient

$$\forall k : \pi_k \geq 0 \quad \sum_{k=1}^K \pi_k = 1$$

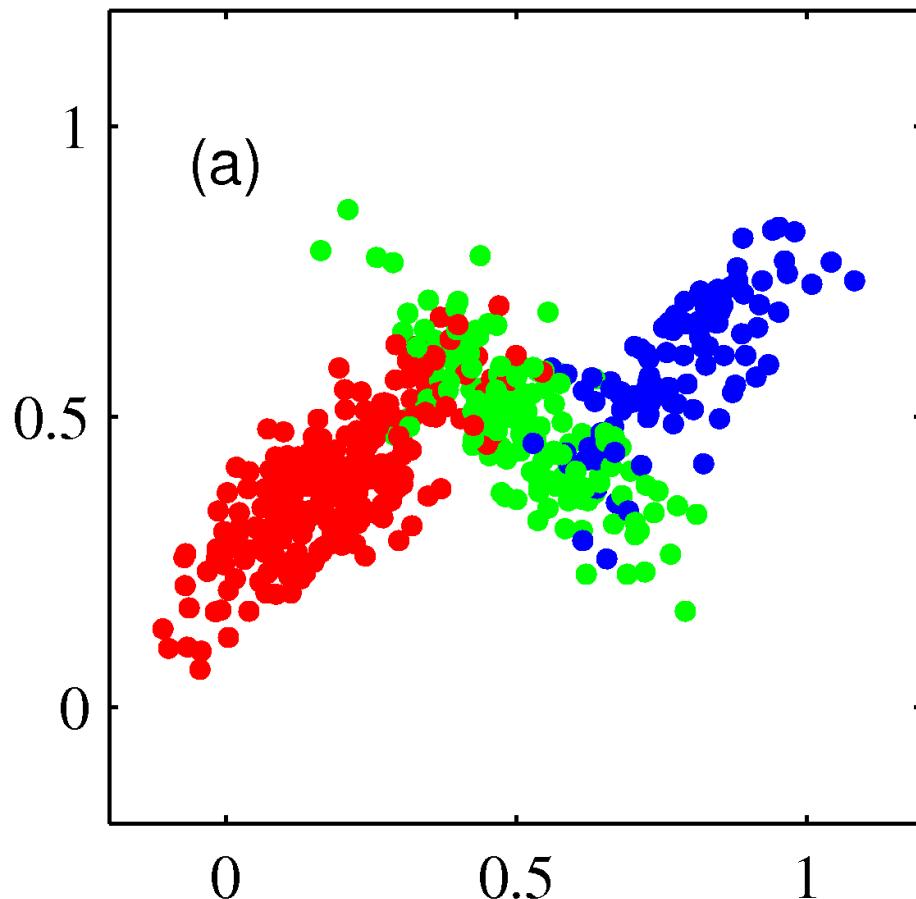


Mixtures of Gaussians (3)



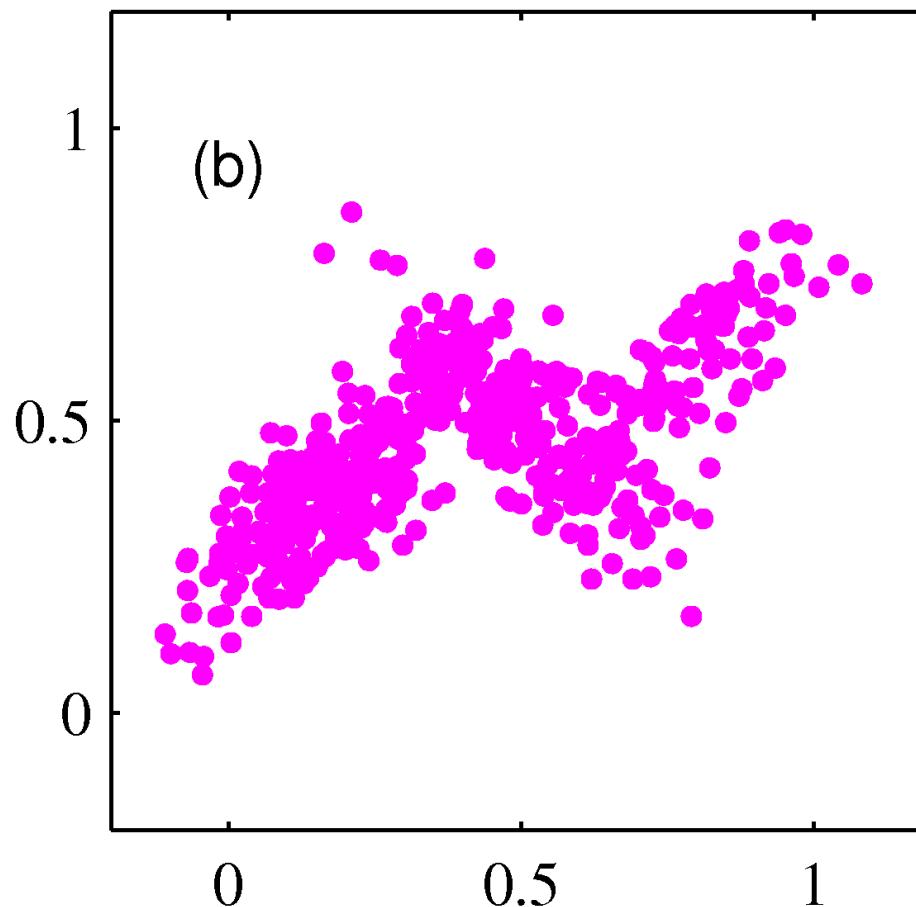
Unsupervised learning

Model data as mixture of multivariate Gaussians



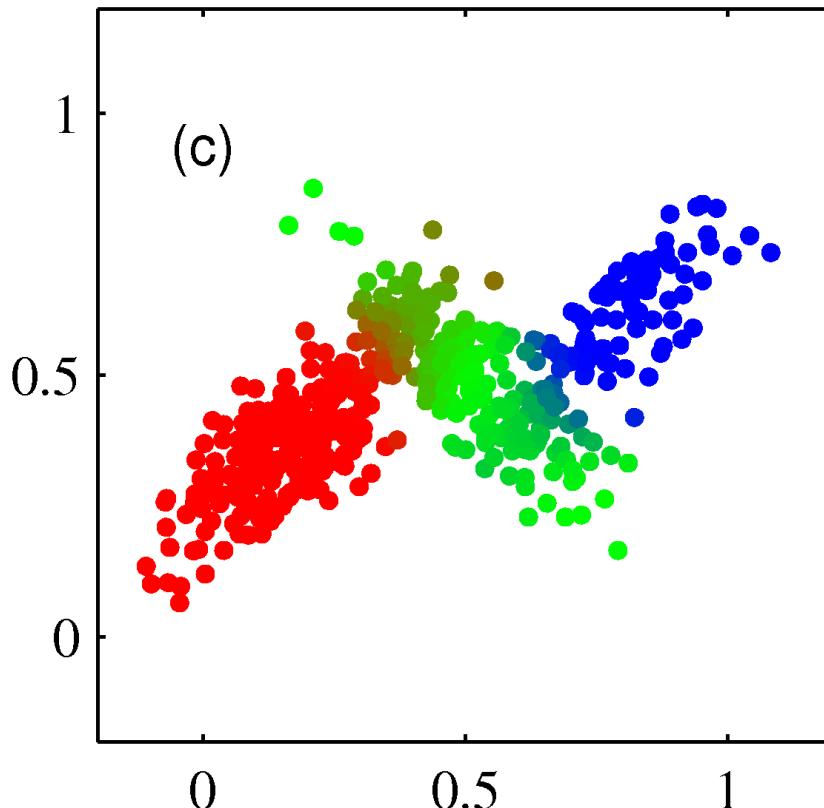
Unsupervised learning

Model data as mixture of multivariate Gaussians



Unsupervised learning

Model data as mixture of multivariate Gaussians



Shown is the *posterior probability* that a point was generated from i^{th} Gaussian: $\Pr(Y = i \mid x)$

ML estimation in supervised setting

- Univariate Gaussian

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x_i \quad \sigma_{MLE}^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})^2$$

- **Mixture of Multivariate Gaussians**

ML estimate for each of the Multivariate Gaussians is given by:

$$\hat{\mu}_{ML} = \frac{1}{n} \sum_{j=1}^n x_j$$

$$\hat{\Sigma}_{ML} = \frac{1}{n} \sum_{j=1}^n (\mathbf{x}_j - \hat{\mu}_{ML})(\mathbf{x}_j - \hat{\mu}_{ML})^T$$

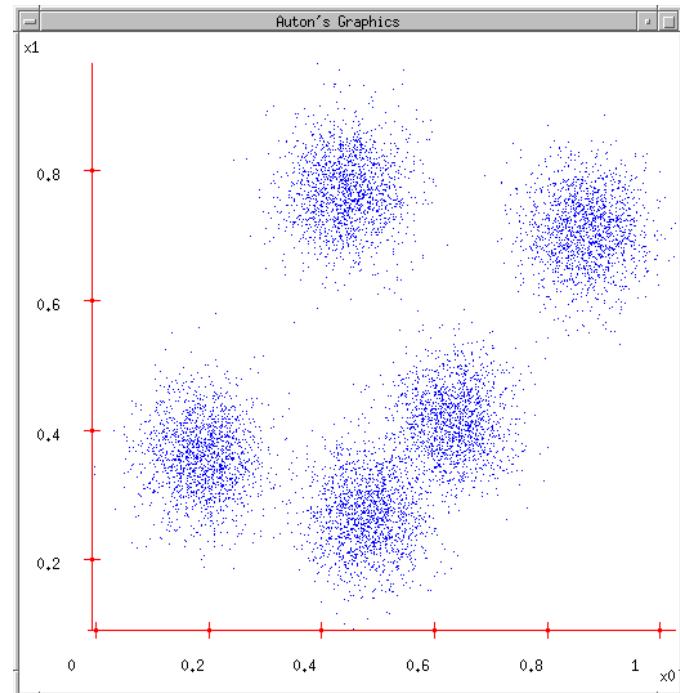


Just sums over x generated from the k 'th Gaussian

That was easy!

But what if *unobserved data*?

- MLE:
 - $\operatorname{argmax}_{\theta} \prod_j P(y_j, x_j)$
 - θ : all model parameters
 - eg, class probs, means, and variances
- But we don't know y_j 's!!!
- Maximize *marginal likelihood*:
 - $\operatorname{argmax}_{\theta} \prod_j P(x_j) = \operatorname{argmax}_{\theta} \prod_j \sum_{k=1}^K P(Y_j=k, x_j)$



EM: Two Easy Steps

Objective: $\operatorname{argmax}_{\theta} \log \prod_j \sum_k P(Y_j=k, x_j | \theta) = \sum_j \log \sum_k P(Y_j=k, x_j | \theta)$

Data: $\{x_j \mid j=1 \dots n\}$

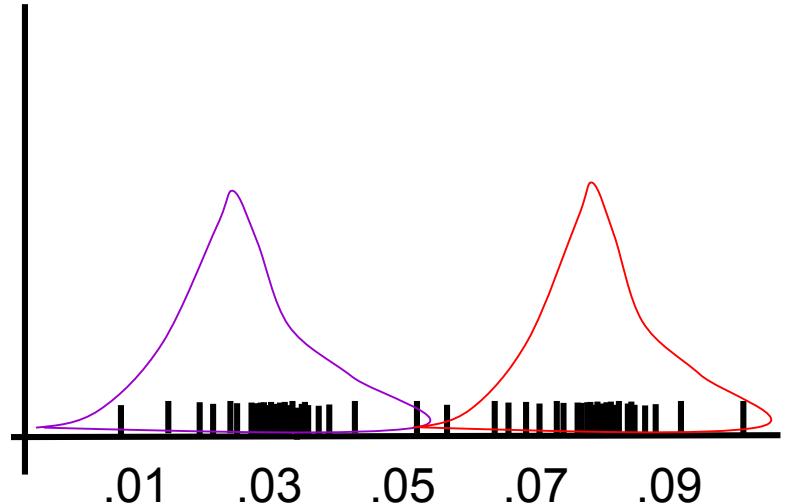
- **E-step:** Compute expectations to “fill in” missing y values according to current parameters, θ
 - For all examples j and values k for Y_j , compute: $P(Y_j=k \mid x_j, \theta)$
- **M-step:** Re-estimate the parameters with “weighted” MLE estimates
 - Set $\theta = \operatorname{argmax}_{\theta} \sum_j \sum_k P(Y_j=k \mid x_j, \theta) \log P(Y_j=k, x_j | \theta)$

Especially useful when the E and M steps have closed form solutions!!!

Simple example: learn means only!

Consider:

- 1D data
- Mixture of $K=2$ Gaussians
- Variances fixed to $\sigma=1$
- Distribution over classes is uniform
- Just need to estimate μ_1 and μ_2



$$\prod_{j=1}^m \sum_{k=1}^K P(x, Y_j = k) \propto \prod_{j=1}^m \sum_{k=1}^{K=2} \exp\left[-\frac{1}{2\sigma^2} \|x - \mu_k\|^2\right] P(Y_j = k)$$

EM for GMMs: only learning means

Iterate: On the t 'th iteration let our estimates be

$$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \dots \mu_K^{(t)} \}$$

E-step

Compute “expected” classes of all datapoints

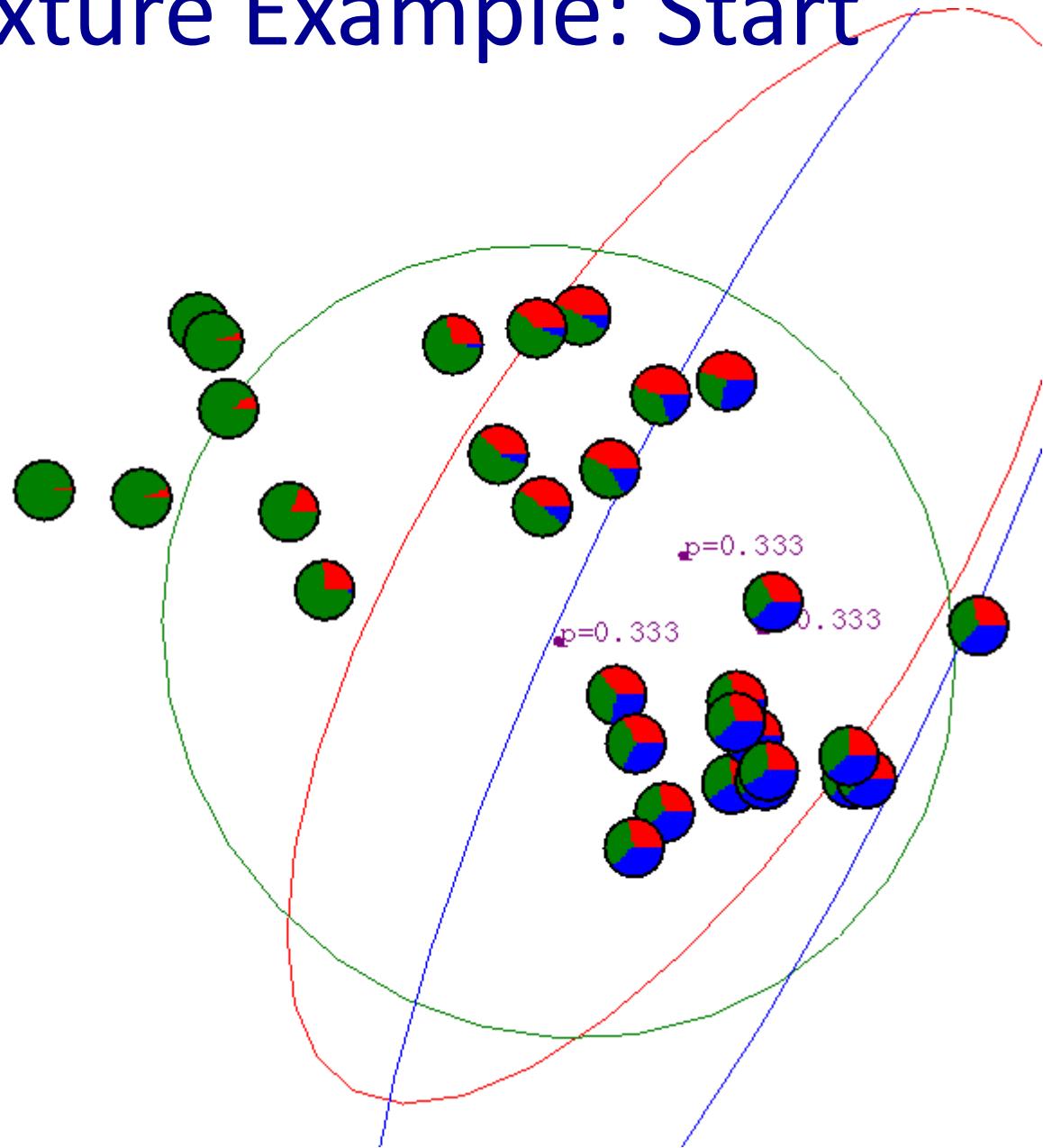
$$P(Y_j = k | x_j, \mu_1 \dots \mu_K) \propto \exp\left(-\frac{1}{2\sigma^2} \|x_j - \mu_k\|^2\right) P(Y_j = k)$$

M-step

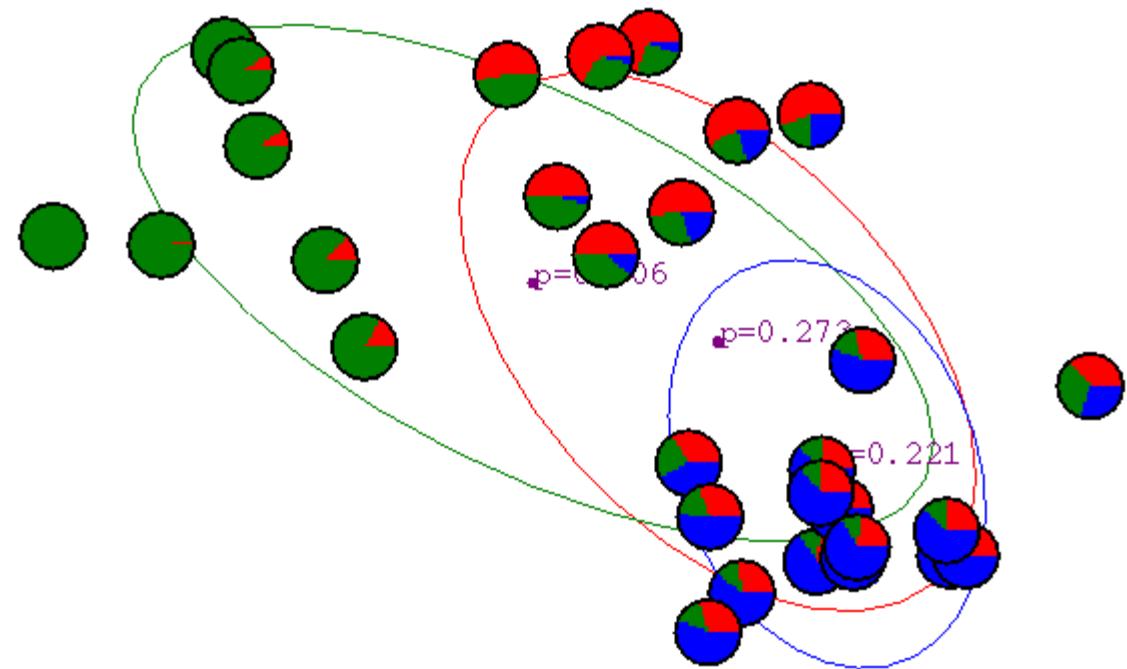
Compute most likely new μ s given class expectations

$$\mu_k = \frac{\sum_{j=1}^m P(Y_j = k | x_j) x_j}{\sum_{j=1}^m P(Y_j = k | x_j)}$$

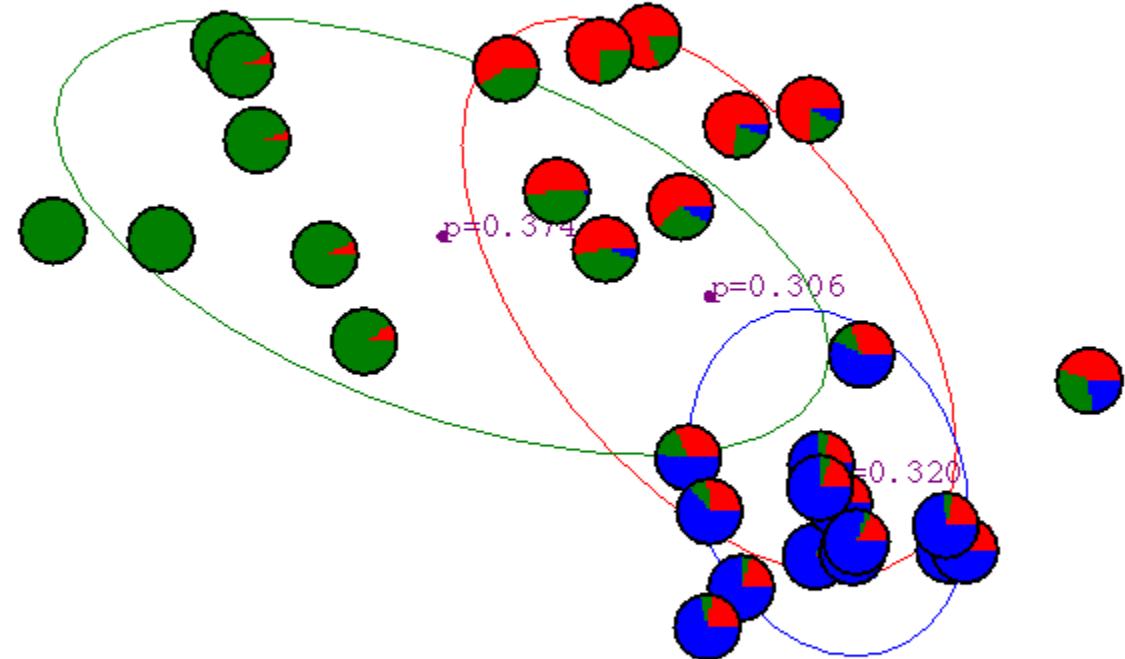
Gaussian Mixture Example: Start



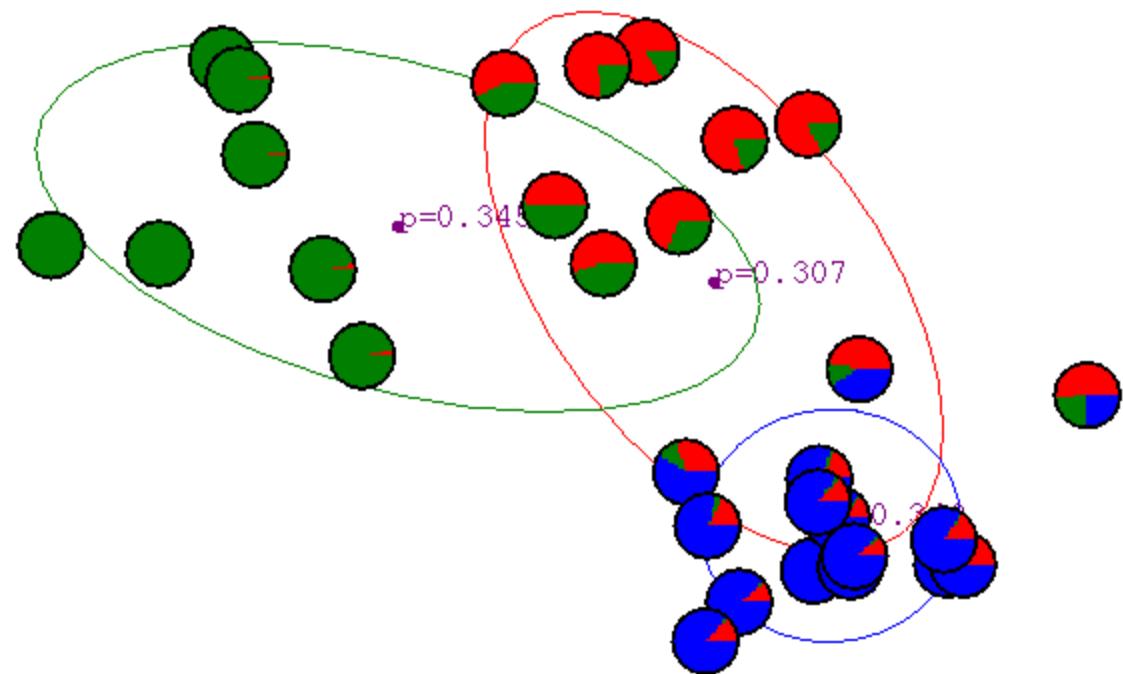
After first iteration



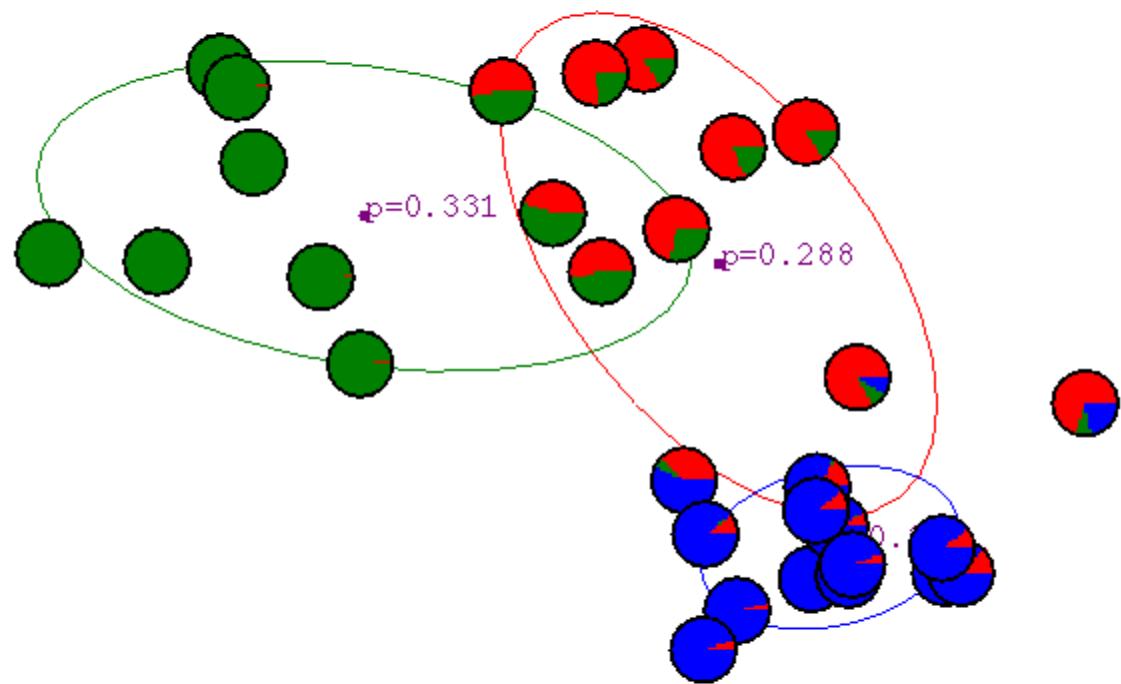
After 2nd iteration



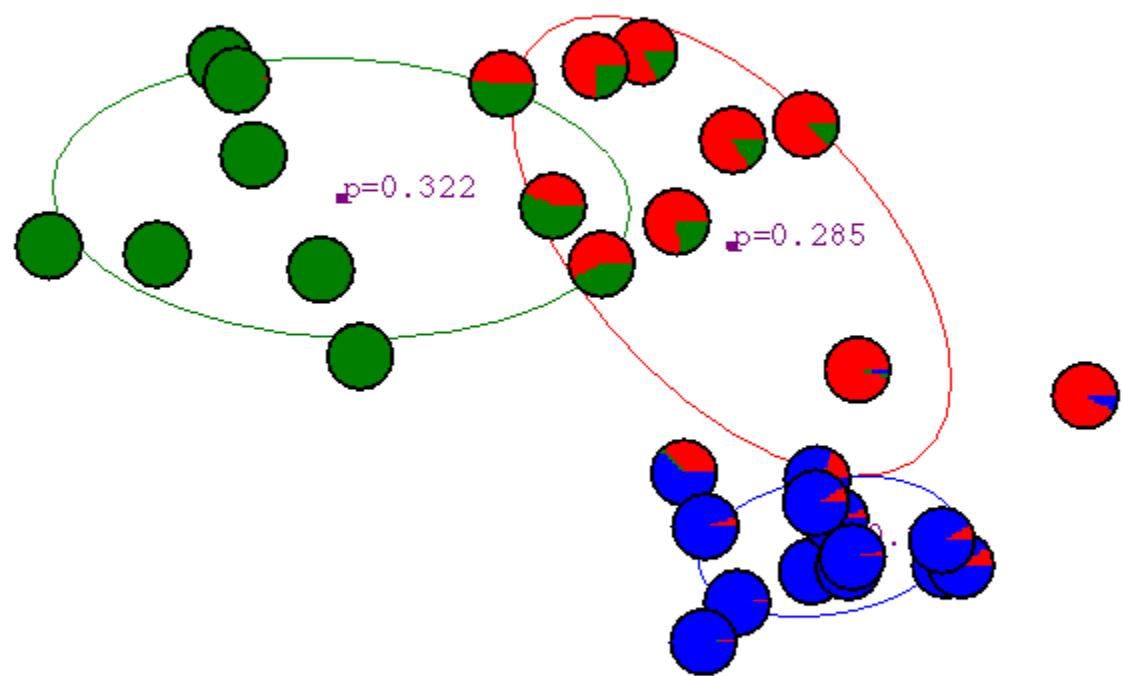
After 3rd iteration



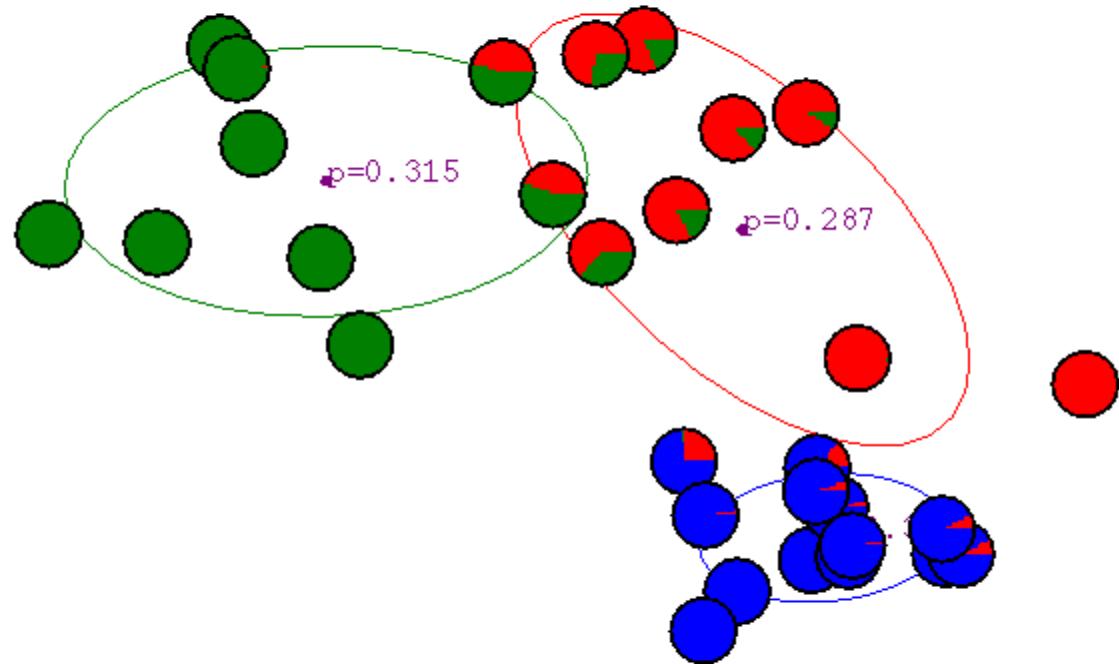
After 4th iteration



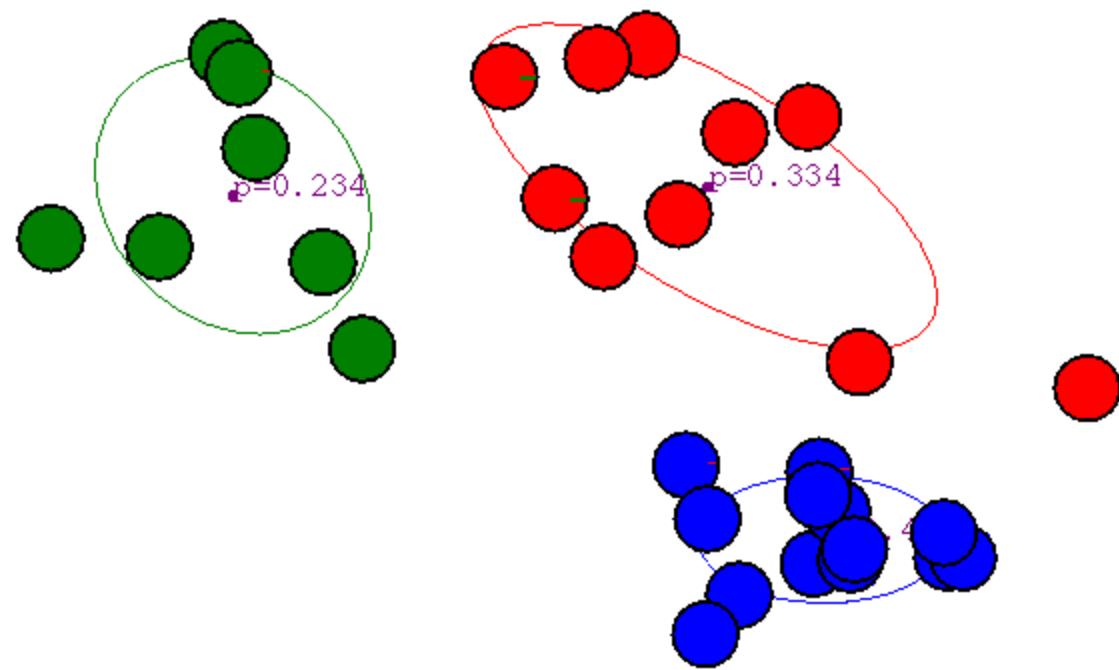
After 5th iteration



After 6th iteration



After 20th iteration



Summary

- Learning parameters with missing data is hard
- Can use an iterative procedure called EM
 - Fill-in missing values (inference)
 - Re-estimate parameters
- Iteratively improves a lower bound on the marginal likelihood
- Useful for unsupervised learning

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 14, February 23, 2017

Today: learning with missing data

- ① Learning with Missing Data
- ② Bayes Net parameter learning with partially observed data
- ③ Expectation Maximization
- ④ Gaussian Mixture models

Partially observed data

- Suppose that our joint distribution is

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

where our samples \mathbf{X} are observed and the variables \mathbf{Z} are never observed in \mathcal{D}

- For example, $\mathcal{D} = \{(0, 1, 0, ?, ?, ?), (1, 1, 1, ?, ?, ?), (1, 1, 0, ?, ?, ?), \dots\}$
- As another example, you might have unlabeled data
 $\{(\text{email}_1, \text{Spam}), (\text{email}_2, \text{Spam}), (\text{email}_3, \text{NotSpam}), (\text{email}_4, ?), \dots\}$

Maximum likelihood

- We can still use the same maximum likelihood approach. The objective that we are maximizing is the (log) **marginal likelihood**

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)\end{aligned}$$

- Because of the sum over \mathbf{z} , there is no longer a closed-form solution for the maximum likelihood estimate θ^* (even in the case of Bayesian networks)

Why is parameter learning in presence of Partially Observed Data challenging?

Likelihood function for Partially Observed Data:

$$\prod_{j=1}^m \sum_{\mathbf{z} \notin \mathbf{x}^{(j)}} p_{\theta}(\mathbf{x}^{(j)}, \mathbf{z})$$

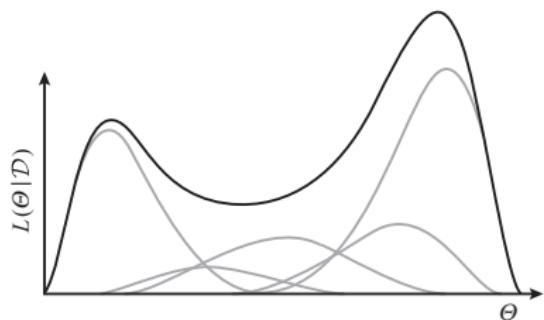
Compare with Likelihood function for Fully Observed Data:

$$\prod_{j=1}^m p_{\theta}(\mathbf{x}^{(j)})$$

Likelihood function for Partially Observed Data:

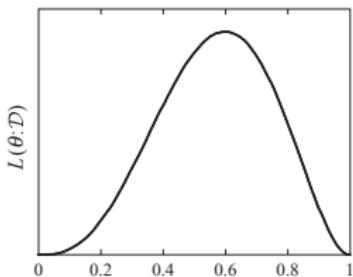
- cannot be expressed in closed form
- is not decomposable (by variable and parent assignment)
- is not unimodal

Why is parameter learning in presence of POD challenging?



Partially Observed case:

Each point in the sum yields a unimodal distribution. When combined, we get a multi-modal distribution.



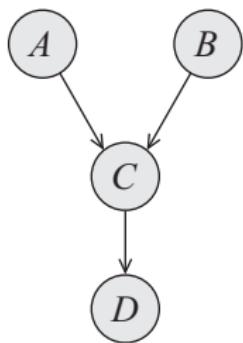
Fully Observed case:
Unimodal distribution

- The optimization problem, i.e. maximizing the *marginal* likelihood of the data, is hard. We need an iterative approach.

The Expectation Maximization (EM) Algorithm

- Start with an initial guess (random) of the parameters $\theta^{(0)}$
- Repeat until convergence
 - ① Complete ("hallucinate") the incomplete data using current parameters (E-step)
 - ② Train: Update the parameters based on the completed data (M-step)

The Expectation Maximization Algorithm: Example



$$\begin{aligned}\theta_a &= .3 \\ \theta_b &= .9 \\ \theta_{c|\bar{a},\bar{b}} &= .83 \\ \theta_{c|\bar{a},b} &= .09 \\ \theta_{c|a,\bar{b}} &= .6 \\ \theta_{c|a,b} &= .2 \\ \theta_{d|\bar{c}} &= .1 \\ \theta_{d|c} &= .8\end{aligned}$$

Data instance: $(a, ?, ?, \bar{d})$

How to complete this example?

For each possible completion

- STEP 1: Compute how likely the completion is (given the observed part).
- STEP 2: Data set is now weighted (similar to importance sampling)

The Expectation Maximization Algorithm: E-Step

- Data set is now **bigger** and **weighted**
- If binary variables, $(a, ?, ?, \bar{d})$ corresponds to four weighted examples
 - (a, b, c, \bar{d}) , weight = .0492
 - (a, b, \bar{c}, \bar{d}) , weight = .8852
 - (a, \bar{b}, c, \bar{d}) , weight = .0164
 - $(a, \bar{b}, \bar{c}, \bar{d})$, weight = .0492
- weight = how likely the completion is according to current parameter estimates

The Expectation Maximization Algorithm: M-Step

Updating: $\theta_{d|\bar{c}}$

- Unweighted MLE estimate:

$$\theta_{d|\bar{c}} = \frac{\#(d, \bar{c})}{\#(\bar{c})}$$

- Weighted MLE estimate:

$$\theta_{d|\bar{c}} = \frac{\sum weight(d, \bar{c})}{\sum weight(\bar{c})} = \frac{\sum_{j=1}^m p_\theta(d, \bar{c} | \mathbf{x}^{(j)})}{\sum_{j=1}^m p_\theta(\bar{c} | \mathbf{x}^{(j)})}$$

E-step

- \mathbf{x} : observed data; \mathbf{z} : hidden data
- $\theta^{(t)}$ current parameter guess
- Hallucinate missing values by computing distribution over hidden variables using current parameter estimate
- For each training example $\mathbf{x}^{(j)}$, compute:

$$Q^{(t+1)}(\mathbf{z}|\mathbf{x}^{(j)}) = P(\mathbf{z}|\mathbf{x}^{(j)}, \theta^{(t)})$$

Towards M-step

- Marginal likelihood does not decompose

$$\ell(\theta; \mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$$

- How do we make progress?
- Idea: use Jensen Inequality (for concave functions)

$$\log \left(\sum_{\mathbf{z}} P(\mathbf{z}) f(\mathbf{z}) \right) \geq \sum_{\mathbf{z}} P(\mathbf{z}) \log f(\mathbf{z})$$

to go from the logarithm of a sum (hard) to a sum of logarithms (easy)

Towards M-step

- From the E-step:

$$Q^{(t+1)}(\mathbf{z}|\mathbf{x}^{(j)}) = P(\mathbf{z}|\mathbf{x}^{(j)}, \theta^{(t)})$$

- Let's use Jensen Inequality

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta) \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{z}; \theta)}{Q^{(t+1)}(\mathbf{z}|\mathbf{x})} \\ &\geq \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{z}; \theta)}{Q^{(t+1)}(\mathbf{z}|\mathbf{x})} \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta) + const\end{aligned}$$

Towards M-step

- Lower bound of marginal likelihood with hidden variables

$$\ell(\theta; \mathcal{D}) \geq \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta) + \text{const}$$

- Recall: Likelihood in fully observable (FO) case:

$$\ell^{FO}(\theta; \mathcal{D}) = \sum_{(\mathbf{x}, \mathbf{z}) \in \mathcal{D}} \log p(\mathbf{x}, \mathbf{z}; \theta)$$

- Lower-bound interpreted as “weighted” data set

M-step: Maximize lower bound

- Lower bound:

$$\ell(\theta; \mathcal{D}) \geq \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta) + \text{const}$$

- Choose $\theta^{(t+1)}$ to maximize lower bound

$$\theta^{(t+1)} = \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta)$$

- Use expected counts. Intuition: Whenever we used $\text{Count}(x, z)$ in fully observable case, replace by $E_{Q^{(t+1)}}[\text{Count}(x, z)]$

M-step: Maximize lower bound

- Lower bound holds for any choice of θ :

$$\ell(\theta; \mathcal{D}) \geq \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta) + \text{const}$$

- What is special about

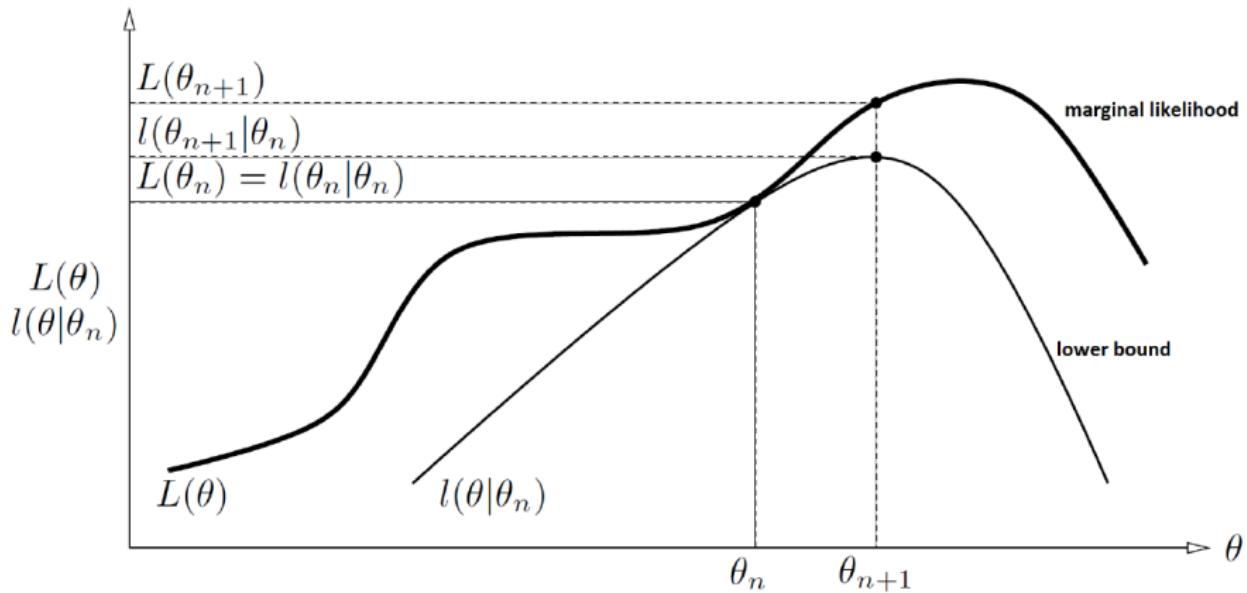
$$Q^{(t+1)}(\mathbf{z}|\mathbf{x}^{(j)}) = P(\mathbf{z}|\mathbf{x}^{(j)}, \theta^{(t)})$$

- The bound is tight at $\theta = \theta^{(t)}$, i.e.

$$\ell(\theta^{(t)}; \mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta^{(t)}) + \text{const}$$

- Any increase in this lower bound during the M-step therefore must also **increase the marginal loglikelihood** (what we actually care about)!

Derivation of EM algorithm



(Figure adapted from tutorial by Sean Borman)

Why is the bound tight

- We derived this lower bound that holds for any choice of $Q^{(t+1)}(\mathbf{z}|\mathbf{x})$:

$$\ell(\theta; \mathcal{D}) \geq \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{z}; \theta)}{Q^{(t+1)}(\mathbf{z}|\mathbf{x})}$$

- If $Q^{(t+1)}(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x}; \theta)$ the bound becomes:

$$\begin{aligned} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}; \theta) \log \frac{p(\mathbf{x}, \mathbf{z}; \theta)}{p(\mathbf{z}|\mathbf{x}; \theta)} &= \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}; \theta) \log \frac{p(\mathbf{z}|\mathbf{x}; \theta)p(\mathbf{x}; \theta)}{p(\mathbf{z}|\mathbf{x}; \theta)} \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}; \theta) \log p(\mathbf{x}; \theta) \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}; \theta) \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) = \ell(\theta; \mathcal{D}) \end{aligned}$$

Expectation maximization

Algorithm is as follows:

- ① Initialize θ_0 , e.g. at random or using a good first guess
- ② Repeat until convergence:

$$\theta_{t+1} = \arg \max_{\theta} \sum_{m=1}^M E_{p(\mathbf{z}_m | \mathbf{x}_m; \theta_t)} [\log p(\mathbf{x}_m, \mathbf{Z}; \theta)]$$

- “E” step corresponds to computing the objective (i.e., the **expectations**)
- “M” step corresponds to **maximizing** the objective

EM: Properties

- EM is indeed searching for estimates that maximize the expected log-likelihood function, which also explains its name.
- Parameters that maximize the expected log-likelihood function cannot decrease the log-likelihood function.
 - Each iteration of EM can only increase the likelihood and never decrease it.
 - It will converge to a local maxima.
- EM may converge to different parameters, with different likelihoods, depending on the initial estimates $\theta^{(0)}$ that it starts with.
- Each iteration of the EM algorithm will have to perform inference on a Bayesian network.
- In each iteration, the algorithm computes the probability of each instantiation (x, \mathbf{u}) given each example as evidence.

M-Step for Bayes Nets

$$\theta^{(t+1)} = \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{\mathbf{z}} Q^{(t+1)}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}; \theta)$$

- Can optimize each CPT independently!
- Data set is now **bigger** (because of $\sum_{\mathbf{z}}$) and **weighted** (because of $Q^{(t+1)}(\mathbf{z}|\mathbf{x})$)
- MLE in fully observed case:

$$\theta_{x_i | \mathbf{x}_{pa(i)}}^{ML} = \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{\sum_{\hat{x}_i} N_{\hat{x}_i, \mathbf{x}_{pa(i)}}}$$

- MLE with hidden variables:

$$\theta_{x_i | \mathbf{x}_{pa(i)}}^{(t+1)} = \frac{E_{Q^{(t+1)}}[N_{x_i, \mathbf{x}_{pa(i)}}]}{E_{Q^{(t+1)}}[\sum_{\hat{x}_i} N_{\hat{x}_i, \mathbf{x}_{pa(i)}}]}$$

Let's see an application to Gaussian Mixture Models.

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 15, February 28, 2017

Today: Bayesian Learning

Bayesian Learning:

- ① integrate prior knowledge into the learning process
- ② reduce learning to an inference problem
- ③ model uncertainty/confidence on the estimates

Bayesian Learning motivation

- ① X Bernoulli variable, unknown parameter $\theta = P(X = H)$
- ② Which one is “better”:
 - ① Observe 1 H and 2 T
 - ② Observe 10 H and 20 T
 - ③ Observe 100 H and 200 T
- ④ What is the Maximum likelihood estimate for θ in these 3 cases?
- ⑤ Maximum Likelihood Estimate is same in all three cases. $\hat{\theta} = 1/3$
- ⑥ However, should be much more “confident” about MLE if we have more data
- ⑦ Want to model **distributions over parameters** $P(\theta|\mathcal{D})$

Black swan events



...



- ① Swans can be either white or black. Suppose we have seen $N = 10$ white swans. What is the probability $p(X_{N+1} = \text{black})$ that swan X_{N+1} is black?
- ② If we plug in the Max Likelihood Estimation, since $N_{\text{black}} = 0$, $N_{\text{white}} = 10$ and

$$\hat{\theta} = \frac{N_{\text{black}}}{N_{\text{black}} + N_{\text{white}}} = 0 \quad , \quad p(X_{N+1} = \text{black} | \hat{\theta}) = 0$$

- ③ We predict black swans are impossible!
- ④ However, this may just be due to insufficient data. Want to capture uncertainty, i.e. $P(\theta | \mathcal{D})$

The Beta-Bernoulli Model

- ① Consider the probability of heads, given a sequence of N coin tosses,
 $\mathcal{D} = \{X_1, \dots, X_N\}$. $X_j \in \{0, 1\}$
- ② Likelihood

$$P(\mathcal{D}|\theta) = \prod_{j=1}^N \theta^{X_j} (1-\theta)^{1-X_j} = \theta^{N_H} (1-\theta)^{N_T}$$

- where N_H, N_T are the numbers of heads and tails in \mathcal{D} respectively.
- ③ Suppose we have some prior knowledge on the parameter $P(\theta)$, e.g.,
a priori, heads and tails are equally likely
 - ④ Use Bayes Rule to update our belief

$$P(\theta|\mathcal{D}) = \frac{P(\theta)P(\mathcal{D}|\theta)}{P(\mathcal{D})} \propto P(\theta)P(\mathcal{D}|\theta)$$

- ⑤ How to represent the prior $P(\theta)$?
 - ① Want computationally “simple” (and still flexible) prior

The Beta-Bernoulli Model

- ① Consider the probability of heads, given a sequence of N coin tosses,
 $\mathcal{D} = \{X_1, \dots, X_N\}$
- ② Likelihood

$$P(\mathcal{D}|\theta) = \prod_{j=1}^N \theta^{X_j} (1-\theta)^{1-X_j} = \theta^{N_H} (1-\theta)^{N_T}$$

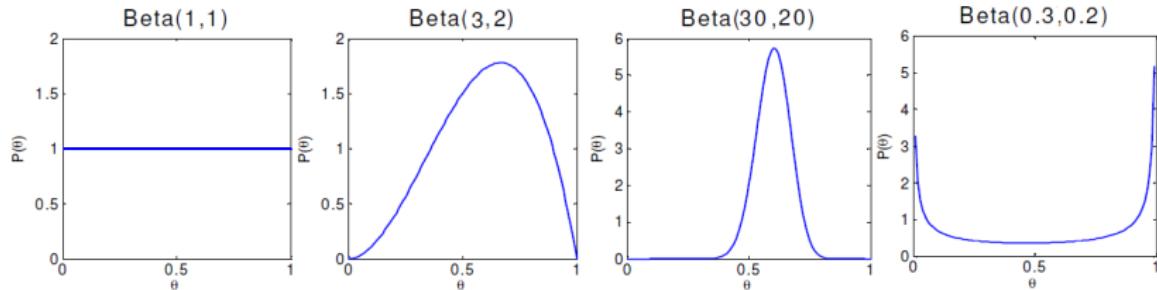
- ③ Natural conjugate prior is the **Beta distribution**, with PDF

$$P(\theta|\alpha_H, \alpha_T) = \text{Beta}(\theta|\alpha_H, \alpha_T) = \frac{1}{B(\alpha_H, \alpha_T)} \theta^{\alpha_H-1} (1-\theta)^{\alpha_T-1}$$

for $\theta \in [0, 1]$. The normalization constant $B(\cdot)$ is the beta function. It guarantees that $\int_0^1 P(\theta|\alpha_H, \alpha_T) d\theta = 1$.

- ④ $\alpha = (\alpha_H, \alpha_T) > 0$ are called “hyperparameters” of the prior

The Beta distribution



- ① The expected value is $E[\theta] = \int_0^1 \theta P(\theta | \alpha_H, \alpha_T) d\theta = \frac{\alpha_H}{\alpha_H + \alpha_T}$
- ② The sum of these exponents, $\alpha_H + \alpha_T$, is interpreted as a **measure of confidence** in the expectations they lead to
- ③ Think of the exponent α_H as the number of heads we have observed before the current data set ("virtual data")
- ④ The hyperparameters (3, 2) and (30, 20) can both be used to encode the belief that heads occurs 60% of the time, yet the second set of exponents imply a stronger belief as they are based on a larger "virtual sample".

