STA 4273H: Statistical Machine Learning

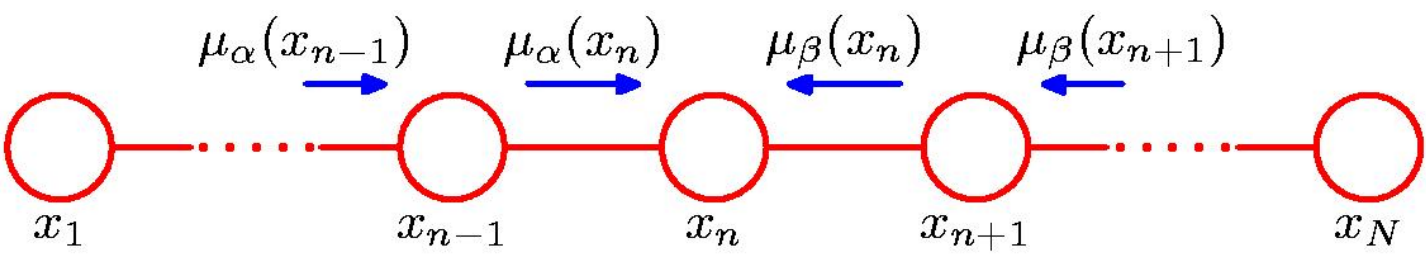
Russ Salakhutdinov

Department of Statistics
rsalakhu@utstat.toronto.edu
http://www.utstat.utoronto.ca/~rsalakhu/
Sidney Smith Hall, Room 6002

Lecture 5

Inference in Graphical Models

- Inference: Some of the nodes are clamped to observed values, and we wish to compute the posterior distribution over subsets of other unobserved nodes.
- We will often exploit the graphical structure to develop efficient inference algorithms.
- Many of the inference algorithms can be viewed as the propagation of local messages around the graph.

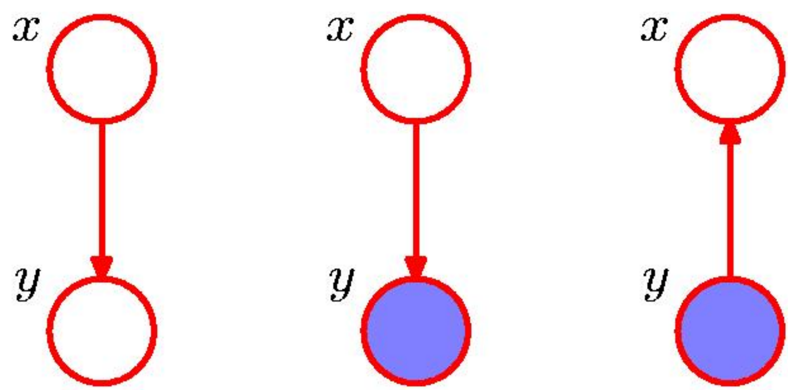


Graphical Representation of Bayes' Rule

• Assume that the joint decomposes into the product:

$$p(x, y) = p(x)p(y|x).$$

• We next observe the value of y.



• We can use Bayes' Rule to calculate the posterior:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

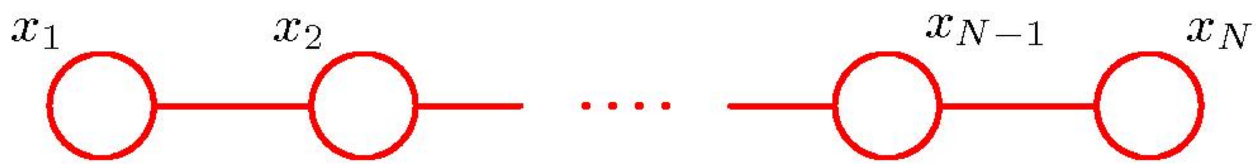
$$p(y) = \sum_{x'} p(y|x')p(x')$$

• The joint is now expressed as: $p(x, y) = p(y)p(x|y)$.

• This represents the simplest example of an inference problem in a graphical model.

Inference on a Chain

• Consider a more complex problem involving a chain of nodes.



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

• We will assume each node is a K-state discrete variable and each potential function is a K by K table, so the joint has (N-1)K^2 parameters.

• Consider the inference problem of finding the marginal distribution over x_n by summing over all values of all other nodes:

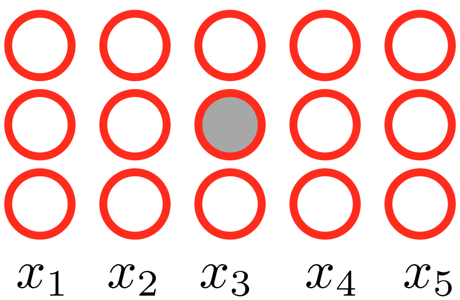
$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$$

• There are N variables with K discrete states, hence there are K^N values for **x**. Hence computation scales exponentially with the length N of the chain!

Pictorial Representation

- Consider the example of N=5 variables, with K=3 states:
- Suppose that we would like to compute the marginal probability $p(x_3)$.

$$p(\mathbf{x}) = \frac{1}{\mathcal{Z}} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \psi_{3,4}(x_3, x_4) \psi_{4,5}(x_4, x_5).$$



$$p(x_3) = \sum_{x_1} \sum_{x_2} \sum_{x_4} \sum_{x_5} p(x_1, x_2, x_3, x_4, x_5).$$

- We can exploit the conditional independence properties of the graphical model:

$$p(x_3) = \frac{1}{\mathcal{Z}} \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \right] \times$$

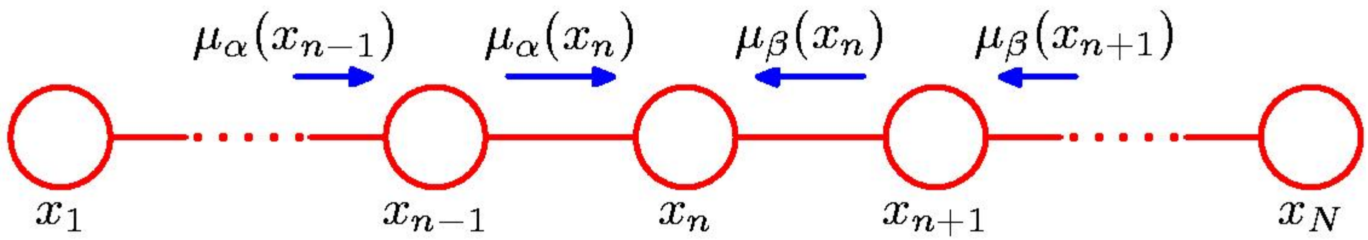
Left branch

$$\left[\sum_{x_4} \psi_{3,4}(x_3, x_4) \left[\sum_{x_5} \psi_{4,5}(x_4, x_5) \right] \right].$$

Right branch

Inference on a Chain

- We can derive the recursive expressions for the left-branch and right-branch messages



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

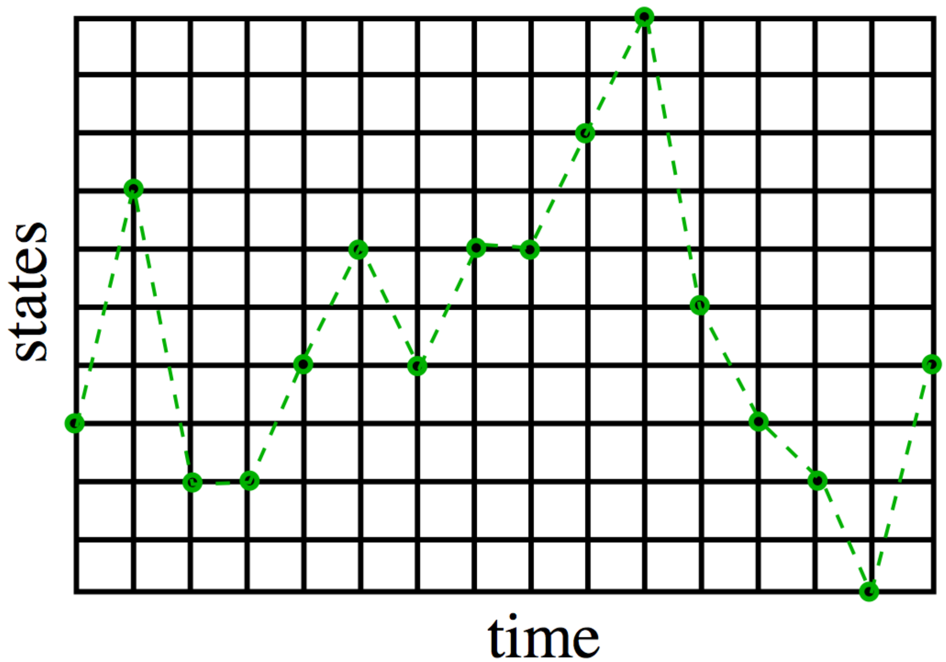
$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right]}_{\mu_\beta(x_n)}$$

