MRFs and Inference in Graphical Models

Mark Dredze

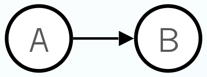
Machine Learning
CS 601.475

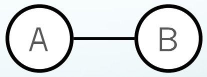
Outline

- Representation
  - What is a graphical model?
  - What does it represent
  - Conditional Independence
  - **Types of probabilistic models**
- **Inference**
  - **How can we compute probabilities?**
  - **Message Passing**
- Examples
  - Learning and inference

Graph Types

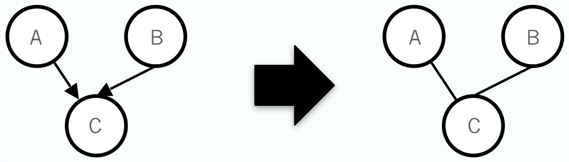
- Edge type determines graph type
- Directed graphs
  - Edges have directions (A -> B)
  - Assume DAGs (no cycles)
  - Typically called Bayesian Networks
    - Popular in AI and stats
- Undirected graphs
  - Edges don't have directions (A – B)
  - Typically called Markov Random Fields (MRFs)
    - Popular in physics and vision





Undirected Networks

- What happens when we use undirected edges?



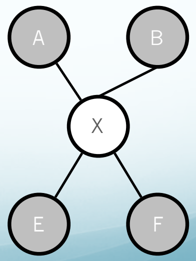
- Called undirected graphical models
  - Markov Random Fields, Markov Networks

Conditional Independence

- D-separation used directionality of the edges
- Can we define it without directionality?
  - Yes!
- If all paths from set A to set B pass through set C, then A is d-separated from B given C
  - Notice no distinction between head to head and tail to tail
  - "Explaining away" not an issue since no causation
  - Actually easier to check ☺

Markov Blanket

- The absence of "explaining away" makes the Markov blanket simple as well
- The Markov blanket of a node contains the neighbors of the node



Factorization

- How can we express the joint distribution as a product of functions over local sets of variables
  - The directions in directed graphs indicated conditional relationship
- Step 1 is easier than with directed graphs
  - Given two nodes x_i and x_j, they are conditionally independent given the entire graph if they are not neighbors

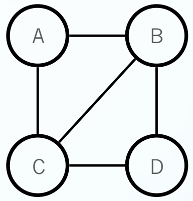
$$p(x_i, x_j \mid \mathbf{x}_{\setminus\{i,j\}}) = p(x_i \mid \mathbf{x}_{\setminus\{i,j\}})\, p(x_j \mid \mathbf{x}_{\setminus\{i,j\}})$$

Defining Factors

- The factorization of the joint distribution is such that x_i and x_j do not appear in the same factor
- Graph concept: clique
  - A set of nodes that are fully connected
    - There exists an edge between every pair of nodes
  - Maximal clique
    - A clique such that adding any other node means it is no longer a clique

Cliques

- Cliques in the graph
  - A/B, A/C, B/D, B/C, C/D
  - Maximal cliques: A/B/C, B/C/D
  - A/B/C/D is not a clique since no edge from A to D
- We just need to use maximal cliques, since they contain all other cliques



Factorization

- Define
  - x_C as all nodes in clique C
  - $\psi_C(x_C)$ is a potential function over clique C
- We can define the joint distribution of the graph as a product of potential functions over maximal cliques

$$p(x) = \frac{1}{Z}\prod_C \psi_C(x_C)$$

  - Cliques have taken the place of nodes and CPTs

Partition Function

$$p(x) = \frac{1}{Z}\prod_C \psi_C(x_C)$$

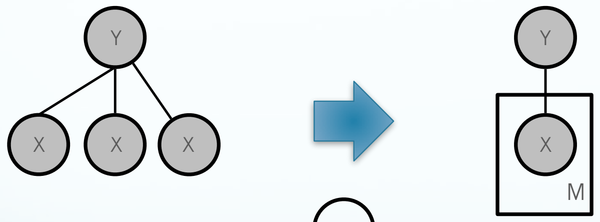
- Notice that $\psi_C(x_C)$ is not a probability
  - Must be $\psi_C(x_C) \geq 0$
  - Will not sum to 1
- Therefore we need to normalize to get a probability