The Beta-Bernoulli Model

- Consider the probability of heads, given a sequence of N coin tosses,
 $\mathcal{D} = \{X_1, \dots, X_N\}$
- Likelihood

$$P(\mathcal{D}|\theta) = \prod_{j=1}^N \theta^{X_j} (1-\theta)^{1-X_j} = \theta^{N_H} (1-\theta)^{N_T}$$

- Natural conjugate prior is the Beta distribution

$$P(\theta|\alpha_H, \alpha_T) = \text{Beta}(\theta|\alpha_H, \alpha_T) = \frac{1}{B(\alpha_H, \alpha_T)} \theta^{\alpha_H-1} (1-\theta)^{\alpha_T-1}$$

- **The posterior is also Beta** (conjugacy), with updated counts

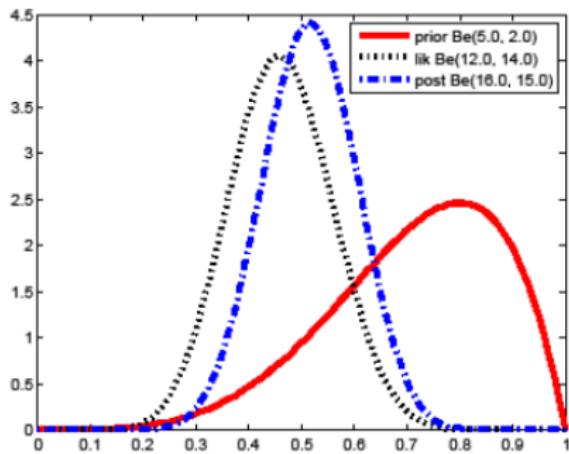
$$\begin{aligned} P(\theta|\mathcal{D}, \alpha_H, \alpha_T) &\propto P(\theta|\alpha_H, \alpha_T)P(\mathcal{D}|\theta) \propto \theta^{\alpha_H+N_H-1} (1-\theta)^{\alpha_T+N_T-1} \\ &= \text{Beta}(\theta|\alpha_H + N_H, \alpha_T + N_T) \end{aligned}$$

- **Conjugacy:** the posterior distribution $P(\theta|\mathcal{D})$ is in the same family as the prior $P(\theta)$. Depends on the likelihood!
- Prior family is closed under Bayesian updating

The Beta-Bernoulli Model

- The posterior is also Beta (conjugacy), with updated counts

$$\begin{aligned} P(\theta|\mathcal{D}, \alpha_H, \alpha_T) &\propto P(\mathcal{D}|\theta)P(\theta|\alpha_H, \alpha_T) \propto \theta^{N_H}(1-\theta)^{N_T}\theta^{\alpha_H-1}(1-\theta)^{\alpha_T-1} \\ &\propto \theta^{(N_H+1)-1}(1-\theta)^{(N_T+1)-1}\theta^{\alpha_H-1}(1-\theta)^{\alpha_T-1} \\ &\propto \text{Beta}(\theta|N_H + 1, N_T + 1)\text{Beta}(\theta|\alpha_H, \alpha_T) \\ &\propto \text{Beta}(\theta|\alpha_H + N_H, \alpha_T + N_T) \end{aligned}$$



Posterior Mean

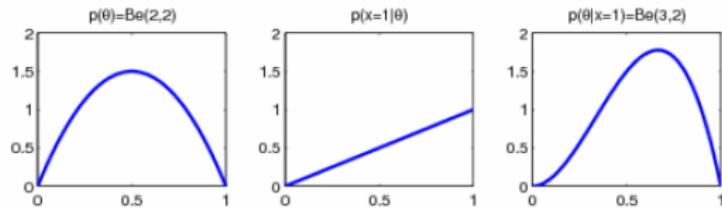
- We assume prior $P(\theta|\alpha_H, \alpha_T) = \text{Beta}(\theta|\alpha_H, \alpha_T)$. What is the **prior** mean of θ ? $\alpha_H/(\alpha_H + \alpha_T)$
- We've just seen that $P(\theta|\mathcal{D}, \alpha_H, \alpha_T) = \text{Beta}(\theta|\alpha_H + N_H, \alpha_T + N_T)$. What is the **posterior** mean of $\theta|\mathcal{D}$? $(\alpha_H + N_H)/(\alpha_H + N_H + \alpha_T + N_T)$
- Let $N = N_H + N_T$ be the number of data points, and $M = \alpha_H + \alpha_T$ be the "strength" of the prior (amount of "virtual data")
- The posterior mean is a convex combination of prior mean α_H/M and max-likelihood mean N_H/N

$$\begin{aligned} E[\theta|\mathcal{D}] &= \frac{\alpha_H + N_H}{\alpha_H + N_H + \alpha_T + N_T} = \frac{\alpha_H + N_H}{N + M} \\ &= \underbrace{\frac{M}{N + M}}_{\text{prior mean}} \underbrace{\frac{\alpha_H}{M}}_{\text{MLE mean}} + \underbrace{\frac{N}{N + M}}_{\text{MLE mean}} \underbrace{\frac{N_H}{N}}_{\text{MLE mean}} \end{aligned}$$

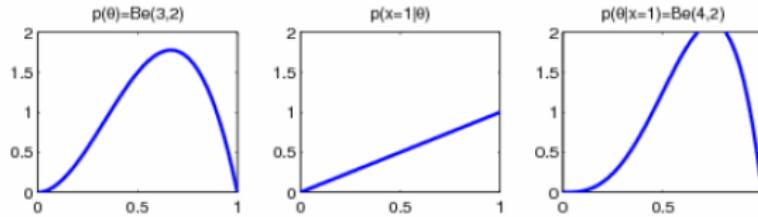
- We shrink our estimate away from the MLE towards the prior (a form of smoothing or regularization)
- The more data we have ($N \rightarrow \infty$), the more confident we become

Posterior Mean

- Prior is $Beta(2, 2)$. We observe one head event. What is the posterior?
Posterior is $Beta(3, 2)$. Mean shifts from $2/4$ to $3/5$



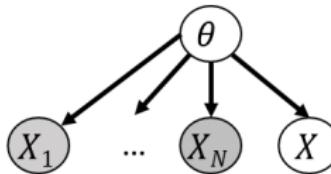
- Prior is $Beta(3, 2)$. We observe one head event. Posterior is $Beta(4, 2)$.
Mean shifts from $3/5$ to $4/6$



- Can recursively update our belief as new data streams in (online learning)

Bayesian Inference

- What is the probability that the next flip is heads in a model with parameter θ^* ? $P(X = H|\theta^*) = \theta^*$. $P(X = T|\theta^*) = 1 - \theta^*$
- Bayesian prediction: what is the probability that the next flip is heads, given that we observed \mathcal{D} ? Don't commit to a particular parameter value θ^* , e.g. the ML estimate, to make predictions



- Instead, we integrate out our uncertainty about θ when predicting the future (hedge our bets)

$$p(X = H|\mathcal{D}) = \int_0^1 P(X = H, \theta|\mathcal{D}) d\theta = \int_0^1 P(X = H|\theta)P(\theta|\mathcal{D}) d\theta$$

- Average the predictions made by an ensemble of models, e.g., in the discrete case: $P(X = H|\theta_1)\frac{1}{2} + P(X = H|\theta_2)\frac{1}{4} + P(X = H|\theta_3)\frac{1}{4}$

Bayesian Inference in Beta-Bernoulli model

- In the Beta-Bernoulli model, the Bayesian predictive distribution is

$$p(X = H|\mathcal{D}) = \int_0^1 P(X = H|\theta)P(\theta|\mathcal{D})d\theta$$

where

- $P(X = H|\theta) = \theta$
 - $P(\theta|\mathcal{D}, \alpha_H, \alpha_T) = \text{Beta}(\theta|\alpha_H + N_H, \alpha_T + N_T)$
 - This is just the posterior mean
- $$p(X = H|\mathcal{D}) = \int_0^1 P(X = H|\theta)P(\theta|\mathcal{D})d\theta = E[\theta] = \frac{\alpha_H + N_H}{\alpha_H + N_H + \alpha_T + N_T}$$
- It's the expected value of the parameter θ (success probability) with respect to an ensemble where the weights are given by the posterior $P(\theta|\mathcal{D})$

Solution to black swan event

- ① If we use a $\text{Beta}(1,1)$ prior for θ , the posterior predictive is

$$P(X_{N+1} = \text{black} | N_{\text{white}}, N_{\text{black}}) = \frac{N_{\text{black}} + 1}{N_{\text{white}} + N_{\text{black}} + 2}$$

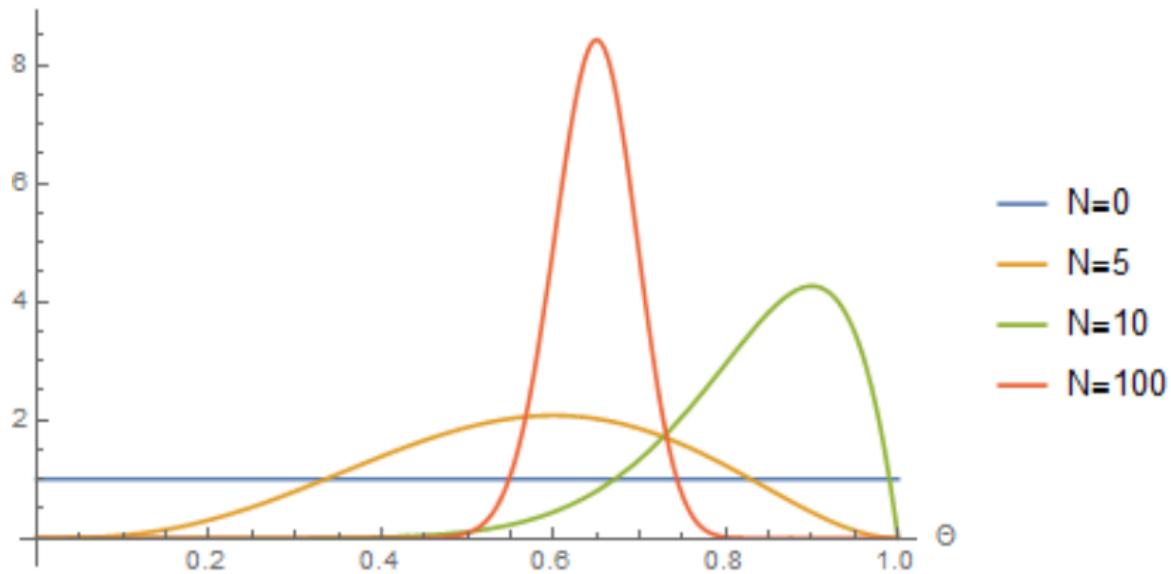
so we will never predict black swans are impossible, even if $N_{\text{black}} = 0$!

- ② However, as we see more and more white swans, we will come to believe that black swans are pretty rare
- ③ We have seen this before: called additive or **Laplace smoothing**
- ④ As $|\mathcal{D}| \rightarrow \infty$, the posterior becomes peaked $P(\theta | \mathcal{D}) \rightarrow \delta_{\theta - \theta^{\text{MLE}}}$ and we get

$$\begin{aligned} p(X = H | \mathcal{D}) &= \int_0^1 P(X = H | \theta) P(\theta | \mathcal{D}) d\theta \approx \int_0^1 P(X = H | \theta) \delta_{\theta - \theta^{\text{MLE}}} d\theta \\ &= P(X = H | \theta^{\text{MLE}}) \end{aligned}$$

MLE convergence

Posterior distribution, using samples from $X \sim \text{Bernoulli}(0.7)$



MAP Estimation

- It is often easier to compute the posterior mode (optimization) than the posterior mean $E[\theta|\mathcal{D}]$ (integration).

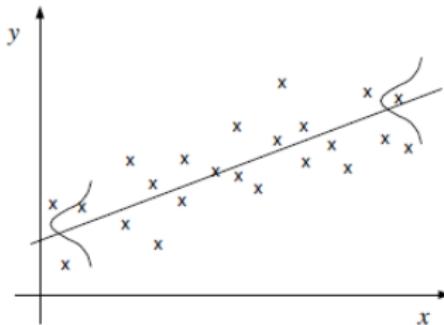
$$\hat{\theta} = \arg \max_{\theta} P(\theta|\mathcal{D}) = \arg \max_{\theta} P(\mathcal{D}|\theta)P(\theta)$$

- This is called **maximum a posteriori estimation**
- Can be interpreted as **regularized** maximum likelihood estimation

$$\hat{\theta} = \arg \max_{\theta} \log P(\theta|\mathcal{D}) = \arg \max_{\theta} \underbrace{\log P(\mathcal{D}|\theta)}_{\text{log-likelihood}} + \underbrace{\log P(\theta)}_{\text{regularizer}}$$

Example: linear regression

- We have some training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$



- Model: $P(Y|\mathbf{X} = \mathbf{x}, \theta)$ is Gaussian, with mean $\theta\mathbf{x}$ and variance 1

Recap: MLE for linear regression

- Model: $P(Y|\mathbf{X} = \mathbf{x}, \theta)$ is Gaussian, with mean $\theta\mathbf{x}$ and variance 1
- The (conditional) likelihood of the data \mathcal{D} is

$$\begin{aligned} P(y_1, \dots, y_m | x_1, \dots, x_m, \theta) &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y_i - \theta\mathbf{x}_i)^2\right) \\ &= \frac{1}{(\sqrt{2\pi})^m} \exp\left(-\frac{1}{2} \sum_{i=1}^m (y_i - \theta\mathbf{x}_i)^2\right) \end{aligned}$$

$$\log P(y_1, \dots, y_m | x_1, \dots, x_m, \theta) = \text{const} - \frac{1}{2} \sum_{i=1}^m (y_i - \theta\mathbf{x}_i)^2$$

- The MLE estimate of the parameter is

$$\theta^* = \arg \min J(\theta) = \frac{1}{2} \sum_{i=1}^m (y_i - \theta\mathbf{x}_i)^2$$

- Maximizing likelihood is equivalent to minimizing least square cost

MAP estimation for linear regression

- Model: $P(Y|\mathbf{X} = \mathbf{x}, \theta)$ is Gaussian, with mean $\theta\mathbf{x}$ and variance 1
- Let's assume a Gaussian **prior** distribution over the weight vector $\theta \in \mathbb{R}^D$ with mean 0 and variance $\lambda^{-1}\mathbf{I}$

$$P(\theta|\lambda) \propto \exp\left(-\frac{\lambda}{2}\theta^T\theta\right)$$

- Maximum a posteriori estimation

$$\hat{\theta}^{MAP} = \arg \max_{\theta} \log P(\theta|\mathcal{D}, \lambda) = \arg \max_{\theta} \underbrace{\log P(\mathcal{D}|\theta)}_{\text{log-likelihood}} + \underbrace{\log P(\theta|\lambda)}_{\text{regularizer}}$$

- The MAP estimate of the weight vector is

$$\hat{\theta}^{MAP} = \arg \max_{\theta} -\frac{1}{2} \sum_{i=1}^m (y_i - \theta^T \mathbf{x}_i)^2 - \frac{\lambda}{2} \theta^T \theta = \arg \min_{\theta} \sum_{i=1}^m (y_i - \theta^T \mathbf{x}_i)^2 + \lambda \theta^T \theta$$

- It's equivalent to the regularized least-squares objective (**ridge regression**). Prior biases solution to "small" θ

Bayesian Learning

What about categorical distributions?

The Dirichlet-Categorical Model

- Consider a sequence of N outcomes of a K -sided dice, $\mathcal{D} = \{X_1, \dots, X_N\}$, $X_j \in \{1, \dots, K\}$. Denote $P[X_j = k] = p_k$, and $\theta = (p_1, \dots, p_K)$
- Likelihood

$$P(\mathcal{D}|\theta) = \prod_{j=1}^N \prod_{k=1}^K p_k^{1[X_j=k]} = \prod_{k=1}^K p_k^{\sum_{j=1}^N 1[X_j=k]}$$

- Natural conjugate prior is the **Dirichlet** distribution

$$P(\theta|\alpha) = Dirichlet(p_1, \dots, p_K|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^K p_k^{\alpha_k - 1}$$

Support is a simplex: p_1, \dots, p_K such that $p_k > 0$ and $\sum_k p_k = 1$

- **The posterior is also Dirichlet** (conjugacy), with updated counts

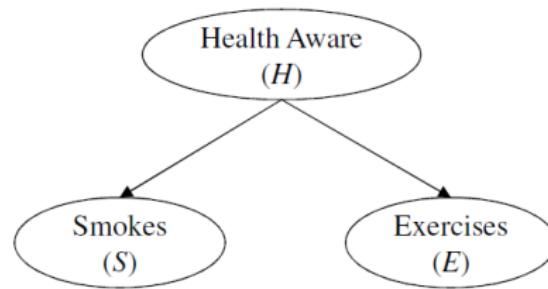
$$\begin{aligned} P(\theta|\mathcal{D}, \alpha) &\propto P(\theta|\alpha)P(\mathcal{D}|\theta) \propto \prod_{k=1}^K p_k^{\sum_{j=1}^N 1[X_j=k]+\alpha_k-1} \\ &= Dirichlet(\theta|\alpha') \quad , \quad \alpha'_k = \sum_j 1[X_j=k] + \alpha_k \end{aligned}$$

Bayesian Learning

What about more general Bayes Nets?

Bayesian Learning

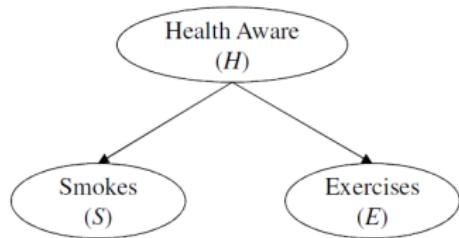
Suppose we have the following Bayes Net structure (variables are binary), and a training dataset \mathcal{D} (fully observed).



Case	H	S	E
1	F	F	T
2	T	F	T
3	T	F	T
4	F	F	F
5	F	T	F

Bayesian Learning

There are 5 sets of parameters to be estimated.



Five parameter sets

$$\theta_H = (\theta_h, \theta_{\bar{h}})$$

$$\theta_{S|h} = (\theta_s|h, \theta_{\bar{s}}|h)$$

$$\theta_{S|\bar{h}} = (\theta_s|\bar{h}, \theta_{\bar{s}}|\bar{h})$$

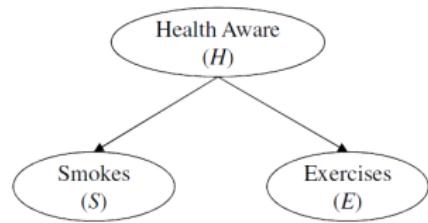
$$\theta_{E|h} = (\theta_e|h, \theta_{\bar{e}}|h)$$

$$\theta_{E|\bar{h}} = (\theta_e|\bar{h}, \theta_{\bar{e}}|\bar{h})$$

How many degrees of freedom for the parameters? 5 independent parameters

Bayesian Learning

Suppose we have some prior knowledge on the possible values of the parameters:



Prior knowledge

$$\theta_{S|h} = (.1, .9)$$

$$\theta_{E|h} = (.8, .2)$$

$$\theta_H \in \{(.75, .25), (.90, .10)\}$$

$$\theta_{S|\bar{h}} \in \{(.25, .75), (.50, .50)\}$$

$$\theta_{E|\bar{h}} \in \{(.50, .50), (.75, .25)\}$$

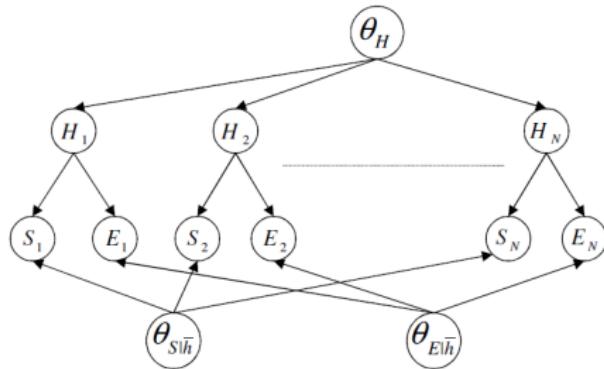
where each of the two values are considered equally likely.

For example, we assume to know the the conditional probabilities when $H = \text{True}$.

Bayesian Learning

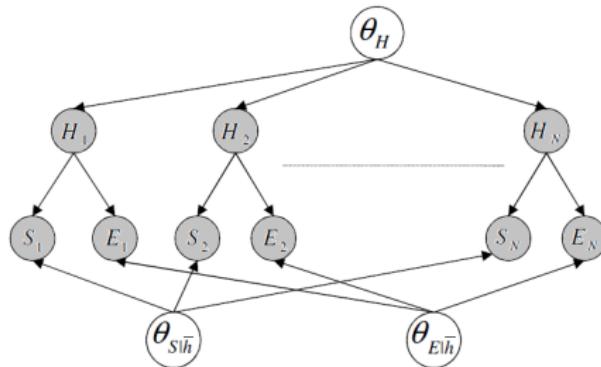
How to incorporate prior knowledge into parameter learning for BN?

Meta networks



- **Meta network:** random variables $\theta_H, \theta_{S|\bar{h}}, \theta_{E|\bar{h}}$ represent the unknown network parameters. The CPTs of these variables encode our prior knowledge about these parameters
- One copy of the original network per training data point. Variables H_i, S_i and E_i represent the values that variables H, S and E take in case i of the data set \mathcal{D} .

Meta networks

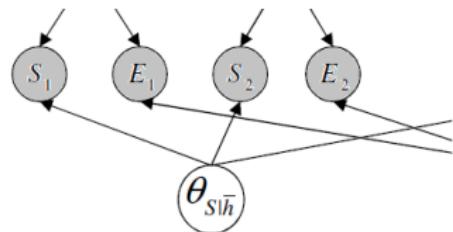


- **Meta network:** random variables θ_H , $\theta_{S|\bar{h}}$, $\theta_{E|\bar{h}}$ represent the unknown network parameters. The CPTs of these variables encode our prior knowledge about these parameters
- One copy of the original network (**base network**) per training data point. Variables H_i , S_i and E_i represent the values that variables H , S and E take in case i of the data set \mathcal{D} .
- Idea: we can then condition on the data by setting the variables as evidence and do inference on θ_H , $\theta_{S|\bar{h}}$, $\theta_{E|\bar{h}}$

Prior Knowledge

- Prior knowledge on network parameters is encoded in the meta network using the CPTs of parameter sets.
- For example, we assumed that the two values of parameter set $\theta_{S|\bar{h}}$ are equally likely. Hence, the CPT is:

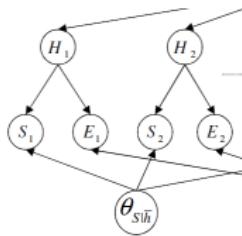
$\theta_{S \bar{h}} = (\theta_{s \bar{h}}, \theta_{\bar{s} \bar{h}})$	$\mathbb{P}(\theta_{S \bar{h}})$
(.25, .75)	50%
(.50, .50)	50%



These CPTs are then given as input to the learning process (they represent our prior knowledge about the domain)

Semantics of the meta network

- Intuition: conditioned on a realization of the parameters, the model is the one specified by the *base network* using that particular realization of the parameters
- Example: variable S_1 has parents $H_1, \theta_{S|\bar{h}}$. We need to specify $P(S_1|H_1, \theta_{S|\bar{h}})$

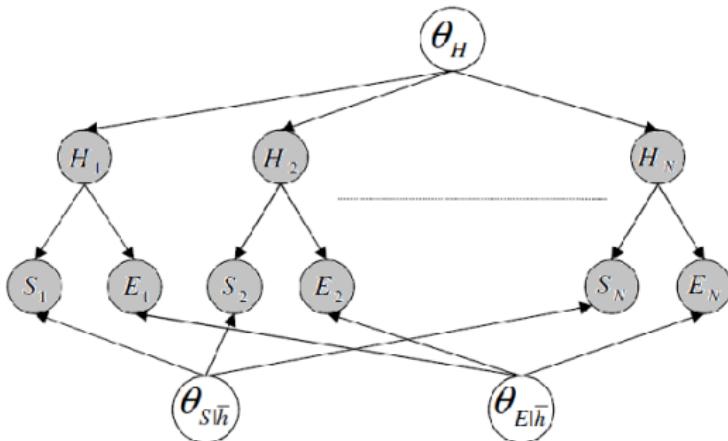


Given $\theta_{S|\bar{h}}$, we know what conditional probability to use

$$P(S_i|H_i = \bar{h}, \theta_{S|\bar{h}}) = \theta_{S|\bar{h}}$$

Data as Evidence

Case	H	S	E
1	h	\bar{s}	e
2	h	\bar{s}	\bar{e}
3	\bar{h}	s	\bar{e}



The data set \mathcal{D} can be viewed as the following variable instantiation:

$$(H_1 = h) \wedge (S_1 = \bar{s}) \wedge (E_1 = e) \wedge \dots \wedge (H_3 = \bar{h}) \wedge (S_3 = s) \wedge (E_3 = \bar{e})$$

One can assert this data set as evidence on the meta network, and then compute the corresponding posterior distribution on network parameters!

Bayesian Learning

- Bayes Rule now gives us a sound way to update our beliefs about the parameters (prior) based on the data
- By explicitly encoding prior knowledge about network parameters, and by treating data as evidence, can **reduce learning to a process of computing posterior distributions**:

$$P(\theta_H, \theta_{S|\bar{h}}, \theta_{E|\bar{h}} | \mathcal{D})$$

where P is the distribution induced by the **meta network**, and \mathcal{D} is the evidence entailed by the data set

- We can now measure uncertainty on the parameter estimates. Not just a point estimate but a probability distribution (recall the black swan example)

Maximum Likelihood approach

- Suppose now that our goal is to compute the probability of observing a smoker who exercises regularly, i.e. $P(S = s, E = e)$
- According to the ML approach, we first need to find the ML estimates θ^{ML} based on the given data, and then use them to compute this probability.
- Among the eight possible parameterizations in this case, the one θ^{ml} with maximum likelihood is:

$$\theta_H = (0.75, 0.25); \theta_{S|\bar{h}} = (0.25, 0.75); \theta_{E|\bar{h}} = (0.50, 0.50)$$

- If we plug in these parameter values in the base network:

$$Pr_{\theta^{ml}}(S = s, E = e) \approx 9.13\%$$

Bayesian Inference

- The Bayesian approach does not commit to a single parameter vector θ and works with a distribution over the possible values of these parameters, $P(\theta|\mathcal{D})$.
- For example, the expected probability of observing a person that both smokes $S = s$ and exercises $E = e$ can be computed as follows:

$$P(S = s, E = e) = \sum_{\theta} Pr_{\theta}(s, e) P(\theta|\mathcal{D}) \approx 11.06\%$$

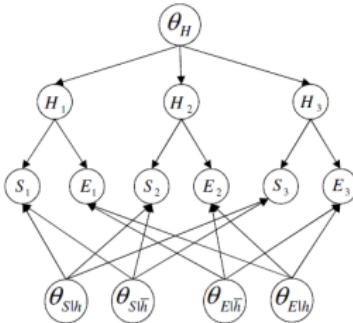
where $Pr_{\theta}(\cdot)$ is the distribution induced by the base network and parametrization θ .

- We consider every possible parametrization θ , compute the probability $Pr_{\theta}(S = s, E = e)$ using the base network, and then take a weighted average of the computed probabilities.
- More robust than MLE (think about the black swan example)

Continuous parameters

- So far we assumed the random variables representing parameters take discrete values. E.g., $\theta_{S|\bar{h}}$ can take only two values (0.25, 0.75) and (0.5, 0.5) (our prior knowledge precluded all other values)
- Not very realistic
- Can allow all possible values for these parameters. Then the parameter sets will be continuous (i.e., having an infinite number of values)
- To apply Bayesian learning in this context, we need suitable prior on continuous parameter sets
- Solution: use Beta prior seen before for distributions over binary variables (and **dirichlet prior** for multinomial)

Beta prior example



- Example: prior on $\theta_h = \Pr[H_i = \text{True}]$. The priors $\theta_h = \text{Beta}(0.75, 0.25)$ and $\theta_h = \text{Beta}(75, 25)$ can both be used to encode the belief that 75% of the individuals are health-aware. A $\text{Beta}(75, 25)$ prior implies a stronger belief (larger virtual sample)

Summary

- Bayesian methods are conceptually simple and elegant, and can handle small sample sizes and complex hierarchical models without overfitting.
- They provide a single mechanism for answering all questions of interest; there is no need to choose between different estimators, etc.
- Issues. Why isn't everybody a Bayesian?
 - Computational issues (requires integrating over all the parameters/structures)
 - Bayes rule requires a prior, which is considered “subjective”.
 - Often we actually have informative prior knowledge. If not, it is possible to create relatively “uninformative” priors to represent prior ignorance.

Bayesian Learning

A Bayesian is one who, vaguely expecting a horse, and catching a glimpse of a donkey, strongly believes he has seen a mule.

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 16, March 2, 2017

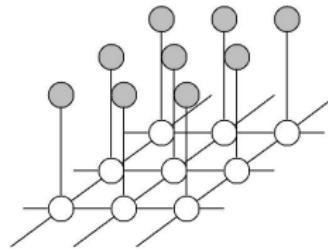
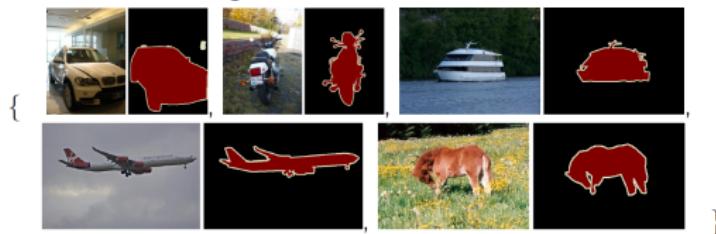
Today: structure learning

Goals for the lecture. You should understand the following concepts

- ① The Chow-Liu algorithm for structure search
- ② Structure learning as search
- ③ Overfitting and model complexity

Learning network structure

- So far: given 1) data and 2) network structure



Goal: find parameters maximize the probability of observing the data

- What if network structure is unknown?

Knowledge discovery

- Network structure gives us a lot of information, e.g. conditional independencies, causal relationships, etc.
- Discover something about p^* , the process that generated the data, e.g.
 - Nature of the dependencies, e.g., positive or negative correlation
 - What are the direct and indirect dependencies
- How to do it? Two main approaches:
 - Constraint-based: test independencies, and add edges accordingly
 - Score-based: search for **network structures** that maximize the probability of observing the given data set

Independence maps

From Lecture 3:

- Let $I(G)$ be the set of all conditional independencies implied by the directed acyclic graph (DAG) G
- Let $I(p)$ denote the set of all conditional independencies that hold for a joint distribution p .
- A DAG G is an **I-map** (independence map) of a distribution p if $I(G) \subseteq I(p)$
 - A fully connected DAG G is an I-map for *any* distribution, since $I(G) = \emptyset \subseteq I(p)$ for all p
- G is a **minimal I-map** for p if it is an I-map and the removal of even a single edge makes it not an I-map

Algorithm for finding minimal I-maps

- Given random variables and known conditional independencies
- Pick ordering X_1, \dots, X_n of the variables
- Idea: use chain rule, then simplify each term as much as possible
- For each X_i
 - Find minimal subset $\mathbf{A}_i \subseteq \{X_1, \dots, X_{i-1}\}$ such that
$$p(X_i | X_1, \dots, X_{i-1}) = p(X_i | \mathbf{A}_i)$$
 - Specify or learn CPD $p(X_i | \mathbf{A}_i)$
- Will produce minimal I-map G !
- Proof sketch: Each variable is independent from its non-descendants given its parents in G , hence p factorizes over G . Factorization implies that G is a valid I-map for p . By construction, it is also **minimal**.
- Not unique (order dependent)
- Why is this important? Same algorithm can be used to **learn BN structure directly from data!**

Algorithm for finding minimal I-maps

- Pick ordering X_1, \dots, X_n of the variables
- For each X_i
 - Find minimal subset $\mathbf{A} \subseteq \{X_1, \dots, X_{i-1}\}$ such that $p(X_i | X_1, \dots, X_{i-1}) = p(X_i | \mathbf{A})$
 - Specify or learn CPD $p(X_i | \mathbf{A})$
- How to check for (conditional) independence if we only have access to data \mathcal{D} ? Use conditional independence test (hypothesis testing)
- If X is independent from Y , then $P(X, Y) = P(X)P(Y)$, and $\widehat{P}(X, Y) \approx \widehat{P}(X)\widehat{P}(Y)$ where \widehat{P} is the empirical distribution in \mathcal{D} . Define

$$d_{\chi^2}(\mathcal{D}) = |\mathcal{D}| \sum_{x,y} \frac{(\widehat{P}(x,y) - \widehat{P}(x)\widehat{P}(y))^2}{\widehat{P}(x)\widehat{P}(y)}$$

if $d_{\chi^2}(\mathcal{D})$ is “small”, conclude that X, Y are independent

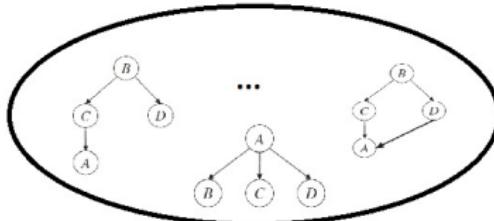
Constraint-based structure learning

- Cons:
 - Independence tests are less reliable when $|\mathcal{D}|$ is small
 - Tests are less reliable when they involve many variables
 - We need to do multiple tests (multiple hypothesis testing)
- Pros:
 - Works well if we impose additional constraints on the maximum number of parents (sparse Bayes nets) and have lots of data

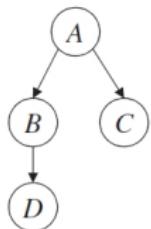
Score based structure learning

Data \mathcal{D} and set of all possible Bayes Nets over the variables A, B, C, D :

\mathcal{D}	A	B	C	D
d_1	a_1	b_1	c_2	d_1
d_2	a_1	b_1	c_2	d_2
d_3	a_1	b_2	c_1	d_1
d_4	a_2	b_1	c_1	d_2
d_5	a_1	b_1	c_2	d_2



Optimal parameters for this network can be learned using MLE



A	θ_a^{ml}
a_1	$4/5$
a_2	$1/5$

A	B	$\theta_{b a}^{ml}$
a_1	b_1	$3/4$
a_1	b_2	$1/4$

A	C	$\theta_{c a}^{ml}$
a_1	c_1	$1/4$
a_1	c_2	$3/4$

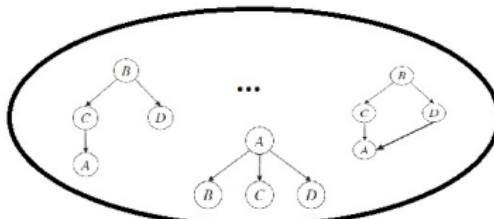
B	D	$\theta_{d b}^{ml}$
b_1	d_1	$1/4$
b_1	d_2	$3/4$

Log-likelihood of data given the structure ≈ -13.3

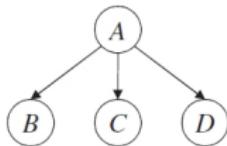
Structure learning

Data \mathcal{D} and set of all possible Bayes Nets over the variables A, B, C, D :

\mathcal{D}	A	B	C	D
d_1	a_1	b_1	c_2	d_1
d_2	a_1	b_1	c_2	d_2
d_3	a_1	b_2	c_1	d_1
d_4	a_2	b_1	c_1	d_2
d_5	a_1	b_1	c_2	d_2



Optimal parameters for another network



A	θ_a^{ml}	A	B	$\theta_{b a}^{ml}$	A	C	$\theta_{c a}^{ml}$	A	D	$\theta_{d a}^{ml}$
a_1	$4/5$	a_1	b_1	$3/4$	a_1	c_1	$1/4$	a_1	d_1	$1/2$
a_1	$4/5$	a_1	b_2	$1/4$	a_1	c_2	$3/4$	a_1	d_2	$1/2$
a_2	$1/5$	a_2	b_1	1	a_2	c_1	1	a_2	d_1	0
a_2	$1/5$	a_2	b_2	0	a_2	c_2	0	a_2	d_2	1

Log-likelihood of data given the structure ≈ -14.1

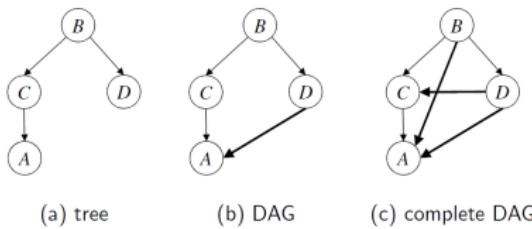
Which one should we prefer?

Learning general DAG structures

- Want to find a graph G that maximizes

$$\arg \max_G LL(\mathcal{D}|G)$$

- Problem: a complete DAG is provably the best structure.



- Theorem:** If a DAG G^* is the result of adding edges to G , then $LL(\mathcal{D}|G^*) \geq LL(\mathcal{D}|G)$. Proof: Homework 3.

Structure learning: Tree Bayesian networks

Idea: find best **tree-structured Bayesian network** (at most one parent per node)

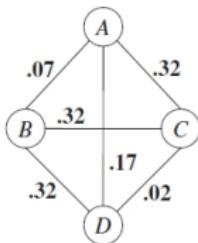
- Issue: too many candidate networks to consider
- **Chow-Liu algorithm:** find ML tree structures with complexity quadratic in the number of variables
- Based on the **mutual information** between two variables in the **empirical distribution**:

$$MI(X, U) = D_{KL}(p(x, u) || p(x)p(u)) = \sum_{x,u} p(x, u) \log \left[\frac{p(x, u)}{p(x)p(u)} \right]$$

- Measures the dependence between the variables. The higher the value, the higher the dependence.
 - What happens if X and U are independent? $MI(X, U) = 0$ if X and U are independent

Chow-Liu Algorithm Step 1

- 1) Compute mutual information $MI(X, U) = \sum_{x,u} \hat{p}(x, u) \log \left[\frac{\hat{p}(x, u)}{\hat{p}(x)\hat{p}(u)} \right]$ between all pairs of variables



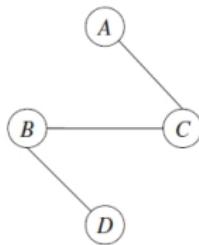
(a) Mutual information graph

Note: based on the empirical data distribution

$$\hat{p}(x_i, x_j) = \frac{\text{Count}(x_i, x_j)}{\# \text{ data points}}$$

Chow-Liu Algorithm Step 2

- 2) Find **maximum** weight spanning tree: a maximal-weight tree that connects all vertices in a graph

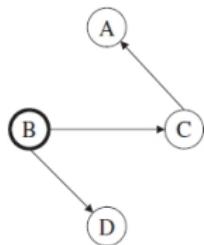


(b) Maximum spanning tree

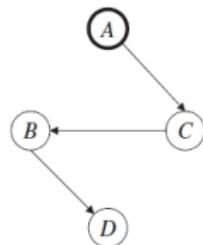
Can use Kruskal or Prim Algorithms. Greedily add edges, just make sure its a tree at every step.

Chow-Liu Algorithm Step 3

- 3) Pick a (any) node, assign directions (arrows going away from it)

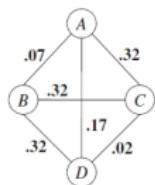


(c) Maximum likelihood tree

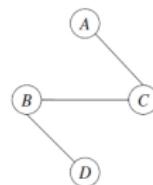


(d) Maximum likelihood tree

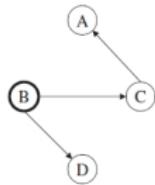
Chow-Liu Algorithm: Learning optimal tree BN



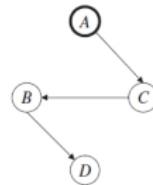
(a) Mutual information graph



(b) Maximum spanning tree



(c) Maximum likelihood tree



(d) Maximum likelihood tree

- What is the complexity? Complexity of computing MI pairs? $O(n^2)$
Maximum spanning tree computation? $O(n^2)$. Total complexity of Chow-Liu: $O(n^2)$
- Optimal log-likelihood ≈ -12.1 , higher than the two tree networks we tried before (as expected)

Why does Chow-Liu work?

Recall data loglikelihood given directed edges for Bayes Nets:

$$\log p(\mathcal{D} | \theta, G) = \sum_i \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} N_{x_i, \mathbf{x}_{pa(i)}} \log p(x_i | \mathbf{x}_{pa(i)}; \theta)$$

Closed form solution for ML parameters:

$$\theta_{x_i | \mathbf{x}_{pa(i)}}^{ML} = \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{\sum_{\hat{x}_i} N_{\hat{x}_i, \mathbf{x}_{pa(i)}}} = \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{N_{\mathbf{x}_{pa(i)}}} = \hat{p}(x_i | \mathbf{x}_{pa(i)})$$

Theorem: The likelihood score decomposes as follows

$$\begin{aligned} \log p(\mathcal{D} | \theta^{ML}, G) &= |\mathcal{D}| \sum_i MI_{\hat{p}}(X_i, \mathbf{X}_{pa(i)}) - |\mathcal{D}| \sum_i H_{\hat{p}}(X_i) \\ \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} N_{x_i, \mathbf{x}_{pa(i)}} \log \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{N_{\mathbf{x}_{pa(i)}}} &= |\mathcal{D}| \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{|\mathcal{D}|} \log \frac{N_{x_i, \mathbf{x}_{pa(i)}} / |\mathcal{D}|}{N_{\mathbf{x}_{pa(i)}} / |\mathcal{D}|} \\ &= |\mathcal{D}| \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} \hat{p}(x_i, \mathbf{x}_{pa(i)}) \log \frac{\hat{p}(x_i, \mathbf{x}_{pa(i)})}{\hat{p}(\mathbf{x}_{pa(i)})} = |\mathcal{D}| \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} \hat{p}(x_i, \mathbf{x}_{pa(i)}) \log \frac{\hat{p}(x_i, \mathbf{x}_{pa(i)}) \hat{p}(x_i)}{\hat{p}(\mathbf{x}_{pa(i)}) \hat{p}(x_i)} \\ &= |\mathcal{D}| \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} \hat{p}(x_i, \mathbf{x}_{pa(i)}) \log \frac{\hat{p}(x_i, \mathbf{x}_{pa(i)})}{\hat{p}(\mathbf{x}_{pa(i)}) \hat{p}(x_i)} + |\mathcal{D}| \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} \hat{p}(x_i, \mathbf{x}_{pa(i)}) \log \hat{p}(x_i) \\ &= |\mathcal{D}| MI_{\hat{p}}(X_i, \mathbf{X}_{pa(i)}) - |\mathcal{D}| H_{\hat{p}}(X_i) \end{aligned}$$

Why does Chow-Liu work?

Recall data loglikelihood given directed edges for Bayes Nets:

$$\log p(\mathcal{D} | \theta, G) = \sum_i \sum_{x_i} \sum_{\mathbf{x}_{pa(i)}} N_{x_i, \mathbf{x}_{pa(i)}} \log p(x_i | \mathbf{x}_{pa(i)}; \theta)$$

The likelihood score decomposes as follows

$$\log p(\mathcal{D} | \theta^{ML}, G) = |\mathcal{D}| \sum_i MI_{\hat{p}}(X_i, \mathbf{X}_{pa(i)}) - |\mathcal{D}| \sum_i H_{\hat{p}}(X_i)$$

Mutual Information and Entropy are based on the empirical distribution \hat{p}

$$\hat{p}(x_i, x_j) = \frac{\text{number of times } X_i = x_i \text{ and } X_j = x_j \text{ in the data}}{\text{number of data points}} = \frac{\text{Count}(x_i, x_j)}{|\mathcal{D}|}$$

Intuition: maximum likelihood prefers networks where parents are informative (high mutual information)

Why does Chow-Liu work?

The likelihood score decomposes as follows

$$\log p(\mathcal{D} | \theta^{ML}, G) = |\mathcal{D}| \sum_i MI_{\hat{p}}(X_i, \mathbf{X}_{pa(i)}) - |\mathcal{D}| \sum_i H_{\hat{p}}(X_i)$$

Want to find a graph G that maximizes

$$\arg \max_G \log P(\mathcal{D} | \theta^{ML}(G), G) = \arg \max_G \sum_i MI(X_i, \mathbf{X}_{pa(i)})$$

If we assume G is a tree, each node has at most one parent

$$\arg \max_{G: G \text{ is tree}} \log P(\mathcal{D} | \theta^{ML}(G), G) = \arg \max_{G: G \text{ is tree}} \sum_{\substack{\text{edges} \\ (i,j) \in G}} MI(X_i, X_j)$$

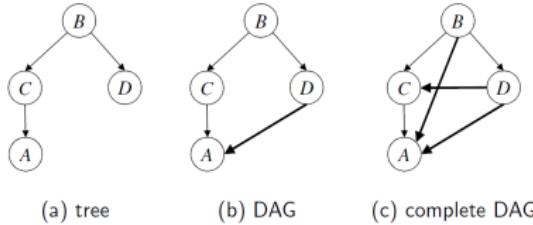
Orientations of edges do not matter for likelihood because MI is symmetric
 $MI(X_i, X_j) = MI(X_j, X_i)$.

Learning general DAG structures

- What about general graphs? Want to find a graph G that maximizes

$$\arg \max_G LL(\mathcal{D}|G) = \arg \max_G \sum_i MI(X_i, \mathbf{X}_{pa(i)})$$

- Problem: a complete DAG is provably the best structure.



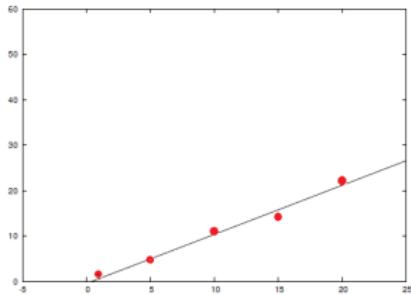
- Intuition: $MI(A, C) \leq MI(A, C \cup D)$. C gives some info about A . $C \cup D$ can only provide more information about A

Learning DAG structures

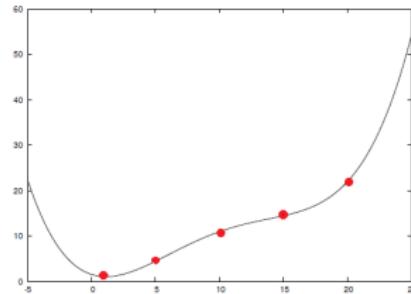
Complete DAGs are undesirable for a number of reasons:

- They make no assertions of **conditional independence**. Topology does not reveal any properties of the distribution. One of the goals of structure learning is to discover knowledge from data
- A complete DAG over n variables has a **treewidth** of $n - 1$ and is therefore hard to do inference (make predictions using the model)
- Complete DAGs suffer from **overfitting** (model has too many parameters compared to the available data)

Overfitting



(a) straight line



(b) 4th degree polynomial

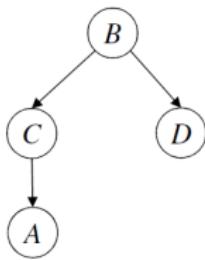
The problem of overfitting: Even though the fit in (b) is perfect, the polynomial does not appear to provide a good generalization of the data beyond the range of the observed data points.

How to avoid overfitting?

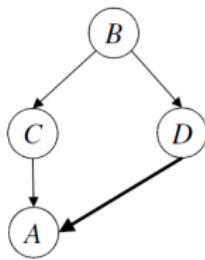
- All available solutions are based on **Occam's razor**: one should prefer simpler models over more complex models, others things being equal
- To realize this principle, one needs
 - ① a measure of model complexity
 - ② a method for balancing the complexity with data fit
- For Bayesian networks: model complexity is measured using the **number of independent parameters** in the model

Model Complexity

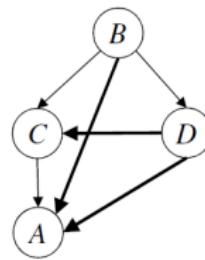
For Bayesian networks G : model complexity $\|G\|$ is measured using the **number of independent parameters** in the model. Assuming binary variables A, B, C, D :



(a) dimension 7



(b) dimension 9



(c) dimension 15

Dimension = number of independent parameters in all the CPTs.

Scoring functions

- Scoring for structure G and data set \mathcal{D} of size $M = |\mathcal{D}|$:

$$Score(G : \mathcal{D}) = LL(\mathcal{D}|G) - \psi(M)||G||$$

Note: Score is ≤ 0

- The first component of this score, $LL(\mathcal{D}|G)$, is the log-likelihood of the data function we considered before.
- The second component, $\psi(M)||G||$, is a **penalty term** that favors simpler models, i.e., ones with a smaller number of independent parameters $||G||$.
- Penalty term has a weight, $\psi(M)$, which is a function of the training data set size $M = |\mathcal{D}|$

Akaike Information Criterion

$$Score(G : \mathcal{D}) = LL(\mathcal{D}|G) - \underbrace{\psi(M)}_{=1} ||G||$$

- When the penalty weight $\psi(M)$ is a constant (independent of M), one gets a score in which model complexity is a secondary issue.
- Log-likelihood function $LL(\mathcal{D}|G)$ grows linearly in the data set size $M = |\mathcal{D}|$ and will quickly dominate the penalty term

$$\log p(\mathcal{D}|\theta^{ML}, G) = |\mathcal{D}| \sum_i MI_{\hat{p}}(X_i, \mathbf{X}_{pa(i)}) - |\mathcal{D}| \sum_i H_{\hat{p}}(X_i)$$

- Model complexity will only be used to distinguish between models that have very similar log-likelihood terms.
- Scoring measure is known as the **Akaike Information Criterion (AIC)**.

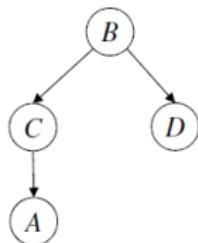
Bayesian Information Criterion

$$\text{Score}(G : \mathcal{D}) = LL(\mathcal{D}|G) - \underbrace{\psi(M)}_{=\log(M)/2} ||G||$$

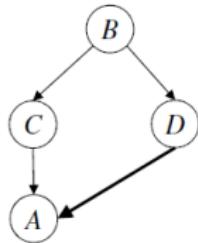
- Another choice of the penalty weight is $\psi(M) = \log(M)/2$ which leads to a “heavier” penalty term.
- Penalty grows as $\log M$ vs. linear growth for the log-likelihood term
- The influence of model complexity will decrease as M grows, allowing the log-likelihood term to eventually dominate the score.
- Scoring is known as **Bayesian Information Criterion** (BIC)

Example

Using BIC to compare models ($M = |\mathcal{D}| = 5$)



$$\begin{aligned} &= -12.1 - \left(\frac{\log_2 5}{2} \right) (7) \\ &= -12.1 - 8.1 \\ &= -20.2 \end{aligned}$$



$$\begin{aligned} &= -10.1 - \left(\frac{\log_2 5}{2} \right) (9) \\ &= -10.1 - 10.4 \\ &= -20.5 \end{aligned}$$

BIC prefers the first model even though it has smaller likelihood.

Consistency

We would like the scores we use to be **consistent**:

- Assume data \mathcal{D} is generated from P^* and G^* is a perfect map for P^* . As $|\mathcal{D}| \rightarrow \infty$, with probability approaching 1:
 - ① For all G , $Score(G : \mathcal{D}) \leq Score(G^* : \mathcal{D})$ (G^* maximizes the score)
 - ② For all G' not I-equivalent to G^* , $Score(G' : \mathcal{D}) < Score(G^* : \mathcal{D})$ (strictly)
- **Theorem:** BIC is consistent.
- What about likelihood score? G^* maximizes, but a fully-connected DAG (which is generally NOT I-equiv to G^*) also maximizes the likelihood
- Consistency only holds in the limit. It says nothing about finite sample size!

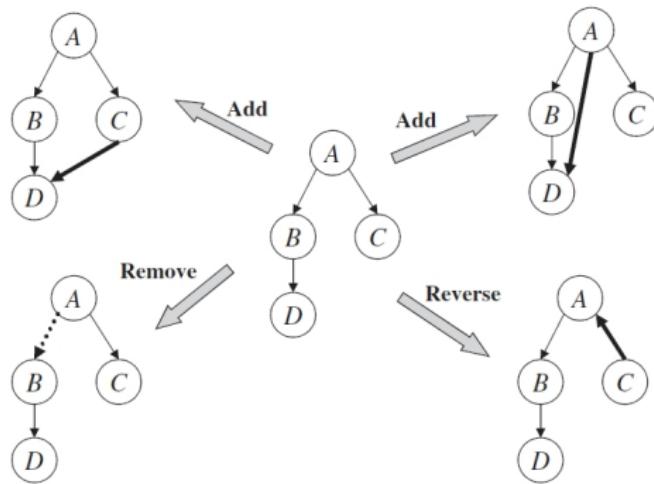
Searching for network structure that optimizes a score

Searching for a network structure that optimizes a score is hard because there are too many possible structures (roughly 2^{n^2} , where n is the number of variables).

There are two main search methods

- Systematic search: very expensive
- Local search style algorithms (greedy style). Fast but suboptimal.

Local search



Note:

- Remove changes one family
- Add changes one family
- Reverse changes two families

Solution found is only locally optimal. Use random restarts, pick the best structure across multiple runs.

Decomposable scores

We would also like our score to be cheap to compute (**decomposable**):

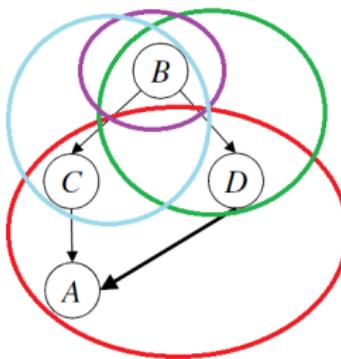
- Score is decomposable if it can be locally computed

$$\text{Score}(G|\mathcal{D}) = \sum_i \text{FamilyScore}(X_i, \text{pa}(X_i)|\mathcal{D})$$

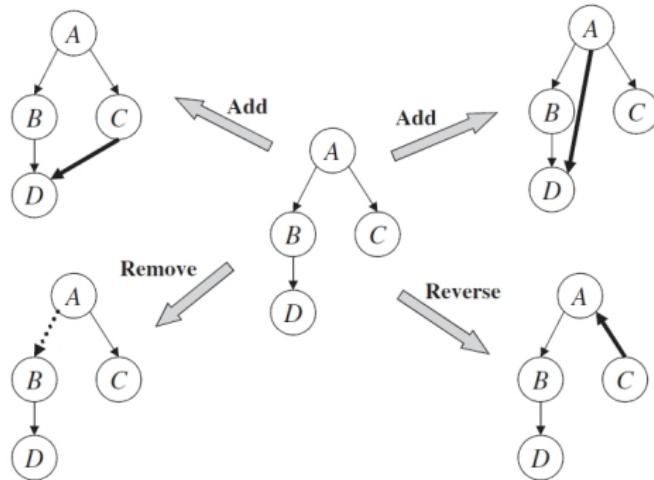
- BIC and AIC are decomposable:

$$\text{FamScore}(X_i, \text{pa}(X_i)|\mathcal{D}) = |\mathcal{D}| (MI(X_i, \mathbf{X}_{\text{pa}(X_i)}) - H(X_i)) - \psi(|\mathcal{D}|) ||\theta_{X_i|\text{pa}(X_i)}||$$

where $||\theta_{X_i|\text{pa}(X_i)}||$ is the number of independent parameters of the CPD $P(X_i|\text{pa}(X_i))$



Local search



Note:

- Remove changes one family
- Add changes one family
- Reverse changes two families

Local score updates: after deleting $A \rightarrow B$ score is

$$\text{Current Score} - \text{FamScore}(B, A | \mathcal{D}) + \text{FamScore}(B | \mathcal{D})$$

Constraining the Search Space

Similar to the algorithm for generating I-maps

- Assume a total ordering of variables
- For each variable find a set of previous variables \mathbf{U}_i that maximize the corresponding local score $Score(X_i, \mathbf{U}_i | \mathcal{D})$. Try to find, for each variable X_i , a good set of parents $U_i \subseteq \{X_1, \dots, X_{i-1}\}$

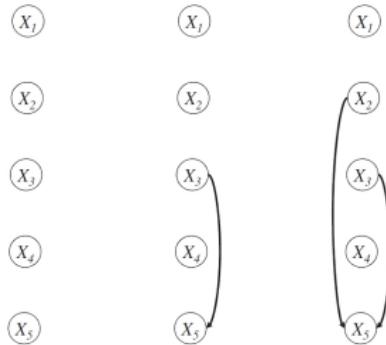
Pros:

- Decomposes into n optimization problems
- No need to worry about creating cycles
- If we bound the maximum number of parents d , then need to search over $\binom{i-1}{d}$ choices of parents

Cons:

- The n optimization problems could still be hard (if d is large)
- Everything depends upon the variable ordering used

Greedy Search: The K3 algorithm



Greedy search for parents of X_5

- X_3 yields the best score
- X_3, X_2 improves the score
- No extension of X_3, X_2 improves the score. Therefore Stop.