- Key concept: multiplication is distributive over addition:

$$ab + ac = a(b + c).$$

Clever Recursion

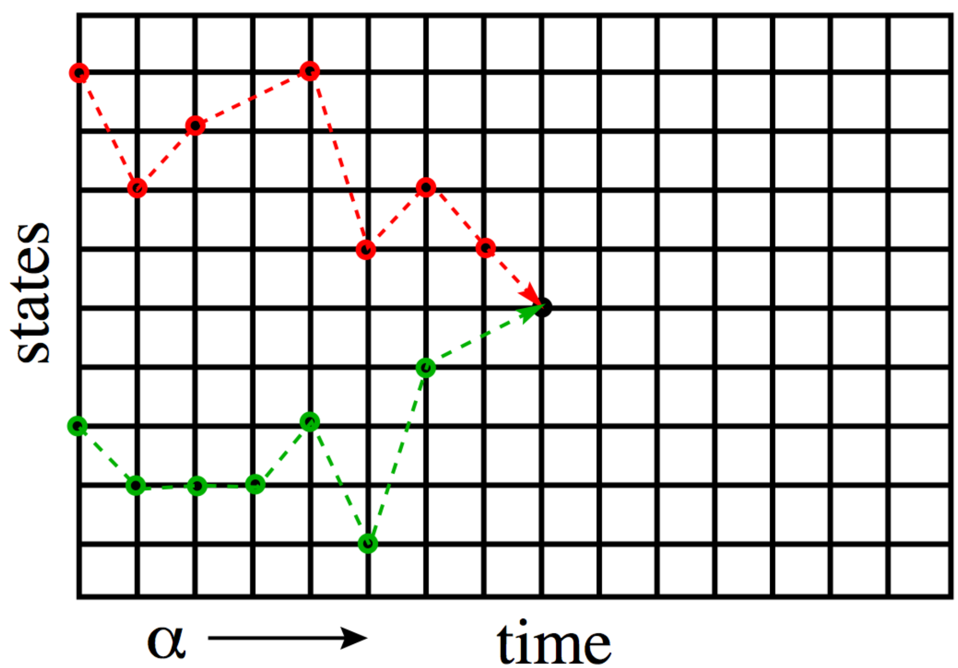
- Naive algorithm:
  1. Start bug in each state at t=1 holding value 0.
  2. Move each bug forward in time by making copies of it and incrementing the value of each copy by the probability of the transition.
  3. Go to 2 until all bugs have reached time T.
  4. Sum up values on all bugs.



Exponential Cost.

Clever Recursion

- Clever recursion:
  - add a step between 2 and 3 above which says:
    - at each node, replace all the bugs with a single bug carrying the sum of their values.
- Also known as dynamic programming.



Computational Cost

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

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

- We have to perform N-1 summations, each of which is over K states.
- Each local summation involves K $\times$ K table of numbers.
- The total cost of evaluating the marginal is O(NK^2) which is linear in the length of the chain (not exponential).
- We were able to exploit the conditional independence properties of this graph to obtain an efficient calculation.
- If the graph were fully connected, we could not exploit its independence properties, and would be forced to work with the full joint.

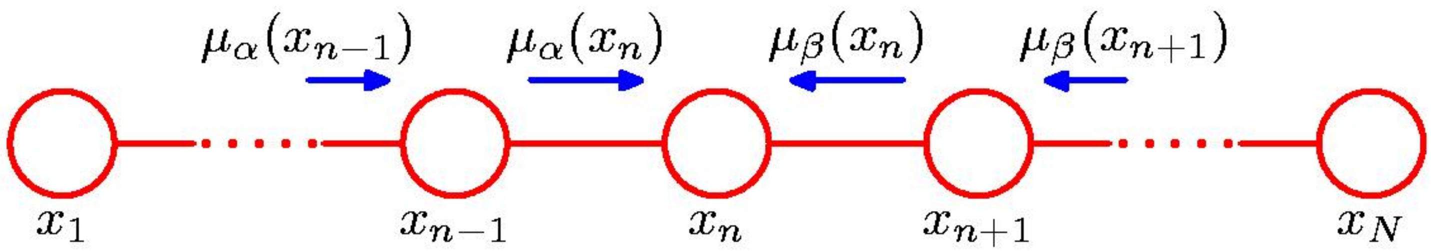
Message Passing

- We can give an interpretation in terms of passing local messages on a graph.
- The marginal decomposes into a product two factors and a normalizing constant.

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

- We can interpret $\mu_\alpha(x_n)$ as a message passed forwards on a chain from node x_{n-1} to node x_n.
- Similarly, $\mu_\beta(x_n)$ can be interpreted as a message passed backwards on a chain from node x_{n+1} to node x_n.
- Each message contains a set of K values, one for each choice of x_n, so the product of two messages represents a point-wise multiplication of the corresponding elements.

Message Passing



- The forward message can be evaluated recursively:

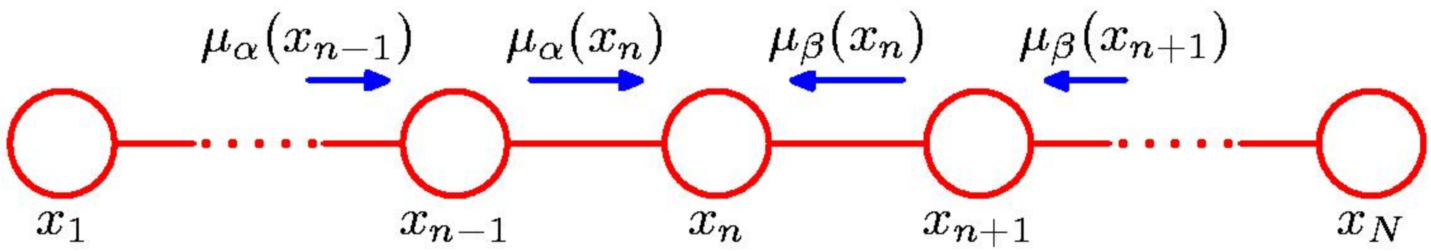
$$\begin{aligned}\mu_\alpha(x_n) &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \left[\sum_{x_{n-2}} \cdots \right] \\ &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1}).\end{aligned}$$

- We therefore first evaluate:

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

- Note that the outgoing message is obtained by multiplying the incoming message by the local potential and summing out the node variable.

Message Passing



- The backward message can be evaluated recursively starting at node x_N.

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

$$\begin{aligned}
\mu_\beta(x_n) &= \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \left[\sum_{x_{n+2}} \cdots \right] \\
&= \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \mu_\beta(x_{n+1}).
\end{aligned}$$

- These types of graphs are called Markov chains and the corresponding message passing equations represent an example of the Chapman-Kolmogorov equations for Markov processes.

Inference on a Chain

- To compute local marginal distributions:
  - Compute and store all forward messages: $\mu_\alpha(x_n)$
  - Compute and store all backward messages: $\mu_\beta(x_n)$
  - Compute normalizing constant Z at any node x_n
  - Compute

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

- The marginals for adjacent pair of nodes:

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

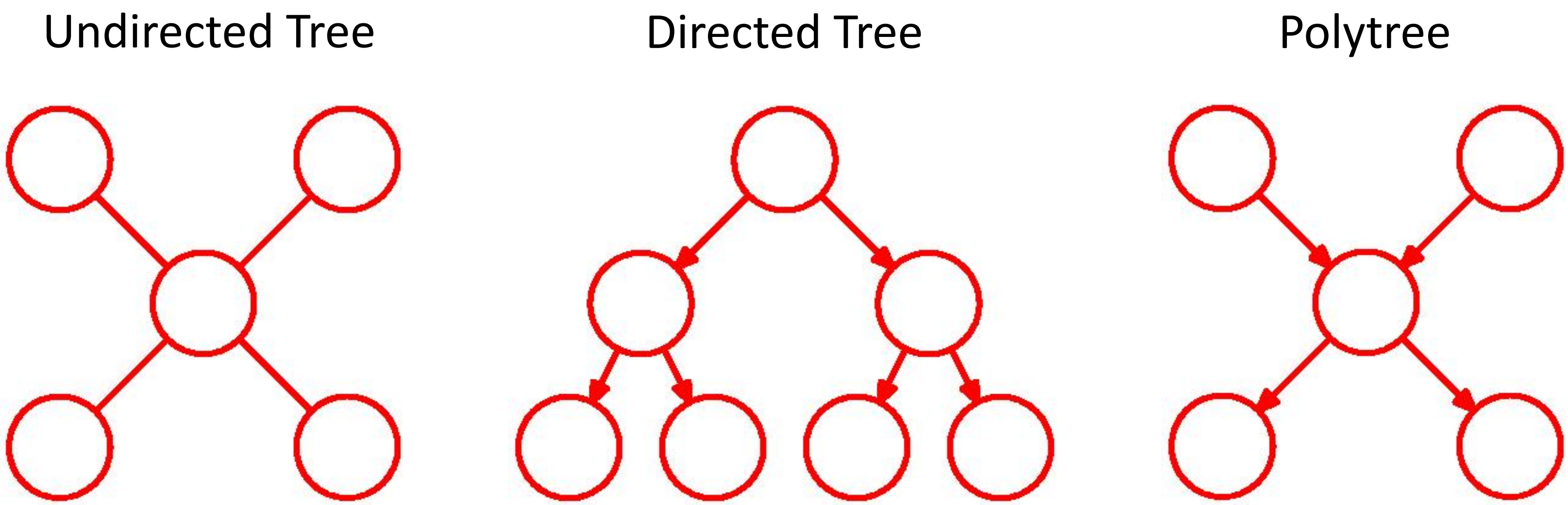
- No additional computation required to compute these pairwise marginals.

- Basic framework for learning Linear Dynamical Systems (LDS) and Hidden Markov Models (HMMs)

Inference on Trees

- This message passing algorithm generalizes easily to any graph which is singly connected. This includes trees and polytrees.

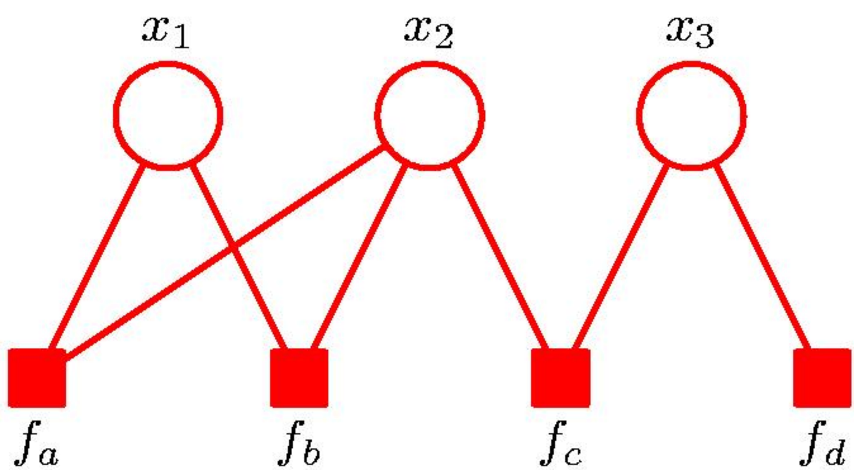
- Each node sends out along each link the product of the messages it receives on its other links.



Factor Graphs

• In addition to nodes representing the random variables, we also introduce nodes for the factors themselves.

$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3) \qquad p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$



• If the potentials are not normalized, we need an extra factor corresponding to 1/Z.

• Each factor f_s is a function of the corresponding variables $\mathbf{x_s}$.

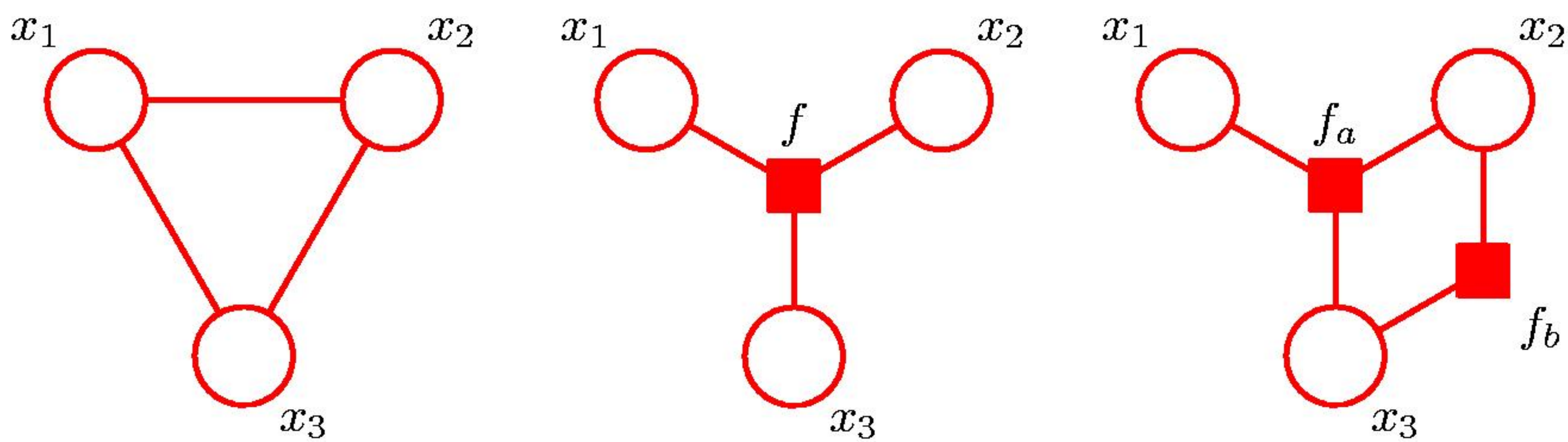
• Note that there two factors that are defined over the same set of variables.

• Each potential has its own factor node that is connected to all the terms in the potential.

• Factor graphs are always bipartite.

Factor Graphs for Undirected Models

• An undirected graph can be readily converted to a factor graph.



$$\psi(x_1, x_2, x_3)$$

$$\begin{aligned} &f(x_1, x_2, x_3) \\ &= \quad \psi(x_1, x_2, x_3) \end{aligned}$$

$$\begin{aligned} &f_a(x_1, x_2, x_3) f_b(x_2, x_3) \\ &= \quad \psi(x_1, x_2, x_3) \end{aligned}$$

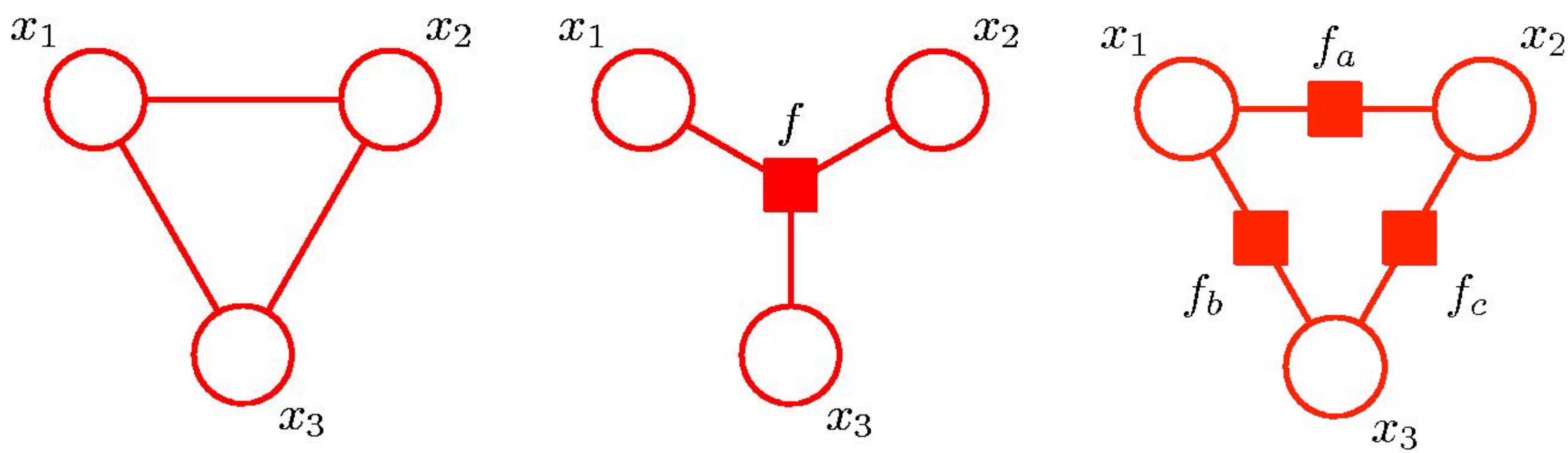
• The third-order factor is more visually apparent than the clique of size 3.