$$Z = \sum_x \prod_C \psi_C(x_C)$$

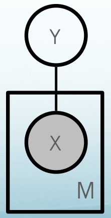
- Z is called the partition function

Example

- A model where we have label Y and example X

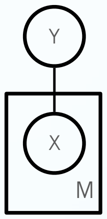


- Test time, no Y
  - Estimate Y using X
- What model is this?



Logistic Regression

- No generative story
  - Graph does not encode causality
  - We can say that X and Y are related
- Learning
  - We observe X and Y, maximum likelihood solution
- Prediction
  - Compute most likely value for Y given X

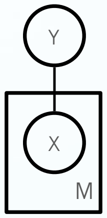


Factorization

$$p(y \mid x) = \frac{1}{Z}\prod_m \psi_m(x_m, y)$$

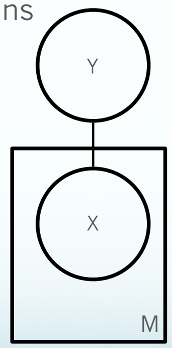
$$\psi_m(x_m, y) = \exp(w_m \cdot f(x_m, y))$$

$$Z = \sum_y \prod_m \psi_m(x_m, y)$$



Potential Functions

- The parameters correspond to potential functions

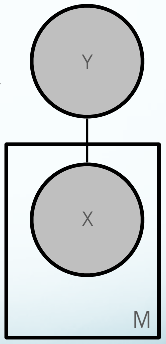


Y/X	0	1
0	.45	3
1	5	.2

M Tables

Learning

- We assumed both examples (X) and labels (Y) for learning logistic regression
  - Maximum likelihood solution
    - Values of potential functions learned using convex optimization



MRF: Pros and Cons

- Pros
  - Define arbitrary potential functions
  - Much more flexibility than directed models
  - Easier to compute condition independence
  - Don't need to express causation relationship
- Cons
  - Z!!!
  - Sum over all states x
  - M discrete nodes, each with K discrete states, K^M
    - However, we just need Z for learning
    - To evaluate we need most likely option, so Z cancels

Generative Model for Generative Models

There are tons of graphical models!

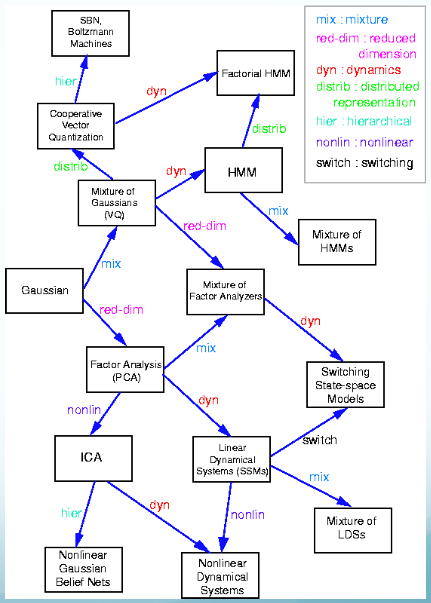


Figure by Ghahramani and Roweis via Murphy

Inference in Graphical Models

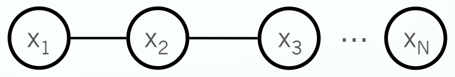
Inference

- Computing probabilities of network configurations
  - We know some values of the network (observed)
  - How do we compute the posterior of a set of nodes?
- Previously, we did this by explicitly working out the probabilities
- How can we do this in an efficient and general way?