For each variable

- Start with empty set of parents
- Successively add variables (as parents) until the score does not increase. At each step, add the variable that increases the score the most.

Summary

- Learning structure is hard: space of possible networks is too big
- Need to be careful about overfitting
 - Restrict the space of possible networks (e.g., trees)
 - Penalize complexity: AIC, BIC, etc.
- Score decomposability reduces overhead during search
- Still need heuristics/approximations
 - Use greedy methods. Fast but suboptimal
 - Fix a variable ordering
 - Combinations, relaxations, etc.
- Confidence on the results?

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 17, March 7, 2017

Today's plan

Plan for today

- Exponential family
- Information geometry

No new algorithms, but general theory/view of some of the concepts we have seen so far. Connections between statistics, graphical models, convex optimization

Recap: parameter learning for MRF

- Let's start with an MRF model:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{c \in C} \theta_c(\mathbf{x}_c) - \ln Z \right)$$

$$Z = \sum_{\mathbf{x}} \prod_{c \in C} \phi_c(\mathbf{x}_c) = \sum_{\mathbf{x}} \exp \left(\sum_{c \in C} \theta_c(\mathbf{x}_c) \right)$$

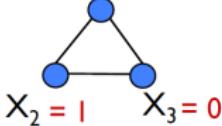
- For parameter learning, we defined a family of probability distributions parameterized by θ (graph structure is fixed):

$$p(\mathbf{x}; \theta) = \frac{1}{Z(\theta)} \prod_{c \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{c \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right)$$

$$= \exp \left(\sum_{c \in C} \sum_{\bar{\mathbf{x}}_c} \theta_c(\bar{\mathbf{x}}_c) \mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c) - \ln Z(\theta) \right) = \exp \left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta) \right)$$

Overcomplete representation: pairwise MRF example

$$f(\mathbf{x}) = \begin{bmatrix} 1[x_1=0] \\ 1[x_1=1] \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{array}{l} \leftarrow \text{Assignment for } X_1 \\ \leftarrow \text{Assignment for } X_2 \\ \leftarrow \text{Assignment for } X_3 \\ \leftarrow \text{Edge assignment for } X_1 X_3 \\ \leftarrow \text{Edge assignment for } X_1 X_2 \\ \leftarrow \text{Edge assignment for } X_2 X_3 \end{array}$$

$X_1 = 0$

 $X_2 = 1$ $X_3 = 0$

For a discrete-variable MRF, $f(\mathbf{x})$ is simply the concatenation of indicator functions:

$$p(\mathbf{x}; \theta) = \exp \left(\sum_{\mathbf{c} \in C} \sum_{\bar{\mathbf{x}}_{\mathbf{c}}} \theta_c(\bar{\mathbf{x}}_{\mathbf{c}}) 1(\mathbf{x}_{\mathbf{c}} = \bar{\mathbf{x}}_{\mathbf{c}}) - \ln Z(\theta) \right) = \exp \left(\theta^T f(\mathbf{x}) - \ln Z(\theta) \right)$$

Exponential family distributions

- Parameterized MRF model (log-linear):

$$p(\mathbf{x}; \theta) = \exp \left(\sum_{c \in C} \sum_{\bar{\mathbf{x}}_c} \theta_c(\bar{\mathbf{x}}_c) \mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c) - \ln Z(\theta) \right) = \exp \left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta) \right)$$

- More generally: **exponential family** distributions

$$p(\mathbf{x}) = h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})]$$

- $h(\mathbf{x})$: base measure (often constant)
- \mathbf{w} : natural parameters
- $\mathbf{f}(\mathbf{x})$: sufficient statistics
- $A(\mathbf{w})$: log-partition function, $A(\mathbf{w}) = \log \left(\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] d\mathbf{x} \right)$
- Herby \mathbf{x} can be continuous (defined over any set). The exponential families include many of the most common distributions!

Example 1: Bernoulli

- The density for a Bernoulli random variable $X \in \{0, 1\}$ can be written as:

$$\begin{aligned} p(x|\pi) &= \pi^x(1-\pi)^{1-x} = \left(\frac{\pi}{1-\pi}\right)^x (1-\pi) \\ &= \exp\left(\log\left(\frac{\pi}{1-\pi}\right)x + \log(1-\pi)\right) \end{aligned}$$

- General functional form for an exponential family

$$p(\mathbf{x}) = h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(w)]$$

- Thus

$$f(x) = x$$

$$w = \log\left(\frac{\pi}{1-\pi}\right)$$

$$A(w) = -\log(1-\pi) = \log(1+e^w)$$

$$h(x) = 1$$

Example 2: Gaussian

- Density for a 1D Gaussian

$$\begin{aligned} p(x|\mu, \sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2 - \frac{1}{2\sigma^2}\mu^2 - \log \sigma\right) \end{aligned}$$

- Exponential family distributions: $p(\mathbf{x}) = h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})]$
- Thus

$$\begin{aligned} \mathbf{f}(x) &= [x, x^2] \\ \mathbf{w} &= \left[\frac{\mu}{\sigma^2}, \frac{-1}{2\sigma^2} \right] = [w_1, w_2] \\ A(\mathbf{w}) &= \frac{\mu^2}{2\sigma^2} + \log \sigma = \frac{-w_1^2}{4w_2} - \frac{1}{2} \log(-2w_2) \\ h(x) &= \frac{1}{\sqrt{2\pi}} \end{aligned}$$

- Other examples: Log-linear models (MRF), Multinomial, Poisson, Exponential, Gamma, Weibull, chi-square, Dirichlet, Geometric

Exponential family distributions

- **Exponential family** distributions

$$p(\mathbf{x}) = h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})]$$

- $h(\mathbf{x})$: base measure (often constant)
- \mathbf{w} : natural parameters
- $\mathbf{f}(\mathbf{x})$: sufficient statistics
- $A(\mathbf{w})$: log-partition function, $A(\mathbf{w}) = \log (\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] d\mathbf{x})$
- Why is this parameterization useful? It reveals useful properties of the probability distribution

Sufficiency

- **Exponential family** distributions

$$p(\mathbf{x}) = h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})]$$

- The sufficient statistic \mathbf{f} is a function that fully “summarizes” the data
- Suppose $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$ are i.i.d. samples from $p(\mathbf{x})$. Then

$$\begin{aligned} p(\mathbf{x}_1, \dots, \mathbf{x}_M) &= h(\mathbf{x}_1) \cdots h(\mathbf{x}_M) \exp \left[\mathbf{w}^T \mathbf{f}(\mathbf{x}_1) - A(\mathbf{w}) \right] \cdots \exp \left[\mathbf{w}^T \mathbf{f}(\mathbf{x}_M) - A(\mathbf{w}) \right] \\ &= h(\mathbf{x}_1) \cdots h(\mathbf{x}_M) \exp \left[\mathbf{w}^T \sum_{m=1}^M \mathbf{f}(\mathbf{x}_m) - M A(\mathbf{w}) \right] \end{aligned}$$

- Suppose we want to estimate \mathbf{w} from $\mathbf{x}_1, \dots, \mathbf{x}_M$ (by MLE). The log-likelihood is

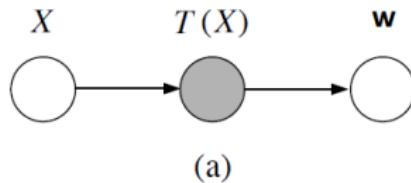
$$\log p(\mathbf{x}_1, \dots, \mathbf{x}_M) = \mathbf{w}^T \sum_{m=1}^M \mathbf{f}(\mathbf{x}_m) - M \cdot A(\mathbf{w}) + const$$

- The likelihood for \mathbf{w} depends on \mathcal{D} **only** through \mathbf{f} ! Similarly for the posterior $p(\mathbf{w}|\mathcal{D})$

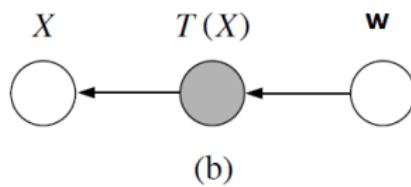
Sufficiency

- Suppose we have a random sample X_1, \dots, X_M taken from a distribution $p(X|\mathbf{w})$ which depends on an unknown parameter \mathbf{w} .
 - Any real-valued function $T(X_1, \dots, X_M)$ of the sample is called a **statistic**. For example, $\sum_i X_i$, $\max(X_1, \dots, X_n)$, $\text{median}(X_1, \dots, X_M)$, etc. are statistics
 - $T(X_1, \dots, X_M)$ is a random variable.
- A statistic $T(X_1, \dots, X_M)$ is said to be **sufficient** for \mathbf{w} if the conditional distribution of \mathbf{w} given $T = t$, does *not* depend on (X_1, \dots, X_n) for any value of t
- Formally, we can say $(X_1, \dots, X_M) \perp \mathbf{w} \mid T(X_1, \dots, X_M)$. $T(X_1, \dots, X_M)$ carries all the information about (X_1, \dots, X_M) needed to estimate \mathbf{w}

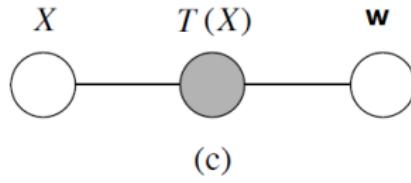
Graphical Model Interpretation



(a)



(b)



(c)

Example

- M independent Bernoulli trials with unknown probability of success w

$$p(x|w) = w^x(1-w)^{1-x} = \exp\left(\log\left(\frac{w}{1-w}\right)x + \log(1-w)\right)$$

- Does the total number of successes contains all the information about w that is in the sample? Does the order in which the successes occurred give additional information about w ?
- Claim: $T(X_1, \dots, X_M) = \sum X_i$ is sufficient for w

$$\begin{aligned} P(X_1 = x_1, \dots, X_M = x_M | T = t) &= \frac{P(X_1 = x_1, \dots, X_M = x_M, T = t)}{P(T = t)} \\ &= \frac{w^t(1-w)^{M-t}}{P(T = t)} = \frac{w^t(1-w)^{M-t}}{w^t(1-w)^{M-t} \binom{M}{t}} \end{aligned}$$

- It does not depend on w , hence T is sufficient

Sufficiency

- **Exponential family** distributions

$$p(\mathbf{x}|\mathbf{w}) = h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})]$$

- Claim: $\mathbf{f}(\mathbf{x})$ is sufficient for \mathbf{w} . Proof:

$$p(\mathbf{f} = \bar{\mathbf{f}} | \mathbf{w}) = \sum_{\mathbf{x}: \mathbf{f}(\mathbf{x}) = \bar{\mathbf{f}}} h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})] = \exp [\mathbf{w}^T \bar{\mathbf{f}} - A(\mathbf{w})] \sum_{\mathbf{x}: \mathbf{f}(\mathbf{x}) = \bar{\mathbf{f}}} h(\mathbf{x})$$

$$p(\mathbf{x} | \mathbf{f} = \bar{\mathbf{f}}, \mathbf{w}) = \frac{p(\mathbf{x}, \mathbf{f} = \bar{\mathbf{f}} | \mathbf{w})}{p(\mathbf{f} = \bar{\mathbf{f}} | \mathbf{w})} = \frac{h(\mathbf{x}) \exp [\mathbf{w}^T \bar{\mathbf{f}} - A(\mathbf{w})]}{\exp [\mathbf{w}^T \bar{\mathbf{f}} - A(\mathbf{w})] \sum_{\mathbf{x}: \mathbf{f}(\mathbf{x}) = \bar{\mathbf{f}}} h(\mathbf{x})}$$

- Suppose $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$ are i.i.d. samples from $p(\mathbf{x}|\mathbf{w})$. Then

$$\begin{aligned} p(\mathbf{x}_1, \dots, \mathbf{x}_M | \mathbf{w}) &= h(\mathbf{x}_1) \cdots h(\mathbf{x}_M) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}_1) - A(\mathbf{w})] \cdots \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}_M) - A(\mathbf{w})] \\ &= h(\mathbf{x}_1) \cdots h(\mathbf{x}_M) \exp \left[\mathbf{w}^T \sum_{m=1}^M \mathbf{f}(\mathbf{x}_m) - M A(\mathbf{w}) \right] \end{aligned}$$

- Claim: $\mathbf{T}(\mathbf{x}_1, \dots, \mathbf{x}_M) = \sum_{m=1}^M \mathbf{f}(\mathbf{x}_m)$ is sufficient for \mathbf{w} .

Derivatives of the log-normalizer

- Exponential family distributions

$$p(x|\mathbf{w}) = h(x) \exp [\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$$

- \mathbf{f} is a sufficient statistic for \mathbf{w} (can compress the data).
- Recall $A(\mathbf{w}) = \log (\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] d\mathbf{x})$ is a log-partition function
- Correspondence between moments and the log-partition function

$$\begin{aligned}\frac{\partial A(\mathbf{w})}{\partial w_i} &= \frac{\partial \log (\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] d\mathbf{x})}{\partial w_i} = \frac{\int h(\mathbf{x}) \frac{\partial \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})]}{\partial w_i} d\mathbf{x}}{\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] d\mathbf{x}} \\ &= \frac{\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] f_i(\mathbf{x}) d\mathbf{x}}{\int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x})] d\mathbf{x}} = \int h(\mathbf{x}) \exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})] f_i(\mathbf{x}) d\mathbf{x} \\ &= \int p(x|\mathbf{w}) f_i(x) dx = E_{p(x|\mathbf{w})}[f_i]\end{aligned}$$

- The derivatives of the log normalizer give the **moments of the sufficient statistics**. Can evaluate an integral computing a derivative

Convexity

- Exponential family distributions

$$p(x|\mathbf{w}) = h(x) \exp [\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$$

- \mathbf{f} is a sufficient statistic for \mathbf{w} (can compress the data)
- Correspondence between moments and the log-partition function

$$\frac{\partial A(\mathbf{w})}{\partial w_i} = E_{p(x|\mathbf{w})}[f_i]$$

- Similarly, you can show that 2nd derivative gives the second-order moments, i.e.

$$\left(\nabla^2 A(\mathbf{w}) \right)_{ij} = \mathbb{E}_{p(x|\mathbf{w})}[f^i(x)f^j(x)] - \mathbb{E}_{p(x|\mathbf{w})}[f^i(x)]\mathbb{E}_{p(x|\mathbf{w})}[f^j(x)] = (\text{cov}[\mathbf{f}(x)])_{ij}$$

- Since covariance matrices are always positive semi-definite, this proves that $A(\mathbf{w})$ is **convex**

Example

- For example, when p is a Gaussian distribution,

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right) = h(x) \exp[\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$$

where

$$\mathbf{f}(x) = [x, x^2]$$

$$\mathbf{w} = \left[\frac{\mu}{\sigma^2}, \frac{-1}{2\sigma^2} \right] = [w_1, w_2]$$

$$A(\mathbf{w}) = \frac{\mu^2}{2\sigma^2} + \log \sigma = \frac{-w_1^2}{4w_2} - \frac{1}{2} \log(-2w_2)$$

- For example

$$\frac{\partial A(\mathbf{w})}{\partial w_1} = \frac{-2w_1}{4w_2} = \frac{-2\frac{\mu}{\sigma^2}}{4\frac{-1}{2\sigma^2}} = \mu = E_{p(x|\mathbf{w})}[x]$$

- The expectation of $\mathbf{f}(x)$ gives the first and second-order (non-central) moments, from which one can solve for μ and σ

Minimal and overcomplete representations

- Exponential family distributions

$$p(x|\mathbf{w}) = h(x) \exp [\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$$

- In a **minimal** exponential family, the components of the sufficient statistics $\mathbf{f}(x)$ are linearly independent:
 - There does not exist a nonzero \mathbf{a} such that $\mathbf{a} \cdot \mathbf{f}(x) = \text{constant}$ for all x .
 - There is a unique \mathbf{w} associated with each distribution
- If an exponential family is not minimal, it is called **overcomplete**
- Consider a Bernoulli random variable $p(x) = w^x(1-w)^{1-x}$
 - $p(x) = \exp(x \log w + (1-x) \log(1-w)) = \exp([\log w, \log 1-w] \cdot [x, 1-x])$
 - $p(x) = \exp\left(x \log \frac{w}{1-w} + \log(1-w)\right)$
- The first parameterization is **overcomplete** ($x + 1 - x = 1$ for all x), while the second is **minimal**.

Mapping of distributions to/from moments

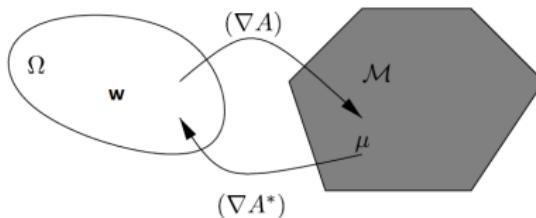
- Exponential family distributions

$$p(x|\mathbf{w}) = h(x) \exp [\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$$

- Correspondence between moments and log-partition function

$$\nabla A(\mathbf{w}) = E_{p(x|\mathbf{w})}[\mathbf{f}(X)] = \boldsymbol{\mu}$$

- In a minimal exponential family, the expected sufficient statistics (**mean parameters**) $\boldsymbol{\mu} = E_{p(x|\mathbf{w})}[\mathbf{f}(X)]$ are another parameterization of the distribution
- That is, there is a 1-1 mapping between \mathbf{w} and $\boldsymbol{\mu}$.



- Many distributions are traditionally parameterized with mean parameters

Example

- For example, when p is a Gaussian distribution,

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right) = h(x) \exp[\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$$

where

$$\mathbf{f}(x) = [x, x^2]$$

$$\mathbf{w} = \left[\frac{\mu}{\sigma^2}, \frac{-1}{2\sigma^2} \right] = [w_1, w_2]$$

$$A(\mathbf{w}) = \frac{\mu^2}{2\sigma^2} + \ln \sigma = \frac{-w_1^2}{4w_2} - \frac{1}{2} \ln(-2w_2)$$

- The expectation of $\mathbf{f}(x)$ gives the first and second-order (non-central) moments, $E[x] = \mu$ and $E[x^2] = \mu^2 + \sigma^2$

Forward mapping

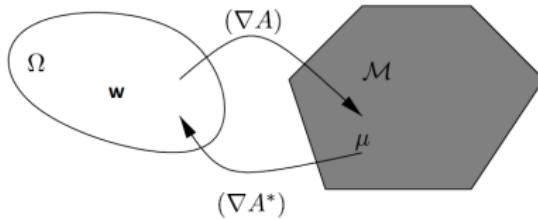
- Parameterized MRF model (log-linear):

$$p(\mathbf{x}|\mathbf{w}) = \exp \left(\sum_{c \in C} \sum_{\bar{\mathbf{x}}_c} \mathbf{w}_c(\bar{\mathbf{x}}_c) \mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c) - \ln Z(\mathbf{w}) \right) = \exp \left(\mathbf{w}^T \mathbf{f}(\mathbf{x}) - \ln Z(\mathbf{w}) \right)$$

- Correspondence between moments and log-partition function

$$\nabla A(\mathbf{w}) = \nabla \ln Z(\mathbf{w}) = E_{p(\mathbf{x}|\mathbf{w})}[\mathbf{f}(\mathbf{X})] = \boldsymbol{\mu}$$

- That is, there is a 1-1 mapping between \mathbf{w} and $\boldsymbol{\mu}$.



- Computing the forward mapping ∇A requires solving an inference problem

$$\frac{\partial A(\mathbf{w})}{\partial \mathbf{w}_c(\bar{\mathbf{x}}_c)} = E_{p(\mathbf{x}|\mathbf{w})}[\mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c)] = p(\mathbf{x}_c = \bar{\mathbf{x}}_c | \mathbf{w})$$

Recall KL-divergence

- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

(measures the expected number of extra bits required to describe *samples from $p(\mathbf{x})$* using a code based on q instead of p)

- $D(p\|q) \geq 0$ for all p, q , with equality if and only if $p = q$
- Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$
- KL-divergences and exponential families fit well together!

M-projections

- Suppose that Q is an exponential family with sufficient statistics \mathbf{f} , and $p(\mathbf{x})$ is an arbitrary distribution
- M-projection (moment projection) of p onto Q is:

$$q^* = \arg \min_{q \in Q} D(p \| q) = \arg \min_{q \in Q} \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- Intuitively: distribution in Q that is as close as possible to p
- Suppose there exists w^* such that $E_{q(x|w^*)}[\mathbf{f}(X)] = E_p[\mathbf{f}(X)] = \boldsymbol{\mu}$. It can be shown (Theorem 8.6) that the the M-projection of p onto Q is $q(x|w^*)$
- M-projection preserves marginals (called “moment matching”)

M-projections

- Recall $q_{\mathbf{w}} \in Q$, $q_{\mathbf{w}}(x) = h(x) \exp [\mathbf{w}^T \mathbf{f}(x) - A(\mathbf{w})]$
- Suppose there exists \mathbf{w}^* such that $E_{q(x|\mathbf{w}^*)}[\mathbf{f}(X)] = E_p[\mathbf{f}(X)] = \mu$
- M-projection (moment projection) of p onto Q is:

$$q^* = \arg \min_{q \in Q} D(p||q) = \arg \min_{q \in Q} \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}$$

$$\begin{aligned} D(p||q_{\mathbf{w}}) - D(p||q_{\mathbf{w}^*}) &= \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q_{\mathbf{w}}(\mathbf{x})} \frac{q_{\mathbf{w}^*}(\mathbf{x})}{p(\mathbf{x})} = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{\exp [\mathbf{w}^{*T} \mathbf{f}(\mathbf{x}) - A(\mathbf{w}^*)]}{\exp [\mathbf{w}^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w})]} \\ &= \sum_{\mathbf{x}} p(\mathbf{x}) (\mathbf{w}^* - \mathbf{w})^T \mathbf{f}(\mathbf{x}) - A(\mathbf{w}^*) + A(\mathbf{w}) \\ &= (\mathbf{w}^* - \mathbf{w})^T \sum_{\mathbf{x}} p(\mathbf{x}) \mathbf{f}(\mathbf{x}) - A(\mathbf{w}^*) + A(\mathbf{w}) \\ &= (\mathbf{w}^* - \mathbf{w})^T \sum_{\mathbf{x}} q(x|\mathbf{w}^*) \mathbf{f}(\mathbf{x}) - A(\mathbf{w}^*) + A(\mathbf{w}) \\ &= D(q_{\mathbf{w}^*}||q_{\mathbf{w}}) - D(q_{\mathbf{w}^*}||q_{\mathbf{w}^*}) = D(q_{\mathbf{w}^*}||q_{\mathbf{w}}) \geq 0 \end{aligned}$$

- Given any $q_{\mathbf{w}}$, $D(p||q_{\mathbf{w}}) \geq D(p||q_{\mathbf{w}^*})$, so $q_{\mathbf{w}^*}$ is the M-projection

Recall connection with Maximum likelihood

- Suppose we are given some data, independent identically distributed (i.i.d.) samples $(\bar{x}_1, \dots, \bar{x}_N)$
- An empirical distribution can be defined as: $\tilde{p}(x) \triangleq \frac{1}{N} \sum_{i=1}^N \delta(x, \bar{x}_i)$
- Let $p(x|w)$ be a family of parametric distributions. Then

$$\begin{aligned} D(\tilde{p} \| p(x|w)) &= \sum_x \tilde{p}(x) \log \frac{\tilde{p}(x)}{p(x|w)} = \sum_x \tilde{p}(x) \log \tilde{p}(x) - \sum_x \tilde{p}(x) \log p(x|w) \\ \sum_x \tilde{p}(x) \log p(x|w) &= \sum_x \frac{1}{N} \sum_{i=1}^N \delta(x, \bar{x}_i) \log p(x|w) \\ &= \frac{1}{N} \sum_{i=1}^N \sum_x \delta(x, \bar{x}_i) \log p(x|w) \\ &= \frac{1}{N} \sum_{i=1}^N \log p(\bar{x}_i|w) \end{aligned}$$

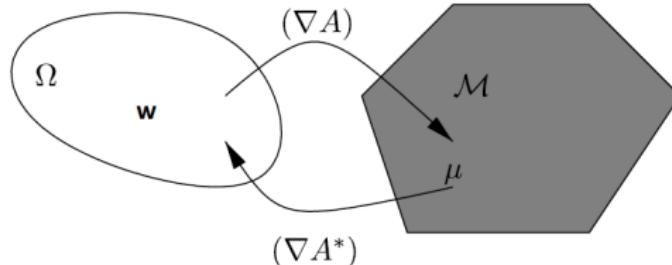
- Minimizing KL divergence (as a function of w), i.e., computing an M-projection of \tilde{p} , is equivalent to maximizing log-likelihood of the data (learning)

Connection with Maximum likelihood

- Now suppose $p(x|w)$ is an exponential family
- As we have seen before, $D(\tilde{p}\|p(x|w))$ is minimized when

$$E_{p(x|w^*)}[\mathbf{f}(x)] = E_{\tilde{p}}[\mathbf{f}(x)] = \frac{1}{N} \sum_{i=1}^N \mathbf{f}(\bar{x}_i)$$

- Moment matching:** choose w^* such that the mean parameters (expected sufficient statistics) match the empirical average in the data
- w^* is also the maximum likelihood estimate of the parameters!
- Natural statistical interpretation for the reverse mapping $(\nabla A)^{-1}$



Connection with Maximum likelihood

- Now suppose $p(x|w)$ is an exponential family
- Suppose we want to estimate \mathbf{w} from $(\bar{x}_1, \dots, \bar{x}_N)$ (by MLE).

$$\begin{aligned} p(\bar{x}_1, \dots, \bar{x}_N) &= h(\bar{x}_1) \cdots h(\bar{x}_N) \exp \left[\mathbf{w}^T \mathbf{f}(\bar{x}_1) - A(\mathbf{w}) \right] \cdots \exp \left[\mathbf{w}^T \mathbf{f}(\bar{x}_N) - A(\mathbf{w}) \right] \\ &= h(\bar{x}_1) \cdots h(\bar{x}_N) \exp \left[\mathbf{w}^T \sum_{m=1}^N \mathbf{f}(\bar{x}_m) - NA(\mathbf{w}) \right] \end{aligned}$$

- The log-likelihood is

$$\log p(\mathbf{x}_1, \dots, \mathbf{x}_K) = \mathbf{w}^T \sum_{m=1}^N \mathbf{f}(\bar{x}_m) - NA(\mathbf{w}) + const$$

- $A(\mathbf{w})$ is convex, so this is a concave optimization problem
- Setting the gradient to zero we get again the moment matching condition

$$\frac{1}{N} \sum_{m=1}^N \mathbf{f}(\bar{x}_m) = \nabla A(\mathbf{w}) = E_{p(x|w)}[\mathbf{f}(x)]$$

Maximum Entropy Interpretation

- Entropy of a probability distribution p measures “how random” p is.

$$H(p) = - \int p(x) \ln p(x) dx$$

- Suppose one seeks a distribution $p(x)$ such that

$$E_{p(x)}[f_i(x)] = t_i$$

and $p(x)$ has **maximum entropy**.

- For example, force desired mean, variance, skewness, etc. but in a sense minimizes all other assumptions
- The solution has exactly the form of the exponential family (proof based on Lagrange multipliers)
- Theorem:** Exponential family distributions maximize the entropy $H(p)$ over all distributions satisfying

$$E_{p(x)}[f_i(x)] = t_i$$

- Involves the computation of the reverse mapping $(\nabla A)^{-1}(\mathbf{t})$

Summary

Key ideas:

- Exponential families are a general class of probabilistic models. Includes MRF, Bayes Net, and many common distributions
- Can read sufficient statistics directly
- Nice convexity properties
- Forward and backward mappings: inference and learning

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 18, March 9, 2017

Today's plan

Plan for today

- Variational Inference
- How to approximate probability distributions (in terms of KL divergence)

Approximate marginal inference

- Given the joint $p(x_1, \dots, x_n)$ represented as a graphical model, how do we perform **marginal inference**, e.g. to compute $p(x_1 | e)$?
- We showed that doing this exactly is NP-hard
- Key idea:** approximate posterior with **simpler** distribution that is as close to $p(x_1 | e)$ as possible
- Loopy BP (and many other algorithms) can be interpreted in this way

Why should we hope that we can find a simple approximation?

- Prior distribution $p(\mathbf{x})$ is complicated. Need to describe all possible states of the world (and relationships between variables)
- Posterior distribution $p(\mathbf{x} | \mathbf{e})$ is often simpler:
 - Have made many observations, thus less uncertainty
 - Variables can become “almost independent”
- For now: Represent posterior as undirected model (and instantiate observations)

$$p(\mathbf{x} | \mathbf{e}) = \frac{1}{Z} \prod_{\mathbf{c} \in C} \phi_{\mathbf{c}}(\mathbf{x}_{\mathbf{c}})$$

Variational methods

- **Goal:** Approximate difficult distribution $p(\mathbf{x} | \mathbf{e})$ with a new distribution $q(\mathbf{x})$ such that:
 - ① $p(\mathbf{x} | \mathbf{e})$ and $q(\mathbf{x})$ are “close” (**approximate**)
 - ② Computation on $q(\mathbf{x})$ is easy ($q(\mathbf{x})$ is **simple**)
- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

(measures the expected number of extra bits required to describe *samples from $p(\mathbf{x})$* using a code based on q instead of p)

- $D(p\|q) \geq 0$ for all p, q , with equality if and only if $p = q$
- Notice that KL-divergence is **asymmetric**

KL-divergence (see Section 8.5 of K&F)

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- Suppose p is the true distribution we wish to do inference with
- What is the difference between the solution to

$$\arg \min_q D(p\|q)$$

(called the *M-projection* of q onto p) and

$$\arg \min_q D(q\|p)$$

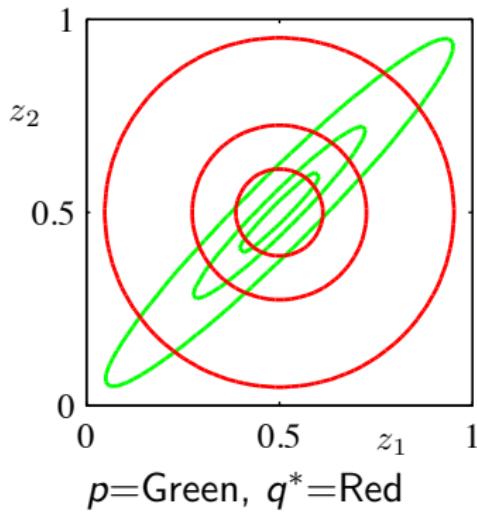
(called the *I-projection*)?

- These two will differ when q is minimized over a restricted set of probability distributions $Q = \{q_1, \dots, q_T\}$, and in particular when $p \notin Q$

KL-divergence – M-projection

$$q^* = \arg \min_{q \in Q} D(p \| q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

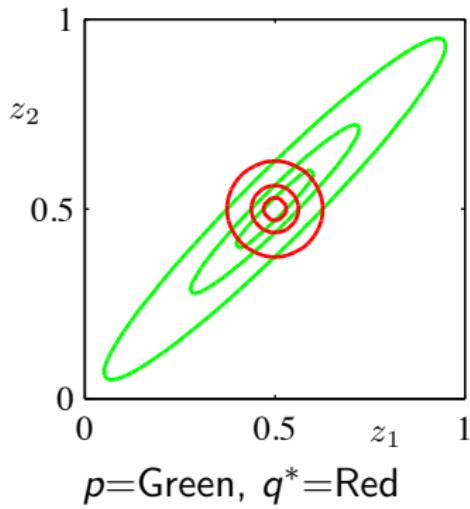
For example, suppose that $p(\mathbf{z})$ is a 2D Gaussian and Q is the set of all Gaussian distributions with diagonal covariance matrices:



KL-divergence – I-projection

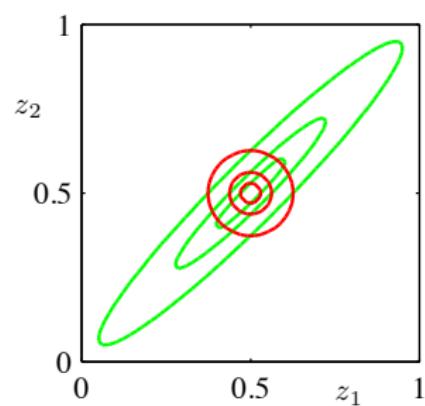
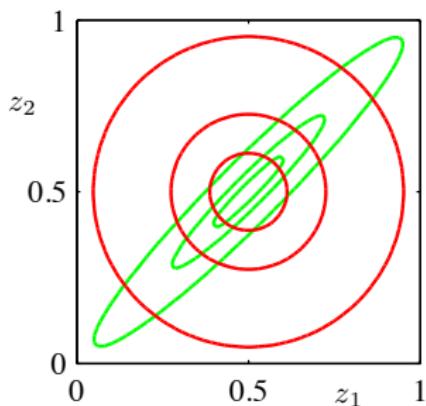
$$q^* = \arg \min_{q \in Q} D(q \| p) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}.$$

For example, suppose that $p(\mathbf{z})$ is a 2D Gaussian and Q is the set of all Gaussian distributions with diagonal covariance matrices:



KL-divergence (single Gaussian)

In this simple example, both the M-projection and I-projection find an approximate $q(\mathbf{x})$ that has the correct mean (i.e. $E_p[\mathbf{z}] = E_q[\mathbf{z}]$):

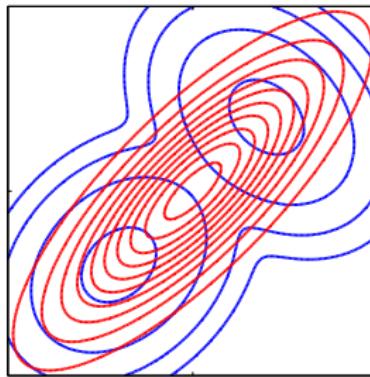


What if $p(\mathbf{x})$ is multi-modal?

KL-divergence – M-projection (mixture of Gaussians)

$$q^* = \arg \min_{q \in Q} D(p \| q) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

Now suppose that $p(x)$ is mixture of two 2D Gaussians and Q is the set of all 2D Gaussian distributions (with arbitrary covariance matrices):

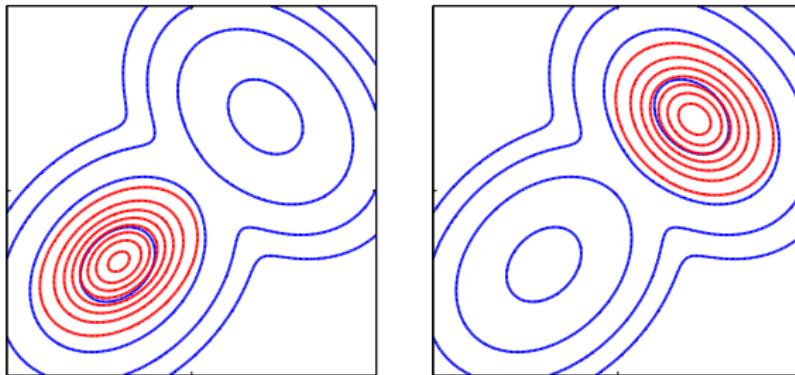


$p=\text{Blue}$, $q^*=\text{Red}$

M-projection yields distribution $q(x)$ with the correct mean and covariance. q chosen to “support” p : $p(x) > 0 \Rightarrow q(x) > 0$

KL-divergence – I-projection (mixture of Gaussians)

$$q^* = \arg \min_{q \in Q} D(q\|p) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}.$$



p =Blue, q^* =Red (two equivalently good solutions!)

Unlike the M-projection, the I-projection does not necessarily yield the correct moments. Underestimates support (overconfident):
 $p(x) = 0 \Rightarrow q(x) = 0$

“Simple” distributions

- **Goal:** Approximate difficult distribution $p(\mathbf{x} | \mathbf{e})$ with a new distribution $q(\mathbf{x})$ such that:
 - ① $p(\mathbf{x} | \mathbf{e})$ and $q(\mathbf{x})$ are “close”
 - ② Computation on $q(\mathbf{x})$ is easy
- Simplest distribution: q is fully factorized (**mean field**).
$$q(X_1, \dots, X_n) = q_1(X_1)q_2(X_2) \cdots q_n(X_n)$$
- Note: it is the product of marginal distributions
$$q_i(X_i) = \sum_{X_1} \sum_{X_{i-1}} \sum_{X_{i+1}} \sum_{X_n} q(X_1, \dots, X_n)$$
- Let $Q = \{q : q(x) = \prod_i q_i(x_i)\}$ be the family of all fully-factored distributions over n variables
- For example, when $n = 2$ and binary variables
$$Q = \{q_\phi(x_1, x_2) = \phi_1^{x_1}(1 - \phi_1)^{1-x_1}\phi_2^{x_2}(1 - \phi_2)^{1-x_2}, 0 \leq \phi_i \leq 1\}$$
- We can try to solve $\arg \min_{q \in Q} D(p \| q)$ or $\arg \min_{q \in Q} D(q \| p)$

Mean Field Inference with M-projection

- Family of distributions: $Q = \{q : q(x) = \prod_i q_i(x_i)\}$

$$\begin{aligned} D(p\|q) &= \sum_x p(x) \log \left(\frac{p(x)}{q(x)} \right) = \sum_x p(x) \log p(x) - \sum_x p(x) \log q(x) \\ &= \sum_x p(x) \log p(x) - \sum_x p(x) \log \prod_i q_i(x_i) \\ &= \sum_x p(x) \log \frac{p(x)}{\prod_i p(x_i)} + \sum_x p(x) \log \prod_i \frac{p(x_i)}{q_i(x_i)} \\ &= \sum_x p(x) \log \frac{p(x)}{\prod_i p(x_i)} + \sum_x p(x) \sum_i \log \frac{p(x_i)}{q_i(x_i)} \\ &= D(p\|q_M) + \sum_i D(p(x_i)\|q_i(x_i)) \geq D(p\|q_M) \end{aligned}$$

where $q_M(x) = \prod_i p(x_i)$. **Moment matching:** Optimal solution is the product of the marginals of p (intractable to compute)!

- M-projection is the “right” notion of distance, but is intractable

Finding the M-projection requires inference

M-projection is:

$$q^* = \arg \min_{q \in Q} D(p \| q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- Suppose that $Q = \{q(x) = h(x) \exp(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}) - A(\mathbf{w})) , \mathbf{w} \in W\}$ is an exponential family ($p(\mathbf{x})$ can be arbitrary) and that we could perform the M-projection, finding q^*
- It can be shown (see Thm 8.6) that the expected sufficient statistics, with respect to $q^*(\mathbf{x})$, match *exactly* the ones with respect to $p(\mathbf{x})$:

$$E_{q^*}[\mathbf{f}(\mathbf{x})] = E_p[\mathbf{f}(\mathbf{x})]$$

- M-projection preserves marginals (called “moment matching”)
- M-projections require expectations with respect to p (inference). Usually intractable

Most variational inference algorithms make use of the l-projection

Variational methods, I-Projections

- Suppose $p(\mathbf{x}; \theta)$ is a *fixed* graphical model we want to approximate:

$$p(\mathbf{x}; \theta) = \frac{1}{Z(\theta)} \prod_{\mathbf{c} \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right)$$

- All of the approaches begin as follows:

$$\begin{aligned} D(q \| p) &= \sum_{\mathbf{x}} q(\mathbf{x}) \ln \frac{q(\mathbf{x})}{p(\mathbf{x})} \\ &= - \sum_{\mathbf{x}} q(\mathbf{x}) \ln p(\mathbf{x}) + \sum_{\mathbf{x}} q(\mathbf{x}) \ln q(\mathbf{x}) \\ &= - \sum_{\mathbf{x}} q(\mathbf{x}) \left(\sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right) - H(q(\mathbf{x})) \\ &= - \sum_{\mathbf{c} \in C} \sum_{\mathbf{x}} q(\mathbf{x}) \theta_c(\mathbf{x}_c) + \sum_{\mathbf{x}} q(\mathbf{x}) \ln Z(\theta) - H(q(\mathbf{x})) \\ &= - \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + \ln Z(\theta) - H(q(\mathbf{x})) \end{aligned}$$

The log-partition function

- Since $D(q\|p) \geq 0$, we have

$$-\sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + \ln Z(\theta) - H(q(\mathbf{x})) = D(q\|p) \geq 0,$$

which implies that

$$\ln Z(\theta) \geq \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + H(q(\mathbf{x})).$$

- Thus, *any* approximating distribution $q(\mathbf{x})$ gives a lower bound on the log-partition function (for a BN, this is the probability of the evidence)
- Recall that $D(q\|p) = 0$ if and only if $p = q$. Thus, if we allow ourselves to optimize over *all* distributions, we have:

$$\ln Z(\theta) = \max_q \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + H(q(\mathbf{x})).$$

- **Variational:** optimization problem over functions

Variational methods

- Suppose that we have an arbitrary graphical model:

$$p(\mathbf{x}; \theta) = \frac{1}{Z(\theta)} \prod_{\mathbf{c} \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right)$$

- Finding the *approximating distribution* $q(\mathbf{x}) \in Q$ that minimizes the I-projection to $p(\mathbf{x})$, i.e. $D(q \| p) = \sum_{\mathbf{x}} q(\mathbf{x}) \ln \frac{q(\mathbf{x})}{p(\mathbf{x})}$, is equivalent to

$$\max_{q \in Q} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + H(q(\mathbf{x}))$$

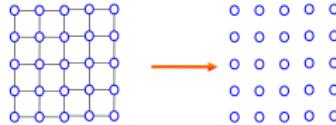
where $E_q[\theta_c(\mathbf{x}_c)] = \sum_{\mathbf{x}_c} q(\mathbf{x}_c) \theta_c(\mathbf{x}_c)$ and $H(q(\mathbf{x}))$ is the *entropy* of $q(\mathbf{x})$

- If $p \in Q$, the value of the objective at optimality is **equal to** $\ln Z(\theta)$
- How should we approximate this? We need a compact way of representing $q(\mathbf{x})$ and finding the maxima

Two types of variational algorithms: Mean-field and relaxation

$$\max_{q \in Q} \sum_{c \in C} \sum_{x_c} q(x_c) \theta_c(x_c) + H(q(x)).$$

- Although this function is concave and thus in theory should be easy to optimize, we need some compact way of representing $q(x)$
- *Mean-field* algorithms assume a factored representation of the joint distribution, e.g.



$$q(x) = \prod_{i \in V} q_i(x_i) \quad (\text{called } \textit{naive} \text{ mean field})$$

Naive mean-field

- Suppose that Q consists of all fully factored distributions, of the form
$$q(\mathbf{x}) = \prod_{i \in V} q_i(x_i)$$
- We can use this to simplify

$$\max_{q \in Q} \sum_{\mathbf{c} \in C} \sum_{\mathbf{x}_{\mathbf{c}}} q(\mathbf{x}_{\mathbf{c}}) \theta_c(\mathbf{x}_{\mathbf{c}}) + H(q)$$

- First, note that $q(\mathbf{x}_c) = \prod_{i \in c} q_i(x_i)$
- Next, notice that the joint entropy decomposes as a sum of local entropies:

$$\begin{aligned} H(q) &= - \sum_{\mathbf{x}} q(\mathbf{x}) \ln q(\mathbf{x}) \\ &= - \sum_{\mathbf{x}} q(\mathbf{x}) \ln \prod_{i \in V} q_i(x_i) = - \sum_{\mathbf{x}} q(\mathbf{x}) \sum_{i \in V} \ln q_i(x_i) \\ &= - \sum_{i \in V} \sum_{\mathbf{x}} q(\mathbf{x}) \ln q_i(x_i) \\ &= - \sum_{i \in V} \sum_{x_i} q_i(x_i) \ln q_i(x_i) \sum_{\mathbf{x}_{V \setminus i}} q(\mathbf{x}_{V \setminus i} \mid x_i) = \sum_{i \in V} H(q_i) \end{aligned}$$

Naive mean-field

- Putting these together, we obtain the following variational objective:

$$(*) \max_q \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \prod_{i \in c} q_i(x_i) + \sum_{i \in V} H(q_i)$$

subject to the constraints

$$q_i(x_i) \geq 0 \quad \forall i \in V, x_i \in \text{Val}(X_i)$$

$$\sum_{x_i \in \text{Val}(X_i)} q_i(x_i) = 1 \quad \forall i \in V$$

- We obtain a *lower bound* on the partition function, i.e. $(*) \leq \ln Z(\theta)$

Example: Naive mean-field for pairwise MRFs

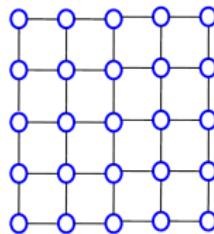
- General graphical model $p(\mathbf{x}; \theta)$ we want to approximate:

$$p(\mathbf{x}; \theta) = \frac{1}{Z(\theta)} \prod_{\mathbf{c} \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right)$$

- Variational objective:

$$(*) \max_q \sum_{\mathbf{c} \in C} \sum_{\mathbf{x}_c} \theta_c(\mathbf{x}_c) \prod_{i \in c} q_i(x_i) + \sum_{i \in V} H(q_i)$$

- If $p(\mathbf{x}; \theta) = \exp \left(\sum_{ij \in E} \theta_{ij}(x_i, x_j) - \ln Z(\theta) \right)$ is a pairwise MRF



$$(*) \max_q \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) q_i(x_i) q_j(x_j) - \sum_{i \in V} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

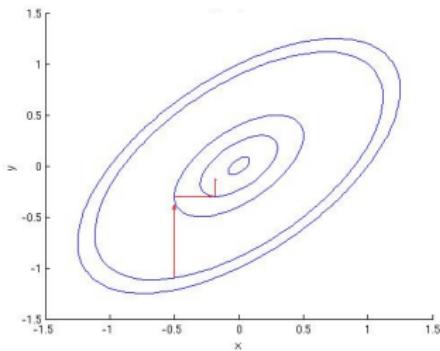
Naive mean-field for pairwise MRFs

- How do we maximize the variational objective (get the best lower bound)?

$$(*) \max_q \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) q_i(x_i) q_j(x_j) - \sum_{i \in V} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

- This is a non-concave optimization problem, with many local maxima!
- Nonetheless, we can greedily maximize it using **block coordinate ascent**:
 - ① Initialize $\{q_i(x_i)\}$ arbitrarily (usually, at random or uniform)
 - ② Iterate over each of the variables $i \in V$. For variable i ,
 - ③ Fully maximize $(*)$ with respect to $\{q_i(x_i), \forall x_i \in \text{Val}(X_i)\}$.
 - ④ Repeat until convergence.

Coordinate ascent



- Maximize along one direction at a time:
 - Initialize x_0, y_0
 - $y_1 = \arg \max_y f(x_0, y)$
 - $x_1 = \arg \max_x f(x, y_1)$
 - $y_2 = \arg \max_y f(x_1, y)$
 - ...
- Objective keeps improving.
$$f(x_0, y_0) \leq f(x_0, y_1) \leq f(x_1, y_1) \leq f(x_1, y_2) \leq \dots$$

Naive mean-field updates for pairwise MRFs

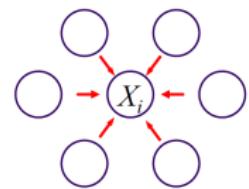
$$(*) \quad \max_q \quad \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) q_i(x_i) q_j(x_j) - \sum_{i \in V} \sum_{x_i} q_i(x_i) \ln q_i(x_i)$$

subject to the constraints

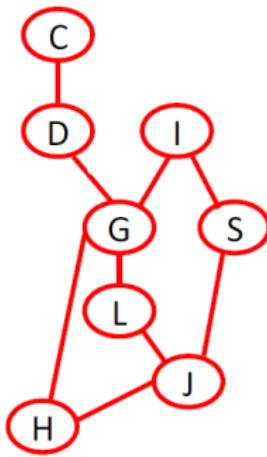
$$q_i(x_i) \geq 0 \quad \forall i \in V, x_i \in \text{Val}(X_i), \quad \sum_{x_i \in \text{Val}(X_i)} q_i(x_i) = 1 \quad \forall i \in V$$

- Iterate over each of the variables $i \in V$ until convergence. For variable i ,
- Fully maximize $(*)$ with respect to $\{q_i(x_i), \forall x_i \in \text{Val}(X_i)\}$.
- How to do that? Constructing the Lagrangian, taking the derivative, setting to zero, and solving yields the update:

$$q(x_i) = \frac{1}{Z_i} \exp \left\{ \sum_{j \in N(i)} \sum_{x_j} q_j(x_j) \theta_{ij}(x_i, x_j) \right\}$$



Example update



$$Q_D(x_D = 1) = \frac{1}{Z_D} \exp \left(\sum_{x_c} Q_C(x_c) \theta_{CD}(x_c, x_D = 1) + \sum_{x_g} Q_G(x_g) \theta_{GD}(x_g, x_D = 1) \right)$$

Naive mean-field interpretation

- Iterate over each of the variables $i \in V$ until convergence. For variable i ,

$$q(x_i) = \frac{1}{Z_i} \exp \left\{ \sum_{j \in N(i)} \sum_{x_j} q_j(x_j) \theta_{ij}(x_i, x_j) \right\}$$

- What is the conditional probability of X_i given its Markov blanket?

$$p(x_i | N(i)) = \frac{1}{\hat{Z}_i} \exp \left\{ \sum_{j \in N(i)} \sum_{x_j} 1[X_j = x_j] \theta_{ij}(x_i, x_j) \right\}$$

- It's a geometric average of conditional probabilities. It iterates until they are consistent with each other
- It's like a Gibbs sampler update, but in "expectation"

General Naive mean-field updates

$$\max_q \quad \sum_{\mathbf{c} \in C} \sum_{\mathbf{x}_{\mathbf{c}}} \theta_c(\mathbf{x}_{\mathbf{c}}) \prod_{i \in c} q_i(x_i) + \sum_{i \in V} H(q_i)$$

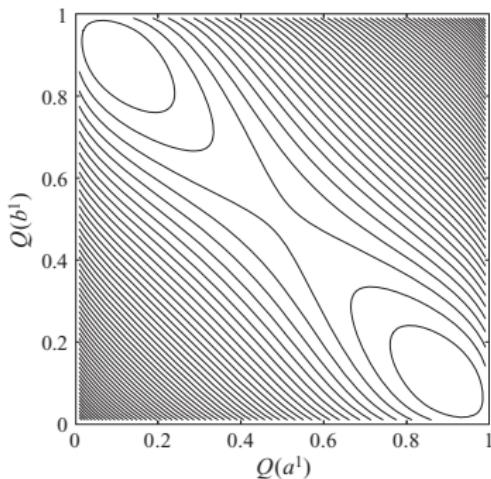
- **Theorem:** For general MRFs with cliques C , Q is a stationary point (local maximum, minimum or stationary point) if and only if for each i and x_i

$$Q(x_i) = \frac{1}{Z_i} \exp \left(\sum_{\mathbf{c} \ni i} \sum_{\mathbf{x}_{\mathbf{c}} \setminus x_i} \theta_c(\mathbf{x}_{\mathbf{c}}, x_i) \prod_{j \in c, j \neq i} q_j(x_j) \right)$$

- Guaranteed to converge (in practice, to a local maximum)!
- Gives both approximating distribution Q (and approximate marginals $Q(x_i)$) and lower bound on $\ln Z$
- Can get stuck in local optimum

How accurate will the approximation be?

- Consider a distribution which is an approximate XOR of two binary variables A and B : $p(a, b) = 0.5 - \epsilon$ if $a \neq b$ and $p(a, b) = \epsilon$ if $a = b$
- The contour plot of the variational objective is:

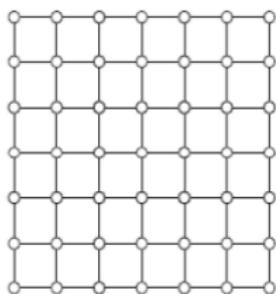


- Even for a single edge, mean field can give very wrong answers!
- Interestingly, once $\epsilon > 0.1$, mean field has a single maximum point at the uniform distribution (thus, exact)

Structured mean-field approximations

- Rather than assuming a fully-factored distribution for q , we can use a *structured* approximation, such as a spanning tree

True dist.



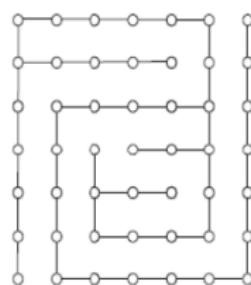
**Fully-factorized
mean field**



$$p(x) \propto \prod_c \phi_c(x_c)$$

$$q(x) \propto \prod_i q_i(x_i)$$

**Structured
mean field**



$$\dot{q}(x)$$

Summary

Key ideas:

- Variational approach: Inference as optimization.
- M-projections: ideal, but intractable
- I-projections: tractable if we use “simple distributions”
- Mean field provides lower bounds and marginal approximations

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 19, March 14, 2017

Today's plan

Plan for today

- Finish Variational Inference
- Geometrical Interpretation
- Understand why loopy belief propagation works so well in practice
- This is not part of the final exam

Recap: Variational methods

- **Goal:** Approximate difficult distribution $p(\mathbf{x} | \mathbf{e})$ with a new distribution $q(\mathbf{x})$ such that:
 - ① $p(\mathbf{x} | \mathbf{e})$ and $q(\mathbf{x})$ are “close” (**approximate**)
 - ② Computation on $q(\mathbf{x})$ is easy ($q(\mathbf{x})$ is **simple**)
- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

(measures the expected number of extra bits required to describe *samples from $p(\mathbf{x})$* using a code based on q instead of p)

- $D(p\|q) \geq 0$ for all p, q , with equality if and only if $p = q$
- Notice that KL-divergence is **asymmetric**

KL-divergence (see Section 8.5 of K&F)

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- Suppose p is the true distribution we wish to do inference with
- What is the difference between the solution to

$$\arg \min_q D(p\|q)$$

(called the *M-projection* of q onto p) and

$$\arg \min_q D(q\|p)$$

(called the *I-projection*)?