• It easy to divide a factor into the product of several simpler factors. This allows additional factorization to be represented.

• There can be several different factor graphs that correspond to the same undirected graph.

Factor Graphs for Undirected Models

• An undirected graph can be readily converted to a factor graph.



$$\psi(x_1, x_2, x_3)$$

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

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

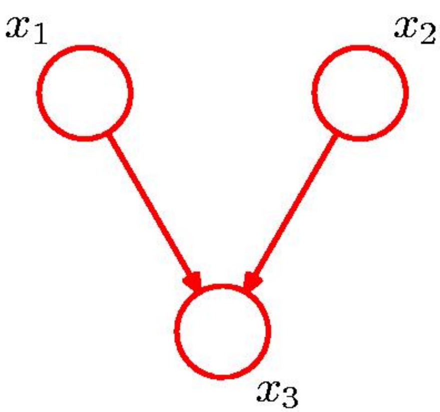
• The third-order factor is more visually apparent than the clique of size 3.

• It easy to divide a factor into the product of several simpler factors. This allows additional factorization to be represented.

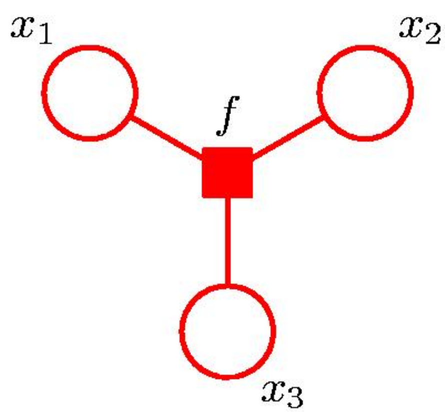
• There can be several different factor graphs that correspond to the same undirected graph.

Factor Graphs for Directed Models

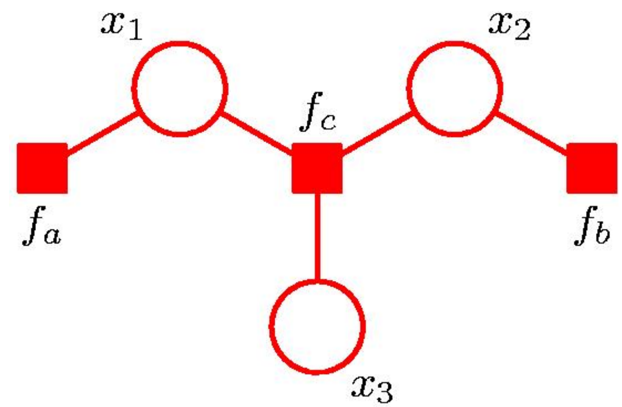
• Directed graphs represent special cases in which the factors represent local conditional distributions.



$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3|x_1,x_2)$$



$$f(x_1,x_2,x_3) = p(x_1)p(x_2)p(3|x_1,x_2)$$



$$\begin{aligned} f_a(x_1) &= p(x_1) \\ f_b(x_2) &= p(x_2) \\ f_c(x_1,x_2,x_3) &= p(x_3|x_1,x_2) \end{aligned}$$

• When converting any singly connected graphical model to a factor graph, it remains singly connected.

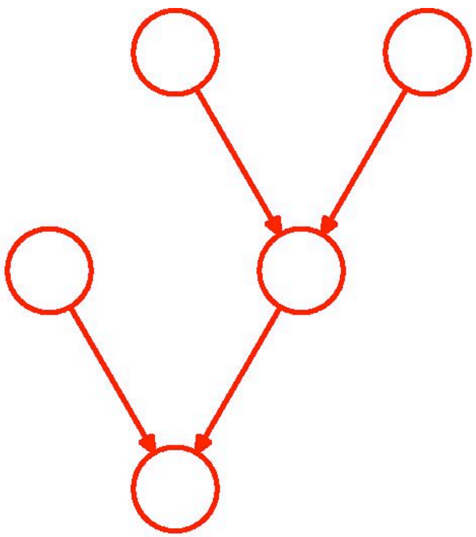
–This preserves the simplicity of inference.

• Converting a singly connected directed graph to an undirected graph may not result in singly connected graph.

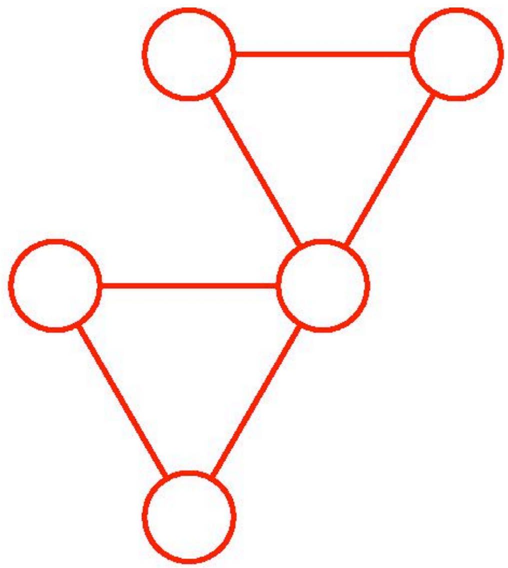
Directed Polytree

- Converting a directed polytree into an undirected graph creates loops.

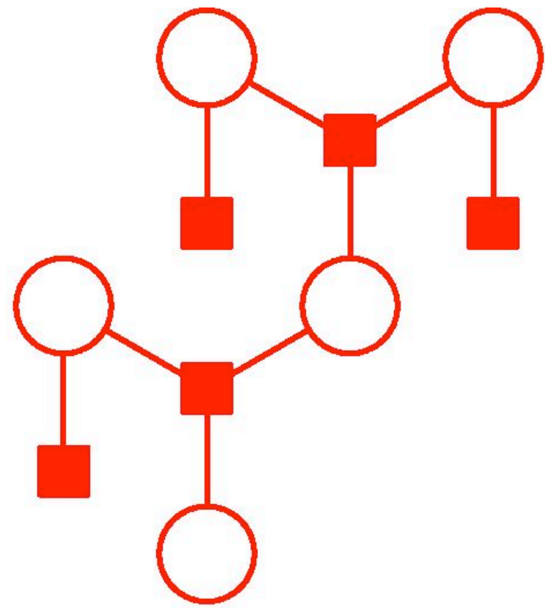
Directed polytree



Converting a polytree into an undirected graph



Converting a polytree into a factor graph



- Converting a directed polytree into a factor graph retains the tree structure.

Computing the Marginals

- We will use factor graph framework to derive a powerful class of efficient exact inference algorithms, known as sum-product algorithms.
- We will focus on the problem of evaluating local marginals over nodes or subset of nodes.
- We will assume that the underlying factor graph has a tree structure.
- The marginal is defined as:

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

- Naive evaluation will take exponential time.

Objective:

i. To obtain an efficient, exact inference algorithm for finding marginals.
ii. In situations where several marginals are required, to allow computations to be shared efficiently.

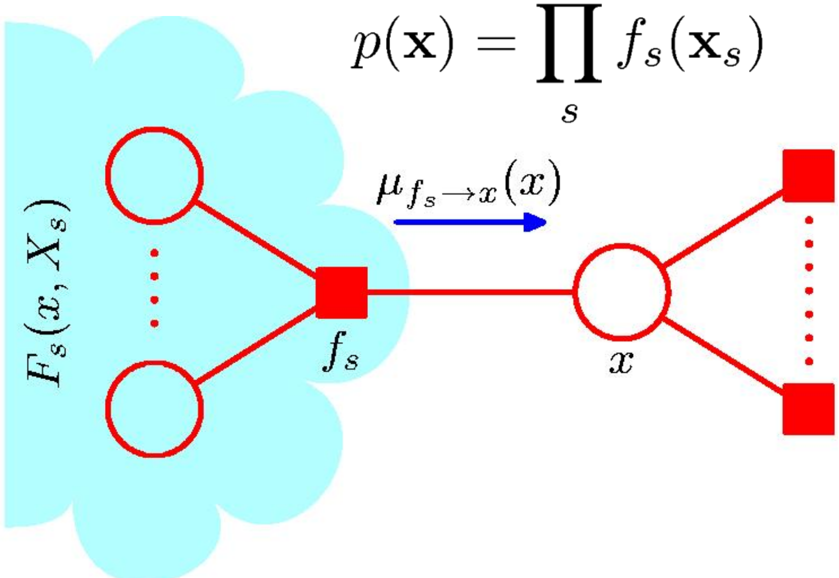
- Key idea: Distributive Law

$$ab + ac = a(b + c)$$

Sum Product Algorithm

- The key idea is to use is to substitute for p(x) using factor graph representation and interchange summations and products in order to obtain and efficient algorithm.