Two Approaches

- Exact inference
  - We get the exact value of the probability we want
  - While some efficient algorithms exist, very slow for some graphs
- Approximate inference
  - Compute an approximation of the desired probability
  - The only solution for some types of graphs

Chain Graphical Model



- Consider a linear chain of random variables
  - Notice this is undirected
  - We can convert directed models to undirected models
    - See book
- Joint distribution of the chain

$$p(x) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\ \psi_{2,3}(x_2,x_3)\ \ldots\ \psi_{N-1,N}(x_{N-1},x_N)$$

  - Given N nodes with K states
  - Each potential function is a K*K table
  - Joint has $(N-1)K^2$ parameters

Chain Inference

- What is $p(x_n)$ for some node n in the chain?
  - Assuming no observed nodes
- Sum over all other nodes in the chain

$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(x)$$

  - Notice there are K^N values to consider in the summation
  - Our computations are **exponential** in the length of the chain

Conditional Independence

- Conditional independence to the rescue!
  - We can write the joint in terms of potentials
  - Each potential depends on 2 nodes
- Plug the factorized joint into the marginal for $p(x_n)$

$$p(x) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\ \psi_{2,3}(x_2,x_3)\ \ldots\ \psi_{N-1,N}(x_{N-1},x_N)$$

$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(x)$$

Conditional Independence

- Consider the final summation for variable x_N
- Only one potential depends on x_N
- We can perform this summation first to give a function of x_{N-1}

$$\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

- Since no other terms depend on x_N we can push this summation all the way in
- The same is true starting from the other side of the chain for x_1

Grouping Potentials

$$p(x_n) = \frac{1}{Z} \overbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2)\right]\right] \cdots\right]}^{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots\right]}_{\mu_\beta(x_n)}$$

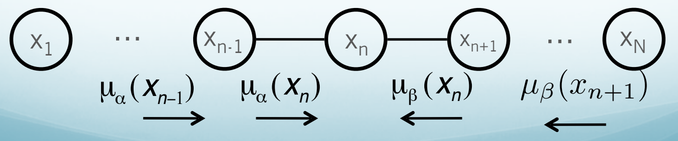
- Cost? N-1 summations over K states
  - Each computation is a factor with a K*K table
  - O(NK²) – linear in the length of the chain

Rewriting As Factors

- The marginal can be written in terms of two factors

$$p(x_n) = \frac{1}{Z} \mu_\alpha(x_n) \mu_\beta(x_n)$$

  - Each factor depends only on the nodes to one side
  - This information is passed along the network to x_n
  - We call this a message
    - Each message contains K values (for every x_n),



Calculating the Message

- Messages are computed recursively (based on other messages)

$$\mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \left[\sum_{x_{n-2}} \cdots\right]$$

$$= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1})$$

- Base case

$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2)$$

- The same is true for $\mu_\beta(x_n)$

Normalization Constant

- Z is a sum over all states in

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

  - which is O(K)
- This is an example of a message passing algorithm

Computing All Marginals?

- What if we wanted to compute $p(x_n)$ for all n?
  - Just run the message passing algorithm ($O(NK^2)$) n times: $O(N^2K^2)$
- This is wasteful since we keep computing the same messages many times
  - Instead, compute them once and save them: $O(NK^2)$

Computing Probabilities

- What about observed variables?
  - Clamp their value, remove the sum
- What about joint distribution over two neighboring nodes?

$$p(x_{n-1}, x_n) = \frac{1}{Z}\mu_\alpha(x_{n-1})\psi_{n-1,n}(x_{n-1}, x_n)\mu_\beta(x_n)$$

- This is great because we can parameterize the potentials and learn them
  - How do we learn these parameters?
  - EM! In fact, this is the E step (assume you have observations, compute the probability given the parameters)