- These two will differ when q is minimized over a restricted set of probability distributions $Q = \{q_1, \dots\}$, and in particular when $p \notin Q$

Most variational inference algorithms make use of the l-projection

Overcomplete representation: pairwise MRF example

Let's assume $p(\mathbf{x})$ is a member of an exponential family (e.g., an MRF)

$$f(\mathbf{x}) = \begin{bmatrix} 1[x_1=0] \\ 1[x_1=1] \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{array}{l} \leftarrow \text{Assignment for } X_1 \\ \leftarrow \text{Assignment for } X_2 \\ \leftarrow \text{Assignment for } X_3 \\ \leftarrow \text{Edge assignment for } X_1 X_3 \\ \leftarrow \text{Edge assignment for } X_1 X_2 \\ \leftarrow \text{Edge assignment for } X_2 X_3 \end{array}$$

$X_1 = 0$
 $X_2 = 1$ $X_3 = 0$

For a discrete-variable MRF, $f(\mathbf{x})$ is simply the concatenation of indicator functions:

$$p(\mathbf{x}; \theta) = \exp \left(\sum_{\mathbf{c} \in C} \sum_{\bar{\mathbf{x}}_{\mathbf{c}}} \theta_{\mathbf{c}} (\bar{\mathbf{x}}_{\mathbf{c}}) 1(\mathbf{x}_{\mathbf{c}} = \bar{\mathbf{x}}_{\mathbf{c}}) - \ln Z(\theta) \right) = \exp \left(\theta^T f(\mathbf{x}) - \ln Z(\theta) \right)$$

Variational methods: rewriting in terms of moments

- Suppose that we have an arbitrary graphical model:

$$\begin{aligned} p(\mathbf{x}; \theta) &= \frac{1}{Z(\theta)} \prod_{\mathbf{c} \in C} \phi_c(\mathbf{x}_c) = \exp \left(\sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right) \\ &= \exp \left(\sum_{\mathbf{c} \in C} \sum_{\bar{\mathbf{x}}_c} \theta_c(\bar{\mathbf{x}}_c) \mathbf{1}(\mathbf{x}_c = \bar{\mathbf{x}}_c) - \ln Z(\theta) \right) = \exp \left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta) \right) \end{aligned}$$

- All of the approaches begin as follows:

$$\begin{aligned} D(q \| p) &= \sum_{\mathbf{x}} q(\mathbf{x}) \ln \frac{q(\mathbf{x})}{p(\mathbf{x})} = - \sum_{\mathbf{x}} q(\mathbf{x}) \ln p(\mathbf{x}) + \sum_{\mathbf{x}} q(\mathbf{x}) \ln q(\mathbf{x}) \\ &= - \sum_{\mathbf{x}} q(\mathbf{x}) (\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta)) - H(q(\mathbf{x})) \\ &= -\theta^T \sum_{\mathbf{x}} q(\mathbf{x}) \mathbf{f}(\mathbf{x}) + \ln Z(\theta) - H(q(\mathbf{x})) \\ &= - \sum_{\mathbf{x}} q(\mathbf{x}) \sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) + \ln Z(\theta) - H(q(\mathbf{x})) \\ &= - \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + \ln Z(\theta) - H(q(\mathbf{x})) \geq 0 \end{aligned}$$

Variational methods

- Suppose that we have an arbitrary graphical model:

$$p(\mathbf{x}; \theta) = \exp \left(\sum_{\mathbf{c} \in C} \theta_c(\mathbf{x}_c) - \ln Z(\theta) \right) = \exp \left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta) \right)$$

- Finding the *approximating distribution* $q(\mathbf{x}) \in Q$ that minimizes the I-projection to $p(\mathbf{x})$, i.e. $D(q||p) = \sum_{\mathbf{x}} q(\mathbf{x}) \ln \frac{q(\mathbf{x})}{p(\mathbf{x})}$, is equivalent to

$$\max_{q \in Q} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + H(q(\mathbf{x})) = \theta^T \sum_{\mathbf{x}} q(\mathbf{x}) \mathbf{f}(\mathbf{x}) + H(q(\mathbf{x}))$$

where $E_q[\theta_c(\mathbf{x}_c)] = \sum_{\mathbf{x}_c} q(\mathbf{x}_c) \theta_c(\mathbf{x}_c)$ and $H(q(\mathbf{x}))$ is the *entropy* of $q(\mathbf{x})$

- If $p \in Q$, the value of the objective at optimality is **equal to** $\ln Z(\theta)$
- How should we approximate this? We need a compact way of representing $q(\mathbf{x})$ and finding the maxima

Two types of variational algorithms: Mean-field and relaxation

$$\max_q \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + H(q(\mathbf{x})).$$

- Need some compact way of representing $q(\mathbf{x})$, and for which the objective function can be evaluated efficiently
- *Mean-field* algorithms assume a factored representation of the joint distribution:

$$q(\mathbf{x}) = \prod_{i \in V} q_i(x_i)$$

- *Relaxation* algorithms introduce some approximations to achieve tractability **[loopy Belief Propagation is an example of this!]**

Re-writing objective in terms of moments

- Assume that $p(\mathbf{x}; \theta) = \exp\left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta)\right)$ is in the exponential family, and let $\mathbf{f}(\mathbf{x})$ be its sufficient statistic vector

$$\begin{aligned}\ln Z(\theta) &= \max_q \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x}_c)] + H(q(\mathbf{x})) \\ &= \max_q \theta^T \sum_{\mathbf{x}} q(\mathbf{x}) \mathbf{f}(\mathbf{x}) + H(q(\mathbf{x}))\end{aligned}$$

- Define $\mu_q = E_q[\mathbf{f}(\mathbf{x})]$ be the *marginals (mean parameters)* of $q(\mathbf{x})$
- Next, instead of optimizing over distributions $q(\mathbf{x})$, optimize over valid marginal vectors μ . We obtain:

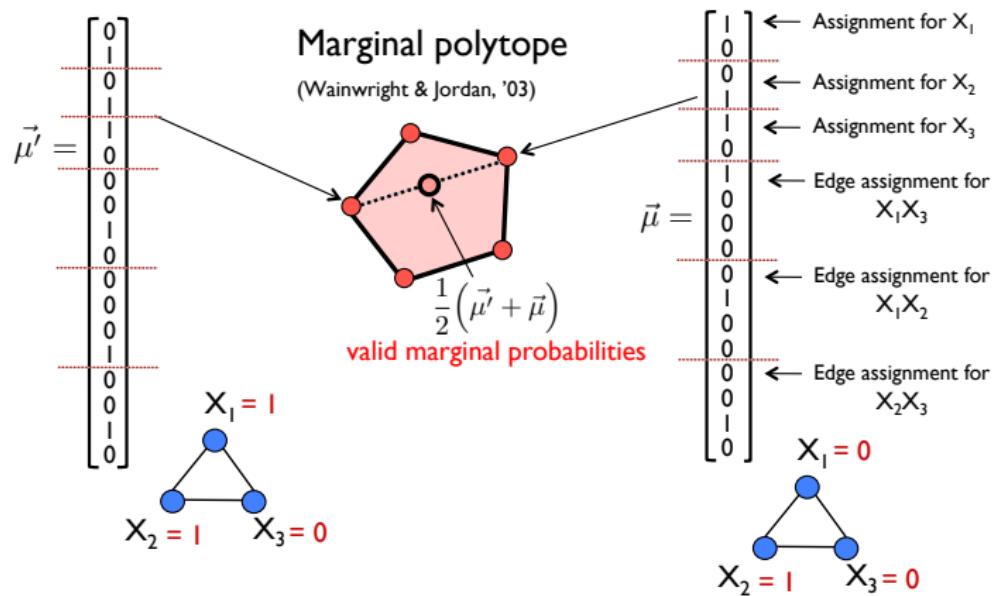
$$\ln Z(\theta) = \max_{\mu \in M} \max_{q: E_q[\mathbf{f}(\mathbf{x})] = \mu} \theta^T \mu + H(q(\mathbf{x}))$$

where M is the **marginal polytope** which contains all possible marginal vectors

Visualization of feasible μ vectors

For binary random variables, the marginal polytope M is given by

$$\begin{aligned} M &= \{\mu : \mu = \sum_{\mathbf{x}} q(\mathbf{x})\mathbf{f}(\mathbf{x}) \text{ for some } q(\mathbf{x}) \geq 0, \sum_{\mathbf{x}} q(\mathbf{x}) = 1\} \\ &= \text{convex-hull}(\{\mathbf{f}(\mathbf{x}), \mathbf{x} \in \{0, 1\}^n\}) \end{aligned}$$



Re-writing objective in terms of moments

- Assume that $p(\mathbf{x}; \theta) = \exp\left(\theta^T \mathbf{f}(\mathbf{x}) - \ln Z(\theta)\right)$ is in the exponential family, and let $\mathbf{f}(\mathbf{x})$ be its sufficient statistic vector
- We can write the variational problem in terms of moments in the **marginal polytope** M

$$\begin{aligned}\ln Z(\theta) &= \max_q \theta^T \sum_{\mathbf{x}} q(\mathbf{x}) \mathbf{f}(\mathbf{x}) + H(q(\mathbf{x})) \\ &= \max_{\mu \in M} \max_{q: E_q[\mathbf{f}(\mathbf{x})] = \mu} \theta^T \mu + H(q(\mathbf{x})) \\ &= \max_{\mu \in M} \theta^T \mu + \max_{q: E_q[\mathbf{f}(\mathbf{x})] = \mu} H(q(\mathbf{x})) \\ &= \max_{\mu \in M} \theta^T \mu + H(\mu)\end{aligned}$$

where

$$H(\mu) = \max_{q: E_q[\mathbf{f}(\mathbf{x})] = \mu} H(q(\mathbf{x}))$$

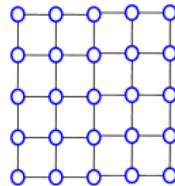
is the entropy of the maximum entropy distribution with marginals μ

Pairwise MRF Example

- Variational objective in general:

$$\begin{aligned}\ln Z(\theta) &= \max_{\mu \in M} \theta^T \mu + H(\mu) \\ &= \max_{\mu \in M} \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \mu_c(x_c) + \max_{q: E_q[f(x)] = \mu} H(q(x))\end{aligned}$$

- If $p(x; \theta) = \exp \left(\sum_{i \in V} \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j) - \ln Z(\theta) \right)$ is a pairwise MRF



$$\ln Z(\theta) = \max_{\mu \in M} \sum_{i \in V} \sum_{x_i} \theta_i(x_i) \mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j) \mu_{ij}(x_i, x_j) + H(\mu)$$

where $H(\mu)$ is the entropy of the maximum entropy distribution with marginals μ

Recall: Maximum Entropy Interpretation

- Given a function $f(x)$, suppose one seeks a distribution $q(x)$ such that

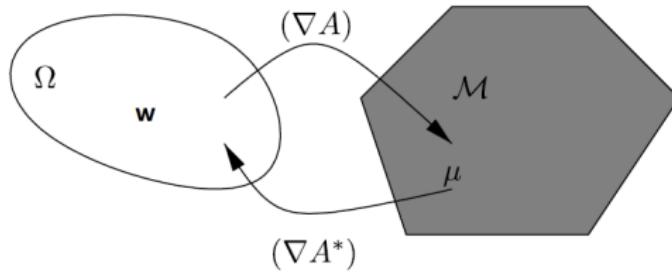
$$E_{q(x)}[f_i(x)] = \mu_i$$

and $q(x)$ has **maximum entropy**.

- The solution has exactly the form of the **exponential family** with sufficient statistic $f(x)$ (proof based on Lagrange multipliers)
- Theorem:** Exponential family distributions maximize the entropy $H(q)$ over all distributions satisfying

$$E_{q(x)}[f_i(x)] = \mu_i$$

- Involves the computation of the reverse mapping $(\nabla A)^{-1}(\mu)$



Relaxation

- Variational objective:

$$\begin{aligned}\ln Z(\theta) &= \max_{\mu \in M} \theta^T \mu + H(\mu) \\ &= \max_{\mu \in M} \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \mu_c(x_c) + H(\mu)\end{aligned}$$

- We still haven't achieved anything, because:
 - ① The marginal polytope M is complex to describe (in general, exponentially many vertices and facets)
 - ② $H(\mu)$ is very difficult to compute or optimize over (requires inference)
- We now make two approximations:
 - ① We replace M with a *relaxation* of the marginal polytope, e.g. the local consistency constraints M_L
 - ② We replace $H(\mu)$ with a function $\tilde{H}(\mu)$ which approximates $H(\mu)$

Local consistency constraints for pairwise MRF

- Force every “cluster” of variables to choose a “soft” local assignment:

$$\mu_i(x_i) \geq 0 \quad \forall i \in V, x_i$$

$$\sum_{x_i} \mu_i(x_i) = 1 \quad \forall i \in V$$

$$\mu_{ij}(x_i, x_j) \geq 0 \quad \forall ij \in E, x_i, x_j$$

$$\sum_{x_i, x_j} \mu_{ij}(x_i, x_j) = 1 \quad \forall ij \in E$$

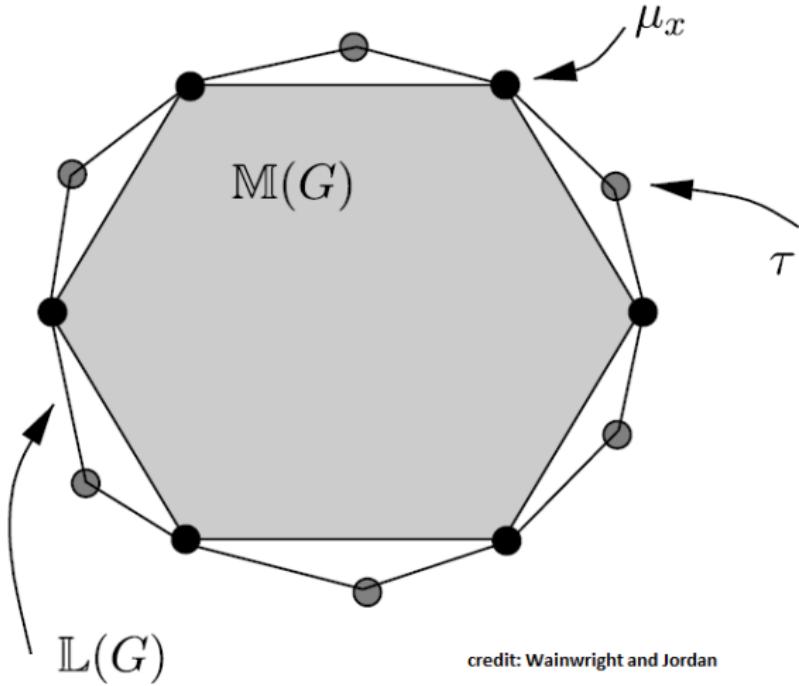
- Enforce that these local assignments are **locally consistent**:

$$\mu_i(x_i) = \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i$$

$$\mu_j(x_j) = \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j$$

- The *local consistency polytope*, M_L is defined by these constraints
- The *local consistency polytope* is equivalent to the marginal polytope if $p(x)$ is a tree structured graphical model!
 - Recall junction trees: local consistency implies global consistency

Marginal vs local consistency polytopes



Entropy of a tree distribution

- Suppose that p is a tree-structured distribution, so that we are optimizing only over marginals $\mu_{ij}(x_i, x_j)$ for $ij \in T$
- The solution to $\arg \max_{q: E_q[f(x)] = \mu} H(q)$ is a tree-structured MRF
- The entropy of q as a function of its marginals decomposes nicely

$$H(\mu) = \sum_{i \in V} H(\mu_i) - \sum_{ij \in T} I(\mu_{ij})$$

where

$$H(\mu_i) = - \sum_{x_i} \mu_i(x_i) \log \mu_i(x_i)$$

$$I(\mu_{ij}) = \sum_{x_i, x_j} \mu_{ij}(x_i, x_j) \log \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i) \mu_j(x_j)}$$

$H(\cdot)$ is the entropy and $I(\cdot)$ is the **mutual information**

Entropy of a tree distribution

- Recall that, if an undirected graph G has a junction tree, then the joint distribution can be expressed as

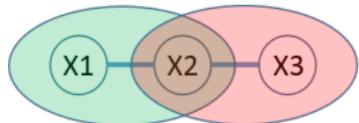
$$q(x) = \frac{\prod_{C \in \mathcal{C}} \beta_C(x_C)}{\prod_{S \in \mathcal{S}} \beta_S(x_S)}$$

where C is the set of cliques and S the set of separators.

- If G is a tree, we can write

$$q(x) = \prod_i \mu_i(x_i) \prod_{ij \in T} \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)}$$

- Example:



$$\begin{aligned} q(x_1, x_2, x_3) &= \frac{\mu_{12}(x_1, x_2)\mu_{23}(x_2, x_3)}{\mu_2(x_2)} \\ &= \mu_1(x_1)\mu_2(x_2)\mu_3(x_3) \frac{\mu_{12}(x_1, x_2)\mu_{23}(x_2, x_3)}{\mu_1(x_1)\mu_2(x_2)\mu_2(x_2)\mu_3(x_3)} \end{aligned}$$

Entropy of a tree distribution

- If G is a tree, we can write

$$q(x) = \prod_i \mu_i(x_i) \prod_{ij \in T} \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)}$$

- Therefore

$$\begin{aligned} H(q) &= -\sum_x q(x) \log q(x) \\ &= -\sum_x q(x) \left(\sum_i \log \mu_i(x_i) + \sum_{ij \in T} \log \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)} \right) \\ &= -\sum_i \sum_{x_i} q(x_i) \log \mu_i(x_i) - \sum_{ij \in T} \sum_{x_i, x_j} q(x_i, x_j) \log \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)} \\ &= \sum_{i \in V} H(\mu_i) - \sum_{ij \in T} I(\mu_{ij}) = H_{\text{bethe}}(\mu) \end{aligned}$$

- Can we use this for non-tree structured models?

Bethe-free energy approximation

- The Bethe entropy approximation is (for any graph)

$$H_{\text{bethe}}(\mu) = \sum_{i \in V} H(\mu_i) - \sum_{ij \in E} I(\mu_{ij})$$

- This gives the following variational approximation:

$$\max_{\mu \in M_L} \sum_{\mathbf{c} \in C} \sum_{\mathbf{x}_{\mathbf{c}}} \theta_c(\mathbf{x}_{\mathbf{c}}) \mu_c(\mathbf{x}_{\mathbf{c}}) + H_{\text{bethe}}(\mu)$$

- For non tree-structured models this is not concave, and is hard to maximize
- Loopy belief propagation, if it converges, finds a stationary point of this objective function (local extrema or saddle point)!

Bethe-free energy approximation

- Bethe approximation: use entropy expression derived for trees, but for loopy graphs
- For a tree-structure graphical model, entropy approximation becomes exact, and unique solution gives true marginals
- For general graphs, there is a correspondence between **fixed points** of the loopy belief propagation algorithm and **stationary points** of the Bethe variational objective
- **Theorem:** If Loopy BP converges, resulting estimated marginals are stationary points (usually local maxima) of the variational objective (Bethe free energy)!

Naive mean-field

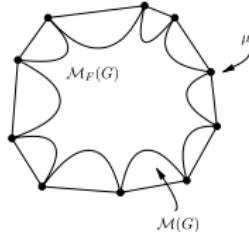
- Recall mean field variational objective:

$$(*) \max_q \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \prod_{i \in c} q_i(x_i) + \sum_{i \in V} H(q_i)$$

subject to the constraints

$$q_i(x_i) \geq 0 \quad \forall i \in V, x_i \in \text{Val}(X_i), \quad \sum_{x_i \in \text{Val}(X_i)} q_i(x_i) = 1 \quad \forall i \in V$$

- Corresponds to optimizing over an *inner bound* $M_F(G)$ on the marginal polytope M , given by $\mu_{ij}(x_i, x_j) = \mu_i(x_i)\mu_j(x_j) = q_i(x_i)q_j(x_j)$ and the above constraints:



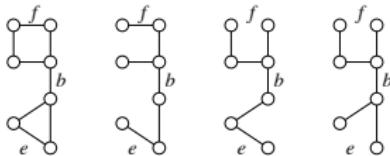
- We obtain a *lower bound* on the partition function, i.e. $(*) \leq \ln Z(\theta)$

Concave relaxation

- Let $\tilde{H}(\mu)$ be an *upper bound* on $H(\mu)$, i.e. $H(\mu) \leq \tilde{H}(\mu)$
- We can obtain the following **upper bound** on the log-partition function:

$$\begin{aligned}\ln Z(\theta) &= \max_{\mu \in M} \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \mu_c(x_c) + H(\mu) \\ &\leq \max_{\mu \in M_L} \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \mu_c(x_c) + H(\mu) \\ &\leq \max_{\mu \in M_L} \sum_{c \in C} \sum_{x_c} \theta_c(x_c) \mu_c(x_c) + \tilde{H}(\mu)\end{aligned}$$

- An example of a **concave** entropy upper bound is the **tree-reweighted** approximation (Jaakkola, Wainwright, & Wilsky, '05), given by specifying a distribution over spanning trees of the graph



Summary

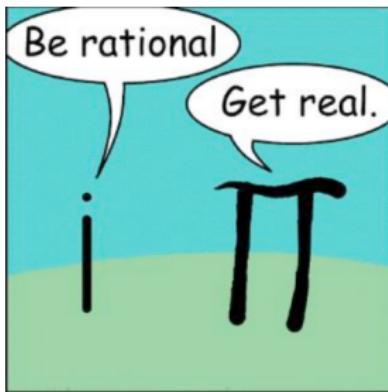
- Key ideas:
 - Variational approach: Inference as optimization.
 - Mean field: approximate marginal polytope M with non-convex inner bound M_F , updates to find stationary points of the objective
 - Loopy belief propagation: approximate marginal polytope M with outer tree-based M_L , approximate entropy with $H_{\text{bethe}}(\vec{\mu})$, updates to find stationary points of the objective
- Active research areas:
 - Alternative, more accurate approximations
 - Alternative, stable algorithms for solving the variational optimization problem

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 2, January 12, 2017



Announcements

- Please sign up on Piazza
- Please sign up on Gradescope
- First Homework released on Tuesday. Due January 24 at midnight (Tuesday)

Example of joint distribution

- Student's Intelligence I . $\text{Val}(I) = \{i_0 \text{ (high)}, i_1 \text{ (very high)}\}$
- Exam's Difficulty D . $\text{Val}(D) = \{d_0 \text{ (easy)}, d_1 \text{ (hard)}\}$
- Student's Grade G . $\text{Val}(G) = \{g_1 \text{ (A)}, g_2 \text{ (B)}, g_3 \text{ (C)}\}$

I	D	G	Prob.
i^0	d^0	g^1	0.126
i^0	d^0	g^2	0.168
i^0	d^0	g^3	0.126
i^0	d^1	g^1	0.009
i^0	d^1	g^2	0.045
i^0	d^1	g^3	0.126
i^1	d^0	g^1	0.252
i^1	d^0	g^2	0.0224
i^1	d^0	g^3	0.0056
i^1	d^1	g^1	0.06
i^1	d^1	g^2	0.036
i^1	d^1	g^3	0.024

How many parameters do we need to specify the joint distribution?

$$2 * 2 * 3 - 1$$

Example of joint distribution

- Suppose X_1, \dots, X_n are binary (Bernoulli) random variables, i.e., $\text{Val}(X_i) = \{0, 1\}$.
- How many possible states?

$$\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ times}} = 2^n$$

- How many parameters to specify the joint distribution $p(x_1, \dots, x_n)$?

$$2^n - 1$$

Structure through independence

- If X_1, \dots, X_n are independent, then

$$p(x_1, \dots, x_n) = p(x_1)p(x_2) \cdots p(x_n)$$

- How many possible states? 2^n
- How many parameters to specify the joint distribution $p(x_1, \dots, x_n)$?
 - How many to specify the marginal distribution $p(x_1)$? 1
- **2^n entries can be described by just n numbers** (if $|\text{Val}(X_i)| = 2$)!
- Independence assumption is too strong. Model not likely to be useful

Key notion: conditional independence

- Two events A, B are conditionally independent given event C if

$$p(A \cap B|C) = p(A|C)p(B|C)$$

- Random variables X, Y are conditionally independent given Z if for all values $x \in \text{Val}(X)$, $y \in \text{Val}(Y)$, $z \in \text{Val}(Z)$

$$p(X = x \cap Y = y | Z = z) = p(X = x | Z = z)p(Y = y | Z = z)$$

- Equivalent: $p(X|Y, Z) = p(X|Z)$. Note the more compact notation
- We write $X \perp Y | Z$
- Similarly for sets of random variables, $\mathbf{X} \perp \mathbf{Y} | \mathbf{Z}$

Properties of conditional independence

- Symmetry: $X \perp Y | Z \Rightarrow Y \perp X | Z$
 - Height \perp Reading Ability | Age \Rightarrow Reading Ability \perp Height | Age
- Decomposition: $X \perp Y, W | Z \Rightarrow X \perp Y | Z$
 - Height \perp (Read Ab., Write Ab.) | Age \Rightarrow Height \perp Read Ab. | Age
- Contraction: $(X \perp Y | Z) \wedge (X \perp W | Y, Z) \Rightarrow X \perp Y, W | Z$
 - Height \perp Read | Age \wedge Height \perp Write | (Age, Read) \Rightarrow Height \perp (Read, Write) | Age
- Weak union: $X \perp Y, W | Z \Rightarrow X \perp Y | Z, W$
 - Height \perp (Read, Write) | Age \Rightarrow Height \perp Read | (Age, Write)
- Intersection, if $p(\cdot) > 0$:
$$(X \perp Y | Z, W) \wedge (X \perp W | Y, Z) \Rightarrow X \perp Y, W | Z$$

Two important rules

- ① **Chain rule** Let S_1, \dots, S_n be events, $p(S_i) > 0$.

$$p(S_1 \cap S_2 \cap \dots \cap S_n) = p(S_1)p(S_2 | S_1) \cdots p(S_n | S_1 \cap \dots \cap S_{n-1})$$

Structure through conditional independence

- If X_1, \dots, X_n are *conditionally independent* given Y , denoted as $X_i \perp X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n \mid Y$, then using Chain Rule

$$\begin{aligned} p(y, x_1, \dots, x_n) &= p(y)p(x_1 \mid y) \prod_{i=2}^n p(x_i \mid x_1, \dots, x_{i-1}, y) \\ &= p(y)p(x_1 \mid y) \prod_{i=2}^n p(x_i \mid y) \\ &= p(y) \prod_{i=1}^n p(x_i \mid y). \end{aligned}$$

- How many parameters? $2n + 1$
- Simple yet *powerful* model (Naive Bayes)

Example: naive Bayes for classification

- Classify e-mails as spam ($Y = 1$) or not spam ($Y = 0$)
 - Let $1 : n$ index the words in our vocabulary (e.g., English)
 - $X_i = 1$ if word i appears in an e-mail, and 0 otherwise
 - E-mails are drawn according to some distribution $p(Y, X_1, \dots, X_n)$
- Suppose that the words are conditionally independent given Y . Then,

$$p(y, x_1, \dots, x_n) = p(y) \prod_{i=1}^n p(x_i | y)$$

Estimate parameters from training data. **Predict** with:

$$p(Y = 1 | x_1, \dots, x_n) = \frac{p(Y = 1) \prod_{i=1}^n p(x_i | Y = 1)}{\sum_{y=\{0,1\}} p(Y = y) \prod_{i=1}^n p(x_i | Y = y)}$$

- Are the independence assumptions made here reasonable?
- Philosophy: Nearly all probabilistic models are “wrong”, but many are nonetheless useful

General Idea

- Use conditional parameterization (instead of joint parameterization)
- For each random variable X_i , specify $p(x_i|\mathbf{x}_{\mathbf{A}_i})$ for set $\mathbf{X}_{\mathbf{A}_i}$ of random variables
- Then use chain rule to get joint parametrization

$$p(x_1, \dots, x_n) = \prod_i p(x_i | \mathbf{x}_{\mathbf{A}_i})$$

- Need to guarantee it is a *legal* probability distribution.
- Use conditional parameterization (instead of joint parameterization). What constraints does a legal probability distribution satisfy?
 - Non negative
 - Sum to one
 - Consistency. For example, $p(x|y)$ and $p(y|x)$ must be consistent

Bayesian networks

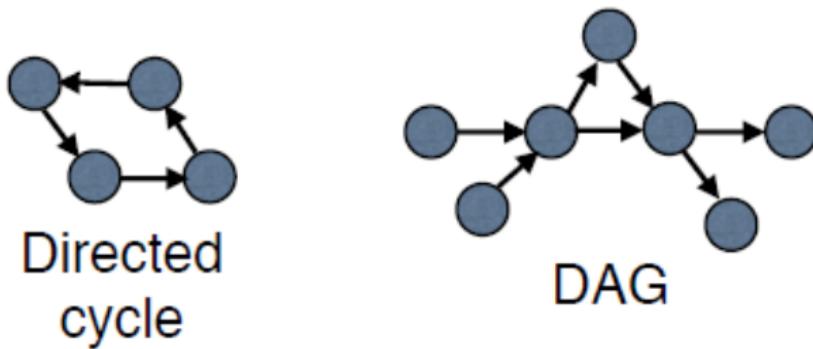
Reference: Chapter 3

- 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, $p(x_i | \mathbf{x}_{\text{Pa}(i)})$, specifying the variable's probability conditioned on its parents' values
- Graph $G = (V, E)$ is called the structure of the Bayesian Network
- Defines a joint distribution:

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

- Claim: $p(x_1, \dots, x_n)$ is a valid probability distribution
- **Economical representation:** exponential in $|\text{Pa}(i)|$, not $|V|$

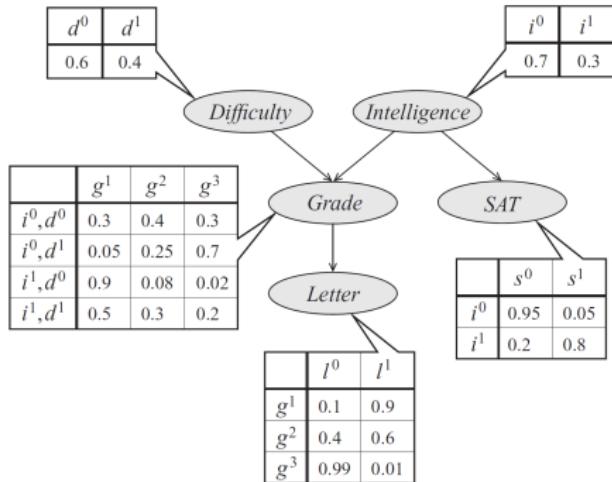
Example



DAG stands for Directed Acyclic Graph

Example

- Consider the following Bayesian network:



- What is its joint distribution?

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

$$p(d, i, g, s, l) = p(d)p(i)p(g | i, d)p(s | i)p(l | g)$$

Example

- Let $q(d, i, g, s, l) = p(d)p(i)p(g | i, d)p(s | i)p(l | g)$. Claim: q is a valid probability distribution.
- Non-negativeness is obvious
- Sums to one:

$$\begin{aligned} \sum_{l \in \{l^0, l^1\}} \sum_{s \in \{s^0, s^1\}} \sum_{g \in \{g^1, g^2, g^3\}} \sum_{i \in \{i^0, i^1\}} \sum_{d \in \{d^0, d^1\}} q(d, i, g, s, l) &= \sum_{l, s, g, i, d} q(d, i, g, s, l) \\ &= \sum_{d, i, g, s, l} q(d, i, g, s, l) = \sum_{d, i, g, s, l} p(d)p(i)p(g | i, d)p(s | i)p(l | g) \end{aligned}$$

Distributive Law

- Two variables and a constant:

$$ax_1 + ax_2 = a(x_1 + x_2)$$

- Many variables and a constant:

$$\sum_i ax_i = a \sum_i x_i$$

- More variables and constants:

$$\sum_{i \in \{1,2\}} \sum_{j \in \{1,2\}} a_i x_i b_j y_j = a_1 x_1 b_1 y_1 + a_1 x_1 b_2 y_2 + a_2 x_2 b_1 y_1 + a_2 x_2 b_2 y_2$$

$$= a_1 x_1 (b_1 y_1 + b_2 y_2) + a_2 x_2 (b_1 y_1 + b_2 y_2) = \sum_{i \in \{1,2\}} a_i x_i \sum_{j \in \{1,2\}} b_j y_j$$

Example

- Let $q(d, i, g, s, l) = p(d)p(i)p(g | i, d)p(s | i)p(l | g)$. Claim: q is a valid probability distribution.
- Non-negativeness is obvious
- Sums to one:

$$\begin{aligned}\sum_{d,i,g,s,l} q(d, i, g, s, l) &= \sum_{d,i,g,s,l} p(d)p(i)p(g | i, d)p(s | i)p(l | g) \\&= \sum_{d,i,g,s} p(d)p(i)p(g | i, d)p(s | i) \sum_l p(l | g) \\&= \sum_{d,i,g,s} p(d)p(i)p(g | i, d)p(s | i) \underbrace{\sum_l p(l | g)}_{=1} \\&= \sum_d p(d) \sum_i p(i) \sum_g p(g | i, d) \sum_s p(s | i) \\&= 1\end{aligned}$$

Example

- Let $q(d, i, g, s, l) = p(d)p(i)p(g | i, d)p(s | i)p(l | g)$.
- Marginal conditional probabilities of p and q match

$$\begin{aligned} q(d) &= \sum_{i,g,s,l} q(d, i, g, s, l) \\ &= \sum_{i,g,s,l} p(d)p(i)p(g | i, d)p(s | i)p(l | g) \\ &= \sum_{i,g,s} p(d)p(i)p(g | i, d)p(s | i) \sum_l p(l | g) \\ &= p(d) \sum_i p(i) \sum_g p(g | i, d) \sum_s p(s | i) \\ &= p(d) \end{aligned}$$

Example

- Let $q(d, i, g, s, l) = p(d)p(i)p(g | i, d)p(s | i)p(l | g)$.
- Marginal conditional probabilities of p and q match

$$\begin{aligned} q(g | i, d) &= \frac{q(g, i, d)}{q(i, d)} = \frac{\sum_{s,l} q(d, i, g, s, l)}{\sum_{s,l,g} q(d, i, g, s, l)} \\ &= \frac{p(d)p(i)p(g | i, d)}{p(d)p(i)} \\ &= p(g | i, d) \end{aligned}$$

MRI Imaging

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

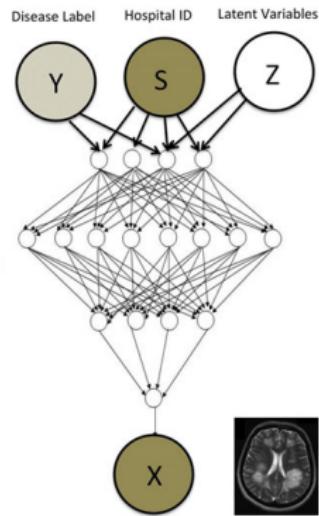


Image credit: Max Welling

More examples

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

What is the differential diagnosis?

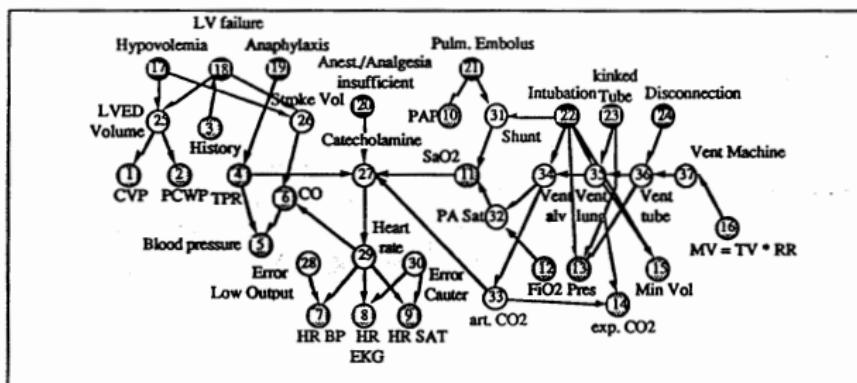


Fig. 1 The ALARM network representing causal relationships is shown with diagnostic (●), intermediate (○) and measurement (◎) nodes. LVED volume: left ventricular end-diastolic volume, LV failure: left ventricular failure, MV: minute ventilation, PA Sat: pulmonary artery oxygen saturation, PAP: pulmonary artery pressure, PCWP: pulmonary capillary wedge pressure, Pres: breathing pressure, RR: respiratory rate, TPR: total peripheral resistance, TV: tidal volume

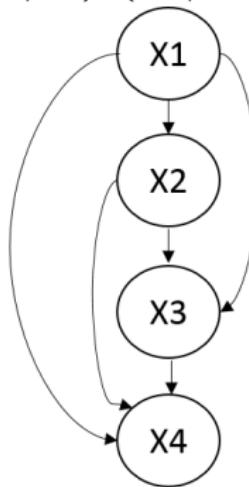
Beinlich et al., The ALARM Monitoring System, 1989

Bayesian networks

- Can every probability distribution be described by a BN?
- Using Chain Rule

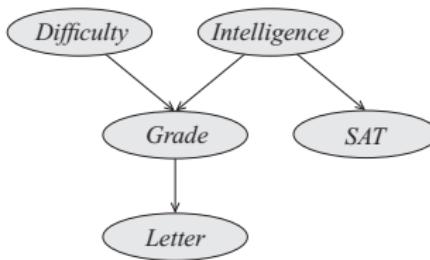
$$p(x_1, \dots, x_n) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_n | x_1, \dots, x_{n-1})$$

$$\bullet p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2)p(x_4 | x_1, x_2, x_3)$$



- No free lunch: high degree nodes, representation not economical

Bayesian network structure implies conditional independencies!



- The joint distribution corresponding to the above BN factors as

$$p(d, i, g, s, l) = p(d)p(i)p(g | i, d)p(s | i)p(l | g)$$

- However, by the chain rule, *any* distribution can be written as

$$p(d, i, g, s, l) = p(d)p(i | d)p(g | i, d)p(s | i, d, g)p(l | g, d, i, s)$$

- Thus, we are assuming the following additional independencies:
 $D \perp I$, $S \perp \{D, G\} | I$, $L \perp \{I, D, S\} | G$.

Proof

- If the joint distribution factors as

$$p(d, i, g, s, l) = p(d)p(i)p(g \mid i, d)p(s \mid i)p(l \mid g)$$

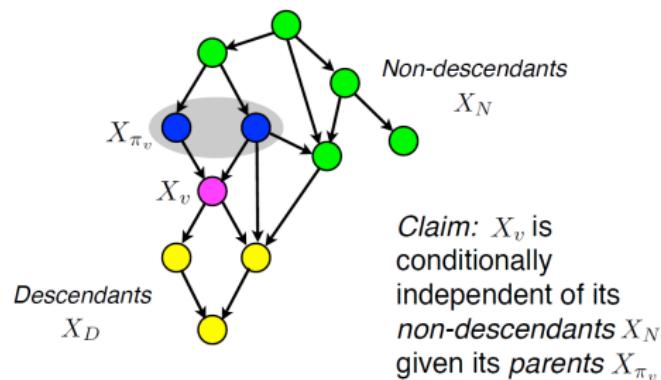
then $D \perp I$.

$$\begin{aligned} p(d, i) &= \sum_{g,s,l} p(d, i, g, s, l) = \sum_{g,s,l} p(d)p(i)p(g \mid i, d)p(s \mid i)p(l \mid g) \\ &= p(d)p(i) \sum_{g,s,l} p(g \mid i, d)p(s \mid i)p(l \mid g) \\ &= p(d)p(i) \end{aligned}$$

Bayesian network structure implies conditional independencies!

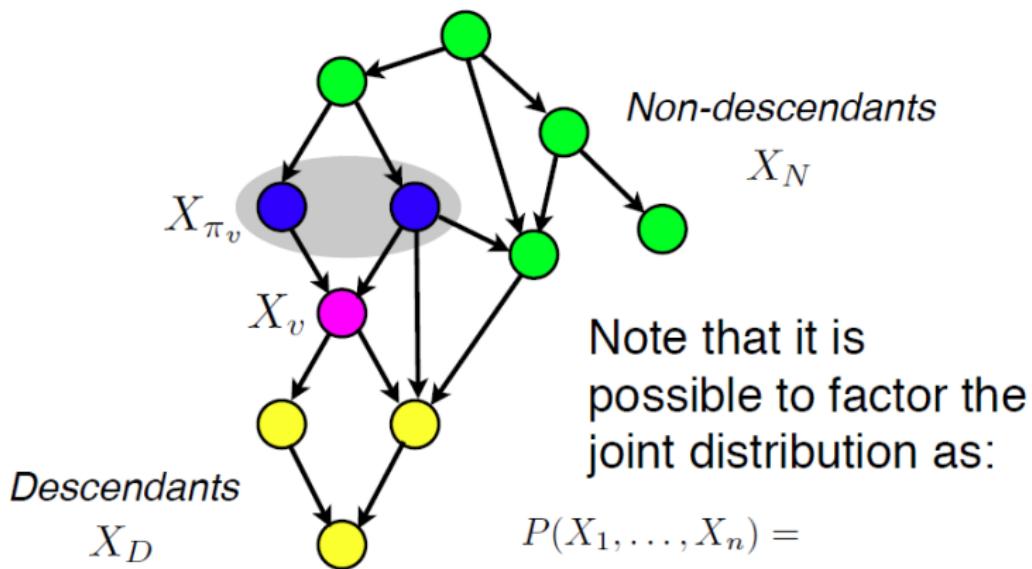
- A Bayesian network structure G implies the following conditional independence assumptions:
 - For each variable X_v , X_v is independent from its non-descendants given its parents

$$X_v \perp \text{NonDescendants}_{X_v} \mid \text{Parents}_{X_v}$$



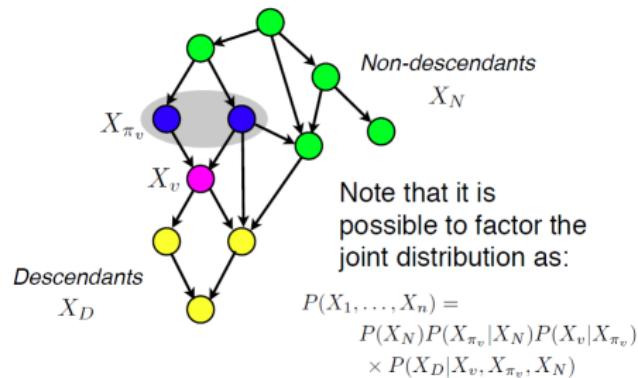
- Called *local independencies*

Bayesian network structure implies conditional independencies!



$$\begin{aligned} P(X_1, \dots, X_n) = \\ P(X_N)P(X_{\pi_v}|X_N)P(X_v|X_{\pi_v}) \\ \times P(X_D|X_v, X_{\pi_v}, X_N) \end{aligned}$$

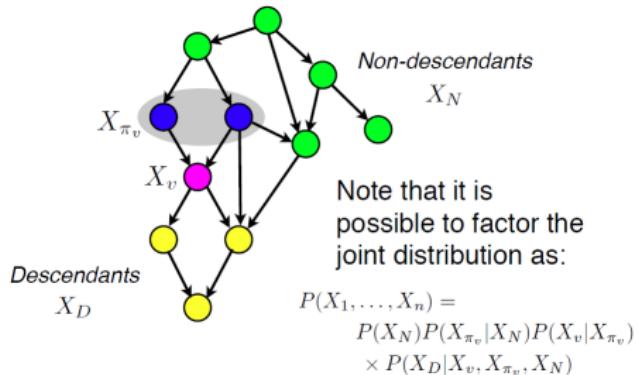
Bayesian network structure implies conditional independencies!



Claim: $X_v \perp \text{NonDescendants}_{X_v} \mid \text{Parents}_{X_v}$. Proof:

$$p(x_v \mid x_{\pi_v}, x_N) = \frac{p(x_v, x_{\pi_v}, x_N)}{p(x_{\pi_v}, x_N)} =$$
$$\frac{\sum_{x_D} p(x_N)p(x_{\pi_v} \mid x_N)p(x_v \mid x_{\pi_v})p(x_D \mid x_v, x_{\pi_v}, x_N)}{\sum_{x_D} \sum_{x_v} p(x_N)p(x_{\pi_v} \mid x_N)p(x_v \mid x_{\pi_v})p(x_D \mid x_v, x_{\pi_v}, x_N)} = p(x_v \mid x_{\pi_v})$$

Conditional independence implies factorization



$X_V \perp \text{NonDescendants}_{X_V} | \text{Parents}_{X_V}$ \Rightarrow Factorization. Proof sketch:
Let P such that it satisfies *local independencies* over directed acyclic graph G . Use chain rule based on *topological variable ordering*:

$$p(x_1, \dots, x_n) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_n | x_1, \dots, x_{n-1})$$

Simplify the formula using conditional independency assumptions.

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 20, March 16, 2017

Announcement

Materials covered this week are **not** part of the final exam.

Plan for today

- ① Revisit EM
- ② Variational EM
- ③ Learning deep generative models

Partially observed data

- Suppose that our joint distribution is

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

where our samples \mathbf{X} are observed and the variables \mathbf{Z} are never observed in \mathcal{D}

- Maximum likelihood learning:

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) \\ &= \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)\end{aligned}$$

- Because of the sum over \mathbf{z} , there is no longer a closed-form solution for the maximum likelihood estimate θ^* (even in the case of Bayesian networks)

The EM Algorithm

- Let $p(\mathbf{z}|\mathbf{x}; \theta)$ be the true posterior distribution
- Suppose $q(\mathbf{z})$ is a probability distribution over the hidden variables

$$\begin{aligned} D_{KL}(q(\mathbf{z}) \| p(\mathbf{z}|\mathbf{x}; \theta)) &= \sum_{\mathbf{z}} q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x}; \theta)} \\ &= - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}|\mathbf{x}; \theta) - H(q) \\ &= - \sum_{\mathbf{z}} q(\mathbf{z}) \log \frac{p(\mathbf{z}, \mathbf{x}; \theta)}{p(\mathbf{x}; \theta)} - H(q) \\ &= - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{x}; \theta) - H(q) \\ &= - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \log p(\mathbf{x}; \theta) - H(q) \geq 0 \end{aligned}$$

The EM Algorithm

- Suppose $q(\mathbf{z})$ is a probability distribution over the hidden variables

$$D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x}; \theta)) = - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \log p(\mathbf{x}; \theta) - H(q) \geq 0$$

- Evidence lower bound holds for any q

$$\log p(\mathbf{x}; \theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

- Equality holds if $q = p(\mathbf{z}|\mathbf{x}; \theta)$

$$\log p(\mathbf{x}; \theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

- This is what we compute in the E-step of the EM algorithm

The EM Algorithm

- $q(\mathbf{z})$ is an arbitrary probability distribution over \mathbf{z}

$$D_{KL}(q \parallel p(\mathbf{z}|\mathbf{x}; \theta)) = - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \log p(\mathbf{x}; \theta) - H(q)$$

- Re-arranging can rewrite as

$$H(q) + \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) = \log p(\mathbf{x}; \theta) - D_{KL}(q \parallel p(\mathbf{z}|\mathbf{x}; \theta)) = F[q, \theta]$$

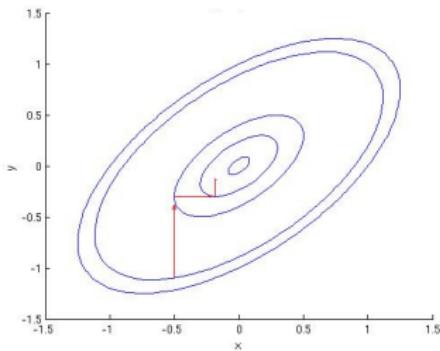
- Can interpret EM as coordinate ascent on $F[q, \theta]$

- ① Initialize $\theta^{(0)}$
- ② $q^{(0)} = \arg \max_q F[q, \theta^{(0)}] = p(\mathbf{z}|\mathbf{x}; \theta^{(0)})$
- ③ $\theta^{(1)} = \arg \max_{\theta} F[q^{(0)}, \theta] = \arg \max_{\theta} \sum_{\mathbf{z}} q^{(0)}(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta)$
- ④ $q^{(1)} = \arg \max_q F[q, \theta^{(1)}]$
- ⑤ ...

- Marginal likelihood never decreases

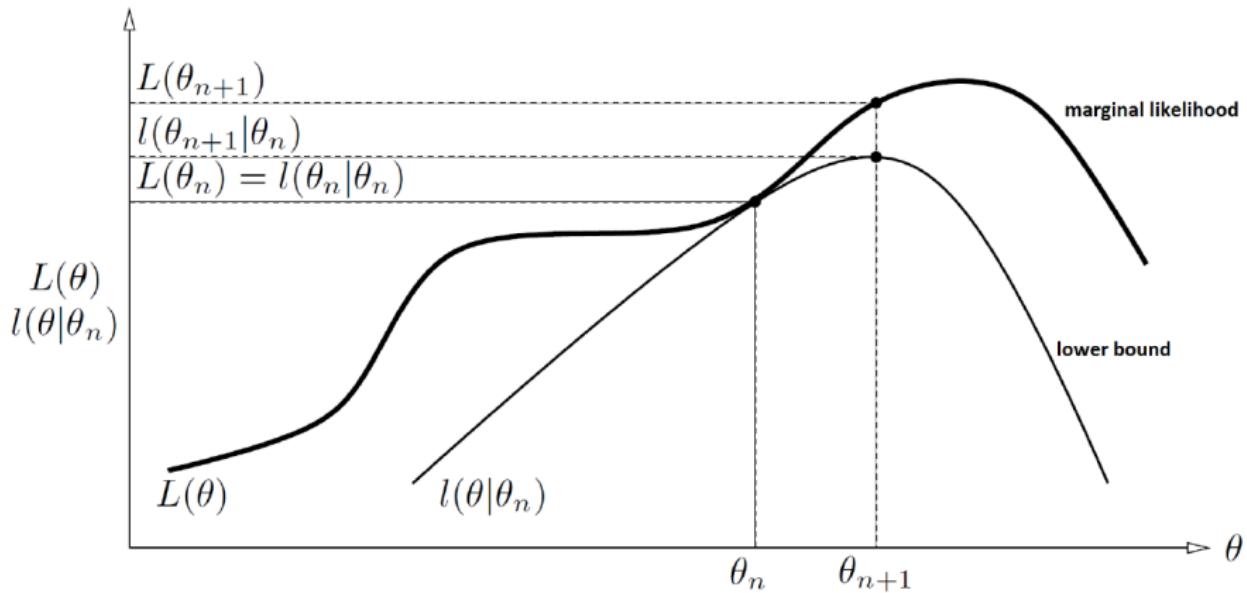
$$\begin{aligned} \log p(\mathbf{x}; \theta^{(0)}) &= F[q^{(0)}, \theta^{(0)}] \leq F[q^{(0)}, \theta^{(1)}] &=& \log p(\mathbf{x}; \theta^{(1)}) - D_{KL}(q^{(0)} \parallel p(\mathbf{z}|\mathbf{x}; \theta^{(1)})) \\ &&\leq& \log p(\mathbf{x}; \theta^{(1)}) \end{aligned}$$

Coordinate ascent



- Maximize along one direction at a time:
 - Initialize x_0, y_0
 - $y_1 = \arg \max_y f(x_0, y)$
 - $x_1 = \arg \max_x f(x, y_1)$
 - $y_2 = \arg \max_y f(x_1, y)$
 - ...
- Objective keeps improving.
$$f(x_0, y_0) \leq f(x_0, y_1) \leq f(x_1, y_1) \leq f(x_1, y_2) \leq \dots$$

Derivation of EM algorithm



(Figure adapted from tutorial by Sean Borman)

The Evidence Lower bound

- What if the posterior $p(\mathbf{z}|\mathbf{x}; \theta)$ is intractable to compute in the E-step?
- Suppose $q(\mathbf{z}|\mathbf{x}, \phi)$ is a (tractable) probability distribution over the hidden variables parameterized by ϕ
 - For example, a fully factored probability distribution (mean field)

$$D_{KL}(q||p(\mathbf{z}|\mathbf{x}; \theta)) = - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \log p(\mathbf{x}; \theta) - H(q) \geq 0$$

- Evidence lower bound (ELBO) holds for any $q(\mathbf{z}|\mathbf{x}, \phi)$

$$\begin{aligned} \log p(\mathbf{x}; \theta) &\geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}, \phi) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q(\mathbf{z}|\mathbf{x}, \phi)) = \underbrace{\mathcal{L}(\mathbf{x}; \theta, \phi)}_{\text{ELBO}} \\ &= \mathcal{L}(\mathbf{x}; \theta, \phi) + D_{KL}(q||p(\mathbf{z}|\mathbf{x}; \theta)) \end{aligned}$$

- The better $q(\mathbf{z}|\mathbf{x}, \phi)$ can approximate the posterior $p(\mathbf{z}|\mathbf{x}; \theta)$, the smaller $D_{KL}(q||p(\mathbf{z}|\mathbf{x}; \theta))$ we can achieve, the closer ELBO will be to $\log p(\mathbf{x}; \theta)$

The Evidence Lower bound

- Evidence lower bound (ELBO) holds for any $q(\mathbf{z}|\mathbf{x}, \phi)$

$$\log p(\mathbf{x}; \theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}, \phi) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q(\mathbf{z}|\mathbf{x}, \phi)) = \underbrace{\mathcal{L}(\mathbf{x}; \theta, \phi)}_{\text{ELBO}}$$

- Maximum likelihood learning (over the entire dataset):

$$\ell(\theta; \mathcal{D}) = \sum_{\mathbf{x}^i \in \mathcal{D}} \log p(\mathbf{x}^i; \theta) \geq \sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi^i)$$

- Therefore

$$\max_{\theta} \ell(\theta; \mathcal{D}) \geq \max_{\theta, \phi^1, \dots, \phi^M} \sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi^i)$$

- Note that we use a different ϕ^i for every data point

The Variational EM Algorithm

- $q_{\phi^i} = q(\mathbf{z}|\mathbf{x}^i, \phi^i)$ is an arbitrary probability distribution over \mathbf{z}

$$D_{KL}(q_{\phi^i} \| p(\mathbf{z}|\mathbf{x}^i; \theta)) = - \sum_{\mathbf{z}} q_{\phi^i}(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}^i; \theta) + \log p(\mathbf{x}^i; \theta) - H(q_{\phi^i})$$

- Re-arranging can rewrite as

$$H(q_{\phi^i}) + \sum_{\mathbf{z}} q_{\phi^i}(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}^i; \theta) = \log p(\mathbf{x}^i; \theta) - D_{KL}(q_{\phi^i} \| p(\mathbf{z}|\mathbf{x}^i; \theta)) = F^i[\phi^i, \theta]$$

- Summing over all data points in \mathcal{D}

$$\sum_i H(q_{\phi^i}) + \sum_{\mathbf{z}} q_{\phi^i}(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}^i; \theta) = \sum_i \log p(\mathbf{x}^i; \theta) - D_{KL}(q_{\phi^i} \| p(\mathbf{z}|\mathbf{x}^i; \theta)) = \sum_i F^i[\phi^i, \theta]$$

- Variational EM as coordinate ascent on $F[\phi^1, \dots, \phi^M, \theta] = \sum_i F^i[\phi^i, \theta]$

① Initialize $\theta^{(0)}$

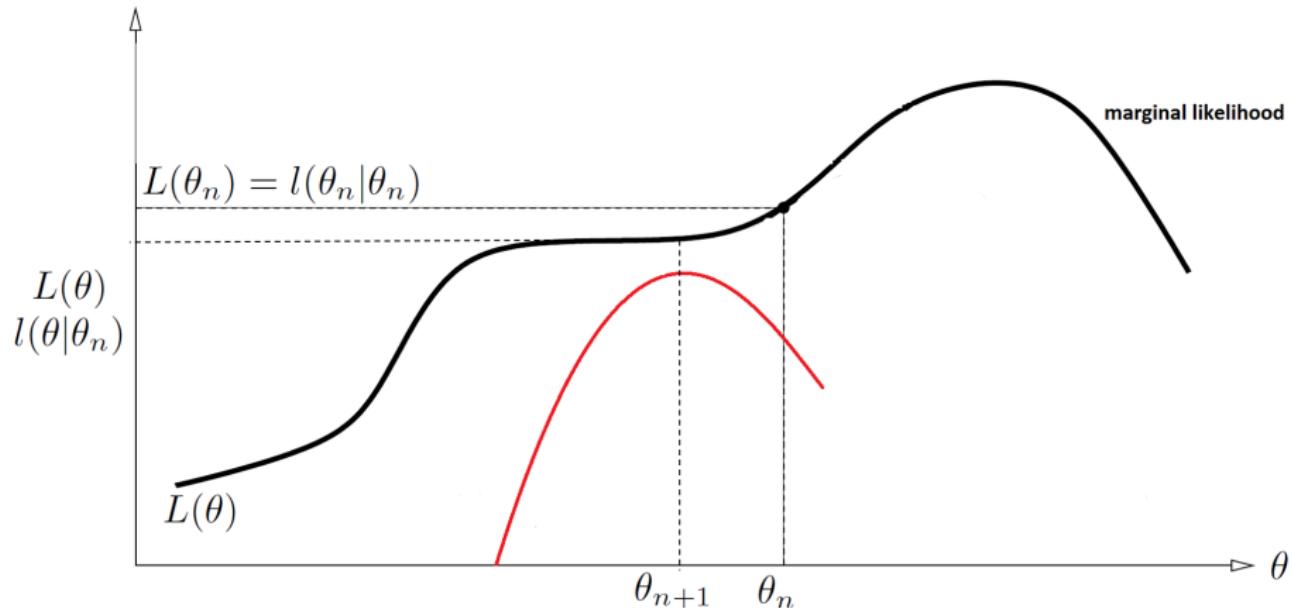
② $\phi^1 = \arg \max_q F^1[\phi^1, \theta^{(0)}] \neq p(\mathbf{z}|\mathbf{x}^1; \theta^{(0)})$ in general

③ $\phi^2 = \arg \max_q F^2[\phi^2, \theta^{(0)}] \neq p(\mathbf{z}|\mathbf{x}^2; \theta^{(0)})$ in general

④ $\theta^{(1)} = \arg \max_{\theta} \sum_i F^i[\phi^i, \theta]$

- Unlike EM, variational EM not guaranteed to reach a local maximum.
Marginal likelihood might decrease!