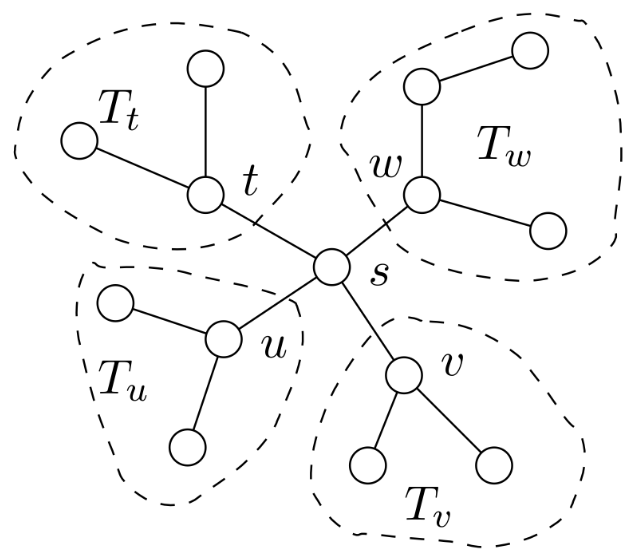
$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$



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

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

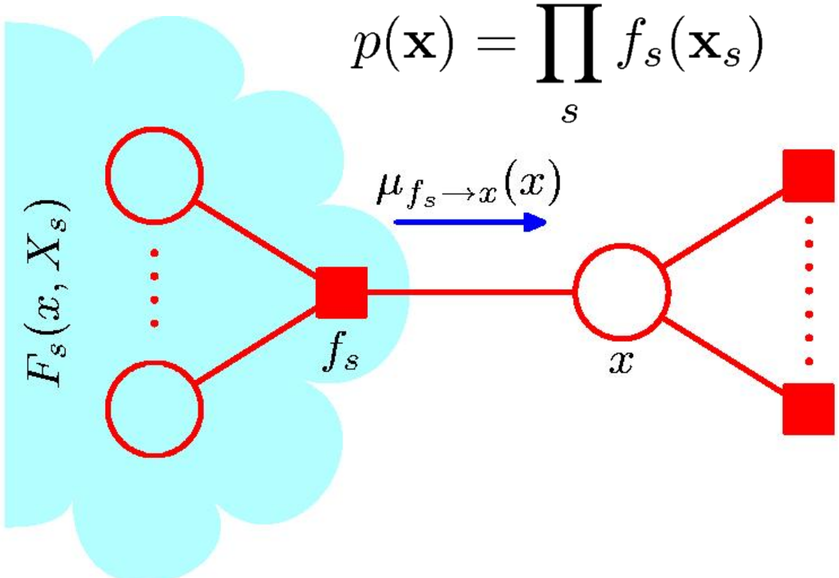
- Partition the factors in the joint distribution into groups.
- Each group is associated with each of the factor nodes that is a neighbor of x.



Sum Product Algorithm

- The key idea is to use is to substitute for p(x) using factor graph representation and interchange summations and products in order to obtain and efficient algorithm.

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



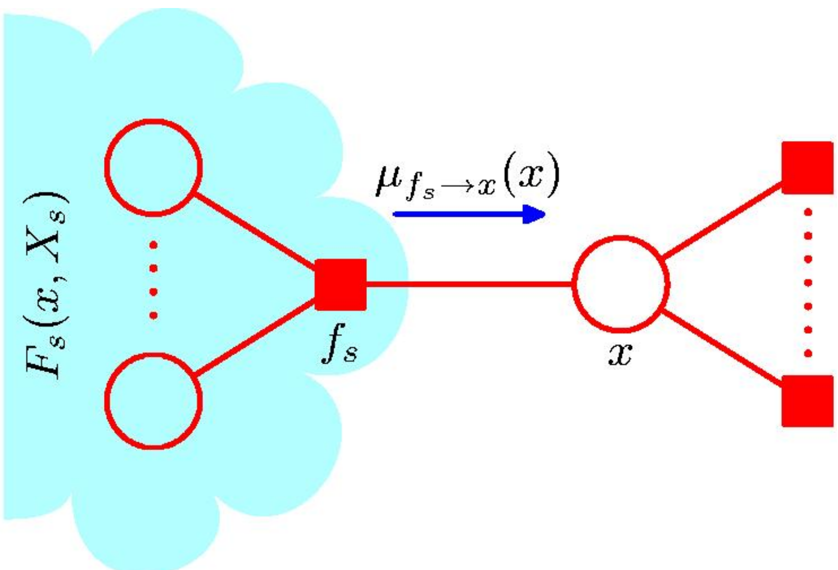
$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

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

- Partition the factors in the joint distribution into groups.
- Each group is associated with each of the factor nodes that is a neighbor of x.
- ne(x) denotes the set of factor nodes that are neighbors of x.
- X_s denotes the set of all variables in the subtree that is connected to x via the factor node f_s
- $F_s(x,X_s)$ denotes the product of all factors in the set associated with f_s.

Messages from Factors to Variables

• Interchanging sums and products we obtain:



$$
\begin{aligned}
p(x) &= \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right] \\
&= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x).
\end{aligned}
$$

• Introduce messages from factor node f_s to variable node x:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

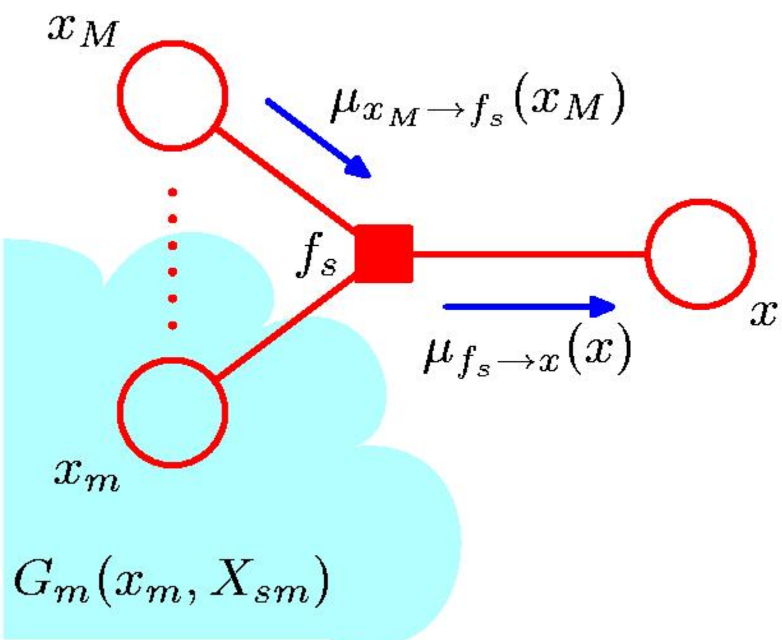
• The required marginal is given by the product of all incoming messages arriving at node x.

• Let us further examine how these messages will look like.

Messages from Factors to Variables

- Note that the term $F_s(x,X_s)$ is described by a product of factors defined over sub-graph.

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M)G_1\left(x_1, X_{s1}\right)\ldots G_M\left(x_M, X_{sM}\right)$$

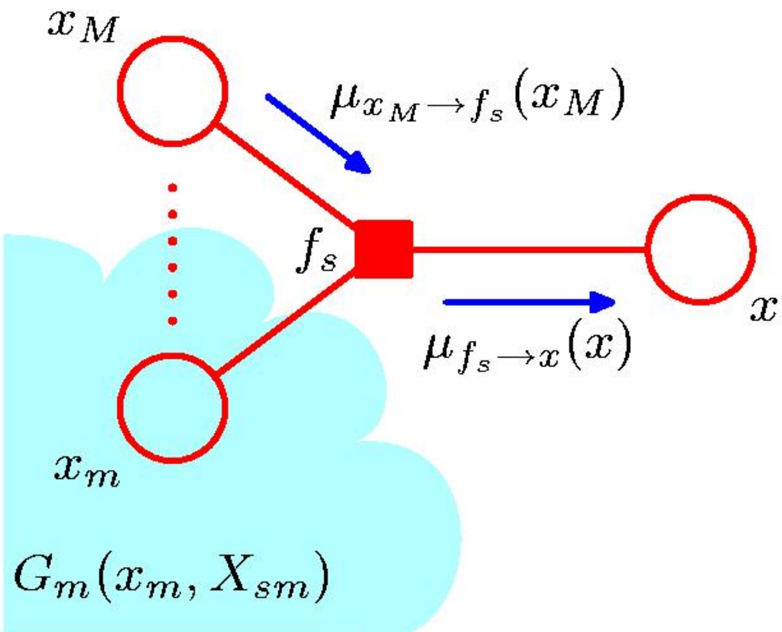


- We denote the variables associated with factor f_s, in addition to x, by $x_1,\ldots,x_M$.