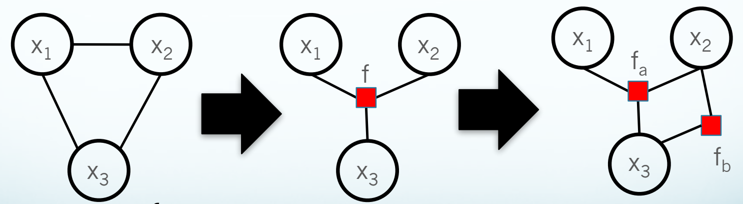
Factor Graphs

- Both types of graphical models (directed/ undirected) allow a global function to be expressed as a product of local factors
  - This factorization is what makes them powerful
- We can make this explicit by representing these factors directly
- Write the joint distribution directly in terms of factors
  - $\mathbf{x}_s$ - a subset of the variables
  - f_s - function of the subset of variables

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

Factor Graphs

- We can construct a factor graph to represent a given graphical model
  - For an undirected model, the factors are potential functions over the maximal cliques



$\psi(x_1, x_2, x_3)$ $\quad f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$

$f_a(x_1, x_2, x_3)\, f_b(x_2, x_3) = \psi(x_1, x_2, x_3)$

Factor Graphs

- Factor graphs are bipartite
  - Two types of nodes: nodes and factors
  - Nodes only link to factors, factors only link to nodes
- Multiple factor graphs represent the same distribution
  - Factor graphs can be more specific about factorization

Sum Product Algorithm

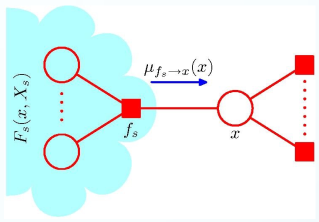
- Factor graphs are used to derive the sum product algorithm
  - A powerful class of efficient exact inference algorithms for tree-structured graphs
- How do we find the marginal

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

  - X\x- the set of variables in x with variable *x* omitted
- We will replace p(**x**) with factor graph to obtain efficient algorithm

Sub-Tree

- Let's consider a fragment of the tree, which allows us to partition the joint into two sets of factors



- We can write the joint as

$$p(\mathbf{x}) = \prod_{s \in ne(x)} F_s(x, X_s)$$

  - ne(x)- factors that are neighbors of x
  - X_s- all variables in sub-tree connected via factor f_s
  - $F_s(x,X_s)$- product of all factors associated with f_s

Factor Messages

- Combining these equations

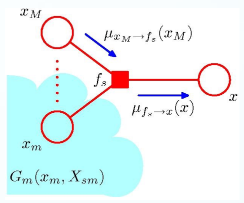
$$p(\mathbf{x}) = \prod_{s\in ne(x)} F_s(x, X_s) \qquad p(x) = \sum_{\mathbf{x}\setminus x} p(\mathbf{x}) \quad \Rightarrow \quad p(x) = \prod_{s\in ne(x)} \left[\sum_{X_s} F_s(x, X_s) \right] = \prod_{s\in ne(x)} \mu_{f\to x}(x)$$

- Messages between factor f_s to node x

$$\mu_{f\to x}(x) = \sum_{X_s} F_s(x, X_s)$$

- The marginal is a product of all incoming messages

Sub-Factors



- Notice that the factors themselves can be broken into factor sub-trees
  - Each message is written

$$\mu_{f_s\to x_m}(x_m) = \sum_{x_1} \cdots \sum_{x_m} f_s(x, x_1 \ldots x_m) \prod_{m\in ne(f_s)\setminus x} \sum_{X_{xm}} G_m(x_m, X_{sm})$$

$$= \sum_{x_1} \cdots \sum_{x_m} f_s(x, x_1 \ldots x_m) \prod_{m\in ne(f_s)\setminus x} \mu_{x_m\to f_s}(x_m)$$

  - G_m is the product of all factors associated with x_m
  - Can replace last term by a message from node to factor