Potential issues with Variational EM algorithm



(Figure adapted from tutorial by Sean Borman)

Generative models



- Generative models can be very expressive:
 - ➊ Sample z from $p(z)$
 - ➋ Sample x from $p(x|z)$
- z is unobserved
- Let θ denote the parameters of this model (both the prior and the conditional)
- $p(x|z)$ could be very complex and non-linear (big neural net). We assume $p(z|x)$ is intractable

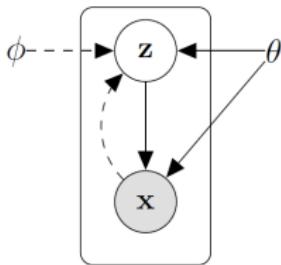
Learning Deep Generative models



- In variational EM, we had a set of variational parameters ϕ^i for each data point
- Now we learn a **single** parametric function that maps x to a set of (good) variational parameters
- We approximate the posterior using this distribution $q_\phi(z|x)$
- Variational EM optimization problem:

$$\max_{\theta} \ell(\theta; \mathcal{D}) \geq \max_{\theta, \phi} \sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$$

Learning Deep Generative models



- Single parametric approximation $q_\phi(z|x)$ for the posterior $p(x|z)$
- Shared set of parameters ϕ for all data points
- **Idea:** Optimize $\sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ as a function of θ, ϕ using gradient descent

Learning Deep Generative models

- Optimize $\sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ as a function of θ, ϕ using (stochastic) gradient descent

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q_{\phi}(\mathbf{z}|\mathbf{x})) \\ &= E_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})]\end{aligned}$$

- Initialize $\theta^{(0)}, \phi^{(0)}$
 - Randomly sample a data point \mathbf{x}^i from \mathcal{D}
 - Compute $\nabla_{\theta} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ and $\nabla_{\phi} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$
 - Update θ, ϕ in the gradient direction
- How to compute the gradients? There might not be a closed form solution for the expectations. So we use Monte Carlo sampling

Learning Deep Generative models

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q_{\phi}(\mathbf{z}|\mathbf{x})) \\ &= E_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})]\end{aligned}$$

- To evaluate the bound, sample $\mathbf{z}^1, \dots, \mathbf{z}^K$ from $q_{\phi}(\mathbf{z}|\mathbf{x})$ and estimate

$$E_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})] \approx \frac{1}{K} \sum_k \log p(\mathbf{z}^k, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}^k|\mathbf{x})$$

- Want to compute $\nabla_{\theta} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ and $\nabla_{\phi} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$
- The gradient with respect to θ is easy

$$\begin{aligned}\nabla_{\theta} E_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})] &= E_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\nabla_{\theta} \log p(\mathbf{z}, \mathbf{x}; \theta)] \\ &\approx \frac{1}{K} \sum_k \nabla_{\theta} \log p(\mathbf{z}^k, \mathbf{x}; \theta)\end{aligned}$$

Learning Deep Generative models

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q_{\phi}(\mathbf{z}|\mathbf{x})) \\ &= E_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})]\end{aligned}$$

- Want to compute $\nabla_{\theta} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ and $\nabla_{\phi} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$
- The gradient with respect to ϕ is more complicated because the expectation depends on ϕ
- We still want to estimate with a Monte Carlo average
- We use a technique called REINFORCE from reinforcement learning

Reinforce

- Set A of possible actions. Each action $\mathbf{z} \in A$ has a reward $r(\mathbf{z})$
- Randomized policy for choosing actions $q_\phi(\mathbf{z})$ parameterized by ϕ . For example, $q_\phi(\mathbf{z}) \propto \exp(f(\mathbf{z}) \cdot \phi)$ where $f(\mathbf{z})$ is a feature vector
- Want to compute a gradient with respect to ϕ of the expected reward

$$E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] = \sum_{\mathbf{z}} q_\phi(\mathbf{z}) r(\mathbf{z})$$

$$\begin{aligned}\frac{\partial}{\partial \phi_i} E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] &= \sum_{\mathbf{z}} \frac{\partial q_\phi(\mathbf{z})}{\partial \phi_i} r(\mathbf{z}) = \sum_{\mathbf{z}} q_\phi(\mathbf{z}) \frac{\partial \log q_\phi(\mathbf{z})}{\partial \phi_i} r(\mathbf{z}) \\ &= \sum_{\mathbf{z}} q_\phi(\mathbf{z}) \frac{1}{q_\phi(\mathbf{z})} \frac{\partial q_\phi(\mathbf{z})}{\partial \phi_i} r(\mathbf{z}) = E_{q_\phi(\mathbf{z})} \left[\frac{\partial \log q_\phi(\mathbf{z})}{\partial \phi_i} r(\mathbf{z}) \right]\end{aligned}$$

Reinforce

- Want to compute a gradient with respect to ϕ of

$$E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] = \sum_{\mathbf{z}} q_\phi(\mathbf{z}) r(\mathbf{z})$$

- The REINFORCE rule is

$$\nabla_\phi E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] = E_{q_\phi(\mathbf{z})} [r(\mathbf{z}) \nabla_\phi \log q_\phi(\mathbf{z})]$$

- We can now construct a Monte Carlo estimate
- Sample $\mathbf{z}^1, \dots, \mathbf{z}^K$ from $q_\phi(\mathbf{z})$ and estimate

$$\nabla_\phi E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] \approx \frac{1}{K} \sum_k r(\mathbf{z}^k) \nabla_\phi \log q_\phi(\mathbf{z}^k)$$

Reinforce

- To learn the variational approximation we need to compute the gradient with respect to ϕ of

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q_{\phi}(\mathbf{z}|\mathbf{x})) \\ &= E_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})]\end{aligned}$$

- The function inside the brackets also depends on ϕ . Want to compute a gradient with respect to ϕ of

$$E_{q_{\phi}(\mathbf{z})} [f(\phi, \mathbf{z})] = \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}) f(\phi, \mathbf{z})$$

- The REINFORCE rule is

$$\nabla_{\phi} E_{q_{\phi}(\mathbf{z})} [f(\phi, \mathbf{z})] = E_{q_{\phi}(\mathbf{z})} [f(\phi, \mathbf{z}) \nabla_{\phi} \log q_{\phi}(\mathbf{z}) + \nabla_{\phi} f(\phi, \mathbf{z})]$$

- We can now construct a Monte Carlo estimate of $\nabla_{\phi} \mathcal{L}(\mathbf{x}; \theta, \phi)$

Learning Deep Generative models

- Optimize $\sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ as a function of θ, ϕ using (stochastic) gradient descent

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q_{\phi}(\mathbf{z}|\mathbf{x})) \\ &= E_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_{\phi}(\mathbf{z}|\mathbf{x})]\end{aligned}$$

- Initialize $\theta^{(0)}, \phi^{(0)}$
 - Randomly sample a data point \mathbf{x}^i from \mathcal{D}
 - Estimate $\nabla_{\theta} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ and $\nabla_{\phi} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ (Monte Carlo)
 - Update θ, ϕ in the gradient direction
- In practice, gradients estimates can be too noisy. Need to use **control variates** (baselines) or **reparameterization trick**

Baselines

- Want to compute a gradient with respect to ϕ of

$$E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] = \sum_{\mathbf{z}} q_\phi(\mathbf{z}) r(\mathbf{z})$$

- The REINFORCE rule is

$$\nabla_\phi E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] = E_{q_\phi(\mathbf{z})} [r(\mathbf{z}) \nabla_\phi \log q_\phi(\mathbf{z})]$$

- Given any constant B (a baseline)

$$\nabla_\phi E_{q_\phi(\mathbf{z})}[r(\mathbf{z})] = E_{q_\phi(\mathbf{z})} [(r(\mathbf{z}) - B) \nabla_\phi \log q_\phi(\mathbf{z})]$$

- To see why,

$$\begin{aligned} E_{q_\phi(\mathbf{z})} [B \nabla_\phi \log q_\phi(\mathbf{z})] &= B \sum_{\mathbf{z}} q_\phi(\mathbf{z}) \nabla_\phi \log q_\phi(\mathbf{z}) = B \sum_{\mathbf{z}} \nabla_\phi q_\phi(\mathbf{z}) \\ &= B \nabla_\phi \sum_{\mathbf{z}} q_\phi(\mathbf{z}) = B \nabla_\phi 1 = 0 \end{aligned}$$

- Same expectation but variance does change

Control variates

- Suppose we want to compute

$$E_{q_\phi(z)}[r(z)] = \sum_z q_\phi(z) r(z)$$

- Define

$$\hat{r}(z) = r(z) + a(h(z) - E_{q_\phi(z)}[h(z)])$$

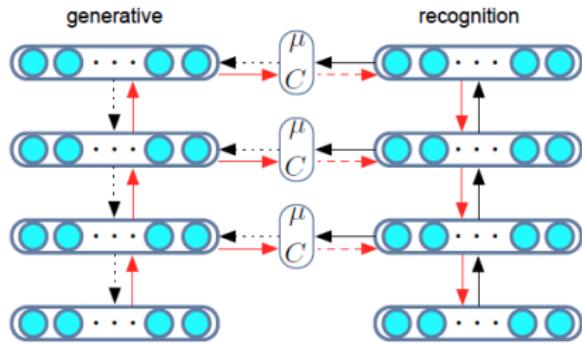
- They have the same expectation

$$E_{q_\phi(z)}[\hat{r}(z)] = E_{q_\phi(z)}[r(z)]$$

but different variances

- Can try to learn and update the control variate during training

Example

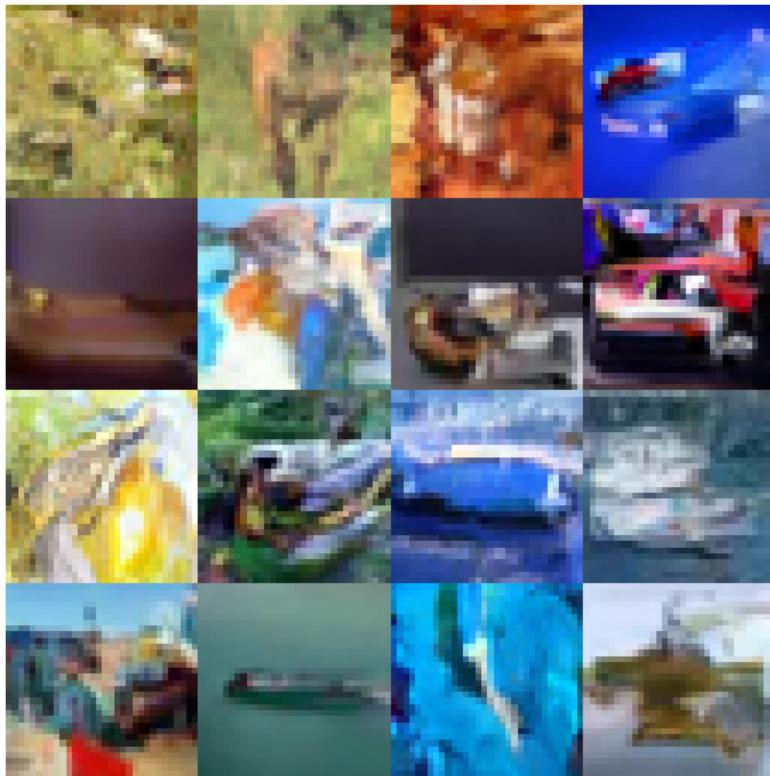


- Can use a neural network to model $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$. Fairly general non-linear transformation

Example



Example



Summary

- Variational Bayes can be used for latent variable models with intractable posteriors
- Deep generative models can learn a broad and powerful class of generative models
- The hidden variables \mathbf{z} are not interpretable
- ELBO might be very loose
- Very active area of research

The END

This is it for CS 228! Thanks for the great quarter together!

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 3, January 17, 2017

Announcements

- Homework 1 due next Tuesday.
- Quizzes available.

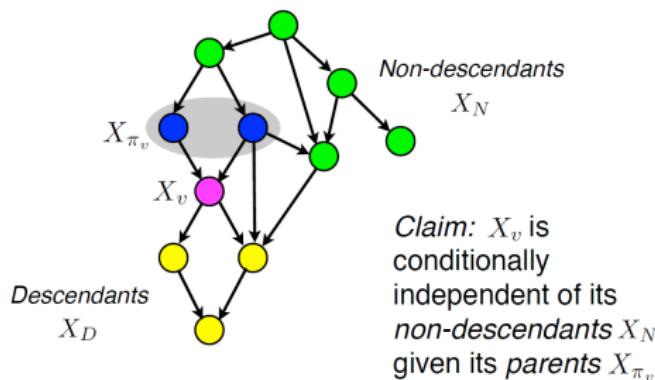
Plan for today

- Probabilistic graphical models: marriage of probability and graph theory
- Last lecture: use graphs as a “data-structure” to specify complex probabilistic models
- This lecture: what does the graph (*structure*) tell us about the statistical properties of a model?
- Key idea: use simple graph algorithms to deduct (complicated) statistical properties of the models

Bayesian network structure implies conditional independencies!

- A Bayesian network structure G implies the following conditional independence assumptions:
 - For each variable X_v , X_v is independent from its non-descendants given its parents

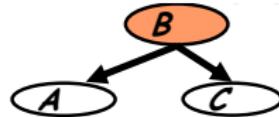
$$X_v \perp \text{NonDescendants}_{X_v} \mid \text{Parents}_{X_v}$$



- Why is it important? As a modeler, you need to understand the assumptions/simplifications your model is making

Networks with 3 vertices

- **Common parent** – $A \perp C$, fixing B *decouples* A and C: $A \perp C | B$



Ex: Height (**A**) \perp Reading Ability (**C**) | Age (**B**)

- **Cascade** – $A \perp C$, knowing B *decouples* A and C: $A \perp C | B$



Ex: Reading Ability (**C**) \perp Age (**A**) | Intelligence (**B**)

- **V-structure** – $A \perp B$, knowing C *couples* A and B : $A \perp B | C$



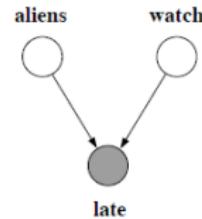
- This important phenomena is called **explaining away** and is what makes Bayesian networks so powerful

Explaining away

- Example: Alice is late. Was she abducted by aliens (**A**) or did she forget to set the alarm on her watch ($\neg W$)?

		remembered watch (W)	
		0	1
aliens (A)	0	0.8	0.1
	1	0.99	0.99

$P(L \mid \dots)$



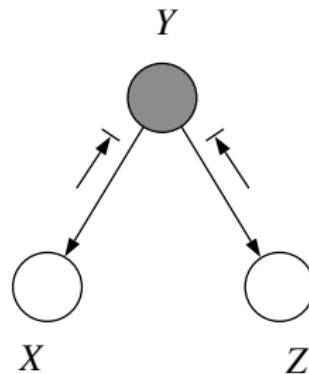
- $P(A = \text{yes}) = 0.001, P(W = \text{yes}) = 0.5$
- $P(A = \text{yes} | L = \text{yes}) \approx 0.002$
- $P(A = \text{yes} | L = \text{yes}, W = \text{no}) \approx 0.0012$
- $P(A = \text{yes} | L = \text{yes}) \neq P(A = \text{yes} | L = \text{yes}, W = \text{no}) \Rightarrow \cancel{A \perp W \mid L}$

D-separation (“directed separated”) in Bayesian networks

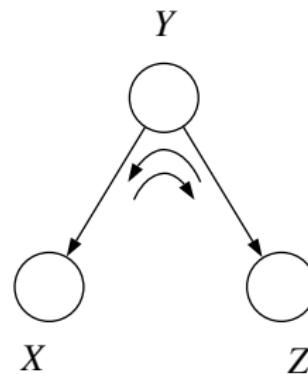
- Can decide conditional independence via algebraic operations: tedious and error prone. Can we just look at the graph?
- Algorithm to calculate whether $Q \perp W | \mathbf{O}$ by looking at **graph separation**
- Look to see if there is **active path** between Q and W when variables \mathbf{O} are observed:
 - An *undirected* path in BN structure G is called **active path** for observed variables $\mathbf{O} \subseteq \{X_1, \dots, X_n\}$, if for every consecutive triple of vars X, Y, Z on the path
 - $X \leftarrow Y \leftarrow Z$ and Y is unobserved ($Y \notin \mathcal{O}$)
 - $X \leftarrow Y \rightarrow Z$ and Y is unobserved ($Y \notin \mathcal{O}$)
 - $X \rightarrow Y \rightarrow Z$ and Y is unobserved ($Y \notin \mathcal{O}$)
 - $X \rightarrow Y \leftarrow Z$ and Y or any of Y 's descendants is observed

D-separation (“directed separated”) in Bayesian networks

- Algorithm to calculate whether $X \perp Z | \mathbf{Y}$ by looking at graph separation
- Look to see if there is **active path** between X and Z when variables \mathbf{Y} are observed:



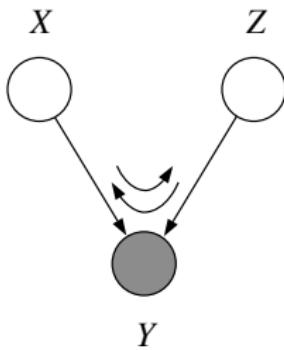
(a)



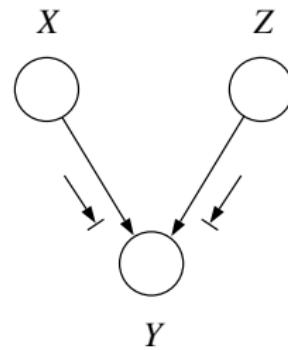
(b)

D-separation (“directed separated”) in Bayesian networks

- Algorithm to calculate whether $X \perp Z | \mathbf{Y}$ by looking at graph separation
- Look to see if there is **active path** between X and Z when variables \mathbf{Y} are observed:



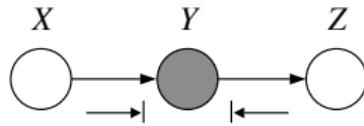
(a)



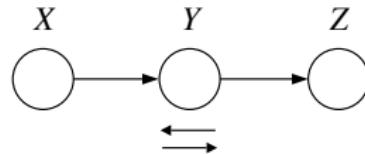
(b)

D-separation (“directed separated”) in Bayesian networks

- Algorithm to calculate whether $X \perp Z | \mathbf{Y}$ by looking at graph separation
- Look to see if there is **active path** between X and Z when variables \mathbf{Y} are observed:



(a)



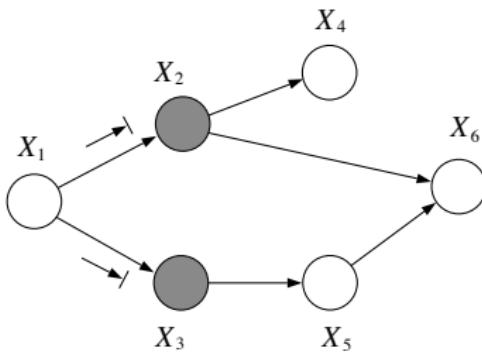
(b)

- If *no such path*, then X and Z are **d-separated** with respect to \mathbf{Y}
- d-separation reduces statistical independencies (hard) to connectivity in graphs (easy)

D-separation and independence

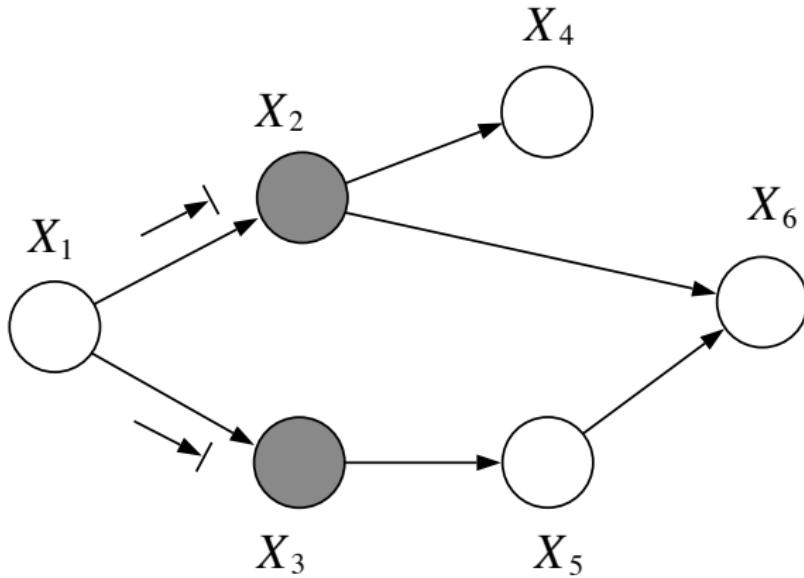
- Idea: $d - sep(X; Y | \mathbf{Z}) \Rightarrow X \perp Y | \mathbf{Z}$
- i.e., X conditionally independent from Y given \mathbf{Z} if there **does not exist** any active path between X and Y for observations \mathbf{Z}

Example



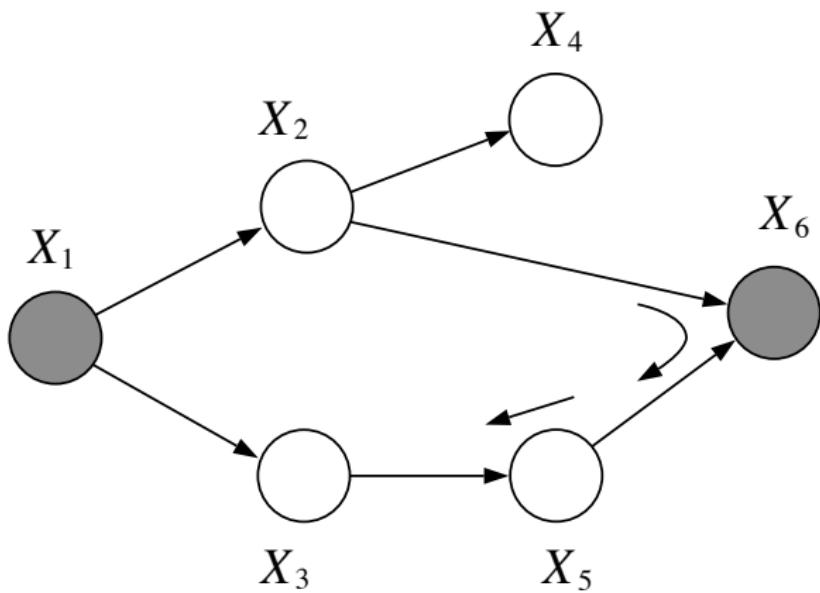
$$d - sep(X_1; X_6 | \{X_2, X_3\}) \Rightarrow X_1 \perp X_6 | \{X_2, X_3\}$$

D-separation example 1



Is $X_1 \perp\!\!\!\perp X_6 | X_2, X_3$? Is $X_5 \perp\!\!\!\perp X_6 | X_2, X_3$? Is $X_4 \perp\!\!\!\perp X_5 | X_2, X_3$?

D-separation example 2



Is $X_4 \perp\!\!\!\perp X_5 | X_1, X_6$? Is $X_4 \perp\!\!\!\perp X_5 | X_1$?

Soundness of d-separation

- Let $I(p)$ denote the set of all conditional independencies that hold for a joint distribution p .
 - Example. Suppose $p(X = 0, Y = 0) = p(X = 0, Y = 1) = p(X = 1, Y = 0) = p(X = 1, Y = 1) = 0.25$. Does $X \perp Y \in I(p)$? Yes!
- Define $I(G) = \{(\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}) : d - sep_G(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})\}$, the set of independencies that correspond to d-separation in G
- Theorem:** If p factorizes according to G , then $I(G) \subseteq I(p)$
- Hence, d-separation captures only true independencies (**soundness**)
- How about $I(G) = I(p)$? (**completeness**)

Completeness?

- Suppose p factorizes over G . We know $I(G) \subseteq I(p)$. Does it hold that $I(p) \subseteq I(G)$?
- NO! **Even if a distribution factorizes, it can still contain additional independencies that are not reflected in the graph structure**
- Consider the network $A \rightarrow B$. $p(A = a^0) = Pr(A = a^1) = 0.5$

	b^0	b^1	
a^0	0.4	0.6	$P(B A = a^0)$
a^1	0.4	0.6	$P(B A = a^1)$

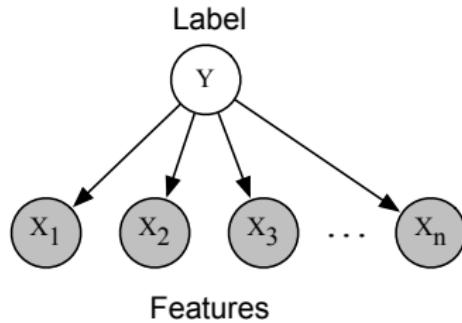
- Is $A \perp B$? Yes! $p(a, b) = p(a)p(b|a) = p(a)p(b)$ We have $I(p) = \{A \perp B\}$
- What can be said using d-separation? $I(G) = \emptyset$
- $I(p) = \{A \perp B\} \not\subseteq \emptyset = I(G)$. Therefore, $I(p) \subseteq I(G)$ does not hold in general!

Completeness?

- However, if $X \perp Y | \mathbf{Z}$ in *all* distributions that factor over G , then $d - sep_G(X; Y | \mathbf{Z})$
- **Theorem:** If X and Y are not d-separated given \mathbf{Z} , then there exists some distribution p factorizing over G in which X and Y are dependent given \mathbf{Z}
- Proof sketch: Pick active path. Parameterize CPD along the path to create dependence. Set everything else to independent to avoid canceling dependencies.

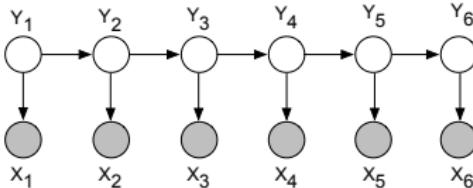
Naive Bayes for single label prediction

- Classify e-mails as spam ($Y = 1$) or not spam ($Y = 0$)
 - Let $1 : n$ index the words in our vocabulary (e.g., English)
 - $X_i = 1$ if word i appears in an e-mail, and 0 otherwise
 - E-mails are drawn according to some distribution $p(Y, X_1, \dots, X_n)$



- $X_1 \perp X_3 ?$
- $X_1 \perp X_3 | Y ?$

Hidden Markov models



- Frequently used for speech recognition, part-of-speech tagging, Kalman filtering. Hidden variables (Y) and Observations (X)
- Joint distribution factors as:

$$p(\mathbf{y}, \mathbf{x}) = p(y_1)p(x_1 | y_1) \prod_{t=2}^T p(y_t | y_{t-1})p(x_t | y_t)$$

- $p(y_1)$ is the distribution for the starting state
 - $p(y_t | y_{t-1})$ is the *transition* probability between any two states
 - $p(x_t | y_t)$ is the *emission* probability
- What are the conditional independencies here? $X_1 \perp X_6$?
 $Y_1 \perp \{Y_3, \dots, Y_6\} | Y_2$?

Representing the world using BN

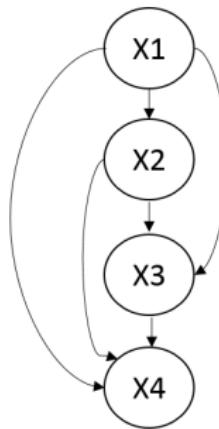
Given a distribution p , can we construct a graph G whose independencies are a reasonable surrogate of the independencies in p ?

Independence maps

- Let $I(G)$ be the set of all conditional independencies implied by a directed acyclic graph (DAG) G
- Let $I(p)$ denote the set of all conditional independencies that hold for a joint distribution p .
- A DAG G is an **I-map** (independence map) of a distribution p if $I(G) \subseteq I(p)$
 - Claim: A fully connected DAG G is an I-map for *any* distribution.

Every probability distribution can be described by a BN

- $p(x_1, \dots, x_n) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_n | x_1, \dots, x_{n-1})$
- For example, for $n = 4$ consider the following Bayesian network:



- A fully connected DAG G is an I-map for *any* distribution, since $I(G) = \emptyset \subseteq I(p)$ for all p . It does not make any independence assumption!
- Can we find a more compact/ economical representation?

Independence maps

- Let $I(G)$ be the set of all conditional independencies implied by the directed acyclic graph (DAG) G
- Let $I(p)$ denote the set of all conditional independencies that hold for a joint distribution p .
- A DAG G is an **I-map** (independence map) of a distribution p if $I(G) \subseteq I(p)$
 - A fully connected DAG G is an I-map for *any* distribution, since $I(G) = \emptyset \subseteq I(p)$ for all p
- G is a **minimal I-map** for p if it is an I-map and the removal of even a single edge makes it not an I-map

Existence of minimal I-maps

- Does every distribution have a minimal I-Map?
- YES. Start with a fully connected DAG G , $I(G) = \emptyset \subseteq I(p)$ for all p . Keep removing edges as long as $I(G) \subseteq I(p)$.
- Not unique!

Perfect Maps

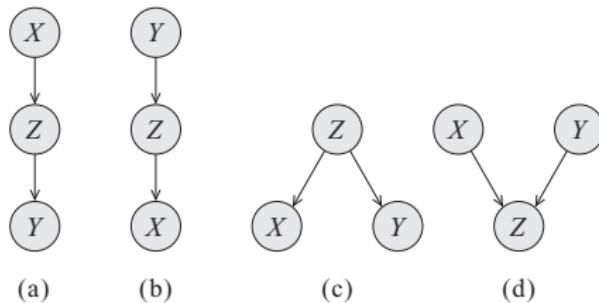
- Minimal I-maps are easy to find, but can contain many unnecessary dependencies, that is, $I(G)$ is much smaller than $I(p)$.
- A BN structure G is called P-map (**perfect map**) for distribution p if $I(G) = I(p)$
- Does every distribution p have a P-map (**perfect map**)?
- No! We will see a counterexample later.

Uniqueness of perfect maps

- If a perfect map exists, is it unique? No
- Consider the following two Bayes Nets structures:
 - $G_1: X \rightarrow Y$
 - $G_2: X \leftarrow Y$
- $I(G_1) = I(G_2)$
- Different BN structures can encode the same conditional independency assertions

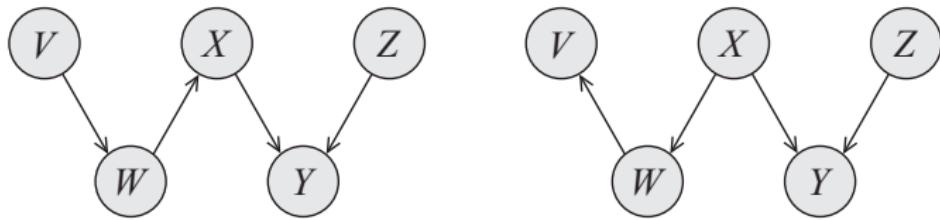
Equivalent structures

- Different Bayesian network structures can be **equivalent** in that they encode precisely the same conditional independence assertions
- Two graphs G, G' are called **I-equivalent** if $I(G) = I(G')$
- Which of these are equivalent?



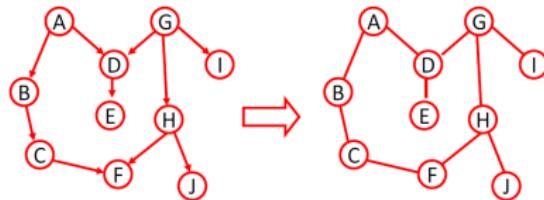
Equivalent structures

- Different Bayesian network structures can be **equivalent** in that they encode precisely the same conditional independence assertions
- Are these equivalent?



Skeleton of BN

- **Theorem:** I-equivalent BNs must have same skeleton



- Converse does not hold. $X \rightarrow Y \leftarrow Z$ is not I-equivalent to $X \rightarrow Y \rightarrow Z$

Importance of v-structures

- **Theorem:** If G, G' have same skeleton and same v-structures, then $I(G) = I(G')$
- Converse does not hold

Summary

- d-separation reduces statistical independencies (hard) to connectivity in graphs (easy)
- The fewer edges in the DAG
 - the more compact the representation is (less parameters)
 - the more independencies are implied by the graph
- Unfortunately,
 - minimal I-maps are not unique (depends on the ordering)
 - there are distributions whose independencies cannot be perfectly captured by a graph (no perfect maps)

2011 Turing Award was for Bayesian networks

A.M. TURING CENTENARY CELEBRATION WEBCAST

Search TYPE HERE



MORE ACM AWARDS

A.M. TURING AWARD

A.M. TURING AWARD WINNERS BY...

ALPHABETICAL LISTING	YEAR OF THE AWARD	RESEARCH SUBJECT
	JUDEA PEARL United States – 2011	CITATION
 Photo-Essay	For fundamental contributions to artificial intelligence through the development of a calculus for probabilistic and causal reasoning.	SHORT ANNOTATED BIBLIOGRAPHY
 BIRTH: September 4, 1936, Tel Aviv.		ACM DL AUTHOR PROFILE
 EDUCATION: B.S., Electrical Engineering (Technion, 1960); M.S., Electronics (Newark College of Engineering, 1961); M.S., Physics (Rutgers University, 1965); Ph.D., Electrical Engineering (Polytechnic Institute of Brooklyn, 1965).		ACM TURING AWARD LECTURE VIDEO
 EXPERIENCE: Research Engineer, New York University Medical School (1960–1961); Instructor,		RESEARCH SUBJECTS
		ADDITIONAL MATERIALS

Judea Pearl created the representational and computational foundation for the processing of information under uncertainty.

He is credited with the invention of *Bayesian networks*, a mathematical formalism for defining complex probability models, as well as the principal algorithms used for inference in these models. This work not only revolutionized the field of artificial intelligence but also became an important tool for many other branches of engineering and the natural sciences. He later created a mathematical framework for *causal inference* that has had significant impact in the social sciences.

Judea Pearl was born on September 4, 1936, in Tel Aviv, which was at that time administered under the British Mandate for Palestine. He grew up in *Bnei Brak*, a Biblical town his grandfather went to reestablish in 1924. In 1956, after serving in the Israeli army and joining a Kibbutz, Judea decided to study engineering. He attended the Technion, where he met his wife, Ruth, and received a B.S. degree in Electrical Engineering in 1960. Recalling the Technion faculty members in a 2012 interview in the *Technion Magazine*, he emphasized the thrill of discovery:

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 4, January 19, 2017

Announcements

- Python review session today, from 16:30 to 17:20 in the Skilling Auditorium. Will be recorded.
- Thanks for all the pull requests for the course notes!

Independence and Perfect Maps

- I-Map: minimal I-maps are easy to find, but can contain many unnecessary dependencies $I(G) \subseteq I(p)$.
- P-Map: A BN structure G is called P-map (perfect map) for distribution p if $I(G) = I(p)$
- Does every distribution p have a P-map? No
- There are types of dependencies that are not easily captured by BN

Plan for today

Introduce a new class of graphical models based on undirected graphs
(undirected graphical models)

- can represent certain dependencies more easily than BN
- used more in practice
- harder to interpret than BN

Example

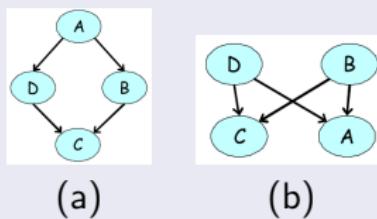
- We have a group of four people: Alex, Bob, Catherine, David
- $A = \text{Alex's hair color}$ (red, green, blue)
 $B = \text{Bob's hair color}$
 $C = \text{Catherine's hair color}$
 $D = \text{David's hair color}$
- Alex and Bob are friends, Bob and Catherine are friends, Catherine and David are friends, David and Alex are friends
- Friends *never* have the same hair color!
- p is the joint distribution over everyone's hair color (uniform over valid assignments)
 - E.g., $p(A = \text{red}, B = \text{red}, C = \text{blue}, D = \text{green}) = 0$,
 $p(A = \text{red}, B = \text{green}, C = \text{blue}, D = \text{green}) > 0$
- The distribution p satisfies $A \perp C \mid \{B, D\}$ and $B \perp D \mid \{A, C\}$

Perfect Maps

- Recall that G is a **perfect map** for distribution p if $I(G) = I(p)$
- Theorem:** Not every distribution has a perfect map as a DAG

Proof.

(By counterexample.) There is a distribution on 4 variables where the only independencies are $A \perp C \mid \{B, D\}$ and $B \perp D \mid \{A, C\}$. This cannot be represented (perfectly) by any Bayesian network.

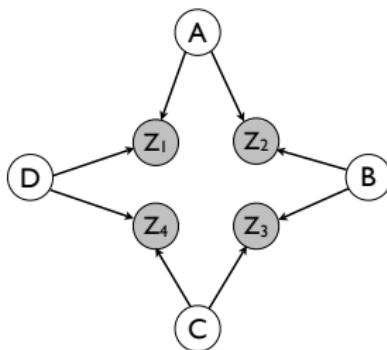


Both (a) and (b) encode $(A \perp C \mid \{B, D\})$, but in both cases $(B \perp D \mid \{A, C\}) \notin I(G)$.



Bayesian networks have limitations

- Can we represent p with a BN? Yes, we can always use a fully connected BN, but this obscures its structure
- Alternatively, we can introduce “dummy” binary variables \mathbf{Z} and work with a **conditional** distribution:



- This satisfies $A \perp C | \{B, D, \mathbf{Z}\}$ and $B \perp D | \{A, C, \mathbf{Z}\}$
- Returning to the previous example, we would set:

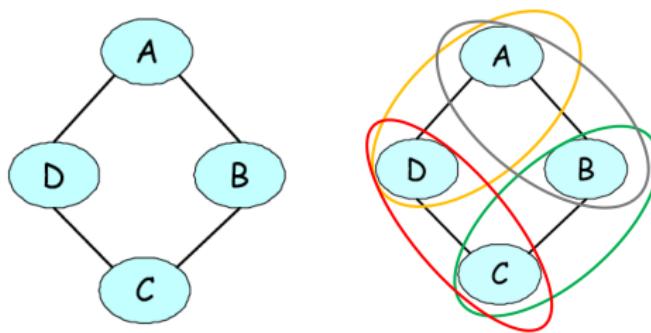
$$p(Z_1 = 1 | a, d) = 1 \text{ if } a \neq d, \text{ and } 0 \text{ if } a = d$$

$$p(Z_1 = 0 | a, d) = 0 \text{ if } a \neq d, \text{ and } 1 \text{ if } a = d$$

Z_1 is the observation that Alice and David have different hair colors

Undirected graphical models

- An alternative representation for joint distributions is as an **undirected graphical model**
- We have an **undirected graph**:
 - one node for each random variable (as in BN)
 - **undirected** edges (representing dependencies)
- Key difference: no topological ordering! Cannot define parents anymore. Instead we use **cliques**. A clique is a **complete subgraph**



Undirected graphical models

- An alternative representation for joint distributions is as an **undirected graphical model**
- A **Markov Random Field** (MRF) is defined by an **undirected graph**:
 - one node for each random variable (as in BN)
 - **undirected** edges (representing dependencies)
- Rather than CPDs, we specify (*non-negative*) **potential functions** (or **factors**) over sets of variables associated with cliques C of the graph,

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

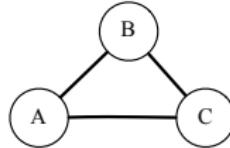
Z is the **partition function** and normalizes the distribution:

$$Z = \sum_{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

- Like Conditional Prob. Distributions, $\phi_c(\mathbf{x}_c)$ can be represented as a table, but it is *not normalized* (does not sum to one). How many parameters for a factor (clique) with 3 binary variables? 8

Example

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c), \quad Z = \sum_{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$


$$\phi_{A,B}(a, b) = \begin{array}{|c|c|} \hline & B \\ \hline 0 & 10 & 1 \\ \hline A & 1 & 1 & 10 \\ \hline 1 & & & \\ \hline \end{array}$$
$$\phi_{B,C}(b, c) = \begin{array}{|c|c|} \hline & C \\ \hline 0 & 10 & 1 \\ \hline B & 1 & 1 & 10 \\ \hline 1 & & & \\ \hline \end{array}$$
$$\phi_{A,C}(a, c) = \begin{array}{|c|c|} \hline & C \\ \hline 0 & 10 & 1 \\ \hline A & 1 & 1 & 10 \\ \hline 1 & & & \\ \hline \end{array}$$

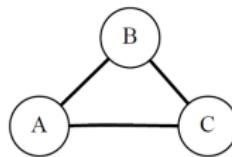
$$p(a, b, c) = \frac{1}{Z} \phi_{A,B}(a, b) \cdot \phi_{B,C}(b, c) \cdot \phi_{A,C}(a, c), \text{ where}$$

$$\begin{aligned} Z &= \sum_{\hat{a}, \hat{b}, \hat{c} \in \{0,1\}^3} \phi_{A,B}(\hat{a}, \hat{b}) \cdot \phi_{B,C}(\hat{b}, \hat{c}) \cdot \phi_{A,C}(\hat{a}, \hat{c}) \\ &= \phi_{A,B}(0,0)\phi_{B,C}(0,0)\phi_{A,C}(0,0) + \phi_{A,B}(0,0)\phi_{B,C}(0,1)\phi_{A,C}(0,1) + \dots \\ &= 10 \cdot 10 \cdot 10 + 10 \cdot 1 \cdot 1 + 10 \cdot 1 \cdot 1 + \dots = 2 \cdot 1000 + 6 \cdot 10 = 2060. \end{aligned}$$

What is $p(0, 0, 0)$? $p(0, 0, 0) = 10 \cdot 10 \cdot 10 / 2060$.

Example

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c), \quad Z = \sum_{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$



$$\phi_{A,B}(a,b) = \begin{array}{cc|cc} & B & & \\ & 0 & 1 & \\ \hline A & 0 & 10 & 10 \\ & 1 & 10 & 10 \end{array}$$

$$\phi_{B,C}(b,c) = \begin{array}{cc|cc} & C & & \\ & 0 & 1 & \\ \hline B & 0 & 10 & 1 \\ & 1 & 1 & 10 \end{array}$$

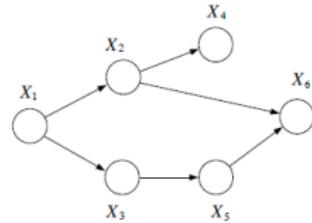
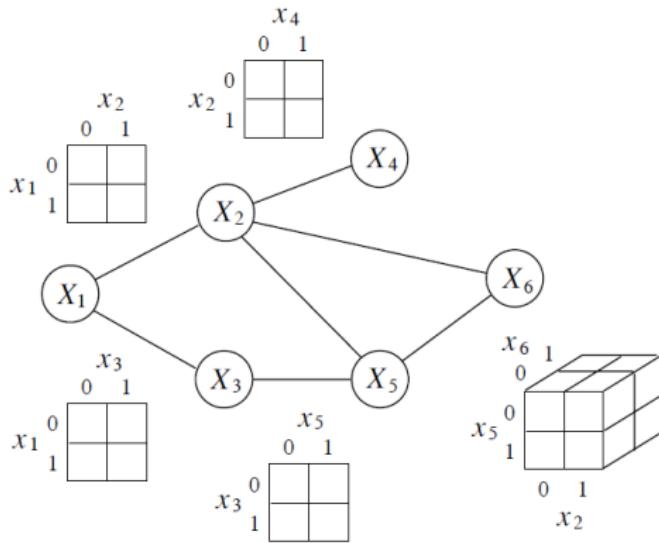
$$\phi_{A,C}(a,c) = \begin{array}{cc|cc} & C & & \\ & 0 & 1 & \\ \hline A & 0 & 10 & 1 \\ & 1 & 1 & 10 \end{array}$$

$$p(a, b, c) = \frac{1}{Z} \phi_{A,B}(a, b) \cdot \phi_{B,C}(b, c) \cdot \phi_{A,C}(a, c), \text{ where}$$

$$\begin{aligned} Z &= \sum_{\hat{a}, \hat{b}, \hat{c} \in \{0,1\}^3} \phi_{A,B}(\hat{a}, \hat{b}) \cdot \phi_{B,C}(\hat{b}, \hat{c}) \cdot \phi_{A,C}(\hat{a}, \hat{c}) \\ &= \phi_{A,B}(0,0)\phi_{B,C}(0,0)\phi_{A,C}(0,0) + \phi_{A,B}(0,0)\phi_{B,C}(0,1)\phi_{A,C}(0,1) + \dots \\ &= 100 \cdot 10 \cdot 10 + 100 \cdot 1 \cdot 1 + 100 \cdot 1 \cdot 1 + \dots = 2 \cdot 1000 + 6 \cdot 100 = 20600. \end{aligned}$$

What is $p(0, 0, 0)$? $p(0, 0, 0) = 100 \cdot 10 \cdot 10 / 20600 = 10 \cdot 10 \cdot 10 / 2060$.

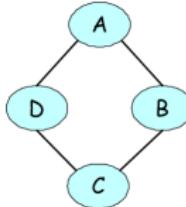
MRF Example



$$p(x_1, x_2, x_3, x_4, x_5, x_6) = \frac{1}{Z} \phi_{12}(x_1, x_2) \phi_{13}(x_1, x_3) \phi_{35}(x_3, x_5) \phi_{24}(x_2, x_4) \phi_{256}(x_2, x_5, x_6)$$

Hair color example as a MRF

- We now have an **undirected** graph:



- The joint probability distribution is parameterized as

$$p(a, b, c, d) = \frac{1}{Z} \phi_{AB}(a, b) \phi_{BC}(b, c) \phi_{CD}(c, d) \phi_{AD}(a, d) \phi_A(a) \phi_B(b) \phi_C(c) \phi_D(d)$$

- Pairwise potentials** enforce that no friend has the same hair color:

$$\phi_{AB}(a, b) = 0 \text{ if } a = b, \text{ and } 1 \text{ otherwise}$$

- Single-node potentials** specify an affinity for a particular hair color, e.g.

$$\phi_D(\text{"red"}) = 0.6, \quad \phi_D(\text{"blue"}) = 0.3, \quad \phi_D(\text{"green"}) = 0.1$$

The normalization Z makes the potentials **scale invariant!** Equivalent to

$$\phi_D(\text{"red"}) = 6, \quad \phi_D(\text{"blue"}) = 3, \quad \phi_D(\text{"green"}) = 1$$

Remarks on Normalization

- Without a topological ordering, there is no natural way to express the joint as a product of consistent local conditional or marginal probabilities; have to sacrifice local normalization
- A consequence is that some parts of the model may end up carrying more “weight” than others
- Without a topological ordering and local normalization, cannot sample efficiently!
- Can make interpretation more difficult

Independences in Markov Nets?

- In Bayes Nets

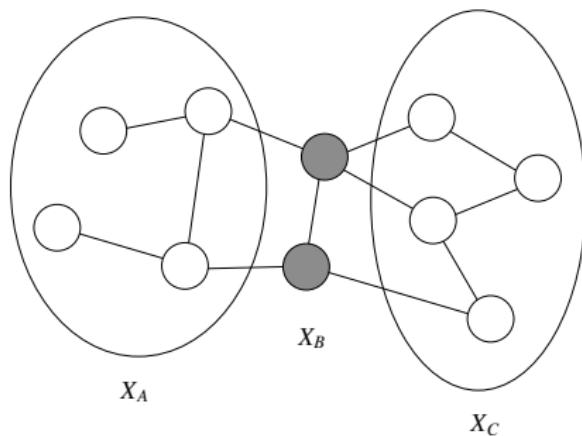
- Local independencies: $X \perp NonDescendants(X) \mid Parents_X$
- Factorization Theorem: p factorizes over $G \Leftrightarrow G$ is an I-map
- Global independencies: d-separation
- Soundness (and completeness) of d-separation

- How about Markov Nets?

- What's the analog of the Local independencies?
- Is there a factorization theorem for Markov Nets?
- What replaces d-separation?

Markov network structure implies conditional independencies

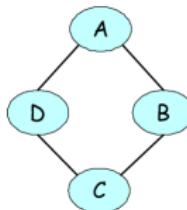
- Let G be the undirected graph. Note we must have one edge for every pair of variables that appear together in a factor
- Conditional independence is given by **graph separation!**



- $X_A \perp X_C | X_B$ if there is no path from $a \in A$ to $c \in C$ after removing all variables in B

Example

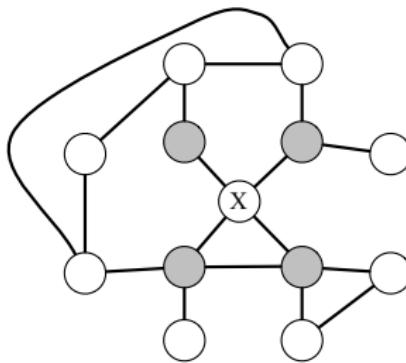
- Returning to hair color example, its undirected graphical model is:



- Is $D \perp B | \{A, C\}$? Since removing A and C leaves no path from D to B , we have $D \perp B | \{A, C\}$
- Is $A \perp C | D$? No, there is path $A - B - C$
- Is $A \perp C | \{D, B\}$? Similarly, since removing D and B leaves no path from A to C , we have $A \perp C | \{D, B\}$
- No other independencies implied by the graph

Markov blanket

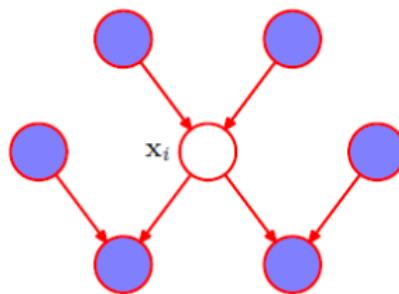
- A set \mathbf{U} is a **Markov blanket** of X if $X \notin \mathbf{U}$ and if \mathbf{U} is a minimal set of nodes such that $X \perp (\mathcal{X} - \{X\} - \mathbf{U}) \mid \mathbf{U}$
- What is the Markov blanket in an undirected graphical model?
- In undirected graphical models, the Markov blanket of a variable is precisely its **neighbors** in the graph:



- In other words, X is independent of the rest of the nodes in the graph given its immediate neighbors

Aside: Markov Blanket in a Bayesian Net

How about a Bayes Net? A variable X_i is conditionally independent of all other variables given...



A variable X_i is conditionally independent of all other variables given its Markov blanket: X_i 's parents, its children, and its children's other parents.

Proof of independence through separation

- We will show that $A \perp C | B$ for the following distribution:



$$p(a, b, c) = \frac{1}{Z} \phi_{AB}(a, b) \phi_{BC}(b, c)$$

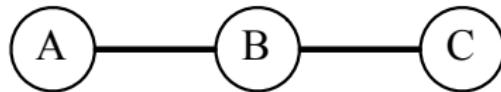
- First, we show that $p(a | b)$ can be computed using only $\phi_{AB}(a, b)$:

$$\begin{aligned} p(a | b) &= \frac{p(a, b)}{p(b)} = \frac{\sum_{\hat{c}} p(a, b, \hat{c})}{p(b)} = \frac{\sum_{\hat{c}} p(a, b, \hat{c})}{\sum_{\hat{a}, \hat{c}} p(\hat{a}, b, \hat{c})} \\ &= \frac{\frac{1}{Z} \sum_{\hat{c}} \phi_{AB}(a, b) \phi_{BC}(b, \hat{c})}{\frac{1}{Z} \sum_{\hat{a}, \hat{c}} \phi_{AB}(\hat{a}, b) \phi_{BC}(b, \hat{c})} \\ &= \frac{\phi_{AB}(a, b) \sum_{\hat{c}} \phi_{BC}(b, \hat{c})}{\sum_{\hat{a}} \phi_{AB}(\hat{a}, b) \sum_{\hat{c}} \phi_{BC}(b, \hat{c})} = \frac{\phi_{AB}(a, b)}{\sum_{\hat{a}} \phi_{AB}(\hat{a}, b)}. \end{aligned}$$

- More generally, the probability of a variable conditioned on its Markov blanket depends *only* on potentials involving that variable

Proof of independence through separation

- We will show that $A \perp C | B$ for the following distribution:



$$p(a, b, c) = \frac{1}{Z} \phi_{AB}(a, b) \phi_{BC}(b, c)$$

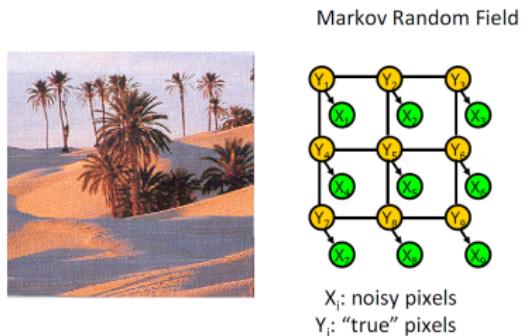
Proof.

$$\begin{aligned} p(a, c | b) &= \frac{p(a, c, b)}{\sum_{\hat{a}, \hat{c}} p(\hat{a}, b, \hat{c})} &= \frac{\frac{1}{Z} \phi_{AB}(a, b) \phi_{BC}(b, c)}{\sum_{\hat{a}, \hat{c}} \frac{1}{Z} \phi_{AB}(\hat{a}, b) \phi_{BC}(b, \hat{c})} \\ &= \frac{\phi_{AB}(a, b) \phi_{BC}(b, c)}{\sum_{\hat{a}} \phi_{AB}(\hat{a}, b) \sum_{\hat{c}} \phi_{BC}(b, \hat{c})} \\ &= p(a | b) p(c | b) \end{aligned}$$



Applications: Image Denoising

- There is a true image $\mathbf{y} \in \{0, 1\}^{3 \times 3}$, and a corrupted image $\mathbf{x} \in \{0, 1\}^{3 \times 3}$. We know \mathbf{x} , and want to somehow recover \mathbf{y} .