Messages from Factors to Variables

- Note that the term $F_s(x,X_s)$ is described by a product of factors defined over sub-graph.

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \ldots G_M(x_M, X_{sM})$$



- Substituting into:

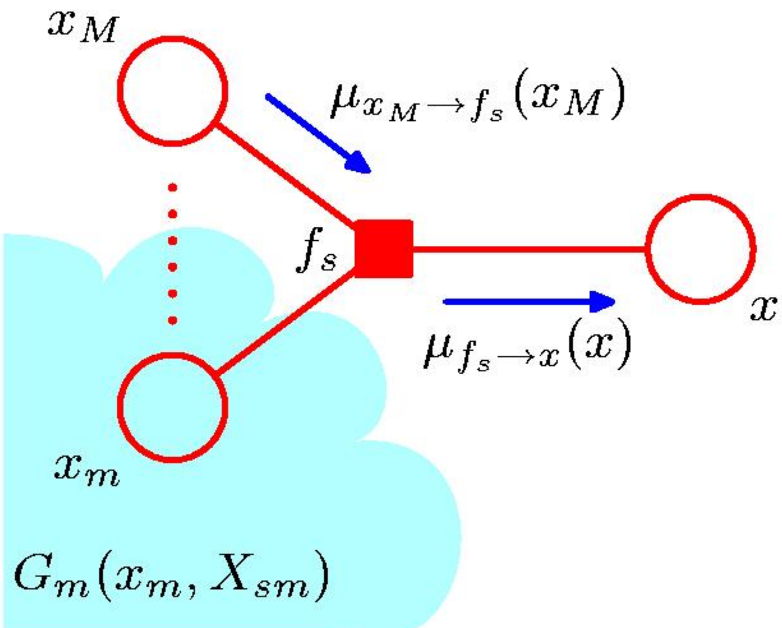
$$\begin{aligned} p(x) &= \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right] \\ &= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x). \end{aligned}$$

$$\begin{aligned} \mu_{f_s \to x}(x) &= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m) \end{aligned}$$

Messages from Factors to Variables

• There are two types of messages: those that go from factor nodes to variables and those that go from variables to factor nodes.

• To evaluate the message sent by a factor node to a variable node:



• Take the product of incoming messages along all other links coming into the factor node.

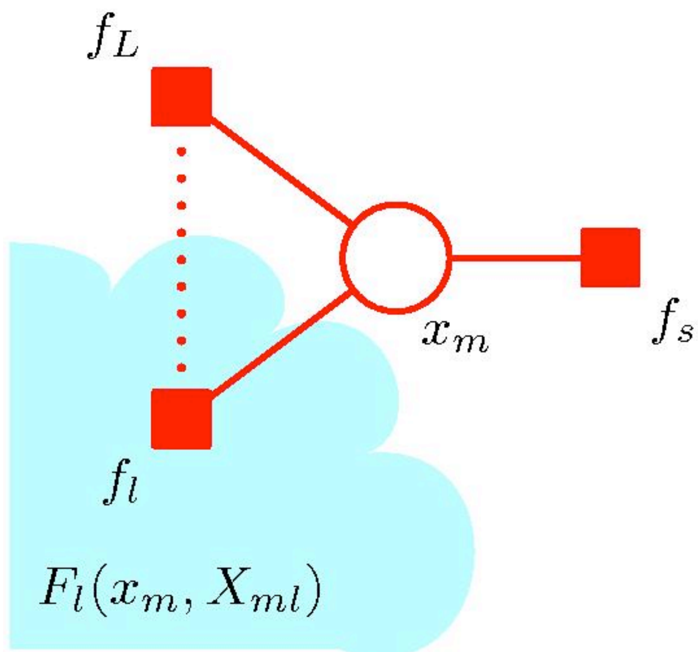
• Multiply by the factor associated with that node.

• Marginalize over the variables associated with incoming messages.

$$\begin{aligned}
\mu_{f_s \to x}(x) &= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right] \\
&= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)
\end{aligned}$$

Messages from Variables to Factors

- To evaluate the messages from variable nodes to factor nodes, we again make use of sub-graph factorization.



- The term $G_m(x_m, S_{sm})$ associated with node x_m is given by the product of terms $F_l(x_m, X_{ml})$.

- Each $F_l(x_m, X_{ml})$ is associated with one of the factor nodes f_l.

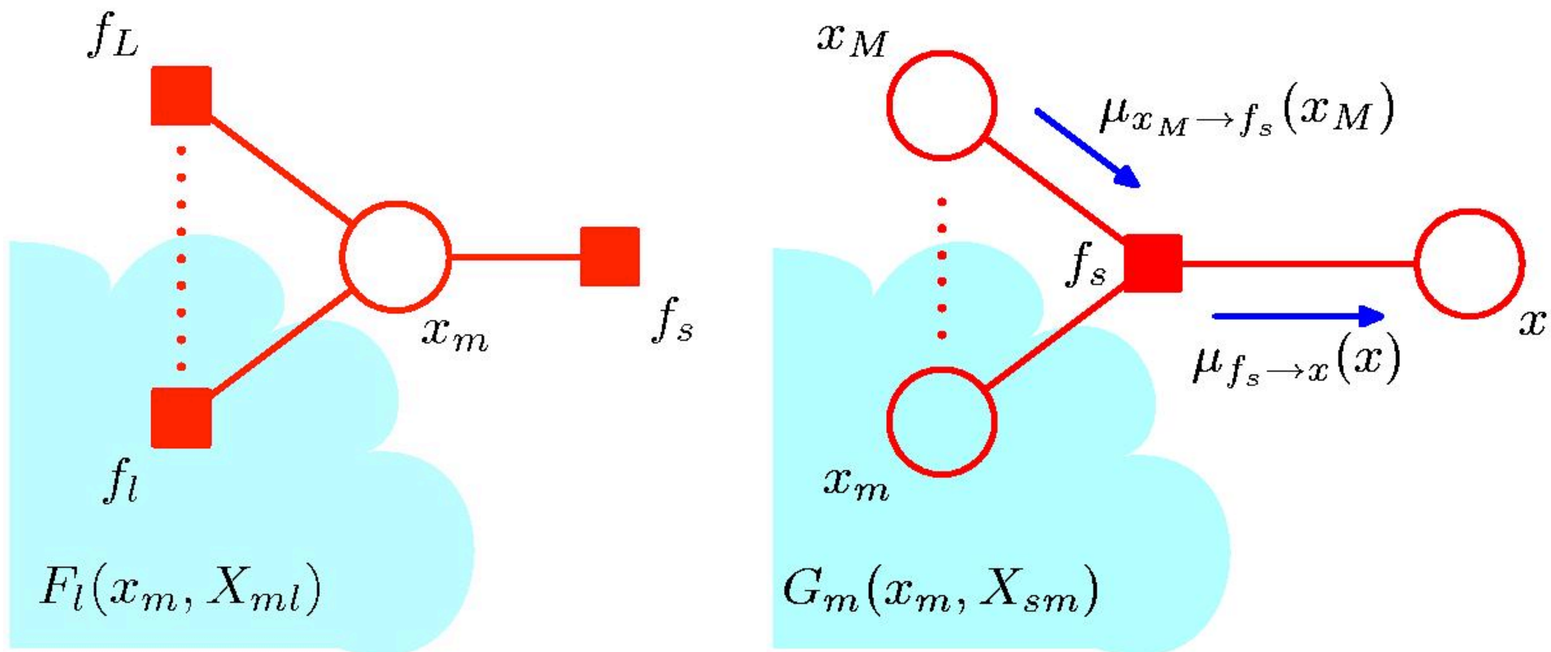
- Messages sent by a variable node to a factor node:
  - Take the product of the incoming messages along all links except for f_s.