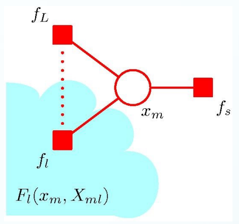
Node Messages

- Messages between factor nodes and factors

$$\mu_{x_m\to f_s}(x_m) = \sum_{X_{sm}} G_m(x_m, X_{sm})$$

- Two different types of messages
  - Nodes to factors
  - Factors to nodes
- Messages passed by a node are a product of variables connected to that node

Evaluating Messages



- How do we evaluate messages from variable nodes to factor nodes?
  - Sub Graph factorization!

$$G_m(x_m, X_{sm}) = \prod_{l\in ne(x_m)\setminus f_s} F_l(x_m, X_{ml})$$

  - $G_m(x_m, X_{sm})$ - associated with node x_m
  - $F_l(x_m, X_{ml})$ - associated with factor f_l linked to x_m
  - Product over all neighbors except f_s

Messages

- Using this definition we can write messages from nodes to factors

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right]$$
$$= \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

- If only two neighbors, then just pass the message
- Requires input from all other factors before can send a message

Recursion

- These messages are all based on recursion
- Base cases (leaves of the network)?

$$\mu_{x \to f}(x) = 1 \qquad \mu_{f \to x}(x) = f(x)$$

Summary: Sum Product

- We want to evaluate marginal p(x) in a tree factor graph
  - Assume x is root, start messages at the leaves
  - Propagate messages until x receives messages
    - Wait until all neighbors except one send message, then create new message
  - When x receives all messages, compute p(x)

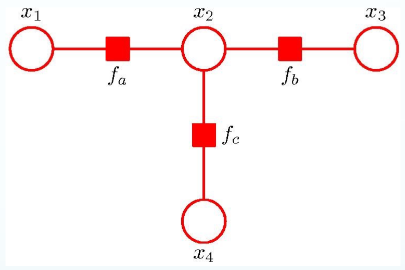
$$p(x) = \prod_{s \in ne(x)} \mu_{f \to x}(x)$$

  - As before, we can compute the messages over the whole graph and use them to compute every marginal

Normalization

- If this is an undirected graph, then the result is a distribution that is not normalized
  - If we started from a directed graph, it is normalized
- Normalization is easy
  - Recall with chains we computed Z by summing over all configurations for the node(s) of interest
  - We have the messages to compute this here

Sum Product Example



Messages

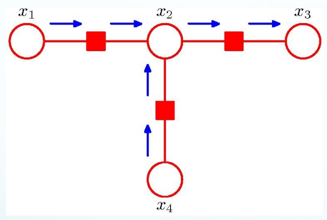
$$\mu_{f\to x}(x) = \sum_{X_s} F_s(x, X_s)$$

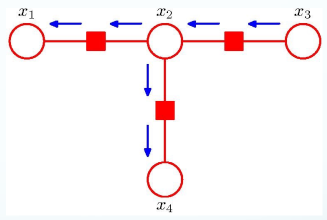
$$\mu_{x_m\to f_s}(x_m) = \prod_{l\in ne(x_m)\setminus f_s} \mu_{f_l\to x_m}(x_m)$$

$$\mu_{f\to x}(x) = f(x)$$

$$\mu_{x\to f}(x) = 1$$

Sum Product Example

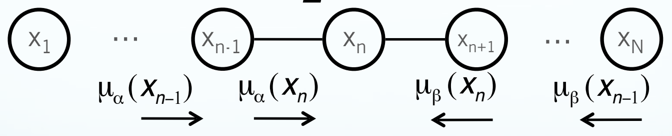




Inference in Chains

- The marginal can be written in terms of two messages

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$



- This is sum product!
    - We just need to add factors.