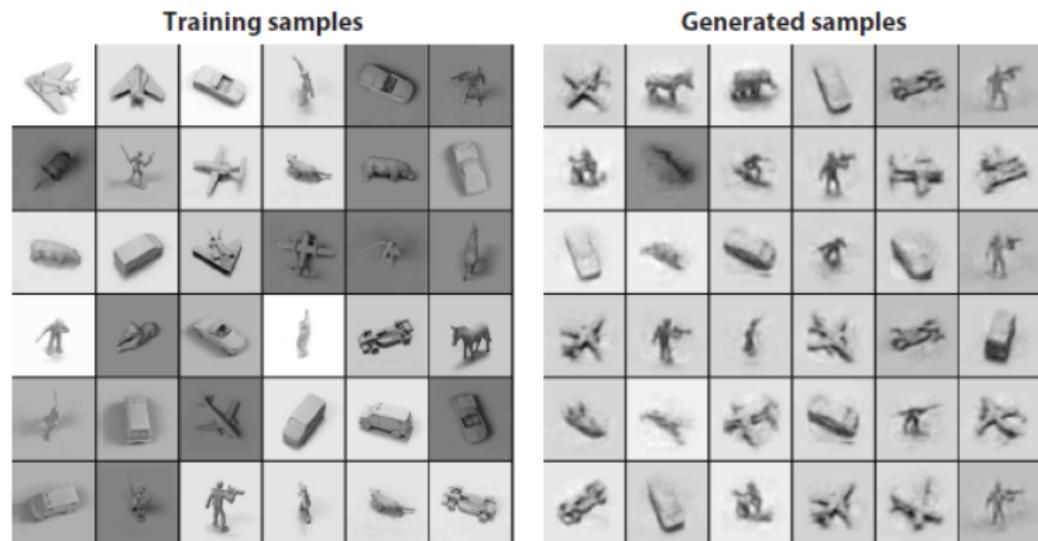


- We model the joint probability distribution $p(\mathbf{y}, \mathbf{x})$ as

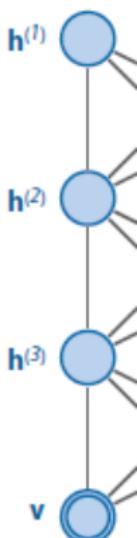
$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_i \phi_i(x_i, y_i) \prod_{(i,j) \in E} \phi_{ij}(y_i, y_j)$$

- $\phi_i(x_i, y_i)$: the i -th corrupted pixel depends on the i -th original pixel
- $\phi_{ij}(y_i, y_j)$: neighboring pixels tend to have the same value
- How did the original image \mathbf{y} look like? Solution: sample from $p(\mathbf{y}|\mathbf{x})$

Applications: Boltzmann Machines



Deep E



Bottom layer variables v are pixel values. Layers above (h) represent

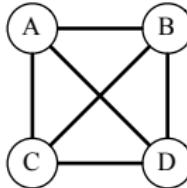
Higher-order potentials

- The examples so far have all been **pairwise MRFs**, involving only node potentials $\phi_i(X_i)$ and pairwise potentials $\phi_{i,j}(X_i, X_j)$
- Often we need **higher-order** potentials, e.g.

$$\phi(x, y, z) = 1 \text{ if and only if } (x + y + z \geq 1),$$

where X, Y, Z are binary, enforcing that at least one of the variables takes the value 1

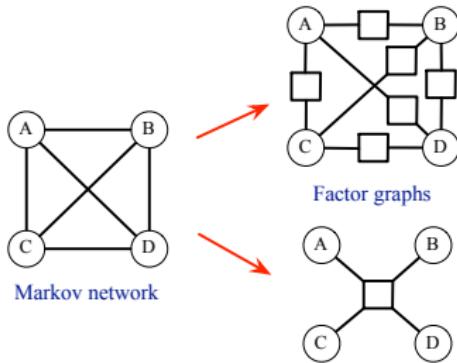
- We say that X, Y, Z is the **scope** of factor ϕ
- Although Markov networks are useful for understanding independencies, they hide much of the distribution's structure:



Does this have pairwise potentials, or one potential for all 4 variables?

Factor graphs

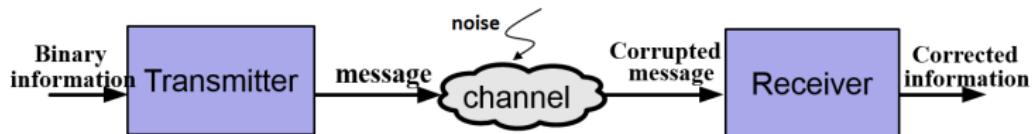
- Factors can be defined over maximal cliques or subsets of them. Cannot tell from the structure! (like in the previous example)
- A **factor graph** is a bipartite undirected graph with **variable nodes** (circles) and **factor nodes** (squares). Edges are only between the variable nodes and the factor nodes
- Each factor node is associated with a single potential, whose *scope* is the set of variables that are neighbors in the factor graph. Less ambiguous than MRF.



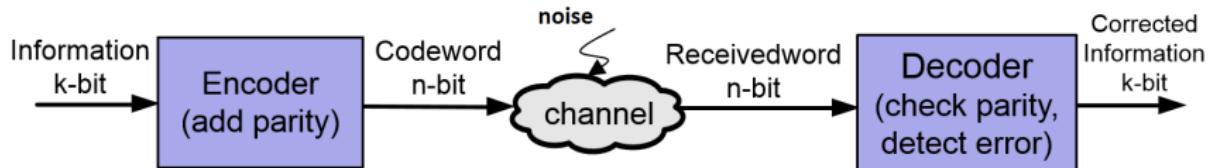
- The distribution is same as the MRF – this is just a different graph data structure

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{f \in \text{Factors}} \phi_f(\mathbf{x}_f)$$

Example: Error Correction in Communication Systems



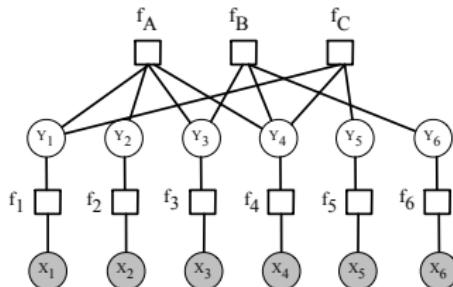
- Example:
 - Transmitted message: 1110011
 - Received message: 1011001
 - How many transmission errors? 3
- Error detection? Correction?



- Example: add parity bit at the end. Message 110001 ($k = 6$) encoded as 110001 | 1 ($n = 7$). If parity does not match, there was a transmission error.

Example: Low-density parity-check codes

- Error correcting codes for transmitting a message over a noisy channel

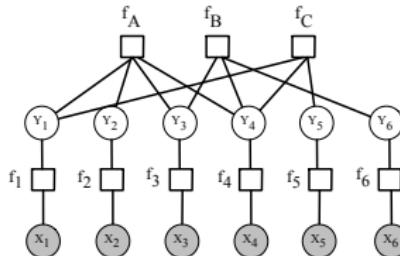


- Probabilistic model over received (\mathbf{x}) and transmitted message (\mathbf{y})
- Each of the top row factors enforce that its variables have even parity, e.g.:

$$f_A(y_1, y_2, y_3, y_4) = 1 \text{ if } y_1 \otimes y_2 \otimes y_3 \otimes y_4 = 0, \text{ and } 0 \text{ otherwise}$$

- Assignments \mathbf{Y} with non-zero probability are called **codewords**
- $f_i(y_i, x_i)$ models the probability of transmission error on the i -th bit: x_i is more often equal to y_i

Example: Low-density parity-check codes



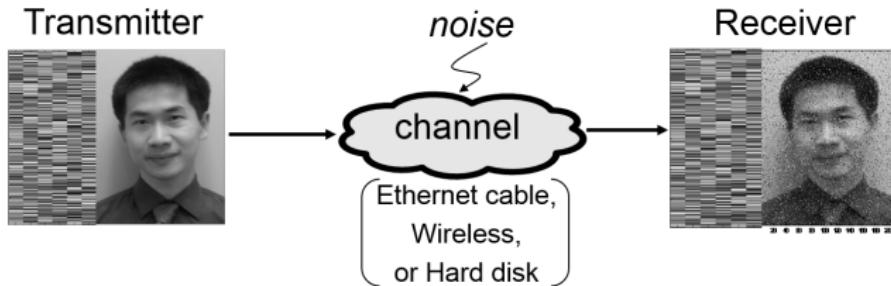
- After receiving \mathbf{x} , we want to guess the original message \mathbf{y} that was sent
 - must have been a valid codeword (f_A, f_B, f_C)
 - “close” to the received message \mathbf{x} (f_1, \dots, f_6)
- The *decoding* problem for LDPCs is to find

$$\operatorname{argmax}_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$$

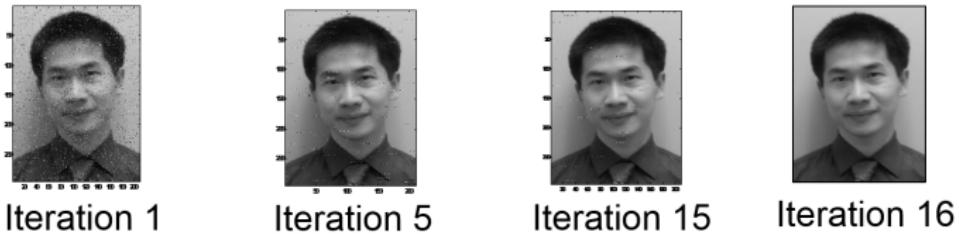
This is called the **maximum a posteriori** (MAP) assignment

- This is a type of inference that corresponds to a (discrete) optimization problem

Demo

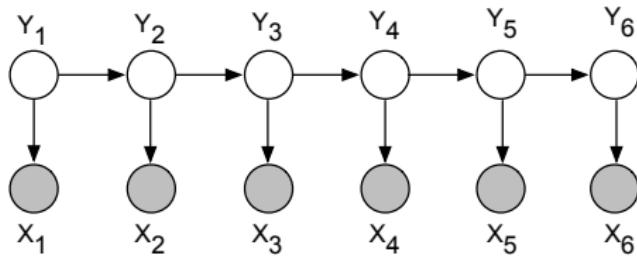


Iterative message passing decoding



Converting BNs to Markov networks

What is the equivalent Markov network for a hidden Markov model?



Many inference algorithms are more conveniently given for undirected models – this shows how they can be applied to Bayesian networks

Moralization of Bayesian networks

- Procedure for converting a Bayesian network into a Markov network
- The **moral graph** $\mathcal{M}[G]$ of a BN $G = (V, E)$ is an undirected graph over V that contains an undirected edge between X_i and X_j if
 - ① there is a directed edge between them (in either direction)
 - ② X_i and X_j are both parents of the same node



(term historically arose from the idea of “marrying the parents” of the node)

- The addition of the moralizing edges leads to the loss of some independence information, e.g., $A \rightarrow C \leftarrow B$, where $A \perp B$ is lost

Converting BNs to Markov networks

- ① Moralize the directed graph to obtain the undirected graphical model:



- ② Introduce one potential function for each CPD:

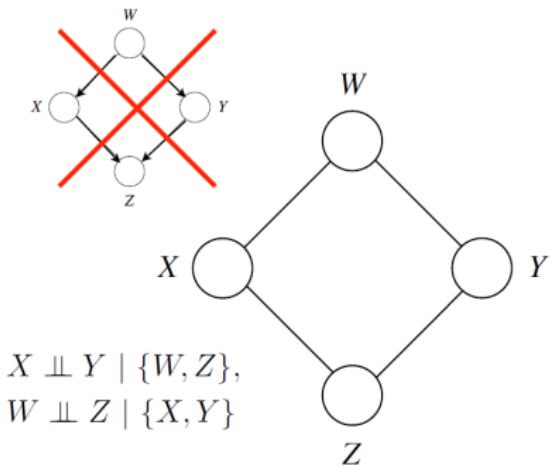
$$\phi_i(x_i, \mathbf{x}_{pa(i)}) = p(x_i | \mathbf{x}_{pa(i)})$$

- So, converting a hidden Markov model to a Markov network is simple:

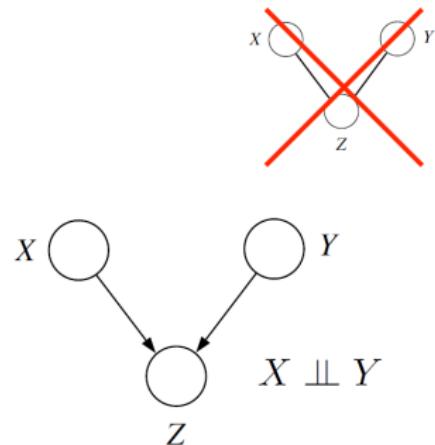


- For variables having > 1 parent, factor graph notation is useful

Representational differences



No directed representation



No undirected representation

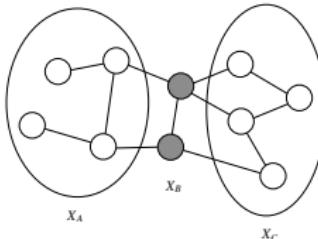
Factorization implies conditional independencies

- $p(\mathbf{x})$ is a *Gibbs distribution* over G if it can be written as

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c),$$

where the variables in each potential $c \in C$ form a clique in G

- Recall that conditional independence is given by graph separation:



- Theorem (**soundness of separation**): If $p(\mathbf{x})$ is a Gibbs distribution for G , then G is an I-map for $p(\mathbf{x})$, i.e. $I(G) \subseteq I(p)$

Proof sketch: Suppose **B** separates **A** from **C**. Then we can write

$$p(\mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C) = \frac{1}{Z} f(\mathbf{X}_A, \mathbf{X}_B) g(\mathbf{X}_B, \mathbf{X}_C).$$

Conditional independencies implies factorization

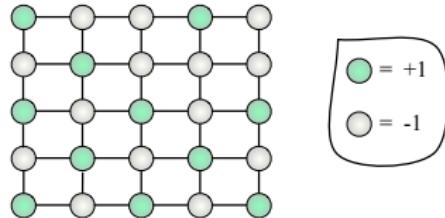
- Theorem (**soundness of separation**): If $p(\mathbf{x})$ is a Gibbs distribution for G , then G is an I-map for $p(\mathbf{x})$, i.e. $I(G) \subseteq I(p)$
- What about the converse? We need one more assumption:
- A distribution is **positive** if $p(\mathbf{x}) > 0$ for all \mathbf{x}
- Theorem (**Hammersley-Clifford**, 1971): If $p(\mathbf{x})$ is a positive distribution and G is an I-map for $p(\mathbf{x})$, then $p(\mathbf{x})$ is a Gibbs distribution that factorizes over G
- Proof is in book (as is counter-example for when $p(\mathbf{x})$ is not positive)
- This is important for **learning**:
 - Global distributions that emerge from local interactions can be characterized and analyzed

Summary

- Undirected models:
 - Graph-based data structure to specify a distribution in factored form
 - Way to define conditional independencies based on a graph structure
 - Equivalent for positive distributions
 - Bayesian network can be seen as Gibbs distribution with $Z = 1$ (partition function)
- Useful for domains with symmetric interactions (where directionality and acyclic assumptions are not natural)
- More flexibility in the model parameters
- Less interpretable: hard to elicit models from experts
- More difficult to learn, and generally more expensive computationally

Example: Ising model

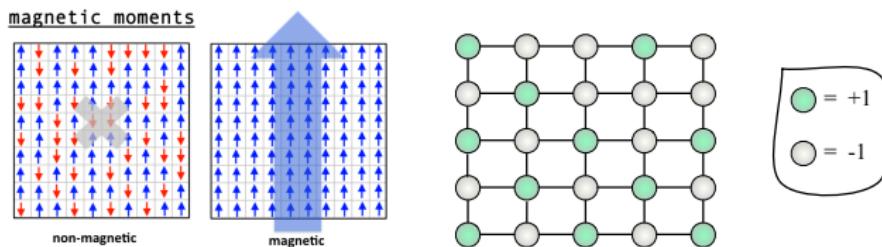
- Mathematical model of ferromagnetism in statistical mechanics. Each variable represents an atom $X_i \in \{-1, +1\}$, whose value is the direction of the atom spin
- Graph $G = (V, E)$. If atoms X_i, X_j are close to each other, then $(i, j) \in E$. Typically G is a grid or a 3D lattice.
- The spin of an atom is biased by the spins of atoms nearby on the material:



$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_i \phi_i(x_i) \prod_{(i,j) \in E} \phi_{ij}(x_i, x_j)$$

Example: Ising model

- Mathematical model of ferromagnetism in statistical mechanics. Each variable represents an atom $X_i \in \{-1, +1\}$, whose value is the direction of the atom spin
- Basic problem: study the probability distribution $p(x_1, \dots, x_n)$. How do samples from this distribution look like?



- Answer depends on the factors ϕ_i, ϕ_{ij} ! It's an inference problem.

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 5, January 24, 2017

Today's lecture

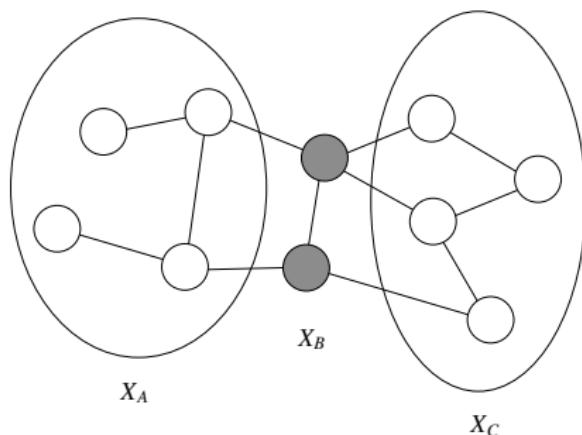
- Finish undirected models
- Conditional models
 - ① Discriminative versus generative classifiers
 - ② Conditional random fields

Independences in Markov Nets?

- In Bayes Nets
 - Local independencies: $X \perp NonDescendants(X) \mid Parents_X$
 - Factorization Theorem: p factorizes over $G \Leftrightarrow G$ is an I-map
 - Global independences: d-separation
 - Soundness (and completeness) of d-separation
- How about Markov Nets?
 - What's the analog of the Local independencies?
 - Is there a factorization theorem for Markov Nets?
 - What replaces d-separation?

Markov network structure implies conditional independencies

- Let G be the undirected graph. Note we must have one edge for every pair of variables that appear together in a potential
- Conditional independence is given by **graph separation!**



- $X_A \perp X_C | X_B$ if there is no path from $a \in A$ to $c \in C$ after removing all variables in B

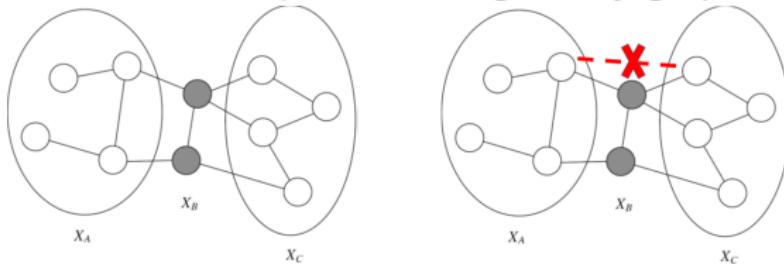
Factorization implies conditional independencies

- $p(\mathbf{x})$ is a *Gibbs distribution* over G if it can be written as

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c),$$

where the variables in each potential $c \in C$ form a clique in G

- Recall that conditional independence is given by graph separation:



- Theorem (**soundness of separation**): If $p(\mathbf{x})$ is a Gibbs distribution for G , then G is an I-map for $p(\mathbf{x})$, i.e. $I(G) \subseteq I(p)$
Proof sketch: Suppose **B** separates **A** from **C**. Then we can write

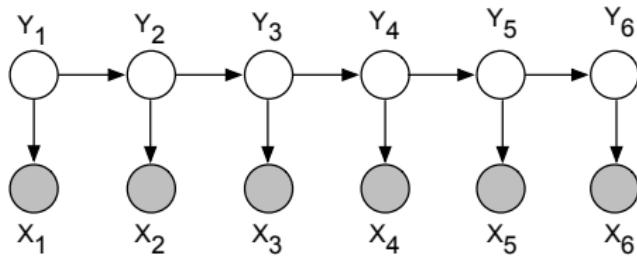
$$p(\mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C) = \frac{1}{Z} f(\mathbf{X}_A, \mathbf{X}_B) g(\mathbf{X}_B, \mathbf{X}_C).$$

Conditional independencies implies factorization

- Theorem (**soundness of separation**): If $p(\mathbf{x})$ is a Gibbs distribution for G , then G is an I-map for $p(\mathbf{x})$, i.e. $I(G) \subseteq I(p)$
- What about the converse? We need one more assumption:
- A distribution is **positive** if $p(\mathbf{x}) > 0$ for all \mathbf{x}
- Theorem (**Hammersley-Clifford**, 1971): If $p(\mathbf{x})$ is a positive distribution and G is an I-map for $p(\mathbf{x})$, then $p(\mathbf{x})$ is a Gibbs distribution that factorizes over G
- Proof is in book (as is counter-example for when $p(\mathbf{x})$ is not positive)
- This is important for **learning**:
 - Prior knowledge is often in the form of conditional independencies (i.e., a graph structure G)
 - Global distributions that emerge from local interactions can be characterized and analyzed

Converting BNs to Markov networks

What is the equivalent Markov network for a hidden Markov model?



Many inference algorithms are more conveniently given for undirected models – this shows how they can be applied to Bayesian networks

Moralization of Bayesian networks

- Procedure for converting a Bayesian network into a Markov network
- The **moral graph** $\mathcal{M}[G]$ of a BN $G = (V, E)$ is an undirected graph over V that contains an undirected edge between X_i and X_j if
 - ① there is a directed edge between them (in either direction)
 - ② X_i and X_j are both parents of the same node

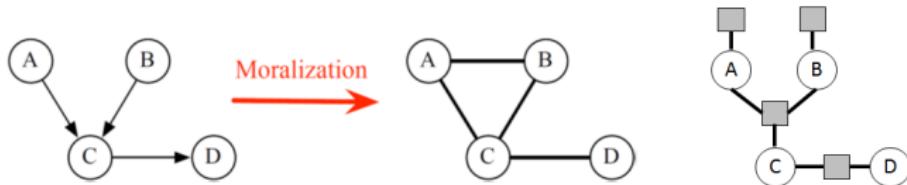


(term historically arose from the idea of “marrying the parents” of the node)

- Without $A - B$ edge, $A \perp B | C$ in the Markov Network (not true in the BN)
- Moralization leads to the loss of some independence information, e.g., $A \rightarrow C \leftarrow B$, where $A \perp B$ is lost after moralization

Converting BNs to Markov networks

- ① Moralize the directed graph to obtain the undirected graphical model:



- ② Introduce one potential function for each CPD:

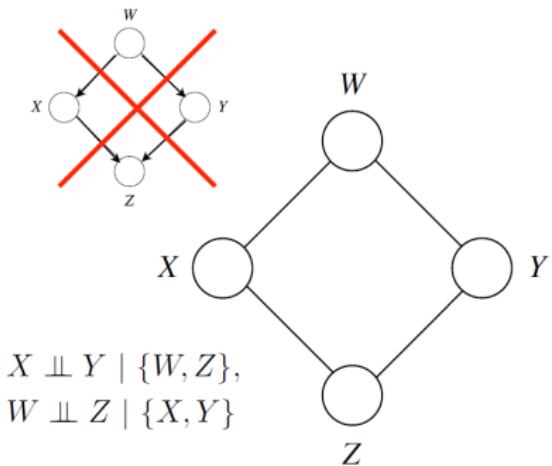
$$\phi_i(x_i, \mathbf{x}_{pa(i)}) = p(x_i | \mathbf{x}_{pa(i)})$$

- ③ What is the normalization constant Z (partition function)? 1

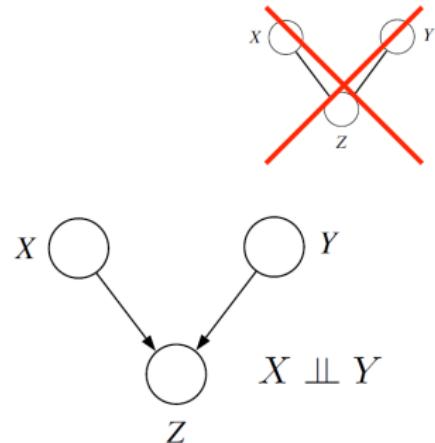
- So, converting a hidden Markov model to a Markov network is simple:



Representational differences



No directed representation



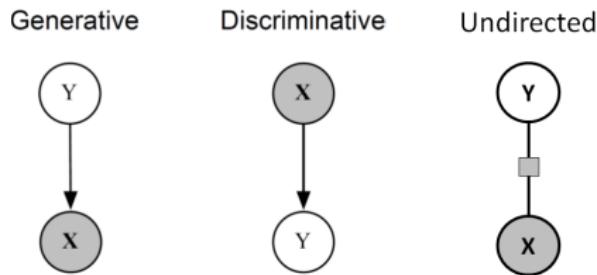
No undirected representation

Summary

- Undirected models:
 - Graph-based data structure to specify a distribution in factored form
 - Way to define conditional independencies based on a graph structure
 - These are equivalent for positive distributions
 - Bayesian network can be seen as Gibbs distribution with $Z = 1$ (partition function)
- Useful for domains with symmetric interactions (where directionality and acyclic assumptions are not natural)
- More flexibility in the model parameters
- Less interpretable: hard to elicit models from experts

Modeling choices

- Suppose you want to build a joint model over labels Y and features X
- There is often significant flexibility in choosing the structure and parameterization of a graphical model

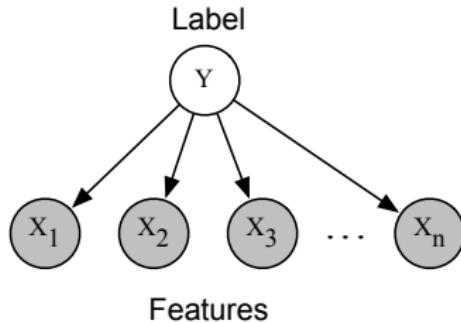


It is important to understand the trade-offs

- In the next few slides, we will study this question in the context of e-mail classification

From lecture 1... naive Bayes for single label prediction

- Classify e-mails as spam ($Y = 1$) or not spam ($Y = 0$)
 - Let $1 : n$ index the words in our vocabulary (e.g., English)
 - $X_i = 1$ if word i appears in an e-mail, and 0 otherwise
 - E-mails are drawn according to some distribution $p(Y, X_1, \dots, X_n)$
- Words are conditionally independent given Y :

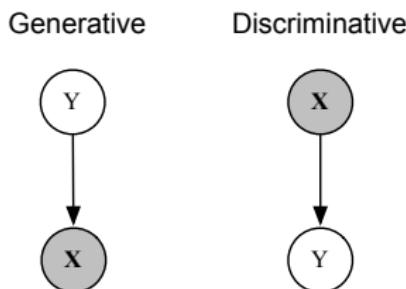


- Prediction given by:

$$p(Y = 1 | x_1, \dots, x_n) = \frac{p(Y = 1) \prod_{i=1}^n p(x_i | Y = 1)}{\sum_{y=\{0,1\}} p(Y = y) \prod_{i=1}^n p(x_i | Y = y)}$$

Discriminative versus generative models

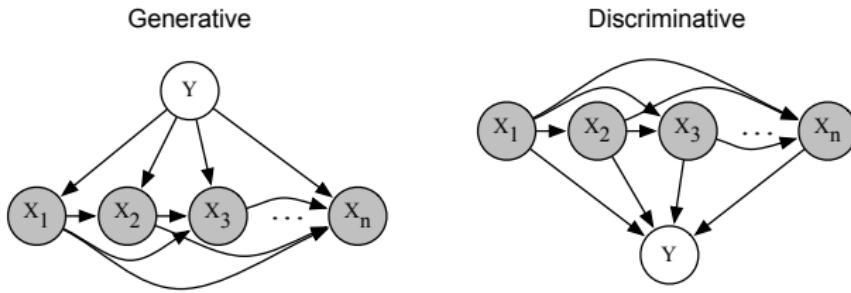
- Recall that these are **equivalent** models of $p(Y, \mathbf{X})$:



- However, suppose all we need for prediction is $p(Y | \mathbf{X})$
- In the left model, we need to estimate *both* $p(Y)$ and $p(\mathbf{X} | Y)$
- In the right model, it suffices to estimate just the **conditional distribution** $p(Y | \mathbf{X})$
 - We never need to use/estimate $p(\mathbf{X})$!
 - Called a **discriminative** model because it is only useful for discriminating Y 's label

Discriminative versus generative models

- Let's go a bit deeper to understand what are the trade-offs inherent in each approach
- Since \mathbf{X} is a random vector, for $Y \rightarrow \mathbf{X}$ to be equivalent to $\mathbf{X} \rightarrow Y$, we must have:

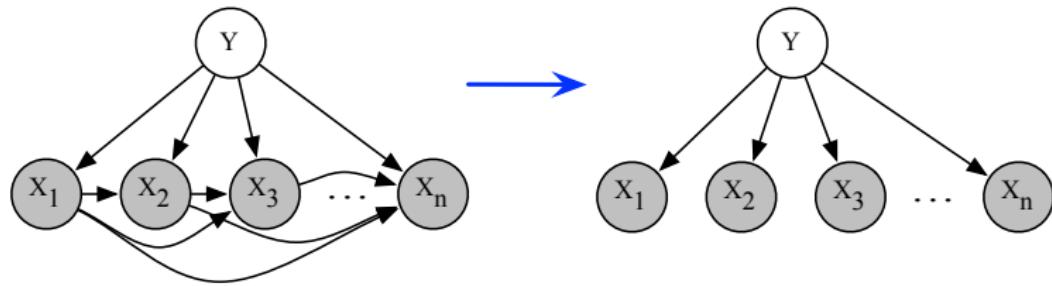


We must make the following choices:

- In the generative model, how do we parameterize $p(X_i | \mathbf{X}_{pa(i)}, Y)$?
- In the discriminative model, how do we parameterize $p(Y | \mathbf{X})$?

Naive Bayes

- ① For the generative model, assume that $X_i \perp \mathbf{X}_{-i} \mid Y$ (**naive Bayes**)



Logistic regression

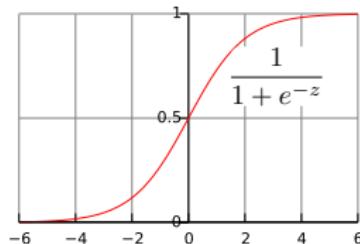
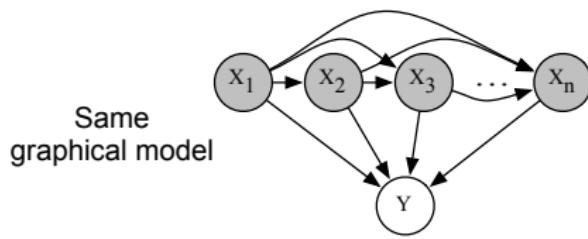
- ① For the discriminative model, assume that

$$p(Y = 1 \mid \mathbf{x}; \boldsymbol{\alpha}) = f(\mathbf{x}, \boldsymbol{\alpha})$$

- ② Not represented as a table anymore. It is a parameterized function of \mathbf{x} (regression)

- Has to be between 0 and 1
- Depend in some *simple* but reasonable way on x_1, \dots, x_n
- Completely specified by a vector $\boldsymbol{\alpha}$ of $n + 1$ parameters (compact representation)

Linear dependence: let $z(\boldsymbol{\alpha}, \mathbf{x}) = \alpha_0 + \sum_{i=1}^n \alpha_i x_i$. Then,
 $p(Y = 1 \mid \mathbf{x}; \boldsymbol{\alpha}) = f(z(\boldsymbol{\alpha}, \mathbf{x}))$, where $f(z) = 1/(1 + e^{-z})$ is called the **logistic function**:



Discriminative versus generative models

- ① For the generative model, assume that $X_i \perp \mathbf{X}_{-i} \mid Y$ (**naive Bayes**)
- ② For the discriminative model, assume that

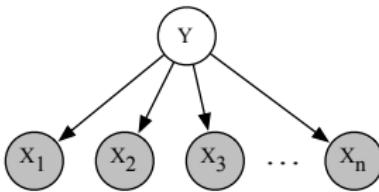
$$p(Y = 1 \mid \mathbf{x}; \alpha) = \frac{e^{\alpha_0 + \sum_{i=1}^n \alpha_i x_i}}{1 + e^{\alpha_0 + \sum_{i=1}^n \alpha_i x_i}} = \frac{1}{1 + e^{-\alpha_0 - \sum_{i=1}^n \alpha_i x_i}}$$

- It can be shown that **assumption 1 \Rightarrow assumption 2**
 - Exercise: assume $p(x_i|y) \sim \mathcal{N}(\mu_{iy}, \sigma_i)$, $p(y) \sim \text{Bernoulli}(\pi)$. Find $P(y|x_1, \dots, x_n)$. Show it has logistic form.
- Thus, every conditional distribution that can be represented using naive Bayes can *also* be represented using the logistic model
- What can we conclude from this?

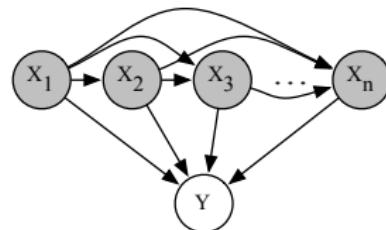
With a large amount of training data, logistic regression will perform at least as well as naive Bayes!

Discriminative models are powerful

Generative (naive Bayes)



Discriminative (logistic regression)



- Logistic model does *not* assume $X_i \perp \mathbf{X}_{-i} \mid Y$, unlike naive Bayes
- This can make a big difference in many applications
- For example, in spam classification, let $X_1 = 1[\text{"bank"} \text{ in e-mail}]$ and $X_2 = 1[\text{"account"} \text{ in e-mail}]$
- Regardless of whether spam, these always appear together, i.e. $X_1 = X_2$
- Learning in naive Bayes results in $p(X_1 \mid Y) = p(X_2 \mid Y)$. Thus, naive Bayes **double counts the evidence**
- Learning with logistic regression sets $\alpha_1 = 0$ or $\alpha_2 = 0$, in effect ignoring it

Generative models are still very useful

- ① Using a conditional model is only possible when \mathbf{X} is always observed
 - When some X_i variables are unobserved, the generative model allows us to compute $p(Y | \mathbf{X}_e)$ by marginalizing over the unseen variables
- ② Estimating the generative model using maximum likelihood is more **efficient** (statistically) than discriminative training
 - Amount of training data needed to get close to infinite data solution
 - **Naive Bayes converges more quickly to its (perhaps less helpful) asymptotic estimates**
 - We will return to these questions in the second half of the course

Structured prediction

- **Standard Machine Learning:**

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

inputs \mathcal{X} are complex, output y is a real number (classification, regression, etc.)

- **Structured Output Learning:**

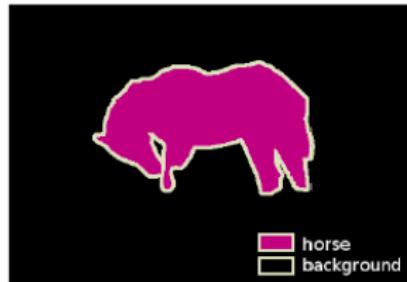
$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

inputs \mathcal{X} are complex , outputs \mathcal{Y} are also complex (structured) objects (sentences, parse trees, segmentations, etc.)

Computer Vision Example



input: images



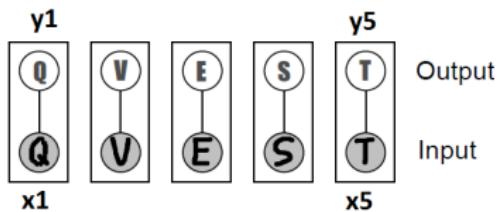
output: segmentation masks

- input space $\mathcal{X} = \{\text{color images}\} = [0, 255]^{3 \cdot M \cdot N}$
- output space $\mathcal{Y} = \{\text{segmentations}\} = \{0, 1\}^{M \times N}$
- (structured output) prediction function: $f : \mathcal{X} \rightarrow \mathcal{Y}$

$$f(x) = \arg \max_{y \in \mathcal{Y}} P(y|x)$$

Handwriting Recognition Example

- input space \mathcal{X} = 5-letter word images , $\mathbf{x} = (x_1, \dots, x_5)$, $x_j \in \{0, 1\}^{300 \times 80}$
- output space \mathcal{Y} = ASCII translation , $\mathbf{y} = (y_1, \dots, y_5)$, $y_j \in \{A, \dots, Z\}$



$$f(\mathbf{x}) = \arg \max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y} | \mathbf{x})$$

- How to build complex (discriminative) models, i.e., how to represent complex conditional probability distributions?
- We don't want to model $p(\mathbf{x})$, i.e., how handwritten characters look like!
We just want to *discriminate* over possible \mathbf{y}

Conditional random fields (CRFs)

- **Conditional random fields** (CRF) are undirected graphical models of conditional distributions $p(\mathbf{Y} \mid \mathbf{X})$
 - \mathbf{Y} is a set of **target variables**
 - \mathbf{X} is a set of **observed variables**
- Potentials are a function of \mathbf{X} and \mathbf{Y}

Formal definition

- A CRF is a Markov network on variables $\mathbf{X} \cup \mathbf{Y}$, which specifies the conditional distribution

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in C} \phi_c(\mathbf{x}_c, \mathbf{y}_c)$$

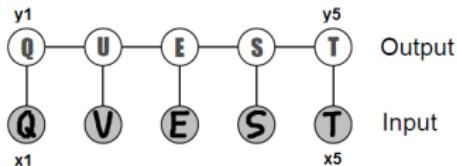
with partition function

$$Z(\mathbf{x}) = \sum_{\hat{\mathbf{y}}} \prod_{c \in C} \phi_c(\mathbf{x}_c, \hat{\mathbf{y}}_c).$$

- As before, two variables in the graph are connected with an undirected edge if they appear together in the scope of some factor
- The only difference with a standard Markov network is the normalization term – before sum over \mathbf{X} and \mathbf{Y} , now only over \mathbf{Y}

Handwriting Recognition Example

- input space \mathcal{X} = 5-letter word images , $\mathbf{x} = (x_1, \dots, x_5)$, $x_j \in \{0, 1\}^{300 \times 80}$
- output space \mathcal{Y} = ASCII translation , $\mathbf{y} = (y_1, \dots, y_5)$, $y_j \in \{A, \dots, Z\}$



- Potentials: $\phi_1(x_1, y_1), \dots, \phi_5(x_5, y_5)$, and $\phi_{12}(y_1, y_2), \dots, \phi_{45}(y_4, y_5)$

$$\begin{aligned} f(\mathbf{x}) &= \arg \max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y} | \mathbf{x}) \\ &= \arg \max_{\mathbf{y} \in \mathcal{Y}} \phi_1(x_1, y_1) \cdots \phi_5(x_5, y_5) \cdot \phi_{12}(y_1, y_2) \cdots \phi_{45}(y_4, y_5) / Z(x) \end{aligned}$$

- Can we use a table representation? How to represent compactly?

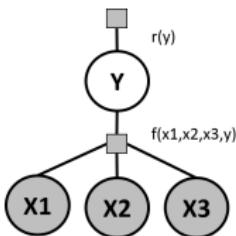
Parameterization of CRFs

- Factors may depend on a large number of variables (discrete and continuous)
- We typically parameterize each factor as a log-linear function,

$$\phi_c(\mathbf{x}_c, \mathbf{y}_c) = \exp\{\mathbf{w}_c \cdot \mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)\}$$

- $\mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)$ is a feature vector.

Example: Logistic Regression as a CRF



- CRF definition: $p(y|x_1, x_2, x_3) = \frac{1}{Z(x_1, x_2, x_3)} r(y) f(x_1, x_2, x_3, y)$
- To get logistic regression, choose
 - $r(y) = \exp(\alpha_0 1[y=1])$
 - $f(x_1, x_2, x_3, y) = \exp((\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3) 1[y=1])$
- $p(Y=1|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3)$
- $p(Y=0|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(0)$
- The normalization constant is $Z(\mathbf{x}) = 1 + \exp(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3)$. This gives the logistic regression formula for $p(Y=1|\mathbf{x})$!

Parameterization of CRFs

- Factors may depend on a large number of variables
- We typically parameterize each factor as a log-linear function,

$$\phi_c(\mathbf{x}_c, \mathbf{y}_c) = \exp\{\mathbf{w}_c \cdot \mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)\}$$

- $\mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)$ is a feature vector. Other examples:
 - Indicators: $f(\mathbf{x}_c, \mathbf{y}_c) = 1[x_i = 1, y_j = 2]$
 - $f(\mathbf{x}_c, \mathbf{y}_c) = (\text{Output of edge detector on } \mathbf{x}_c) \cdot 1[y_j = k]$
 - Output (final layer) of a convolutional neural network with \mathbf{x} as input
 - Is word \mathbf{x}_c capitalized?
 - Combinations of the above: $f(\mathbf{x}_c, \mathbf{y}_c) = 1[y_j = 2]x_i$
- Key idea: \mathbf{x} is always observed. Can use extremely complex features, no need to worry about modeling $p(\mathbf{x})$
- \mathbf{w} is a weight vector which is typically learned – we will discuss this extensively in later lectures

Main Strengths

- Avoid encoding dependencies between the variables in X (features)
- Allows incorporating into model rich set of observed variables, whose dependencies are complex or poorly understood
- Allows including continuous variables. Even if distributions do not have simple parametric forms
- Can incorporate domain knowledge: rich features without modeling joint distribution

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 6, January 25, 2017

Roadmap for the course

- **Representation:** How do we specify distributions that satisfy particular independence properties? ✓
- **Inference:** How can we exploit independence properties for efficient computation?
- **Learning:** How can we acquire a model from data?

How to acquire a model?

- A graphical model has two components: **graph structure** and the **associated potentials**
- Several possible ways to acquire a model:
 - Use expert knowledge to determine the graph and the potentials.
 - Use data+learning to determine the potentials, i.e., **parameter learning**.
 - Use data+learning to determine the graph, i.e., **structure learning**.
- Manual design is difficult to do and can take a long time for an expert
- We usually have access to a set of examples from the distribution we wish to model, e.g., a set of emails annotated by a human (spam or not-spam).

More rigorous definition

- Lets assume that the domain is governed by some underlying distribution p^*
- We are given a dataset \mathcal{D} of m samples from p^*
- The standard assumption is that the data instances are **independent and identically distributed (IID)**
- We are also given a family of models \mathcal{M} , and our task is to learn some “good” model $\hat{\mathcal{M}} \in \mathcal{M}$ (i.e., in this family) that defines a distribution $p_{\hat{\mathcal{M}}}$
 - For example, all Bayes nets with a given graph structure, for all possible choices of the CPD tables
- We can learn model parameters for a fixed structure, or both the structure and model parameters

Goal of learning

- The goal of learning is to return a model \hat{M} that precisely captures the distribution p^* from which our data was sampled
- This is in general not achievable because of
 - limited data only provides a rough approximation of the true underlying distribution
 - computational reasons
- Example. Suppose we represent each email with a vector X of 1000 binary features ($X_i = 1[\text{"is } i\text{-th word present?}]$), plus a binary label $y \in \{0, 1\}$. How many possible states in the model? 2^{1001} . Even 10^7 training examples provide extremely sparse coverage!
- We want to select \hat{M} to construct the "best" approximation to the underlying distribution p^*
- What is "best"?

What is “best”?

This depends on what we want to do

- ① Density estimation: we are interested in the full distribution (so later we can compute whatever conditional probabilities we want)
- ② Specific prediction tasks: we are using the distribution to make a prediction
 - Is this email spam or not?
- ③ Structure or knowledge discovery: we are interested in the model itself
 - How do some genes interact with each other?

1) Learning as density estimation

- We want to learn the full distribution so that later we can answer *any* probabilistic inference query
- In this setting we can view the learning problem as **density estimation**
- We want to construct \hat{M} as "close" as possible to p^* (recall we assume we are given a dataset \mathcal{D} of samples from p^*)
- How do we evaluate "closeness"?

KL-divergence

- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

(measures the expected number of extra bits required to describe *samples from $p(x)$* using a code based on q instead of p)

- $D(p\|q) \geq 0$ for all p, q , with equality if and only if $p = q$
- Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$

Detour on KL-divergence

- To compress, it is useful to know the probability distribution the data is sampled from
- For example, let X_1, \dots, X_{100} be samples of an unbiased coin. Roughly 50 heads and 50 tails. Optimal compression scheme is to record heads as 0 and tails as 1. In expectation, use 1 bit per sample, and cannot do better
- Suppose the coin is biased, and $P[H] \gg P[T]$. Then it's more efficient to use fewer bits on average to represent heads and more bits to represent tails, e.g.
 - Batch multiple samples together
 - Use a short sequence of bits to encode $HHHH$ (common) and a long sequence for $TTTT$ (rare).
- KL-divergence: if your data comes from p , but you use a scheme optimized for q , the divergence $D_{KL}(p||q)$ is the number of extra bits you'll need on average

1) Learning as density estimation

- We want to learn the full distribution so that later we can answer *any* probabilistic inference query
- In this setting we can view the learning problem as **density estimation**
- We want to construct \hat{M} as "close" as possible to p^* (recall we assume we are given a dataset \mathcal{D} of samples from p^*)
- How do we evaluate "closeness"?
- **KL-divergence** is one possibility:

$$\mathbf{D}(p^* || \hat{p}) = \mathbf{E}_{\mathbf{x} \sim p^*} \left[\log \left(\frac{p^*(\mathbf{x})}{\hat{p}(\mathbf{x})} \right) \right] = \sum_{\mathbf{x}} p^*(\mathbf{x}) \log \frac{p^*(\mathbf{x})}{\hat{p}(\mathbf{x})}$$

- $\mathbf{D}(P^* || \hat{P}) = 0$ iff the two distributions are the same.
- It measures the "compression loss" (in bits) of using \hat{P} instead of P^* .

Expected log-likelihood

- We can simplify this somewhat:

$$\mathbf{D}(p^* || \hat{p}) = \mathbf{E}_{\mathbf{x} \sim p^*} \left[\log \left(\frac{p^*(\mathbf{x})}{\hat{p}(\mathbf{x})} \right) \right] = \mathbf{E}_{\mathbf{x} \sim p^*} [\log p^*(\mathbf{x})] - \mathbf{E}_{\mathbf{x} \sim p^*} [\log \hat{p}(\mathbf{x})]$$

- The first term does not depend on \hat{p} .
- Then, minimizing KL divergence is equivalent to *maximizing the expected log-likelihood*

$$\mathbf{E}_{\mathbf{x} \sim p^*} [\log \hat{p}(\mathbf{x})]$$

- Asks that \hat{p} assign high probability to instances sampled from p^* , so as to reflect the true distribution
- Because of log, samples \mathbf{x} where $\hat{p}(\mathbf{x}) \approx 0$ weigh heavily in objective
- Although we can now compare models, since we are ignoring $\mathbf{H}(p^*)$, we don't know how close we are to the optimum
- Problem: In general we do not know p^* .

Maximum likelihood

- Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x} \sim p^*} [\log \hat{p}(\mathbf{x})]$$

with the *empirical log-likelihood*:

$$\mathbf{E}_{\mathcal{D}} [\log \hat{p}(\mathbf{x})] = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log \hat{p}(\mathbf{x})$$

- **Maximum likelihood learning** is then:

$$\max_{\hat{\mathcal{M}}} \quad \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log \hat{p}(\mathbf{x})$$

- Equivalently, maximize likelihood of the data $\hat{p}(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}) = \prod_{\mathbf{x} \in \mathcal{D}} \hat{p}(\mathbf{x})$

Example

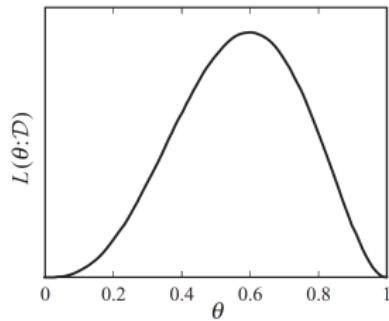
Single variable example: A biased coin

- Two outcomes: *heads* (h) and *tails* (t)
- Data set: Tosses of the biased coin
- Assumption: the process is controlled by a probability distribution $p^*(x)$ where $x \in \{h, t\}$
- Class of models \mathcal{M} : all probability distributions over $\{h, t\}$.
- Example learning task: How should we choose $\hat{p}(x)$ from \mathcal{M} if 60 out of 100 tosses are heads?

MLE scoring for the coin example

We represent our model: $\hat{p}(x = H) = \theta$ and $\hat{p}(x = T) = 1 - \theta$

- Example data: $\mathcal{D} = \{H, H, T, H, T\}$
- Likelihood of data = $\prod_i \hat{p}(x_i) = \theta \cdot \theta \cdot (1 - \theta) \cdot \theta \cdot (1 - \theta)$



- Optimize for θ which makes \mathcal{D} most likely. What is the solution in this case?

MLE scoring for the coin example: Analytical derivation

Distribution: $\hat{p}(x = H) = \theta$ and $\hat{p}(x = T) = 1 - \theta$

- More generally, log-Likelihood function

$$\begin{aligned}L(\theta) &= \theta^{\#\text{heads}} \cdot (1 - \theta)^{\#\text{tails}} \\ \log L(\theta) &= \log(\theta^{\#\text{heads}} \cdot (1 - \theta)^{\#\text{tails}}) \\ &= \#\text{heads} \cdot \log(\theta) + \#\text{tails} \cdot \log(1 - \theta)\end{aligned}$$

- MLE Goal: Find θ^* such that $\log L(\theta^*)$ is maximum.
- Differentiate the likelihood function with respect to θ and set the derivative to zero. We get:

$$\theta^* = \frac{\#\text{heads}}{\#\text{heads} + \#\text{tails}}$$

MLE scoring for the multinomial case

Distribution for a dice: $\hat{p}(x = i) = \theta_i$ for $i = 1, 2, \dots, 6$

- Log-Likelihood function

$$\begin{aligned}\log L(\theta) &= \log(\theta_1^{\#ones} \cdots \theta_6^{\#sixes}) \\ &= \#ones \cdot \log(\theta_1) + \cdots + \#sixes \cdot \log(\theta_6)\end{aligned}$$

- MLE Goal: Find $\theta^* = (\theta_1, \dots, \theta_6)$ such that $\sum_i \theta_i = 1$ and $\log L(\theta^*)$ is maximum.
- Build Lagrangian, differentiate with respect to θ_i and set the derivative to zero. We get:

$$\theta_i^* = \frac{\#\text{samples with } i\text{th outcome}}{\#\text{total samples}}$$

- Laplace smoothing: what if a count is zero?

$$\theta_i^* = \frac{\#\text{samples with } i\text{th outcome} + \alpha}{\#\text{total samples} + 6\alpha}$$

Extending the MLE principle to a Bayesian network

Given a Bayesian network structure with n variables and factorization

$$\hat{p}(\mathbf{x}) = \prod_{i=1}^n \theta_{x_i | pa(x_i)}$$

Training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$. Maximum likelihood estimate of the parameters (the CPDs)?

- Decomposition of Likelihood function

$$L(\theta, \mathcal{D}) = \prod_{j=1}^m \hat{p}(\mathbf{x}^{(j)}) = \prod_{j=1}^m \prod_{i=1}^n \theta_{x_i^{(j)} | pa(x_i^{(j)})} = \prod_{i=1}^n \prod_{j=1}^m \theta_{x_i^{(j)} | pa(x_i^{(j)})}$$

- Goal : maximize $\max_{\theta} L(\theta, \mathcal{D})$
- Constraints? $\theta_{x_i | pa(x_i)} \geq 0$; for each i , for each possible $pa(x_i)$,
 $\sum_{x_i} \theta_{x_i | pa(x_i)} = 1$

Extending the MLE principle to a Bayesian network

Bayesian network $\hat{p}(\mathbf{x}) = \prod_{i=1}^n \theta_{x_i|pa(x_i)}$, data $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$

$$L(\theta, \mathcal{D}) = \prod_{i=1}^n \prod_{j=1}^m \theta_{x_i^{(j)}|pa(x_i^{(j)})}$$

- Let $\#(x_i, pa(x_i))$ be the number of times the tuple $(x_i, pa(x_i))$ appears in the data set. We can write Likelihood function as:

$$L(\theta, \mathcal{D}) = \prod_{i=1}^n \prod_{pa(x_i)} \prod_{x_i} \theta_{x_i|pa(x_i)}^{\#(x_i, pa(x_i))}$$

$$\max_{\theta} L(\theta, \mathcal{D}) = \prod_{i=1}^n \underbrace{\max_{\theta_i} \prod_{pa(x_i)} \prod_{x_i} \theta_{x_i|pa(x_i)}^{\#(x_i, pa(x_i))}}_{\text{Only depends on CPD of } x_i}$$

- Maximization of the (log) likelihood function decomposes into separate maximizations for the local conditional distributions!