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

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

Summary

- Two distinct kind of messages:

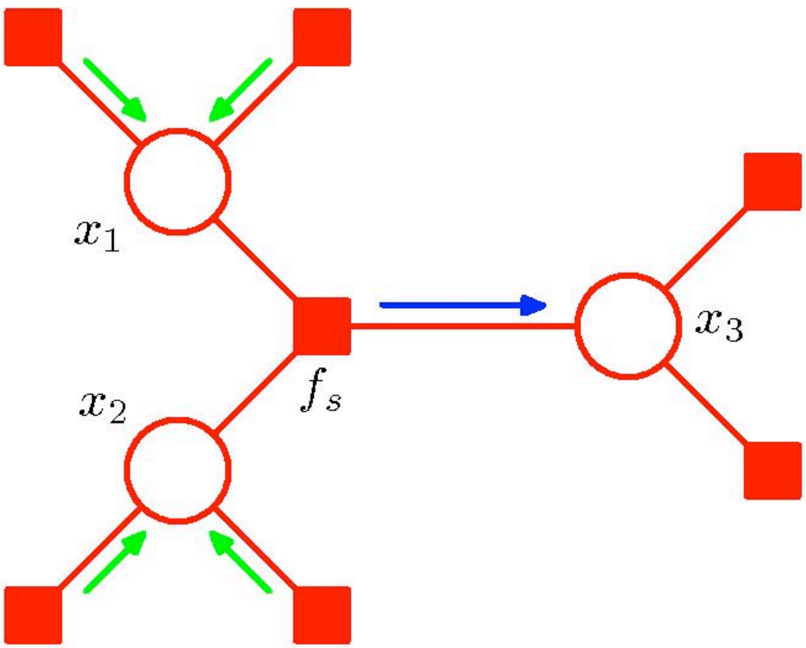


$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \backslash f_s} \mu_{f_l \to x_m}(x_m)$$

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

Another Interpretation

- The sum-product algorithm can be viewed in terms of messages sent out by factor nodes to other factor nodes.



The outgoing message (blue) is obtained by taking the product of all the incoming messages (green), multiplying by f_s, and marginalizing other x_1 and x_2.

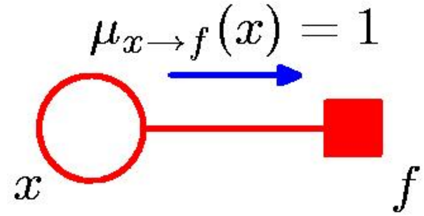
$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

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

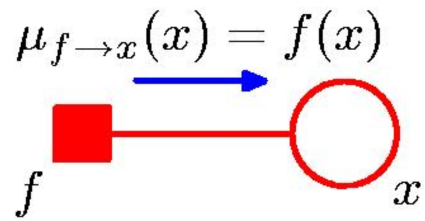
Initialization

• Messages can be computed recursively in terms of other messages.

- If the leaf node is a variables node then:

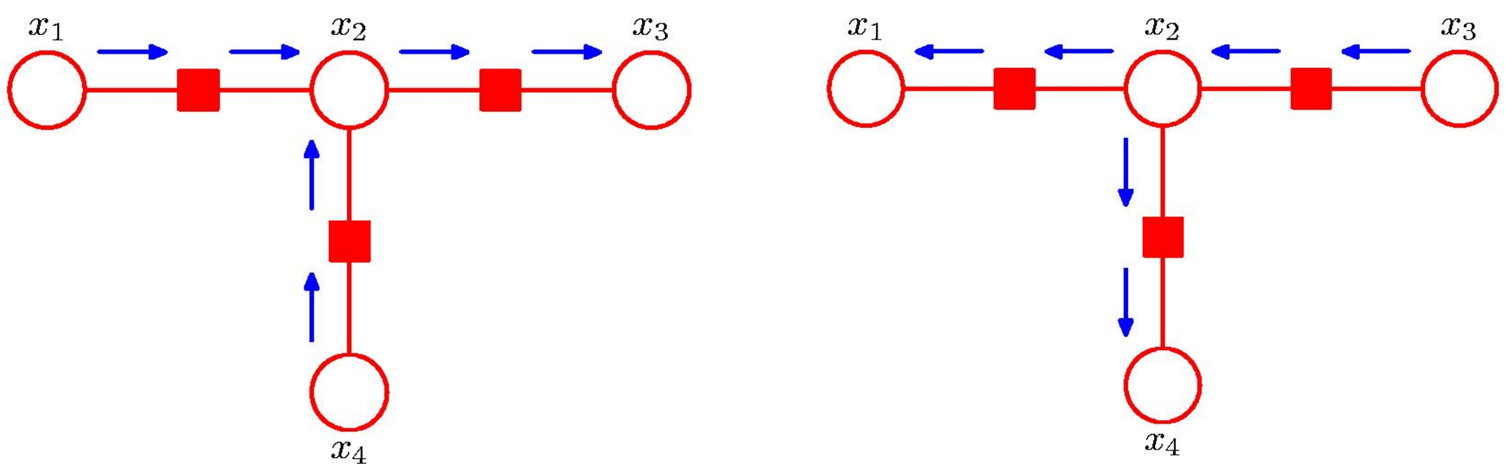


- If the leaf node is a factor node then:



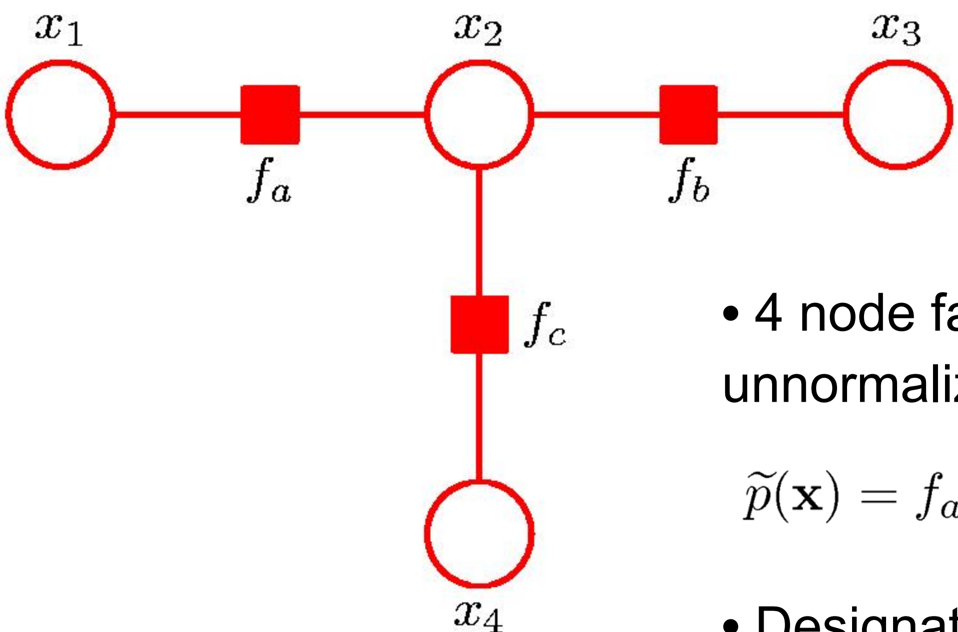
Sum Product Algorithm

- We will often be interested in finding the marginals for every node.
- To compute local marginals:
  - Pick an arbitrary node as root.
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.



Example

- Consider a simple example that will illustrate the sum-product algorithm.



- 4 node factor graph whose unnormalized probability distribution is:

$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

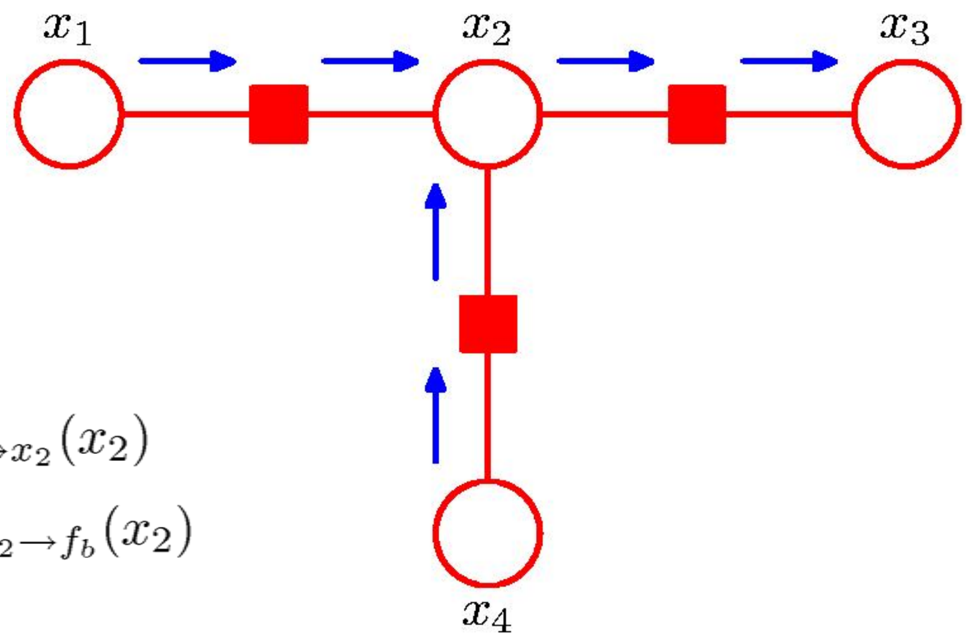
- Designate node x_3 as a root node.