Most Likely Configuration

- We know how to find the probability of configuration
- How do we find the most likely configuration?
  - Run the Sum Product algorithm for each marginal
  - Select the most probable value for each node

  - Problem: this gives a node specific max probability
  - We want most likely values for all of the nodes together

Inference

- The goal of inference is to find the probability of a certain configuration
- Examples
  - The probability of a configuration x?
  - The probability of $x_2=1$?
  - **The most likely configuration?**

Max Product Algorithm

- Computes the configuration of the graph that gives the highest global probability
- Replace sum in Sum Product with max

$$\mathbf{x}^{\max} = \arg\max_{\mathbf{x}} p(\mathbf{x})$$

- Key trick in Sum Product
  - $ab + ac = a(b + c)$
- Key trick in Max Product
  - $\arg\max(ab, ac) = a\arg\max(b, c)$

Moving Out Terms

- Write out the max terms
  - $\mathbf{x}^{\max} = \arg\max_{\mathbf{x}} p(\mathbf{x}) = \arg\max_{x_1} \ldots \arg\max_{x_N} p(\mathbf{x})$
- For chain nodes, we write
  - $$\max_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z}\max_{x_1}\cdots\max_{x_N}\left[\psi_{1,2}(x_1,x_2)\cdots\psi_{N-1,N}(x_{N-1},x_N)\right]$$
    $$= \frac{1}{Z}\max_{x_1}\left[\psi_{1,2}(x_1,x_2)\cdots\left[\max_{x_N}\psi_{N-1,N}(x_{N-1},x_N)\right]\cdots\right]$$

Message Passing

- To avoid underflow, take log(max)
- Using the message passing formulation from Sum Product, we obtain new messages for Max Product

$$\mu_{f_s \to x}(x) = \max_{x_1 \cdots x_N}\left[\log f_s(x, x_1 \ldots x_N) + \sum_{m \in ne(f_s)\backslash x} \mu_{x_m \to f}(x_m)\right]$$

$$\mu_{x \to f_s}(x) = \sum_{l \in ne(x)\backslash f_s} \mu_{f_l \to x_m}(x)$$

- Base case

$$\mu_{f_s \to x}(x) = \log f_s(x)$$

$$\mu_{x \to f_s}(x) = 0$$

Root node

$$x^{\max} = \arg\max_x \left[\sum_{s \in ne(x)} \mu_{f_s \to x}(x)\right]$$

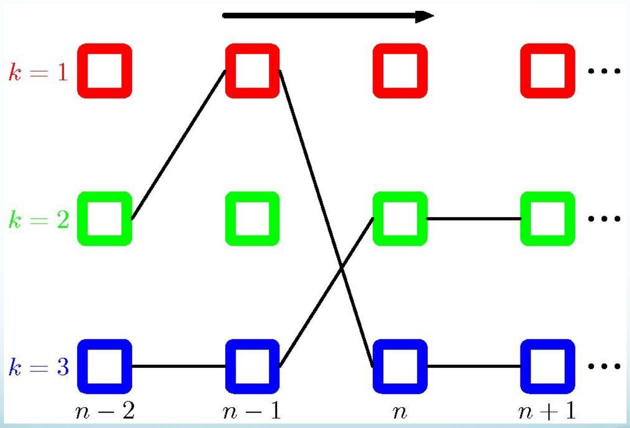
Messages from Root

- We can't pass messages back as before
  - There may be multiple configurations that have max value for p(**x**)
  - Need to distinguish between these configurations
- Solution: keep track of which states correspond to the same max configuration
  - Store quantities:

$$\phi(x_n) = \arg\max_{x_{n-1}}[\log f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \to f_{n-1,n}}(x_n)]$$

    - The argument at x_n that gave the highest probability
    - To find the best path we follow the values for ϕ(x)

Example for a Chain



Inference Review

- Message Passing Algorithms for Exact Inference
  - Each node computes a local message to send to its neighbors
  - Sum Product
    - Compute marginals for nodes in the graph
    - Efficient way to compute all marginals
  - Max Product
    - Find the highest probability configuration
    - Back track through graph to decode entire configuration
- Non-tree structured graphs
  - Exact inference algorithms exist for certain types of graphs