Extending the MLE principle to a Bayesian network

Bayesian network $\hat{p}(\mathbf{x}) = \prod_{i=1}^n \theta_{x_i | pa(x_i)}$, data $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$

$$L(\theta, \mathcal{D}) = \prod_{i=1}^n \prod_{j=1}^m \theta_{x_i^{(j)} | pa(x_i^{(j)})}$$

- Let $\#(x_i, pa(x_i))$ be the number of times the tuple $(x_i, pa(x_i))$ appears in the data set. We can write Likelihood function as:

$$L(\theta, \mathcal{D}) = \prod_{i=1}^n \prod_{pa(x_i)} \prod_{x_i} \theta_{x_i | pa(x_i)}^{\#(x_i, pa(x_i))}$$

$$\max_{\theta} L(\theta, \mathcal{D}) = \prod_{i=1}^n \prod_{pa(x_i)} \underbrace{\max_{\theta_{pa(x_i)}} \prod_{x_i} \theta_{x_i | pa(x_i)}^{\#(x_i, pa(x_i))}}_{\text{one row of the CPD}}$$

- Maximization of the (log) likelihood function decomposes into separate maximizations for the local conditional distributions!

Extending the MLE principle to a Bayesian network

Bayesian network $\hat{p}(\mathbf{x}) = \prod_{i=1}^n \theta_{x_i|pa(x_i)}$, data $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$

- Given **fully observed** data \mathcal{D} , MLE solution is:

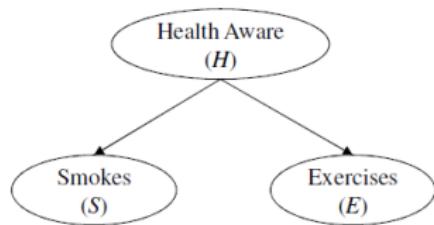
$$\theta_{x_i|pa(x_i)}^* = \frac{\#(x_i, pa(x_i))}{\#(pa(x_i))}$$

where $\#(x_i, pa(x_i))$ is the number of times the tuple $(x_i, pa(x_i))$ appears in \mathcal{D} . $\#(pa(x_i))$ is the number of times the tuple $pa(x_i)$ appears in \mathcal{D} .

- It reduces to standard multinomial estimation per local conditional

MLE Learning example: Bayesian network

Suppose we have a network with 3 binary variables: H, S, E



(a) network structure

Case	H	S	E
1	T	F	T
2	T	F	T
3	F	T	F
4	F	F	T
5	T	F	F
6	T	F	T
7	F	F	F
8	T	F	T
9	T	F	T
10	F	F	T
11	T	F	T
12	T	T	T
13	T	F	T
14	T	T	T
15	T	F	T
16	T	F	T

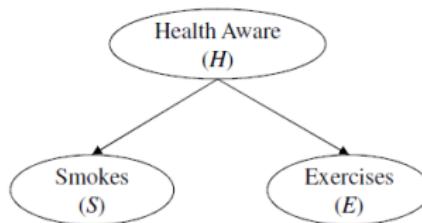
(b) complete data

H	S	E	$\Pr_{\mathcal{D}}(.)$
T	T	T	2/16
T	T	F	0/16
T	F	T	9/16
T	F	F	1/16
F	T	T	0/16
F	T	F	1/16
F	F	T	2/16
F	F	F	1/16

(c) empirical distribution

MLE Learning example: Bayesian network

H	S	E	$\Pr_{\mathcal{D}}(.)$
T	T	T	$2/16$
T	T	F	$0/16$
T	F	T	$9/16$
T	F	F	$1/16$
F	T	T	$0/16$
F	T	F	$1/16$
F	F	T	$2/16$
F	F	F	$1/16$



We have the following parameter estimates:

H	θ_H^{ml}
h	$3/4$
\bar{h}	$1/4$

H	S	$\theta_{S H}^{ml}$
h	s	$1/6$
h	\bar{s}	$5/6$
\bar{h}	s	$1/4$
\bar{h}	\bar{s}	$3/4$

H	E	$\theta_{E H}^{ml}$
h	e	$11/12$
h	\bar{e}	$1/12$
\bar{h}	e	$1/2$
\bar{h}	\bar{e}	$1/2$

2) Likelihood, Loss and Risk

- We now generalize by introducing the concept of a **loss function**
- A **loss function** $\text{loss}(\mathbf{x}, \mathcal{M})$ measures the loss that a model \mathcal{M} makes on a particular instance \mathbf{x}
- Assuming training dataset sampled from some distribution p^* , our goal is to find the model that minimizes the **expected loss** or **risk**,

$$\mathbf{E}_{\mathbf{x} \sim p^*} [\text{loss}(\mathbf{x}, \mathcal{M})]$$

- What is the loss function which corresponds to maximum likelihood estimation? Log-loss,

$$\text{loss}(\mathbf{x}, \hat{\mathcal{M}}) = -\log \hat{p}(\mathbf{x}).$$

- p^* is unknown, but we can approximate the expectation using the empirical average, i.e., **empirical risk**

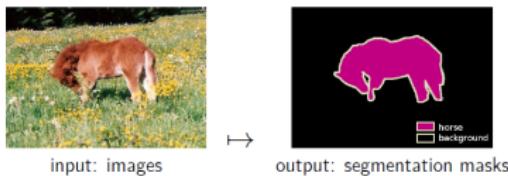
$$\mathbf{E}_{\mathcal{D}} [\text{loss}(\mathbf{x}, \hat{\mathcal{M}})] = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \text{loss}(\mathbf{x}, \hat{\mathcal{M}})$$

Example: conditional log-likelihood

- Suppose we want to predict a set of variables \mathbf{Y} given some others \mathbf{X} , e.g., for segmentation or stereo vision
- We concentrate on predicting $p(\mathbf{Y}|\mathbf{X})$, and use a **conditional** loss function

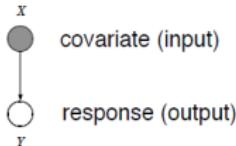
$$\text{loss}(\mathbf{x}, \mathbf{y}, \hat{\mathcal{M}}) = -\log \hat{p}(\mathbf{y} | \mathbf{x}).$$

- Since the loss function only depends on $\hat{p}(\mathbf{y} | \mathbf{x})$, suffices to estimate the conditional distribution, not the joint
- This is the objective function we use to train conditional random fields (CRFs) and logistic regression classifiers

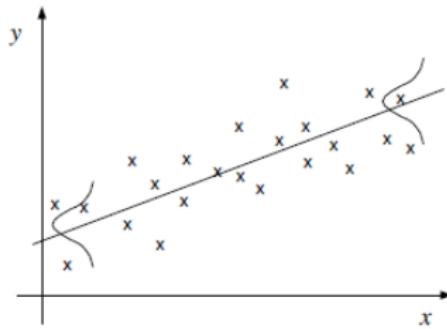


Example: linear regression

- Want to model the relationship between some covariates and an output value (continuous)



- We have some training data $\mathcal{D} = \{(x_1, y_1), \dots, (x_m, y_m)\}$



- Could try to model $p(X, Y)$, but it is easier to just model $p(Y|X)$
- Models: $p(Y|X = x; \alpha, \beta)$ is Gaussian, with mean $\alpha + \beta x$ and variance σ^2

Example: linear regression

- Can be generalized to multiple covariates: $p(Y|\mathbf{X} = \mathbf{x})$ is Gaussian, with mean $\theta\mathbf{x}$ and variance σ^2
- The (conditional) likelihood of the data \mathcal{D} is

$$\begin{aligned} p(y_1, \dots, y_m | x_1, \dots, x_m, \theta) &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - \theta\mathbf{x}_i)^2\right) \\ &= \frac{1}{(\sqrt{2\pi\sigma^2})^m} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^m (y_i - \theta\mathbf{x}_i)^2\right) \end{aligned}$$

$$\log p(y_1, \dots, y_m | x_1, \dots, x_m, \theta) = \text{const} - \frac{1}{2\sigma^2} \sum_{i=1}^m (y_i - \theta\mathbf{x}_i)^2$$

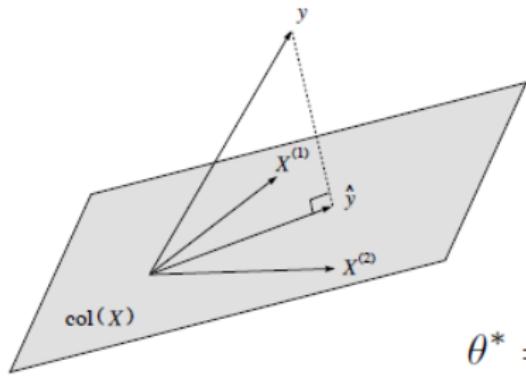
- The MLE estimate of the parameter is

$$\theta^* = \arg \min J(\theta) = \sum_{i=1}^m (y_i - \theta\mathbf{x}_i)^2$$

Normal Equations

Orthogonal projection

$$\begin{aligned}\hat{y} &= X\theta \\ X^T(y - X\theta^*) &= 0 \\ X^T X \theta^* &= X^T y\end{aligned}$$



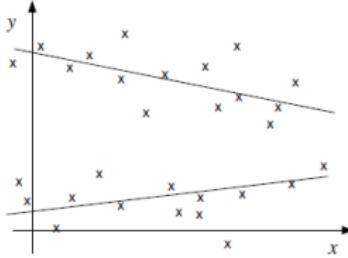
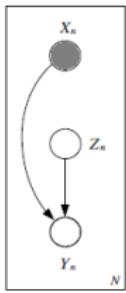
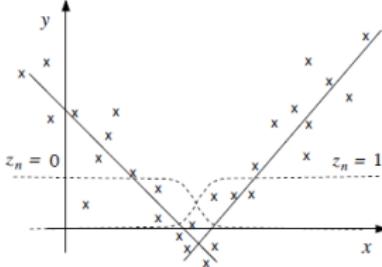
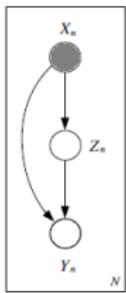
Least squares

$$\begin{aligned}J(\theta) &= \frac{1}{2} \sum_{n=1}^N e_n^2 \\ &= \frac{1}{2} \sum_{n=1}^N (y_n - \theta^T x_n)^2 \\ &= \frac{1}{2} (y - X\theta)^T (y - X\theta)\end{aligned}$$

$$\begin{aligned}J'(\theta) &= -X^T(y - X\theta) \\ X^T(y - X\theta^*) &= 0 \\ X^T X \theta^* &= X^T y\end{aligned}$$

$$\theta^* = (X^T X)^{-1} X^T y$$

More complex regression models



Empirical Risk and Overfitting

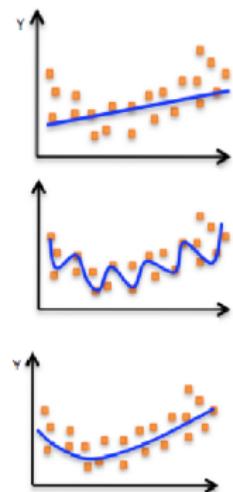
- Empirical risk minimization can easily **overfit** the data
 - Extreme example: The data is the model (remember all training data).
- Generalization: the data is a sample, usually there is vast amount of samples that you have never seen. Your model should generalize well to these “never-seen” samples.
- Thus, we typically restrict the **hypothesis space** of distributions that we search over

Bias-Variance trade off

- If the hypothesis space is very limited, it might not be able to represent p^* , even with unlimited data
 - This type of limitation is called **bias**, as the learning is limited on how close it can approximate the target distribution
- If we select a highly expressive hypothesis class, we might represent better the data
 - When we have small amount of data, multiple models can fit well, or even better than the true model. Moreover, small perturbations on \mathcal{D} will result in very different estimates
 - This limitation is call the **variance**.

Bias-Variance trade off

- There is an inherent **bias-variance trade off** when selecting the hypothesis class. Error in learning due to both things: bias and variance.
- Hypothesis space: linear relationship
 - Does it fit well? Underfits
- Hypothesis space: high degree polynomial
 - Overfits
- Hypothesis space: low degree polynomial
 - Right tradeoff



How to avoid overfitting?

- Hard constraints, e.g. by selecting a less expressive hypothesis class:
 - Bayesian networks with at most d parents
 - Pairwise MRFs (instead of arbitrary higher-order potentials)
- Soft preference for “simpler” models: **Occam Razor**.
- Augment the objective function with **regularization**:

$$\text{objective}(\mathbf{x}, \mathcal{M}) = \text{loss}(\mathbf{x}, \mathcal{M}) + R(\mathcal{M})$$

Summary of how to think about learning

- ① Figure out what you care about, e.g. expected loss

$$\mathbf{E}_{\mathbf{x} \sim P^*} [\text{loss}(\mathbf{x}, \mathcal{M})]$$

- ② Figure out how best to estimate this from what you have, i.e. data \mathcal{D} and prior knowledge. For example, use regularization

$$\mathbf{E}_{\mathcal{D}} [\text{loss}(\mathbf{x}, \mathcal{M})] + R(\mathcal{M})$$

- ③ Figure out how to optimize over this objective function, e.g. the minimization

$$\min_{\mathcal{M}} \mathbf{E}_{\mathcal{D}} [\text{loss}(\mathbf{x}, \mathcal{M})] + R(\mathcal{M})$$

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 7, January 31, 2017

Inference

Now that we know everything about representation, how do we compute something interesting with these models?

Probabilistic inference

- Suppose we have a probabilistic model (BN, MRF, CRF,...) over variables \mathcal{X}
- We are typically interested in conditional probability queries,

$$p(\mathbf{Y}|\mathbf{E} = \mathbf{e}) = \frac{p(\mathbf{Y}, \mathbf{e})}{p(\mathbf{e})}$$

(e.g., the probability that a Congressman is a Democrat \mathbf{Y} given some observed votes $\mathbf{E} = \mathbf{e}$)

- Let $\mathbf{W} = \mathcal{X} - \mathbf{Y} - \mathbf{E}$ be the random variables that are neither the query nor the evidence (e.g., the result of a vote that was not recorded). Each of these joint distributions can be computed by marginalizing over the other variables:

$$p(\mathbf{Y}, \mathbf{e}) = \sum_{\mathbf{w}} p(\mathbf{Y}, \mathbf{e}, \mathbf{w}), \quad p(\mathbf{e}) = \sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{e})$$

- Note: $\sum_{\mathbf{w}}$ means a sum over all possible values \mathbf{W} can take. For example, if $\mathbf{Y} \in \{0, 1\}$, $\sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{e}) = p(\mathbf{y} = 0, \mathbf{e}) + p(\mathbf{y} = 1, \mathbf{e})$

Other Probabilistic Inference Queries

- MAP inference (maximum a posteriori, or most probable explanation). Let $\mathbf{Y} = \mathcal{X} - \mathbf{E}$

$$MAP(\mathbf{Y}|\mathbf{E} = \mathbf{e}) = \arg \max_{\mathbf{y}} p(\mathbf{y}|\mathbf{e})$$

Most likely **joint** assignment to all non-evidence variables (optimization problem). Ex: in HW2 find most likely party affiliation and all votes that were not recorded

- Marginal MAP. Let $\mathbf{Z} = \mathcal{X} - \mathbf{Y} - \mathbf{E}$.

$$MMAP(\mathbf{Y}|\mathbf{E} = \mathbf{e}) = \arg \max_{\mathbf{y}} \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z}|\mathbf{e})$$

Involves both summation and optimization. Ex: in HW2 find most likely party affiliation, marginalizing out all votes that were not recorded

Computational complexity of probabilistic inference

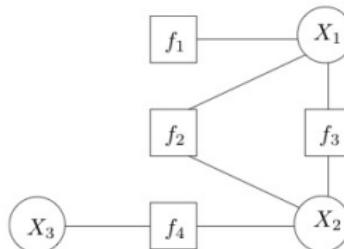
- Here we show that probabilistic inference is hard in the worst-case. Unless P=NP, there does *not* exist an efficient algorithm
- We show this by reducing 3-SAT, which is NP-complete, to probabilistic inference in Bayesian networks
- 3-SAT asks about the *satisfiability* of a logical formula defined on n binary variables X_1, \dots, X_n , e.g.

$$(\neg X_3 \vee \neg X_2 \vee X_3) \wedge (X_2 \vee \neg X_4 \vee \neg X_5) \wedge (X_1 \vee X_2 \vee X_3) \dots$$

- Each of the disjunction terms is called a *clause*, e.g. $(\neg X_3 \vee \neg X_2 \vee X_3)$ is a clause
- In 3-SAT, each clause is defined on at most 3 variables. Canonical NP complete problem.

Reducing satisfiability to MAP inference

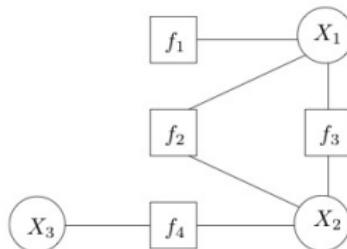
- **Input:** 3-SAT formula with n variables X_1, \dots, X_n and m clauses C_1, \dots, C_m



- One variable $X_i \in \{0, 1\}$ for each variable in the SAT formula.
- One factor for each clause f_i for each clause C_i : $f_i = 1$ if and only if C_i is satisfied
 - Example clause $C_4: X_2 \vee \neg X_3$. We add a factor $f_4(X_2, X_3) = 1$ iff $X_2 \vee \neg X_3$. More explicitly, $f_4(X_2 = 0, X_3 = 0) = 1 = f_4(1, 0) = f_4(1, 1)$, while $f_4(X_2 = 0, X_3 = 1) = 0$.
- Note: $p(X_1, \dots, X_n) > 0$ if and only if X_1, \dots, X_n satisfies all the clauses C_1, \dots, C_m

Reducing satisfiability to MAP inference

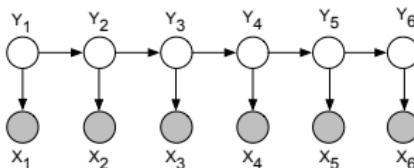
- **Input:** 3-SAT formula with n variables X_1, \dots, X_n and m clauses C_1, \dots, C_m



- Note: $p(X_1, \dots, X_n) > 0$ if and only if X_1, \dots, X_n satisfies all the clauses
- Thus, we can find a satisfying assignment (whenever one exists) by constructing this MRF and finding the MAP assignment:
 $\text{argmax}_{X_1, \dots, X_n} p(X_1, \dots, X_n)$
- This proves that MAP inference in MRFs is NP-hard (also in Bayesian Networks)
- What is the normalization constant $Z = \sum_{X_1, \dots, X_n} \prod_i f_i$? It is the total number of satisfying assignments. Also hard to compute.

Probabilistic inference in practice

- NP-hardness simply says that there **exist** difficult inference problems
- Real-world inference problems are not necessarily as hard as these worst-case instances
- Some graphs are **easy** to do inference in! For example, inference in hidden Markov models



and other tree-structured graphs can be performed in **linear time**

- In practice, due to the structure of the Bayesian network, we can sometimes cache computations that are otherwise computed exponentially many times

Distributive Law

- Two variables and a constant:

$$ax_1 + ax_2 = a(x_1 + x_2)$$

- Many variables and a constant:

$$\sum_i ax_i = a \sum_i x_i$$

- More variables and constants:

$$\begin{aligned} \sum_{i \in \{1,2\}} \sum_{j \in \{1,2\}} a_i x_i b_j y_j &= a_1 x_1 b_1 y_1 + a_1 x_1 b_2 y_2 + a_2 x_2 b_1 y_1 + a_2 x_2 b_2 y_2 \\ &= a_1 x_1 (b_1 y_1 + b_2 y_2) + a_2 x_2 (b_1 y_1 + b_2 y_2) = \sum_{i \in \{1,2\}} a_i x_i \sum_{j \in \{1,2\}} b_j y_j \end{aligned}$$

- Many variables, many constants (“push sum inside”):

$$\sum_{x_1, \dots, x_n} \prod_{i=1}^n a_{x_i} x_i = \sum_{x_1} \cdots \sum_{x_n} a_{x_1} x_1 \cdots a_{x_n} x_n = \sum_{x_1} a_{x_1} x_1 \sum_{x_2} a_{x_2} x_2 \cdots \sum_{x_n} a_{x_n} x_n$$

Variable elimination (VE)

- Exact algorithm for probabilistic inference in **any** graphical model
- Running time will depend on the *graph structure*
- Uses **distributive law** and **dynamic programming** to circumvent enumerating all possible assignments to the variables
- First we introduce the concept for computing marginal probabilities, $p(X_i)$, in Bayesian networks
- After this, we will generalize to MRFs and conditional queries

Basic idea

- Suppose we have a simple chain, $A \rightarrow B \rightarrow C \rightarrow D$, and we want to compute $p(D)$
- $p(D)$ is a **set** of values, $\{p(D = d), d \in \text{Val}(D)\}$. Algorithm computes sets of values at a time – an entire distribution
- By the chain rule and conditional independence, the joint distribution factors as

$$p(A, B, C, D) = p(A)p(B | A)p(C | B)p(D | C)$$

- In order to compute $p(D)$, we have to marginalize over A, B, C :

$$\begin{aligned} p(D) &= \sum_{a,b,c} p(A = a, B = b, C = c, D) \\ &= \sum_{a \in \text{Val}(A)} \sum_{b \in \text{Val}(B)} \sum_{c \in \text{Val}(C)} p(A = a, B = b, C = c, D) \end{aligned}$$

Let's be a bit more explicit...

$$\begin{array}{cccc} P(a^1) & P(b^1 | a^1) & P(c^1 | b^1) & P(d^1 | c^1) \\ + P(a^2) & P(b^1 | a^2) & P(c^1 | b^1) & P(d^1 | c^1) \\ + P(a^1) & P(b^2 | a^1) & P(c^1 | b^2) & P(d^1 | c^1) \\ + P(a^2) & P(b^2 | a^2) & P(c^1 | b^2) & P(d^1 | c^1) \\ + P(a^1) & P(b^1 | a^1) & P(c^2 | b^1) & P(d^1 | c^2) \\ + P(a^2) & P(b^1 | a^2) & P(c^2 | b^1) & P(d^1 | c^2) \\ + P(a^1) & P(b^2 | a^1) & P(c^2 | b^2) & P(d^1 | c^2) \\ + P(a^2) & P(b^2 | a^2) & P(c^2 | b^2) & P(d^1 | c^2) \end{array}$$

$$\begin{array}{cccc} P(a^1) & P(b^1 | a^1) & P(c^1 | b^1) & P(d^2 | c^1) \\ + P(a^2) & P(b^1 | a^2) & P(c^1 | b^1) & P(d^2 | c^1) \\ + P(a^1) & P(b^2 | a^1) & P(c^1 | b^2) & P(d^2 | c^1) \\ + P(a^2) & P(b^2 | a^2) & P(c^1 | b^2) & P(d^2 | c^1) \\ + P(a^1) & P(b^1 | a^1) & P(c^2 | b^1) & P(d^2 | c^2) \\ + P(a^2) & P(b^1 | a^2) & P(c^2 | b^1) & P(d^2 | c^2) \\ + P(a^1) & P(b^2 | a^1) & P(c^2 | b^2) & P(d^2 | c^2) \\ + P(a^2) & P(b^2 | a^2) & P(c^2 | b^2) & P(d^2 | c^2) \end{array}$$

- There is structure to the summation, e.g., repeated $P(c^1 | b^1)P(d^1 | c^1)$
- Use **distributive law!**

Using distributive law

- Our goal was to compute

$$\begin{aligned} p(D) = \sum_{a,b,c} p(a, b, c, D) &= \sum_{a,b,c} p(a)p(b | a)p(c | b)p(D | c) \\ &= \sum_c \sum_b \sum_a p(D | c)p(c | b)p(b | a)p(a) \end{aligned}$$

- We can push the summations inside (*distributive law*) to obtain:

$$p(D) = \sum_c p(D | c) \sum_b p(c | b) \underbrace{\sum_a p(b | a)p(a)}_{\underbrace{\psi_1(a, b)}_{\tau_1(b)}}$$

- What's the size of $\psi_1(A, B) = P(B|A)P(A)$? What about $\tau_1(B)$?
- We now have to compute: $p(D) = \sum_c p(D | c) \sum_b p(c | b)\tau_1(b)$

Using distributive law

- We now have to compute:

$$p(D) = \sum_c p(D | c) \sum_b p(c | b) \tau_1(b)$$

$$p(D) = \sum_c p(D | c) \sum_b \underbrace{p(c | b) \tau_1(b)}_{\psi_2(c, b)}$$

$$p(D) = \sum_c p(D | c) \underbrace{\sum_b \underbrace{p(c | b) \tau_1(b)}_{\psi_2(c, b)}}_{\tau_2(c)}$$

- We now have to compute: $p(D) = \sum_c p(D | c) \tau_2(c)$
- This procedure is dynamic programming: cache and reuse computation

Inference in a chain

- Generalizing the previous example, suppose we have a chain $X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_n$ where each variable has k states ($|Val(X_i)| = k$)
- For $i = 1$ up to $n - 1$, compute (and cache)

$$p(X_{i+1}) = \sum_{x_i \in Val(X_i)} p(X_{i+1} | x_i) p(x_i)$$

- How long does each step take? Each step takes k^2 time
- The total running time is $\mathcal{O}(nk^2)$
- In comparison, naively marginalizing over all variables has complexity $\mathcal{O}(k^n)$
- We did inference over the joint without ever explicitly constructing it!

Summary so far

- Worst-case analysis says that inference is NP-hard
- Even approximating it is NP-hard
- In practice, due to the structure of the Bayesian network, we can sometimes cache computations that are otherwise computed exponentially many times

Sum-product inference task

- Given a Bayes Net with n variables $\mathbf{X} = \mathbf{Y}, \mathbf{Z}$
- We want to give an algorithm to compute

$$p(\mathbf{y}) = \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z}) = \sum_{\mathbf{z}} \prod_{X_i \in \mathbf{X}} p(x_i | \mathbf{x}_{\text{Pa}(X_i)}) \quad \forall \mathbf{y} \in \text{Val}(\mathbf{Y})$$

- Let's think of CPDs as factors (like in an MRF)
- This can be reduced to the following **sum-product** inference task:

$$\text{Compute } \tau(\mathbf{y}) = \sum_{\mathbf{z}} \prod_{\phi_i \in \Phi} \phi_i(x_i, \mathbf{x}_{\text{Pa}(X_i)}) \quad \forall \mathbf{y},$$

where Φ is given by the conditional probability distributions (one for each variable),

$$\Phi = \{\phi_i\}_{i=1}^n = \{p(X_i | \mathbf{X}_{\text{Pa}(X_i)})\}_{i=1}^n,$$

- What is the scope of ϕ_i ? $\text{Scope}[\phi_i] = \{X_i, \text{Pa}(X_i)\}$

Sum-product inference task

- Suppose we have a Bayes Net $A \leftarrow B \rightarrow C \leftarrow D$
- Suppose we want to compute $p(C)$

$$\begin{aligned} p(c) &= \sum_{a,b,d} p(a, b, c, d) &=& \sum_{a,b,d} p(b)p(d)p(a|b)p(c|b,d) \\ & &=& \sum_{a,b,d} \phi_1(b)\phi_2(d)\phi_3(a,b)\phi_4(c,b,d) \\ & &=& \sum_{\mathbf{z}} \prod_{\phi_i \in \Phi} \phi_i(x_i, \mathbf{x}_{\text{Pa}(X_i)}) \end{aligned}$$

- In this case, $\mathbf{Z} = \{A, B, D\}$ and $\Phi = \{\phi_1(B), \phi_2(D), \phi_3(A, B), \phi_4(C, B, D)\}$
- Why are we doing this? Easier to describe algorithms in terms of operations over factors, and same algorithm will also work for MRFs!

Factor marginalization

- Recall that the local conditional distributions (in BN) and factors (in MRF) can be thought of as tables
- Summing a factor over a variable x_i can be thought of as producing a new factor
- This new factor is defined a table that no longer depends on x_i
- Efficient algorithms for inference depend on exploiting these intermediate tables

Factor marginalization

- Let $\phi(\mathbf{X}, B)$ be a factor where \mathbf{X} is a set of variables and $B \notin \mathbf{X}$
- Factor marginalization** of ϕ over B (also called “summing out B in ϕ ”) gives a new factor:

$$\tau(\mathbf{X}) = \sum_B \phi(\mathbf{X}, B)$$

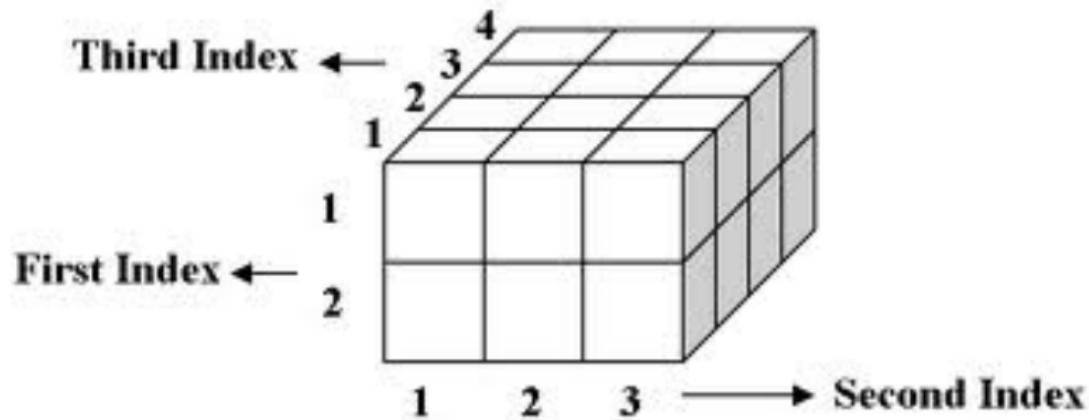
For example, marginalizing $\phi(A, B, C)$ over B gives:

a^1	b^1	c^1	0.25
a^1	b^1	c^2	0.35
a^1	b^2	c^1	0.08
a^1	b^2	c^2	0.16
a^2	b^1	c^1	0.05
a^2	b^1	c^2	0.07
a^2	b^2	c^1	0
a^2	b^2	c^2	0
a^3	b^1	c^1	0.15
a^3	b^1	c^2	0.21
a^3	b^2	c^1	0.09
a^3	b^2	c^2	0.18

a^1	c^1	0.33
a^1	c^2	0.51
a^2	c^1	0.05
a^2	c^2	0.07
a^3	c^1	0.24
a^3	c^2	0.39

A visualization

You can think of a factor $\phi(A, B, C)$ over 3 variables as a 3-D array:



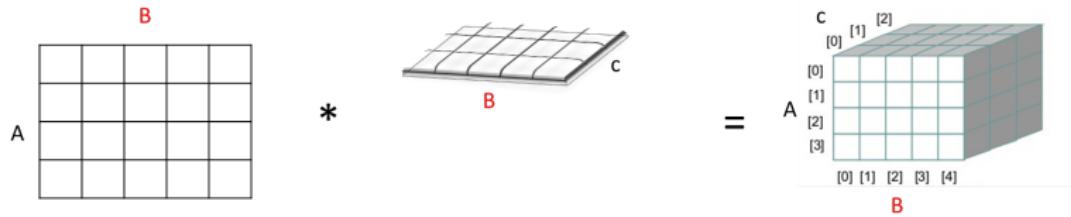
Marginalizing is equivalent to summing along one dimension. For example, by summing along the third dimension, one obtains a new 2D array of size 2×3 .

Factor product

- Let $\phi_1(A, B)$, $\phi_2(B, C)$ be two factors
- Their product $\phi_1(A, B) \cdot \phi_2(B, C)$ is a new factor $\phi_3(A, B, C)$ such that for all a, b, c

$$\phi_3(a, b, c) = \phi_1(a, b)\phi_2(b, c)$$

- Note the product $\phi_1(a, b)\phi_2(b, c)$ is just a product of two real numbers



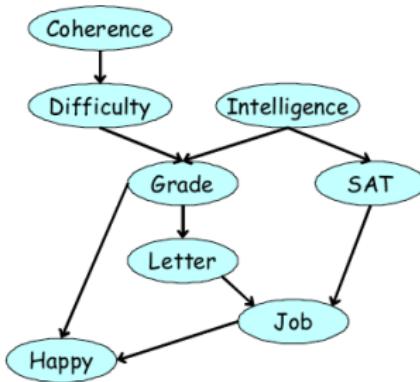
Sum-product variable elimination

- Remember our goal (marginal inference) is to compute

$$\tau(\mathbf{Y}) = \sum_{\mathbf{Z}} \prod_{\phi_c \in \Phi} \phi_c(\mathbf{X}_c)$$

- Order the variables \mathbf{Z} (called the **elimination ordering**)
- Iteratively marginalize out variable Z_i , one at a time (based on this ordering)
- For each i ,
 - Multiply all factors that have Z_i in their scope, generating a new product factor
 - Marginalize this product factor over Z_i , generating a new smaller factor
 - Add this new smaller factor to the set of all factors
 - Remove all factors that have Z_i in their scope

Example



- What is $p(\text{Job})$? Joint distribution factorizes as:

$$p(C, D, I, G, S, L, H, J) = p(C)p(D|C)p(I)p(G|D, I)p(L|G)p(S|I)p(J|S, L)p(H|J, G)$$

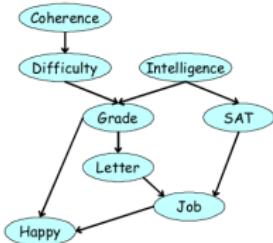
with factors

$$\Phi = \{\phi_C(C), \phi_D(C, D), \phi_I(I), \phi_G(G, D, I), \phi_L(L, G), \\ \phi_S(S, I), \phi_J(J, S, L), \phi_H(H, J, G)\}$$

- Let's do variable elimination with ordering $\{C, D, I, H, G, S, L\}$ on the board!

Elimination ordering

- We can pick any order we want, but some orderings introduce factors with much larger scope



Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D)$, $\tau_1(D)$	G, I, D	$\tau_2(G, I)$
3	I	$\phi_I(I)$, $\phi_S(S, I)$, $\tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H, G, J	$\tau_4(G, J)$
5	G	$\tau_4(G, J)$, $\tau_3(G, S)$, $\phi_L(L, G)$	G, J, L, S	$\tau_5(J, L, S)$
6	S	$\tau_5(J, L, S)$, $\phi_J(J, L, S)$	J, L, S	$\tau_6(J, L)$
7	L	$\tau_6(J, L)$	J, L	$\tau_7(J)$

- Alternative ordering...

Step	Variable eliminated	Factors used	Variables involved	New factor
1	G	$\phi_G(G, I, D)$, $\phi_L(L, G)$, $\phi_H(H, G, J)$	G, I, D, L, J, H	$\tau_1(I, D, L, J, H)$
2	I	$\phi_I(I)$, $\phi_S(S, I)$, $\tau_1(I, D, L, S, J, H)$	S, I, D, L, J, H	$\tau_2(D, L, S, J, H)$
3	S	$\phi_J(J, L, S)$, $\tau_2(D, L, S, J, H)$	D, L, S, J, H	$\tau_3(D, L, J, H)$
4	L	$\tau_3(D, L, J, H)$	D, L, J, H	$\tau_4(D, J, H)$
5	H	$\tau_4(D, J, H)$	D, J, H	$\tau_5(D, J)$
6	C	$\tau_5(D, J)$, $\phi_D(D, C)$	D, J, C	$\tau_6(D, J)$
7	D	$\tau_6(D, J)$	D, J	$\tau_7(J)$

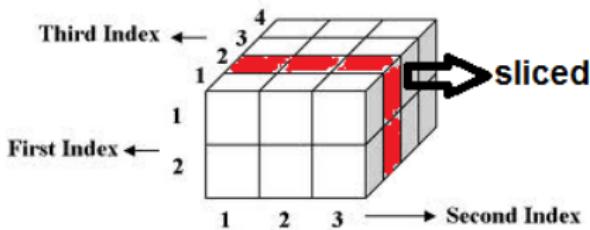
How to introduce evidence?

- Recall that our original goal was to answer *conditional* probability queries,

$$p(\mathbf{Y} | \mathbf{E} = \mathbf{e}) = \frac{p(\mathbf{Y}, \mathbf{e})}{p(\mathbf{e})}$$

- Apply variable elimination algorithm to the task of computing $P(\mathbf{Y}, \mathbf{e})$
- Replace each factor $\phi \in \Phi$ that has $\mathbf{E} \cap \text{Scope}[\phi] \neq \emptyset$ with

$$\phi'(\mathbf{x}_{\text{Scope}[\phi] - \mathbf{E}}) = \phi(\mathbf{x}_{\text{Scope}[\phi] - \mathbf{E}}, \mathbf{e}_{\mathbf{E} \cap \text{Scope}[\phi]})$$



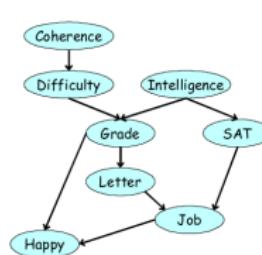
- Then, eliminate the variables in $\mathcal{X} - \mathbf{Y} - \mathbf{E}$. The returned factor $\phi^*(\mathbf{Y})$ is $p(\mathbf{Y}, \mathbf{e})$. To obtain the conditional $p(\mathbf{Y} | \mathbf{e})$, sum also over \mathbf{Y} to get $p(\mathbf{e})$.

Undirected Graphs

- Algorithm is the same in the MRF case
- Use potentials instead of CPDs
- Can be used to compute the normalization constant Z (eliminating all variables)

$$Z = \sum_{\hat{x}_1, \dots, \hat{x}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

Running time of variable elimination

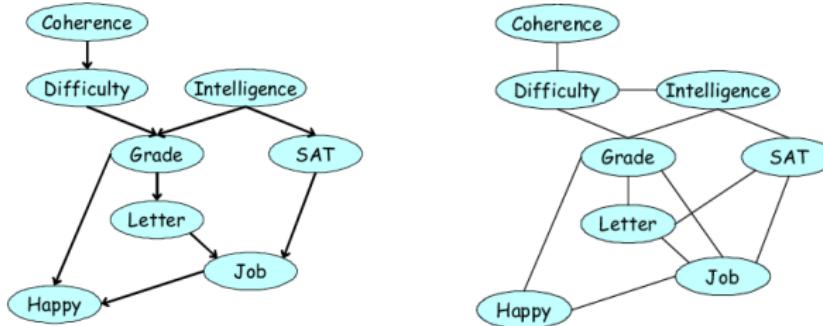


Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D)$, $\tau_1(D)$	G, I, D	$\tau_2(G, I)$
3	I	$\phi_I(I)$, $\phi_S(S, I)$, $\tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H, G, J	$\tau_4(G, J)$
5	G	$\tau_4(G, J)$, $\tau_3(G, S)$, $\phi_L(L, G)$	G, J, L, S	$\tau_5(J, L, S)$
6	S	$\tau_5(J, L, S)$, $\phi_J(J, L, S)$	J, L, S	$\tau_6(J, L)$
7	L	$\tau_6(J, L)$	J, L	$\tau_7(J)$

- Let n be the number of variables, and m the number of initial factors (for BN, $m = n$)
- At each step, we pick a variable X_i and multiply all factors involving X_i , resulting in a single factor ψ_i
- At most $n + m$ factors are ever in the set. Each factor is multiplied only once.
- Let N_i be the number of variables in the factor ψ_i , and let $N_{\max} = \max_i N_i$
- The running time of VE is then $O((n + m)v^{N_{\max}})$, where $v = \max_i |\text{Val}(X_i)|$.
- The primary concern is that N_{\max} can potentially be as large as n

Running time in graph-theoretic concepts

- Let's try to analyze the complexity in terms of the graph structure
- G_ϕ is the undirected graph with one node per variable, where there is an edge (X_i, X_j) if these appear together in the scope of some factor ϕ
- Ignoring evidence, this is either the original MRF (for sum-product VE on MRFs) or the moralized Bayesian network:



Elimination as graph transformation

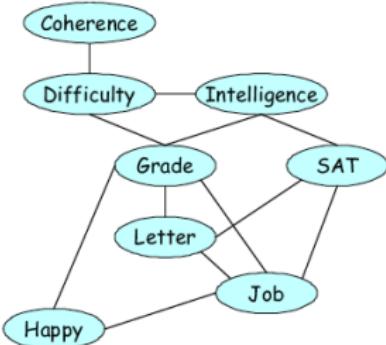
When a variable X is eliminated,

- We create a single factor ψ that contains X and all of the variables \mathbf{Y} with which X appears in factors
- We eliminate (marginalize) X from ψ , replacing it with a new factor τ that contains all of the variables \mathbf{Y} , but not X . Let's call the new set of factors Φ_X (removing all the factors containing X and adding τ)

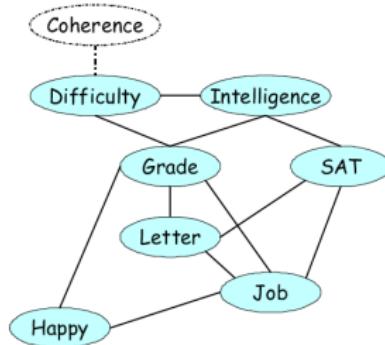
How does this modify the graph, going from G_Φ to G_{Φ_X} ?

- Constructing ψ generates edges between all of the variables $Y \in \mathbf{Y}$
- Some of these edges were already in G_Φ , some are new
- The new edges are called **fill edges**
- The step of removing X from Φ to construct Φ_X removes X and all its incident edges from the graph

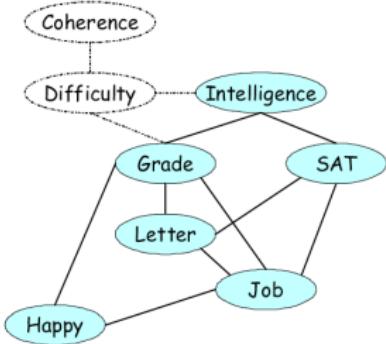
Example



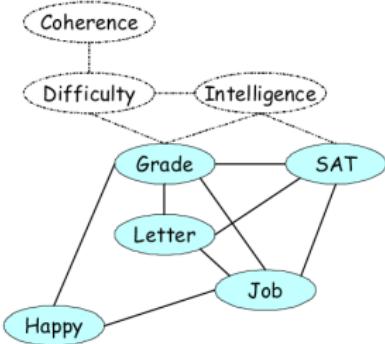
(Graph)



(Elim. C)

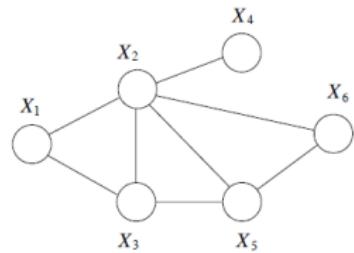
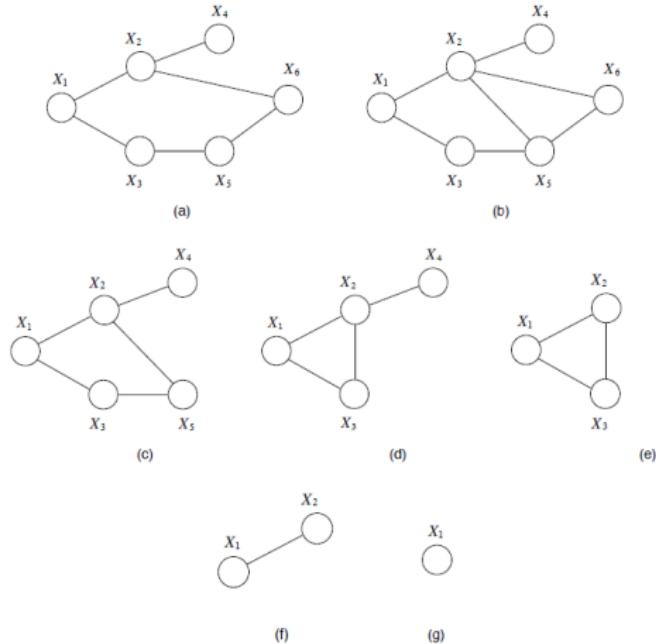


(Elim. D)



(Elim. I)

Another Example

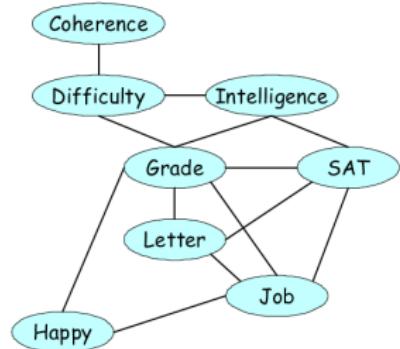
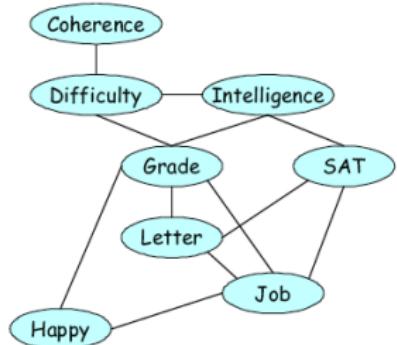


Each time a node is eliminated, all of its neighbors are connected, forming a clique

Induced graph

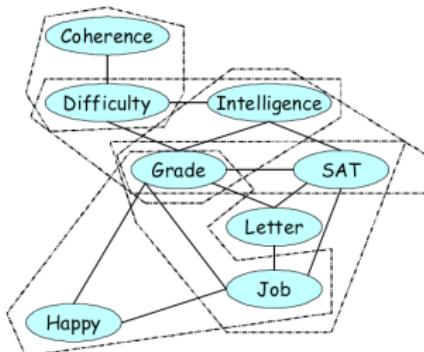
- We can summarize the computation cost using a single graph that is the union of all the graphs resulting from each step of the elimination
- We call this the **induced graph** $\mathcal{I}_{\Phi, \prec}$, where \prec is the elimination ordering

Example



(Induced graph)

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C), \phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D), \tau_1(D)$	G, I, D	$\tau_2(G, I)$
3	I	$\phi_I(I), \phi_S(S, I), \tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H, G, J	$\tau_4(G, J)$
5	G	$\tau_4(G, J), \tau_3(G, S), \phi_L(L, G)$	G, J, L, S	$\tau_5(J, L, S)$
6	S	$\tau_5(J, L, S), \phi_J(J, L, S)$	J, L, S	$\tau_6(J, L)$
7	L	$\tau_6(J, L)$	J, L	$\tau_7(J)$



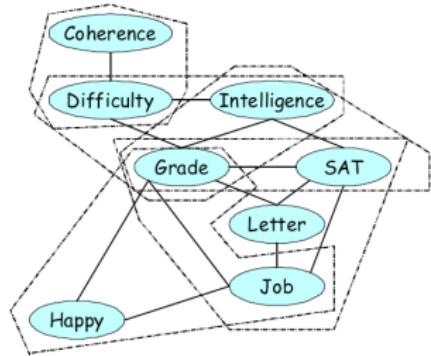
(Maximal Cliques)

Properties of the induced graph

- **Theorem:** Let $\mathcal{I}_{\Phi, \prec}$ be the induced graph for a set of factors Φ and ordering \prec , then
 - ① Every factor generated during VE has a scope that is a clique in $\mathcal{I}_{\Phi, \prec}$
 - ② Every maximal clique in $\mathcal{I}_{\Phi, \prec}$ is the scope of some intermediate factor in the computation

(see book for proof)
- Thus, N_{max} is equal to the size of the largest clique in $\mathcal{I}_{\Phi, \prec}$
- The running time, $O(mv^{N_{max}})$, is exponential in the size of the largest clique of the induced graph, $v = \max_i |\text{Val}(X_i)|$

Example



(Maximal Cliques)

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C), \phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D), \tau_1(D)$	G, I, D	$\tau_2(G, I)$
3	I	$\phi_I(I), \phi_S(S, I), \tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H, G, J	$\tau_4(G, J)$
5	G	$\tau_4(G, J), \tau_3(G, S), \phi_L(L, G)$	G, J, L, S	$\tau_5(J, L, S)$
6	S	$\tau_5(J, L, S), \phi_J(J, L, S)$	J, L, S	$\tau_6(J, L)$
7	L	$\tau_6(J, L)$	J, L	$\tau_7(J)$

(VE)

- Maximal cliques in $\mathcal{I}_{G,\prec}$

$$\begin{aligned}
 \mathbf{C}_1 &= \{C, D\} \\
 \mathbf{C}_2 &= \{D, I, G\} \\
 \mathbf{C}_3 &= \{G, L, S, J\} \\
 \mathbf{C}_4 &= \{G, J, H\} \\
 \mathbf{C}_5 &= \{I, G, S\}
 \end{aligned}$$

Choosing an elimination ordering

Some possible heuristics:

- **Min-neighbors:** The cost of a vertex is the number of neighbors it has in the current graph.
- **Min-weight:** the cost of a vertex is the product of weights (domain cardinality) of its neighbors.
- **Min-fill:** the cost of a vertex is the number of edges that need to be added to the graph due to its elimination.

Which one better?

- None of these criteria is better than others.
- Often will try several.

Summary

What you need to know:

- Inference is generally hard
- However, models we use in practice typically have structure
- We can leverage this structure using dynamic programming style algorithms
- Such algorithms can be extremely efficient if one chooses the right elimination ordering
- Existence of a good elimination ordering depends on properties of the graph

Probabilistic Graphical Models

Stefano Ermon

Stanford University

Lecture 8, February 2, 2017

Today's lecture

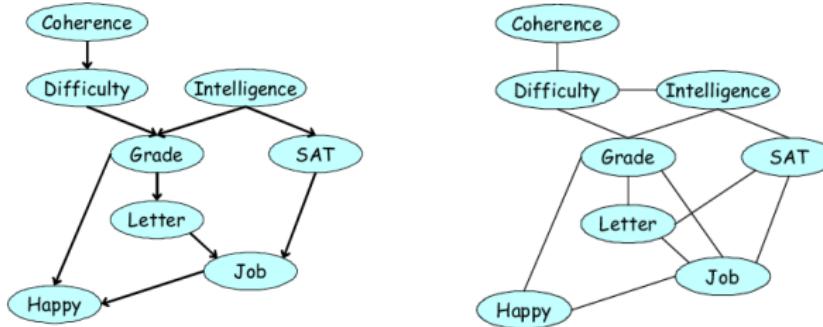
- ① How to solve multiple probabilistic inference queries on the same graph at the same time?
 - ① Message passing algorithms
- ② How to solve other types of inference queries, such as MAP inference

Variable Elimination Recap

- Variable elimination algorithm in a nutshell:
 - ① Pick an elimination ordering
 - ② Eliminate each variable in turn:
 - ① Combine factors that include this variable into single factor (take product)
 - ② Marginalize variable from new factor
- Complexity depends on how big the product factors are
- Complexity depends both on the **graph structure** and the **elimination ordering**

Running time in graph-theoretic concepts

- Let's try to analyze the complexity in terms of the graph structure
- G_ϕ is the undirected graph with one node per variable, where there is an edge (X_i, X_j) if these appear together in the scope of some factor ϕ
- Ignoring evidence, this is either the original MRF (for sum-product VE on MRFs) or the moralized Bayesian network:



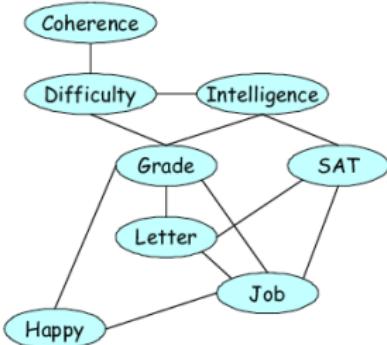
Elimination as graph transformation

When a variable X is eliminated,

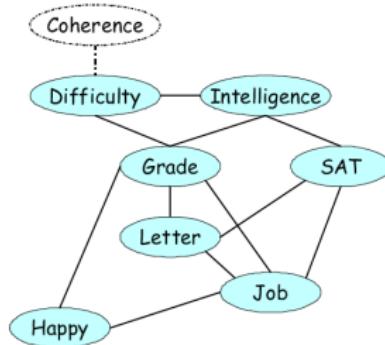
- We create a single factor ψ that contains X and all of the variables \mathbf{Y} with which X appears in factors
- We eliminate (marginalize) X from ψ , replacing it with a new factor τ that contains all of the variables \mathbf{Y} , but not X .

How does this modify the graph?

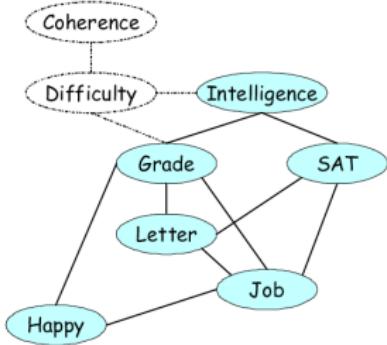
Induced Graph



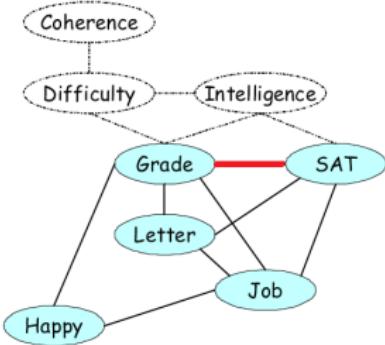
(Graph)



(Elim. C)



(Elim. D)



(Elim. I)

Elimination as graph transformation

When a variable X is eliminated,

- We create a single factor ψ that contains X and all of the variables \mathbf{Y} with which X appears in factors
- We eliminate (marginalize) X from ψ , replacing it with a new factor τ that contains all of the variables \mathbf{Y} , but not X .

How does this modify the graph?

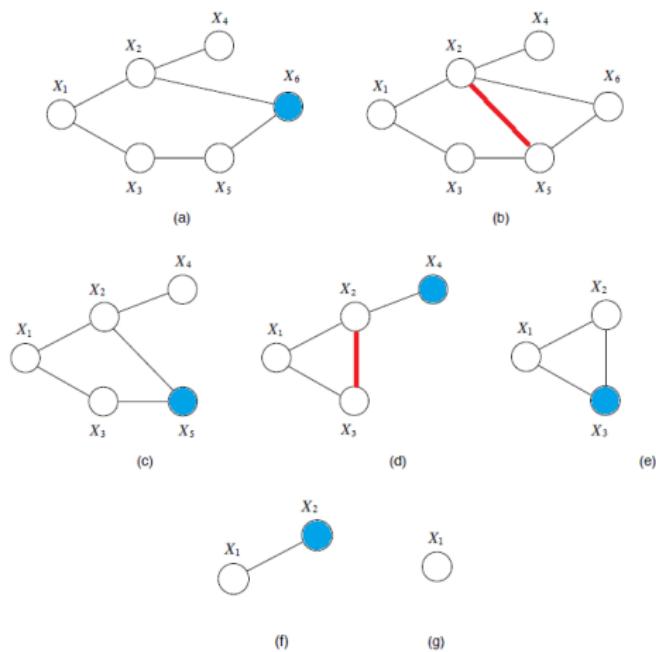
- Constructing ψ generates edges between all of the variables $Y \in \mathbf{Y}$
- Some of these edges were already there, some are new
- The new edges are called **fill edges**
- The step of removing X from Φ removes X and all its incident edges from the graph

Induced graph

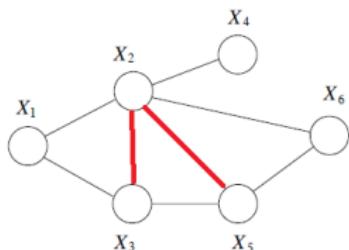
- We can summarize the computation cost using a single graph that is *the union of all the graphs resulting from each step of the elimination*
- We call this the **induced graph** $\mathcal{I}_{\Phi, \prec}$, where \prec is the elimination ordering

Another Example

Elimination ordering: $X_6, X_5, X_4, X_3, X_2, X_1$



Induced graph:



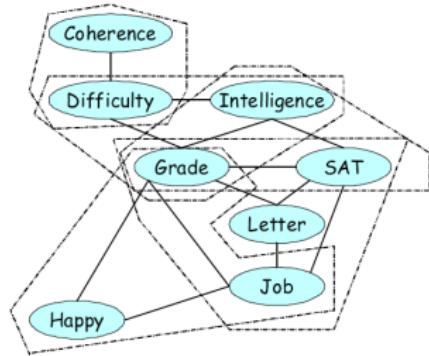
Each time a node is eliminated, all of its neighbors are connected, forming a clique

Structure of the induced graph captures the complexity of VE

Properties of the induced graph

- **Theorem:** Let $\mathcal{I}_{\Phi, \prec}$ be the induced graph for a set of factors Φ and ordering \prec , then
 - ① Every factor generated during VE has a scope that is a clique in $\mathcal{I}_{\Phi, \prec}$
 - ② Every maximal clique in $\mathcal{I}_{\Phi, \prec}$ is the scope of some intermediate factor in the computation
- (see book for proof)
- Thus N_{max} , the maximum number of variables involved in a factor, is equal to the size of the largest clique in $\mathcal{I}_{\Phi, \prec}$
- The running time, $O(mv^{N_{max}})$, is **exponential in the size of the largest clique of the induced graph**, $v = \max_i |\text{Val}(X_i)|$

Example



(Maximal Cliques)

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C), \phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D), \tau_1(D)$	G, I, D	$\tau_2(G, I)$
3	I	$\phi_I(I), \phi_S(S, I), \tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H, G, J	$\tau_4(G, J)$
5	G	$\tau_4(G, J), \tau_3(G, S), \phi_L(L, G)$	G, J, L, S	$\tau_5(J, L, S)$
6	S	$\tau_5(J, L, S), \phi_J(J, L, S)$	J, L, S	$\tau_6(J, L)$
7	L	$\tau_6(J, L)$	J, L	$\tau_7(J)$

(VE)

- Maximal cliques in $\mathcal{I}_{G,\prec}$

$$\begin{aligned}
 \mathbf{C}_1 &= \{C, D\} \\
 \mathbf{C}_2 &= \{D, I, G\} \\
 \mathbf{C}_3 &= \{G, L, S, J\} \\
 \mathbf{C}_4 &= \{G, J, H\} \\
 \mathbf{C}_5 &= \{I, G, S\}
 \end{aligned}$$

Induced width

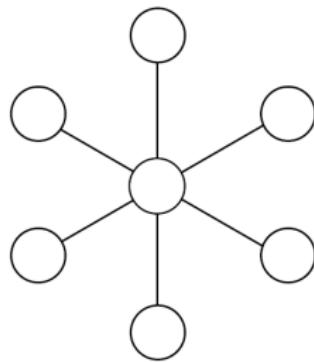
- The **width** of an induced graph is #nodes in largest clique - 1
- We define the **induced width** $w_{\mathcal{G}, \prec}$ to be the width of the graph $\mathcal{I}_{\mathcal{G}, \prec}$ induced by applying VE to \mathcal{G} using ordering \prec
- The **treewidth**, or “minimal induced width” of graph \mathcal{G} is

$$w_{\mathcal{G}}^* = \min_{\prec} w_{\mathcal{G}, \prec}$$

- The treewidth provides a bound on the best running time achievable by VE on a distribution that factorizes over \mathcal{G} : $O(mv^{w_{\mathcal{G}}^*})$,
- Unfortunately, finding the **best** elimination ordering (equivalently, computing the treewidth) for a graph is NP-hard
- In practice, heuristics like picking a node with few neighbors (*min-neighbors*) or that introduces a small number of fill-edges (*min-fill*) are used to find a good elimination ordering

Treewidth examples

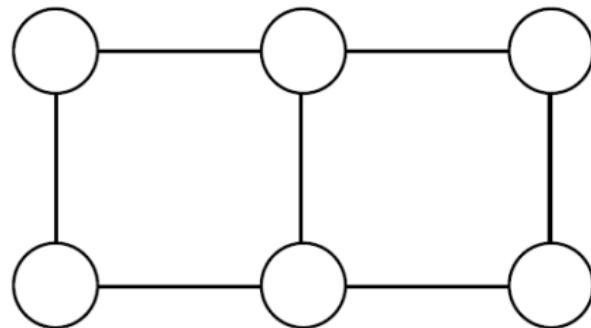
Treewidth = size of largest clique -1
for the best variable ordering



What is the treewidth of this star graph? 1

Treewidth examples

Treewidth = size of largest clique -1
for the best variable ordering



What is the treewidth of this graph? 2

Sum-product algorithm

- What if we want to compute several marginal probabilities on the *same* graph? For example, compute both $p(X)$ and $p(Y)$
- The elimination algorithm works for all graphs, but it must be run separately for each query
- We can be more efficient by caching some of the intermediate computations!
- The **sum-product algorithm** efficiently computes *all single-variable marginals* in the special case of tree-structured graphs
 - What is the treewidth of a tree?
- Later will generalize for graphs with cycles (**junction tree algorithm**)

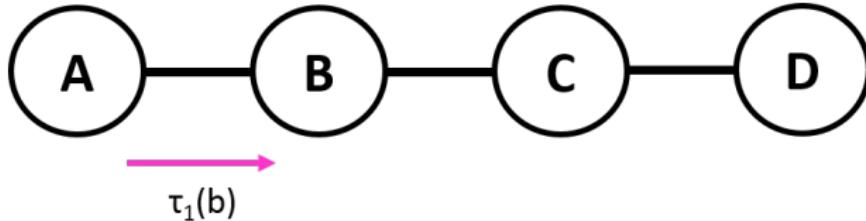
VE on a chain

- Example BN from last lecture: $A \rightarrow B \rightarrow C \rightarrow D$

$$\begin{aligned} p(D) = \sum_{a,b,c} p(a, b, c, D) &= \sum_{a,b,c} p(a)p(b | a)p(c | b)p(D | c) \\ &= \sum_{a,b,c} \phi_{DC}(D, c)\phi_{CB}(c, b)\phi_{BA}(b, a)\phi_A(a) \end{aligned}$$

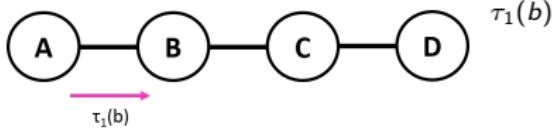
- We can push the summations inside (*distributive law*) to obtain:

$$p(D) = \sum_c \sum_b \phi_{DC}(D, c)\phi_{CB}(c, b) \underbrace{\sum_a \underbrace{\phi_{BA}(b, a)\phi_A(a)}_{\psi_1(a, b)}}_{\tau_1(b)}$$



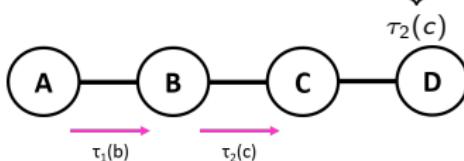
VE on a chain

$$p(D) = \sum_c \sum_b \phi_{DC}(D, c) \phi_{CB}(c, b) \underbrace{\sum_a \phi_{BA}(b, a) \phi_A(a)}_{\tau_1(b)}$$

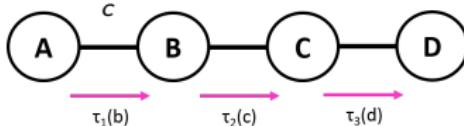


$$p(D) = \sum_c \sum_b \phi_{DC}(D, c) \phi_{CB}(c, b) \tau_1(b)$$

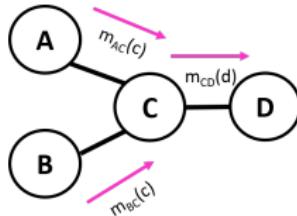
$$p(D) = \sum_c \phi_{DC}(D, c) \underbrace{\sum_b \phi_{CB}(c, b) \tau_1(b)}_{\tau_2(c)}$$



$$p(D) = \sum_c \phi_{DC}(D, c) \tau_2(c) = \tau_3(D)$$



Example



$$\sum_{a,b,c} \phi_C(c) \phi_{DC}(D, c) \phi_B(b) \phi_{BC}(b, c) \phi_A(a) \phi_{AC}(a, c) =$$

$$\sum_{b,c} \phi_C(c) \phi_{DC}(D, c) \phi_B(b) \phi_{BC}(b, c) \sum_a \phi_A(a) \phi_{AC}(a, c) =$$

$$\sum_{b,c} \phi_C(c) \phi_{DC}(D, c) \phi_B(b) \phi_{BC}(b, c) m_{AC}(c) =$$

$$\sum_c \phi_C(c) \phi_{DC}(D, c) m_{AC}(c) \sum_b \phi_B(b) \phi_{BC}(b, c) =$$

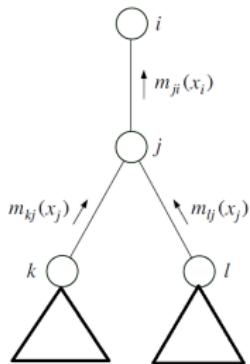
$$\sum_c \phi_C(c) \phi_{DC}(D, c) m_{AC}(c) m_{BC}(c) = m_{CD}(D)$$

More generally, to eliminate variable j on a tree, VE computes

$$m_{ji}(x_i) = \sum_{x_j} \phi_j(x_j) \phi_{ij}(x_i, x_j) \prod_{\ell \in \text{Neighbors}(j) \setminus i} m_{\ell j}(x_j)$$

Message Passing

Suppose we run VE on a tree, eliminating variables from the leaves to the root. VE operations on a tree can be thought of as passing messages:



To eliminate variable j , VE computes

$$m_{ji}(x_i) = \sum_{x_j} \phi_j(x_j) \phi_{ij}(x_i, x_j) \prod_{\ell \in \text{Neighbors}(j) \setminus i} m_{\ell j}(x_j)$$

Here $m_{kj}(x_j)$ is the result of eliminating variable k and all its descendants. It “summarizes” all the information from the subtree rooted at k .

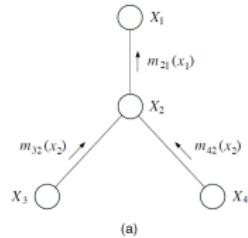
Termination

- At the final node x_f in the elimination ordering (the query node, which is the root of the tree), the marginal $p(x_f)$ is computed as a (normalized) product over incoming messages (and possibly a singleton potential $\phi(x_f)$)

$$p(x_f) \propto \phi_f(x_f) \prod_{\ell \in \text{Neighbors}(x_f)} m_{\ell f}(x_f)$$

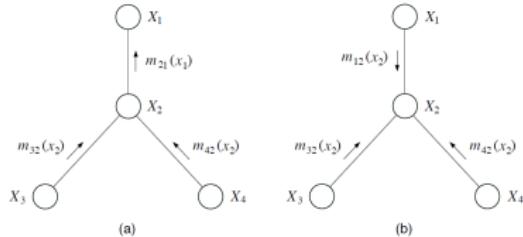
- Why is this useful? The key insight is that, if we try to compute marginal probabilities for each node in turn in this way, *the same messages are generated over and over again*
- The complete set of messages scales linearly with the size of the tree

All Messages



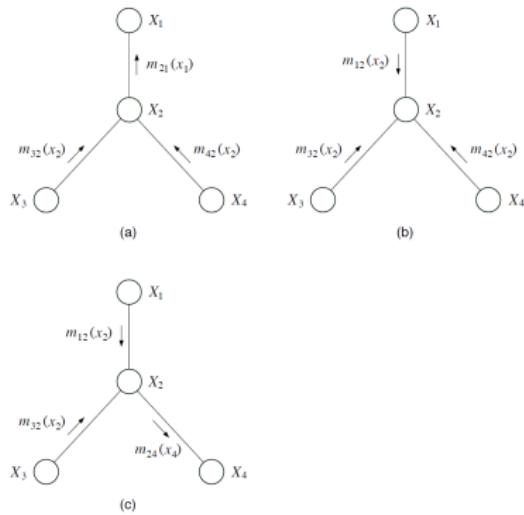
- (a) Compute $P(X_1)$, eliminating X_3, X_4, X_2 . Intermediate factors generated by VE represented as messages.

All Messages



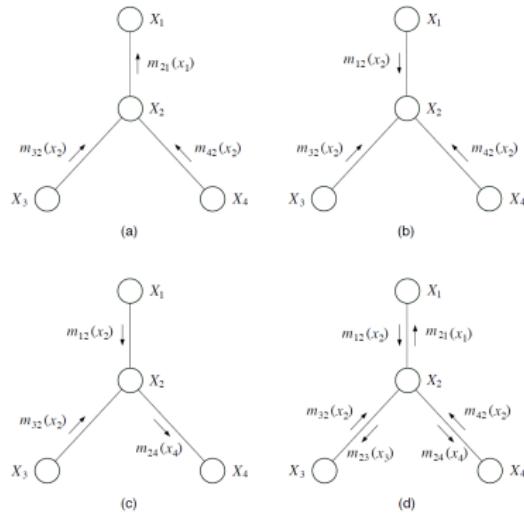
- (a) Compute $P(X_1)$, eliminating X_3 , X_4 , X_2 . Intermediate factors generated by VE represented as messages.
- (b) Compute $P(X_2)$, eliminating X_3 , X_4 , X_1 . Messages $m_{32}(x_2), m_{42}(x_2)$ are identical to those generated for case (a).

All Messages



- (a)** Compute $P(X_1)$, eliminating X_3 , X_4 , X_2 . Intermediate factors generated by VE represented as messages.
- (b)** Compute $P(X_2)$, eliminating X_3 , X_4 , X_1 . Messages $m_{32}(x_2), m_{42}(x_2)$ are identical to those generated for case (a).
- (c)** Compute $P(X_4)$, eliminating X_1 , X_3 , X_2 . Some messages (intermediate factors) in common with previous computations.

All Messages



- (a) Compute $P(X_1)$, eliminating X_3, X_4, X_2 . Intermediate factors generated by VE represented as messages.
- (b) Compute $P(X_2)$, eliminating X_3, X_4, X_1 . Messages $m_{32}(x_2), m_{42}(x_2)$ are identical to those generated for case (a).
- (c) Compute $P(X_4)$, eliminating X_1, X_3, X_2 . Some messages (intermediate factors) in common with previous computations.
- (d) If we have all the messages, we can compute *all marginal probabilities!*

Message Passing Protocol

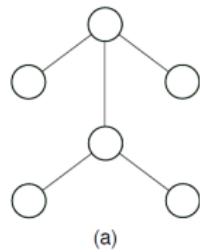
- All that is needed at each query node is the complete set of incoming messages
- However, there are dependencies to consider: a message $m_{ji}(x_i)$ cannot be sent from j to i until messages have been received from all **other** neighbors of j

$$m_{ji}(x_i) = \sum_{x_j} \phi_j(x_j) \phi_{ij}(x_i, x_j) \prod_{\ell \in \text{Neighbors}(j) \setminus i} m_{\ell j}(x_j)$$

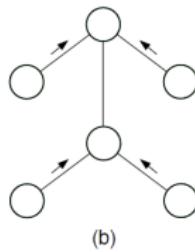
- One way to think about it is as a parallel algorithm: each node has a processor that polls its neighbors, sends each message when all others received
- The messages will flow from the leaves up
- Sometimes called Belief Propagation

Parallel Protocol

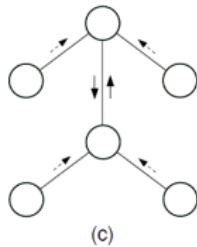
Protocol: Node j sends to neighbor i when (and only when) it has received messages from all *other* neighbors



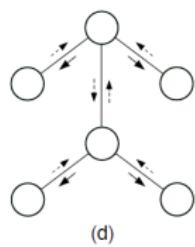
(a)



(b)



(c)

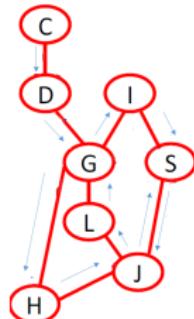


(d)

All incoming messages received by all nodes in $O(|E|)$ steps.

Loopy Belief Propagation

- What if we apply BP to a (high treewidth) graph with loops?



- How about the order of the updates?

- Initialize all the messages (e.g., uniformly) $\delta_{i \rightarrow j}^0(X_j) = \frac{1}{|\text{Val}(X_j)|}$
- Keep doing BP updates until it converges
 - Repeat

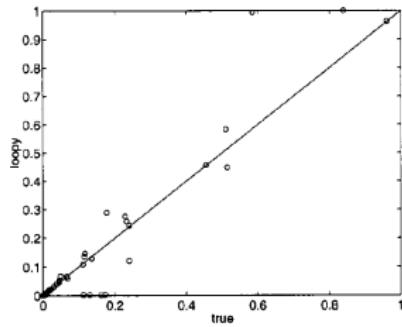
$$\delta_{i \rightarrow j}^{t+1}(X_j) = \sum_{x_i} \phi_{ij}(x_i, X_j) \phi_i(x_i) \prod_{k \in (\mathcal{N}(i) \setminus \{j\})} \delta_{k \rightarrow i}^t(x_i)$$

until $\delta_{i \rightarrow j}^{t+1} - \delta_{i \rightarrow j}^t$ is small, for all edges (i, j)

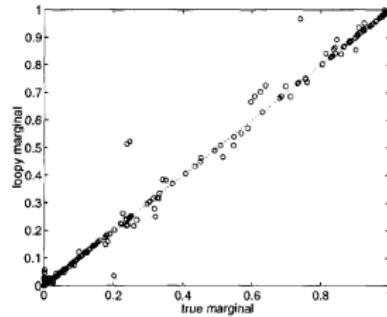
- It will generally not converge. If it converges, it will not necessarily give correct marginals
- Still, often **very useful in practice!**

Some Empirical Results

Example from medical diagnosis (from *Loopy belief propagation for approximate inference: An empirical study*)



Example from intensive care patient monitoring



Practical Issues

- General update

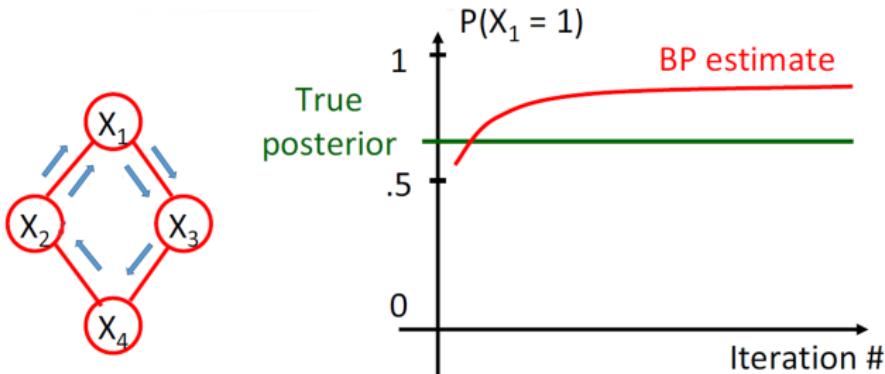
$$\delta_{i \rightarrow j}(X_j) = \sum_{x_i} \phi_{ij}(x_i, X_j) \phi_i(x_i) \prod_{k \in (\mathcal{N}(i) \setminus \{j\})} \delta_{k \rightarrow i}(x_i)$$

- In loopy BP, factors are multiplied together many times. Can be **numerically unstable**: very large or very small numbers
- Solution: normalize so that messages sum to one

$$\delta_{i \rightarrow j}(X_j) = \frac{1}{Z_{i \rightarrow j}} \sum_{x_i} \phi_{ij}(x_i, X_j) \phi_i(x_i) \prod_{k \in (\mathcal{N}(i) \setminus \{j\})} \delta_{k \rightarrow i}(x_i)$$

- Multiplication by a constant will not affect the estimated marginals (product of all incoming messages)

Loopy Belief Propagation



- Loopy BP multiplies the same potentials multiple times. It is often over-confident.
- Suppose factors encourage neighboring variables to take the same value. Suppose we know $P[X_2 = 0] > P[X_2 = 1]$ (evidence). Message $\delta_{2 \rightarrow 1}$ will encourage $X_1 = 0$. Message $\delta_{1 \rightarrow 3}$ will encourage $X_3 = 0$. Message $\delta_{3 \rightarrow 4}$ will encourage $X_4 = 0$. Message $\delta_{4 \rightarrow 2}$ will encourage $X_2 = 0$. Double counting! We are led to believe $P[X_2 = 0] \gg P[X_2 = 1]$

Convergence and accuracy guarantees

Can we prove convergence?

- For certain classes of random graphs (coding theory)
- Other special conditions (e.g., single loop)

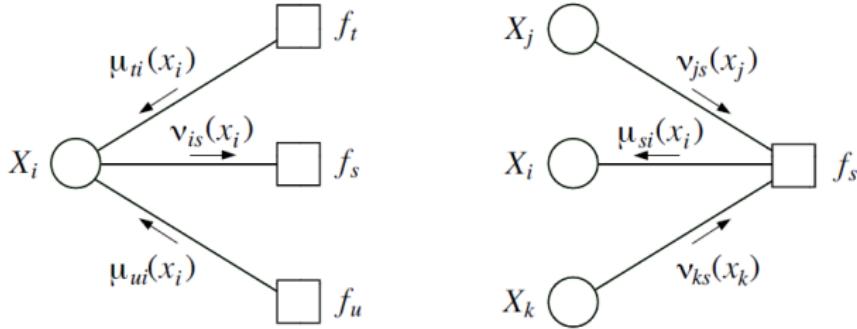
Can we prove accuracy guarantees?

- Sometimes can provide bounds if certain conditions on the potentials are satisfied

Very active area of research.

Sum Product for Factor Graphs

Two types of messages: variables \rightarrow factors (ν), factors \rightarrow variables (μ)

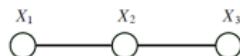


Both updates require product. Only messages factor-to-variables require summing.

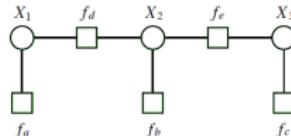
$$\nu_{\text{var}(i) \rightarrow \text{fac}(s)}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{\text{fac}(t) \rightarrow \text{var}(i)}(x_i)$$

$$\mu_{\text{fac}(s) \rightarrow \text{var}(i)}(x_i) = \sum_{\mathbf{x}_{\mathcal{N}(s) \setminus i}} f_s(\mathbf{x}_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{\text{var}(j) \rightarrow \text{fac}(s)}(x_j)$$

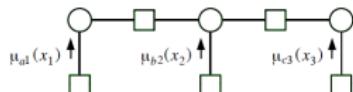
Example on Factor Trees



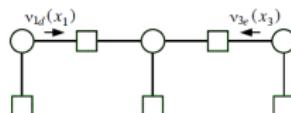
(a)



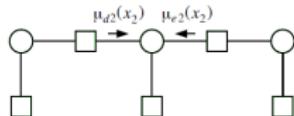
(b)



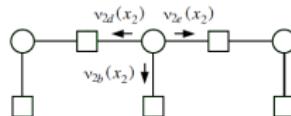
(c)



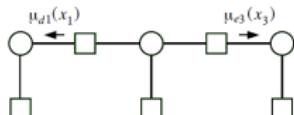
(d)



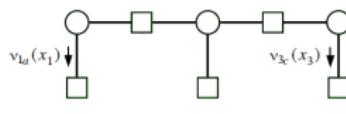
(e)



(f)



(g)



(h)

Summary

- Runtime of VE is exponential in the size of the largest clique, which depends on the elimination ordering used
 - Treewidth is the best over all possible elimination orderings
- If graph is a tree, then it has small treewidth and inference is efficient.
 - Can get *all* marginals with a two pass procedure
 - VE can be thought of as a distributed algorithm based on message passing
- Even if graph has very large treewidth, can still run (heuristically) Belief Propagation

Probabilistic Graphical Models

Stefano Ermon

Stanford University

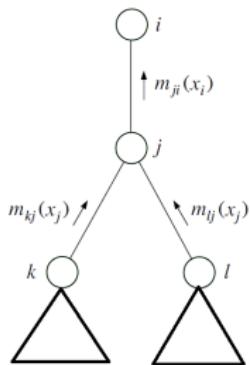
Lecture 9, February 7, 2017

Today's lecture

- ➊ How to solve multiple probabilistic inference queries on the same *loopy* graph at the same time?
 - ➊ Junction trees, another data structure for inference

Message Passing

Suppose we run VE on a tree, eliminating variables from the leaves to the root. VE operations on a tree can be thought of as passing messages:



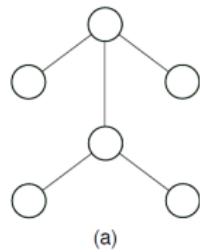
To eliminate variable j , VE computes

$$m_{ji}(x_i) = \sum_{x_j} \psi_j(x_j) \psi_{ij}(x_i, x_j) \prod_{\ell \in N(j) \setminus i} m_{\ell j}(x_j)$$

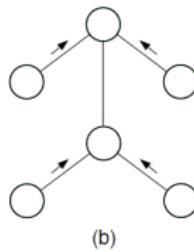
Here $m_{kj}(x_j)$ is the result of eliminating variable k and all its descendants. It “summarizes” all the information from the subtree rooted at k .

Parallel Protocol

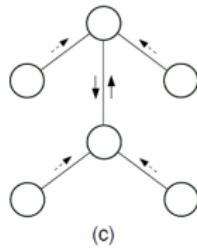
Protocol: Node j sends to neighbor i when (and only when) it has received messages from all *other* neighbors



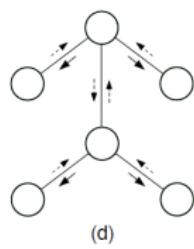
(a)



(b)



(c)



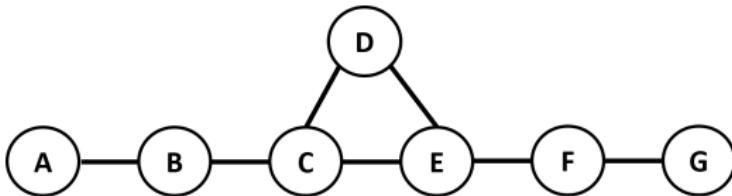
(d)

All incoming messages received by all nodes in $O(|E|)$ steps.

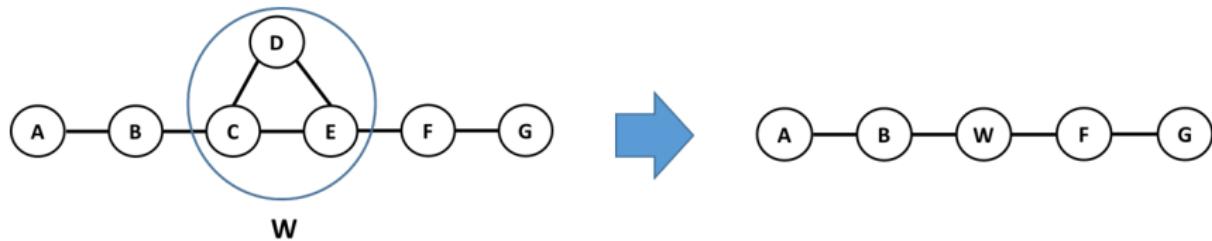
Junction Tree Idea

- We have good methods for probabilistic inference on trees
- The difficulty of handling loopy graphs has to do with the complex dependencies that emerge as nodes are eliminated (cliques in induced graph)
- The idea of the junction tree algorithm is essentially to manage this complexity by converting the graph to a *tree of cliques*
- Message passing can then be applied to this new tree (with some modifications)
 - Simultaneous computation of all marginals
 - Guaranteed two-pass convergence

Cluster graph

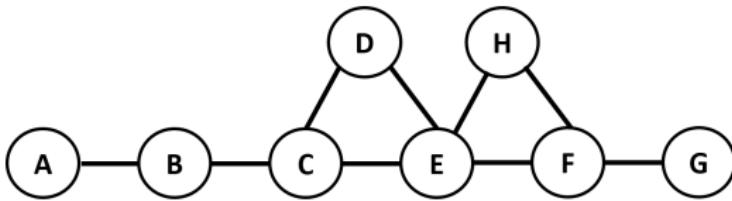


Can we use Belief propagation? No, because there is a loop $C - D - E$.

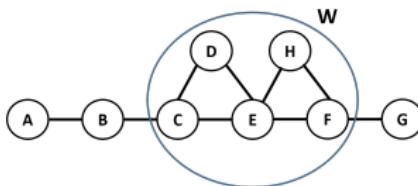


- Idea: define an equivalent model $p'(A, B, \mathbf{W}, F, G)$ that is a **tree**
- $\text{Val}(\mathbf{W}) = \text{Val}(C) \times \text{Val}(D) \times \text{Val}(E)$, $\mathbf{w} = (c, d, e)$
- $\phi'_{BW}(b, \mathbf{w}) = \phi'_{BW}(b, (c, d, e)) = \phi_{BC}(b, c)$ for all d, e
- $\phi'_{WF}(\mathbf{w}, f) = \phi'_{WF}((c, d, e), f) = \phi_{EF}(e, f)$ for all c, d
- $\phi'_W(\mathbf{w}) = \phi_{CD}(c, d)\phi_{DE}(d, e)\phi_{CE}(c, e)$

Cluster graph



Option 1

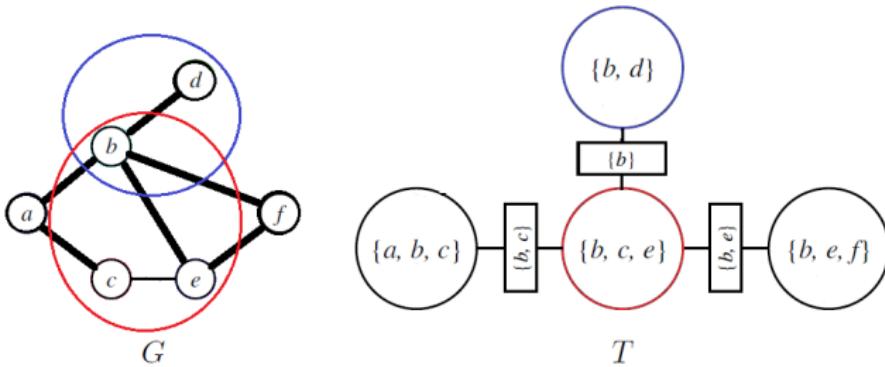


We'd like to do something like this



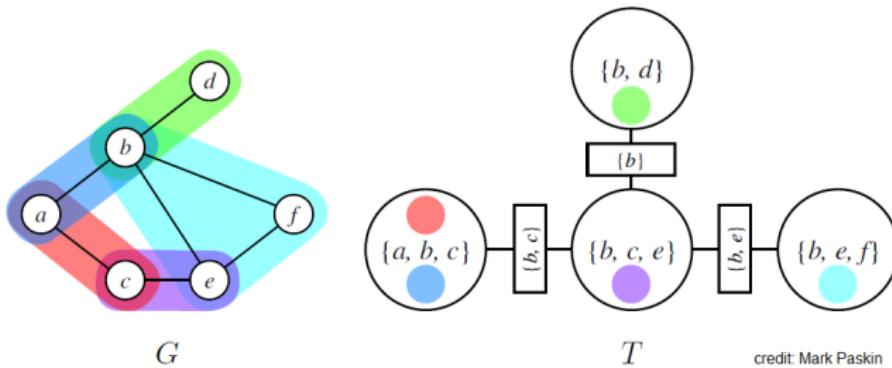
Issue: ensure that $w = (c, d, e)$ is consistent with $v = (e, f, h)$ w.r.t E

Cluster graph



A **cluster graph** T is an undirected graph where each node i is associated to a subset of variables \mathbf{C}_i (a cluster). An edge between clusters $\mathbf{C}_i, \mathbf{C}_j$ is associated with a **sep-set** $S_{i,j} = \mathbf{C}_i \cap \mathbf{C}_j$

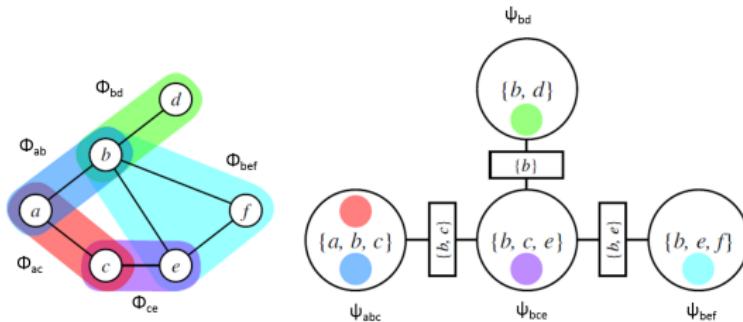
Cluster graph



A **cluster graph** T is an undirected graph where each node i is associated to a subset of variables \mathbf{C}_i (a cluster). A cluster graph is

- ➊ family preserving for a set of factors Φ : for each factor $\phi \in \Phi$ there is some cluster C such that $\text{Scope}[\phi] \subseteq C$. (in the figure, factors shown as cliques on the left, and colored dots on the right)

A data structure for inference



Suppose we have a model defined by a set of factors Φ

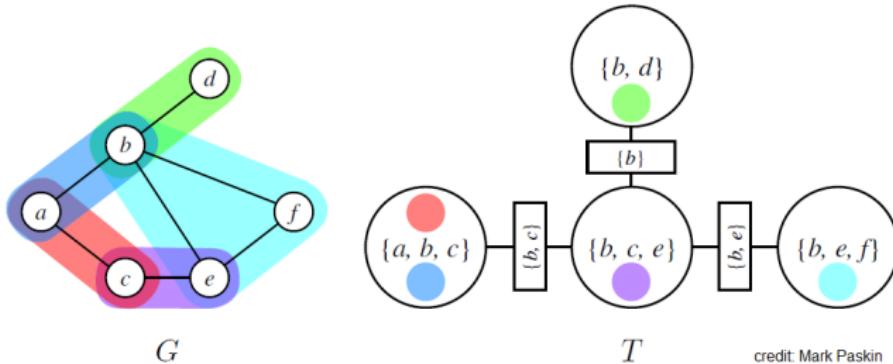
$$p(\mathbf{x}) = \frac{1}{Z} \prod_{\phi_i \in \Phi} \phi_i(\mathbf{x}_i)$$

and a family preserving **cluster graph** T for Φ . Then we can rewrite

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in T} \psi_C(\mathbf{x}_C)$$

How? Create a potential ψ_C for each cluster C of T and initialize it to 1. Then multiply each potential $\phi_i \in \Phi$ into the cluster potential ψ_C of one cluster C such that $\text{Scope}[\phi_i] \subseteq C$. Possible because it is **family preserving**.

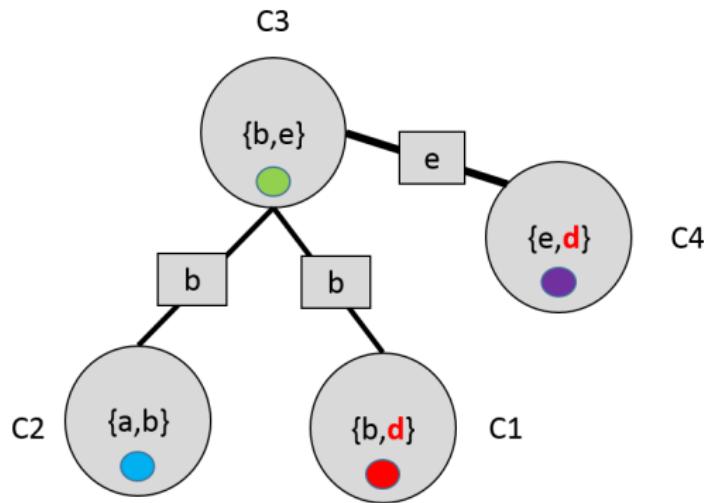
Junction Tree



A cluster graph T is a **junction tree** (or clique tree) for a set of factors Φ if it has these three properties:

- ① it is a tree
- ② family preserving: for each factor ϕ there is some cluster C such that $\text{Scope}[\phi] \subseteq C$.
- ③ running intersection: for every pair of clusters V and W , all clusters on the path between V and W contain $V \cap W$

Example of running intersection

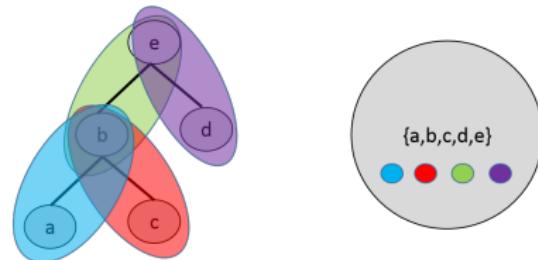


This is **not** a valid junction tree because of variable **d**. Violates **running intersection**:

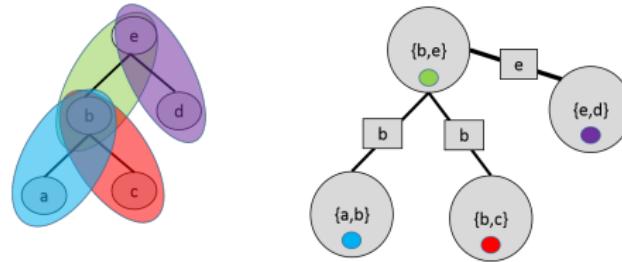
$$C_1 \cap C_4 = \{d\} \not\subseteq C_3$$

Illustration

It's easy to come up with a (useless) junction tree:



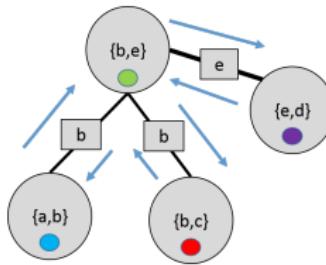
If the model is simple (tree), then obtaining a junction tree with small cliques is easy:



We'll see later how to construct junction trees (through VE).

Junction tree inference

We start with a set of factors Φ and a **junction tree** T for Φ . Inference is just message passing:



- Each cluster sends one message (potential function) to each neighbor
- Cluster B is allowed to send a message to a neighbor C only after it has received messages from all neighbors except C

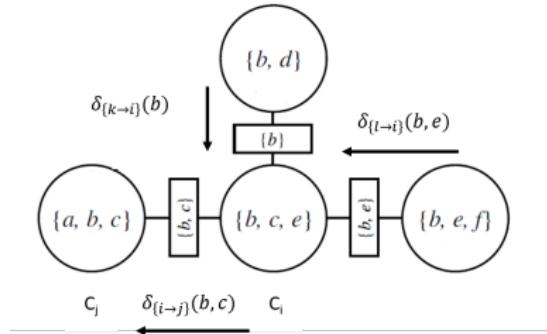
At the end, by combining its local potential with the messages it receives, each cluster is able to compute the marginal probability of its variables.

Shafer-Shenoy Inference

- The message sent from i to j is defined as

$$\delta_{i \rightarrow j}(C_i \cap C_j) = \sum_{C_i \setminus C_j} \psi_{C_i}(C_i) \prod_{k \in (\mathcal{N}(i) \setminus \{j\})} \delta_{k \rightarrow i}(C_k \cap C_i)$$

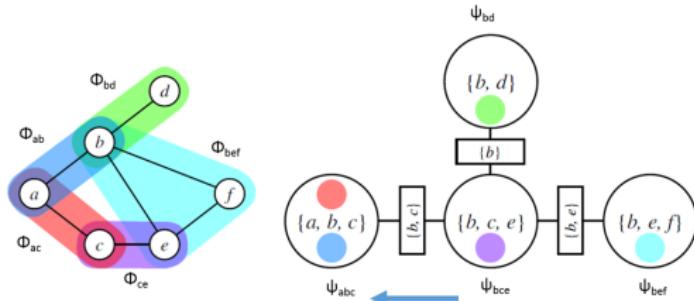
- Cluster i computes the product of its local potential ψ_{C_i} and the messages from all clusters **except** j , marginalizes out all variables that are not in j , and then sends the result to j



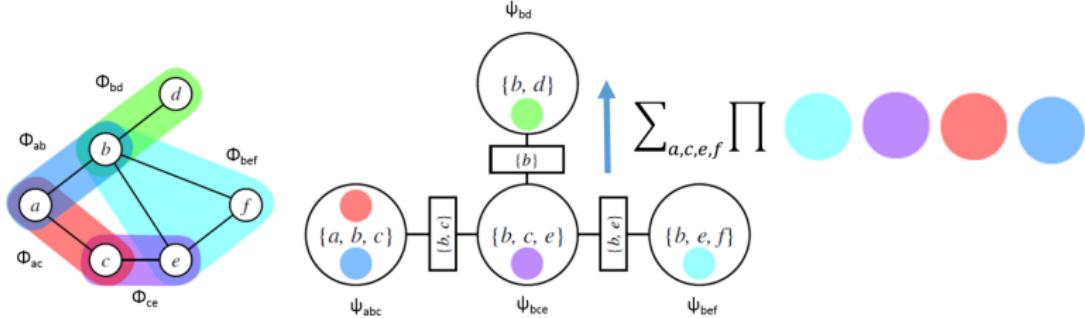
$$\delta_{i \rightarrow j}(b, c) = \sum_e \psi_{C_i}(b, c, e) \delta_{k \rightarrow i}(b) \delta_{l \rightarrow i}(b, e)$$

Intuition

We are just doing Variable Elimination

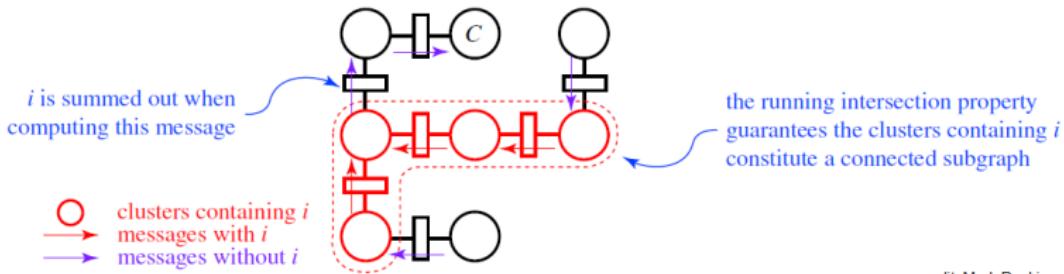


$$\sum_{e,d,f} \prod$$



Intuition for these 3 requirements

- ① **Tree:** there is a natural order, e.g., can choose a root and pass messages from leaves to the root
- ② **Family preserving:** we can define a potential for each cluster such that their product defines the same distribution
- ③ **Running intersection:** clusters that contain variable X form a connected component. We know when it is “safe” to eliminate a variable

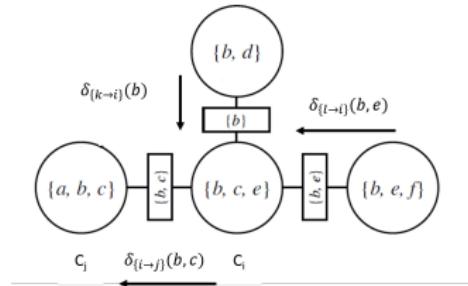


credit: Mark Paskin

Shafer-Shenoy Inference

- Once a cluster has received messages from all its neighbors, we can compute its **belief** (marginal probability) as the product of **all** incoming messages and local potential:

$$p(C_r) \propto \psi_{C_r} \prod_{k \in \mathcal{N}(r)} \delta_{k \rightarrow r}$$



$$p(a, b, c) \propto \delta_{i \rightarrow j}(b, c) \psi_{C_j}(a, b, c)$$

How to compute individual marginal probabilities?

$$p(a) = \sum_{b,c} p(a, b, c)$$

Calibration

- When all clusters have received all incoming messages, we say that the clique tree is **calibrated**. This simply means that the beliefs (= product of incoming messages) are consistent with each other.

$$\beta(C_r) = \psi_{C_r} \prod_{k \in \mathcal{N}(r)} \delta_{k \rightarrow r}$$

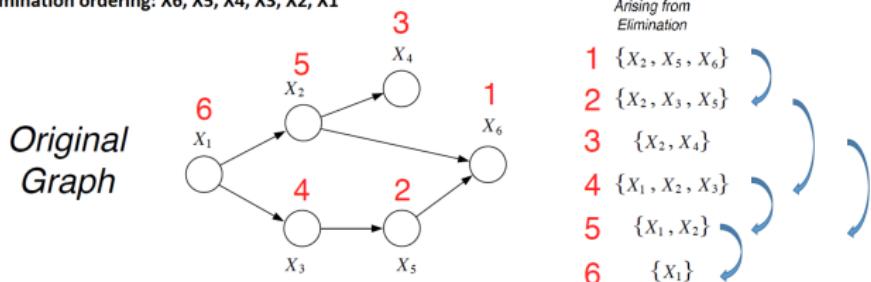
- Let C_1 and C_2 be two clusters, both containing X : $X \in C_1, X \in C_2$.

$$\sum_{C_1 \setminus X} \beta(C_1) = \sum_{C_2 \setminus X} \beta(C_2)$$

This is equal to the (unnormalized) marginal probability. How to get a junction tree?

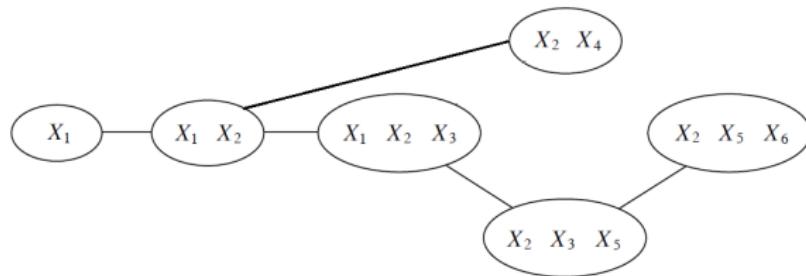
Clique trees from VE

Elimination ordering: $X_6, X_5, X_4, X_3, X_2, X_1$



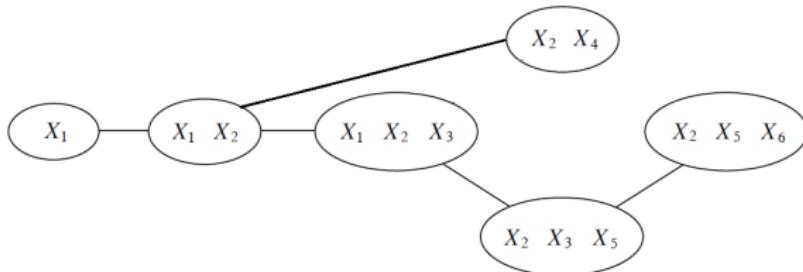
Draw an edge between C_i and C_j if the intermediate factor produced eliminating a variable in C_i is used in the computation of the product factor of C_j

Tree of Cliques



Why is this a valid junction tree

*Tree of
Cliques*

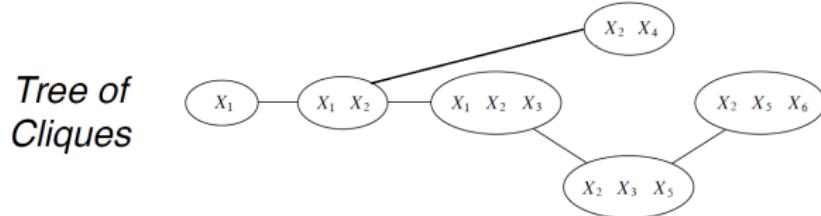


Intuition for why this is a valid junction tree (see book for proofs)

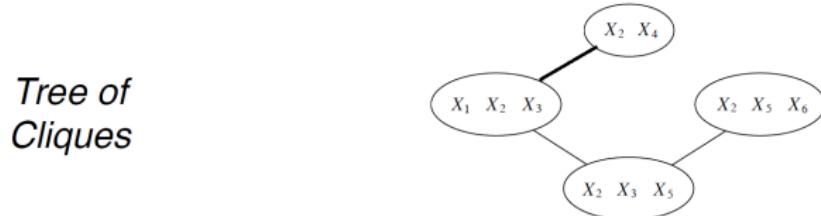
- ① **Tree:** every intermediate factor is used once and then removed
- ② **Family preserving:** every original factor $\phi \in \Phi$ can be assigned to the cluster (= clique) where it is multiplied in
- ③ **Running intersection:** a variable appears in every factor from the moment it is introduced (by multiplying in the factor that mentions it) until it is summed out

Junction trees are not unique

Note that junction trees are not unique:



For example, we can get rid of cliques that are not maximal:

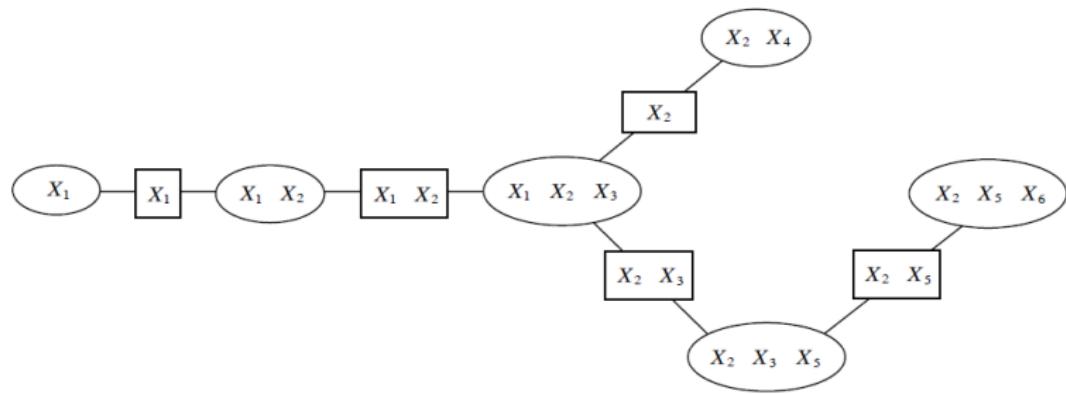


Also depends on the elimination ordering used.

Computational Complexity

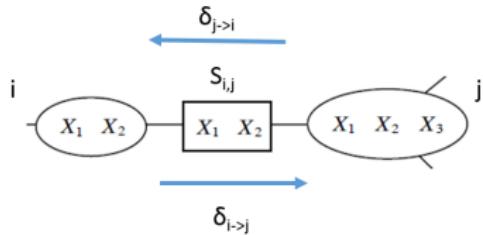
- The algorithm can be divided into 2 phases:
 - ① compilation (moralization, elimination)
 - ② propagation (introduction of evidence, propagation of messages)
- Compilation can be done once in preprocessing. Given an elimination ordering, it is polynomial in the number of variables
- Propagation is limited by marginalization and multiplication of clique potentials. It is linear in the number of cliques but exponential in clique size.
- At convergence, we can compute **all** clique marginals
- A calibrated clique tree can be seen as an alternative representation of the original probability distribution, one where we can easily access marginal probabilities of groups of variables

Reminder: Separator Sets



Between every two clique nodes is a separator node associated with the intersection of the variables in the adjoining cliques (the separator set)

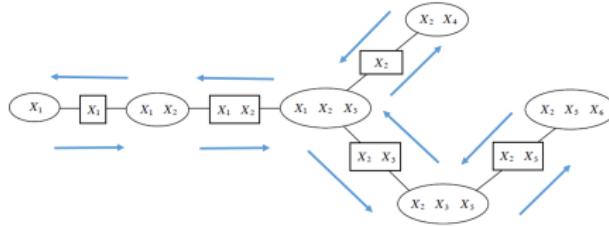
Extended Representation



- Let's compute the marginal probability over the sep-set $\mathbf{S}_{i,j}$

$$\begin{aligned}\mu_{i,j}(\mathbf{S}_{i,j}) &= \sum_{\mathbf{C}_i \setminus \mathbf{S}_{i,j}} \beta_i(\mathbf{C}_i) \\ &= \sum_{\mathbf{C}_i \setminus \mathbf{S}_{i,j}} \psi_{\mathbf{C}_i} \prod_{k \in \mathcal{N}(i)} \delta_{k \rightarrow i} \\ &= \sum_{\mathbf{C}_i \setminus \mathbf{S}_{i,j}} \psi_{\mathbf{C}_i} \delta_{j \rightarrow i} \prod_{k \in \mathcal{N}(i) \setminus j} \delta_{k \rightarrow i} \\ &= \delta_{j \rightarrow i} \sum_{\mathbf{C}_i \setminus \mathbf{S}_{i,j}} \psi_{\mathbf{C}_i} \prod_{k \in \mathcal{N}(i) \setminus j} \delta_{k \rightarrow i} \\ &= \delta_{j \rightarrow i} \delta_{i \rightarrow j}\end{aligned}$$

Extended Representation Continued



- If we associate factors to both the clusters *and* the separator sets, we get a very useful alternative representation for $p = \prod_i \psi_i(C_i)$

$$\frac{\prod_i \beta_i(\mathbf{C}_i)}{\prod_{(i,j) \in E} \mu_{i,j}(\mathbf{S}_{i,j})} = \frac{\prod_i \psi_{C_i} \prod_{k \in N(i)} \delta_{k \rightarrow i}}{\prod_{(i,j) \in E} \delta_{j \rightarrow i} \delta_{i \rightarrow j}} = \prod_i \psi_{C_i}$$

Every message appears exactly once in the numerator and once in the denominator!

- It is an extended reparameterization of the original probability distribution that exposes directly the marginals of the clusters.

An alternative view of junction tree inference

- Assume $p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)$ where \mathcal{C} is a set of cliques
- Goal: reparameterize so that potentials \propto clique marginals
- Normally clique potentials do not correspond to marginals; some can be conditionals. Example BN: $A \rightarrow B \rightarrow C$. Potentials are $\psi_{AB} = p(A)p(B|A) = p(A, B)$, but $\psi_{BC} = p(C|B) \neq p(B, C)$.
- **Extended Representation:** Let's associate potentials with separators as well, so that:

$$p(x) = \frac{\prod_{C \in \mathcal{C}} \psi'_C(x_C)}{\prod_{S \in \mathcal{S}} \psi'_S(x_S)}$$

Now we can have (1) clique potentials proportional to marginals and (2) a valid representation of the joint distribution at the same time.
In previous example $A \rightarrow B \rightarrow C$:

$$p(A, B, C) = \frac{\psi'_{AB}(A, B)\psi'_{BC}(B, C)}{\psi'_B(B)} = \frac{p(A, B)p(B, C)}{p(B)}$$

Recap: what you need to know

- The Junction Tree algorithms generalize Variable Elimination to the **simultaneous execution** of probability queries.
- The algorithms take the form of message passing on a graph called a junction tree, whose nodes are clusters (sets of variables).
- Each cluster starts with one potential of the factorized density. By combining this potential with the potentials it receives from its neighbors, it can compute the marginal over its variables.
- The complexity of the algorithms scales (exponentially) with the **size of the largest clique**.