Graphical Models Review

Graphical Models Overview

- Types of models
  - Directed Graphical Models
    - Bayesian Networks
  - Undirected Graphical Models
    - Markov Random Fields
  - Factor Graphs

Model Types

- Directed Graphical Models
  - CPTs (everything is already normalized)
  - Training often by counting
- Undirected Graphical Models
  - Factors replace CPTs
  - Learn the factors by discriminative training
  - Generative training requires more complex Z

Efficient Algorithms

- Message passing algorithms
- Computing marginal probabilities
  - Sum Product Algorithm for trees
- Computing max probability configuration
  - Max Product Algorithm for trees
  -

Learning Settings

- Unsupervised learning
  - We don't observe all of the variables
- Semi-supervised learning
  - We observe variables for only some examples
- Supervised learning
  - We observe all the variables at training

Remaining Issues

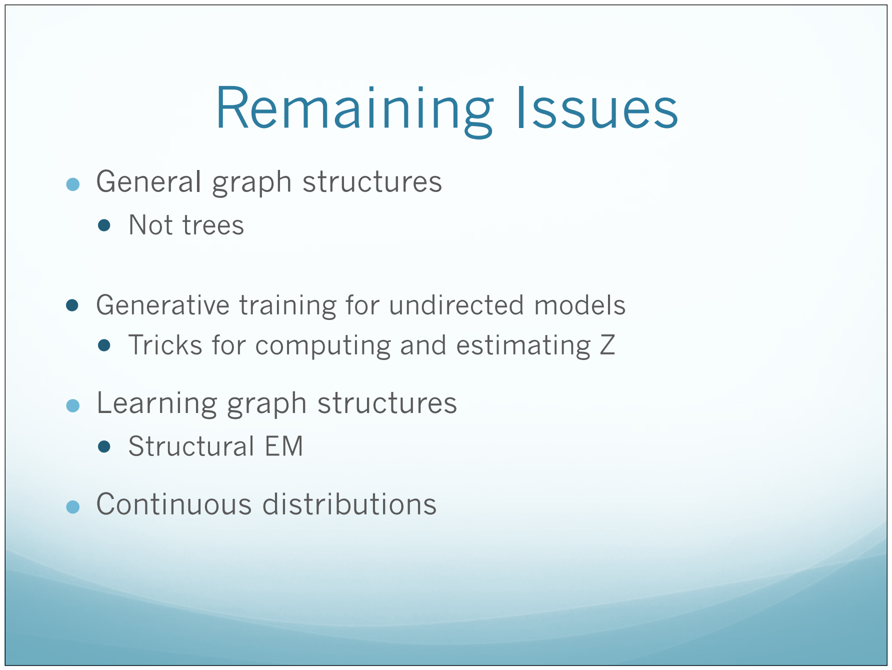
- Approximate Inference
  - Computing probabilities in non-tractable distributions
    - Distribution is dimensionality of latent space is too high
    - Posterior distribution is complex and can't be computed analytically
- Examples
    - Variational inference
    - Sampling methods

Variational Inference

- We can't compute the marginal p(**X**,**Z**)
  - Approximate p(**Z**|**X**) and p(**X**)
  - Maximize p(**X**)
    - Decompose p(**X**) using EM idea into lower bound and KL
- Want to minimize KL between p and our estimate q
  - q is non-tractable
  - Replace with an approximating distribution q*

Sampling Methods

- We want to compute the marginal for p(**X**)
  - Cannot compute the expectation of the distribution
  - We can draw samples from the distribution
- Idea: approximate the expectation by taking many samples
- Examples
  - Rejection sampling
    - Sample and throw away points that don't match what you want
  - Gibbs Sampling (Markov Chain Monte Carlo)
    - Draw a new value for one variable, update the others



Remaining Issues

- General graph structures
  - Not trees
- Generative training for undirected models
  - Tricks for computing and estimating Z
- Learning graph structures
  - Structural EM
- Continuous distributions