From Leaf Nodes to Root

- Starting at the leaf nodes:

$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

$$\begin{aligned}
\mu_{x_1 \to f_a}(x_1) &= 1 \\
\mu_{f_a \to x_2}(x_2) &= \sum_{x_1} f_a(x_1, x_2) \\
\mu_{x_4 \to f_c}(x_4) &= 1 \\
\mu_{f_c \to x_2}(x_2) &= \sum_{x_4} f_c(x_2, x_4) \\
\mu_{x_2 \to f_b}(x_2) &= \mu_{f_a \to x_2}(x_2) \mu_{f_c \to x_2}(x_2) \\
\mu_{f_b \to x_3}(x_3) &= \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \to f_b}(x_2)
\end{aligned}$$



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

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

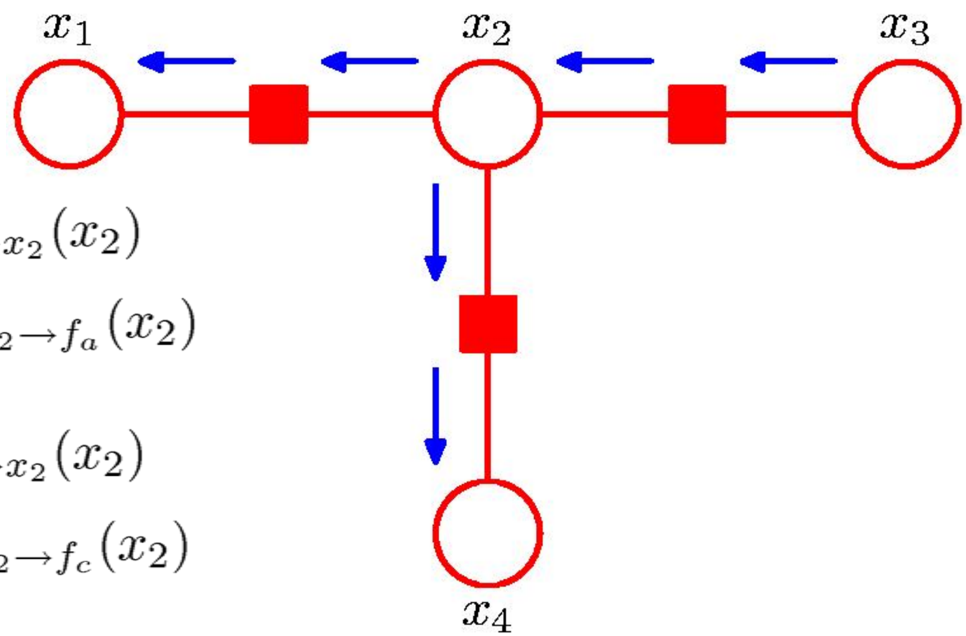
- Once this message propagation is complete, we start from the root node back to the leaf nodes.

From Root Node to Leafs

- From the root node out to the leaf nodes:

$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

$$\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2) \mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2) \mu_{f_b \to x_2}(x_2) \\
\mu_{f_c \to x_4}(x_4) &= \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \to f_c}(x_2)
\end{aligned}$$



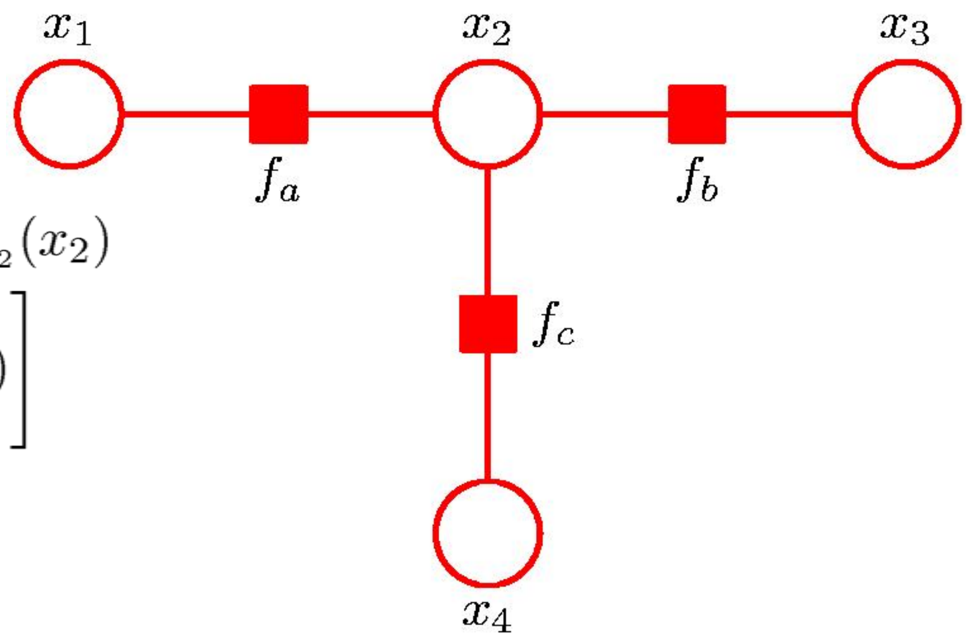
$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

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

Example

- We can check that the marginal is given by the correct expression:

$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$



$$\widetilde{p}(x_2) = \mu_{f_a \to x_2}(x_2) \mu_{f_b \to x_2}(x_2) \mu_{f_c \to x_2}(x_2)$$

$$\begin{aligned} &= \left[\sum_{x_1} f_a(x_1, x_2)\right] \left[\sum_{x_3} f_b(x_2, x_3)\right] \\ &\quad \left[\sum_{x_4} f_c(x_2, x_4)\right] \\ &= \sum_{x_1} \sum_{x_3} \sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\ &= \sum_{x_1} \sum_{x_3} \sum_{x_4} \widetilde{p}(\mathbf{x}) \end{aligned}$$

Performing Inference

- So far we assumed that all variables are unobserved.
- Typically, some subsets of nodes will be observed, and we wish to calculate the posterior distribution conditioned on these observations.
- Let is partition **x** into hidden variables **h** and observed variables **v**.
- Let us denote the values of observed variables by $\hat{\mathbf{v}}$.
- Can multiply the joint distribution by:

$$\tilde{p}(\mathbf{x}) \prod_i I(v_i, \hat{v}_i).$$

- By running the sum-product algorithm, we can efficiently calculate unnormalized posterior margnals:

$$\tilde{p}(h_i | \mathbf{v} = \hat{\mathbf{v}}).$$

- Normalization can be performed using local computation.

Max Sum Algorithm

- The sum-product algorithm allows us to find marginal distributions.
- Sometimes we might be interested in finding a setting of the variables that has the largest probability.
- This can be accomplished by a closely related algorithm called max-sum.
- Objective: an efficient algorithm for finding:

  i. the value x^{max} that maximises $p(x)$;

  ii. the value of $p(x^{max})$.

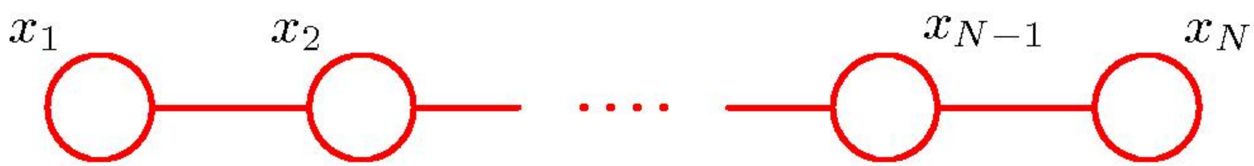
- In general, maximum marginals ≠ joint maximum.

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

$$\arg\max_x p(x, y) = 1 \qquad \arg\max_x p(x) = 0$$

Inference on a Chain

- The key idea is to exchange products with maximizations.



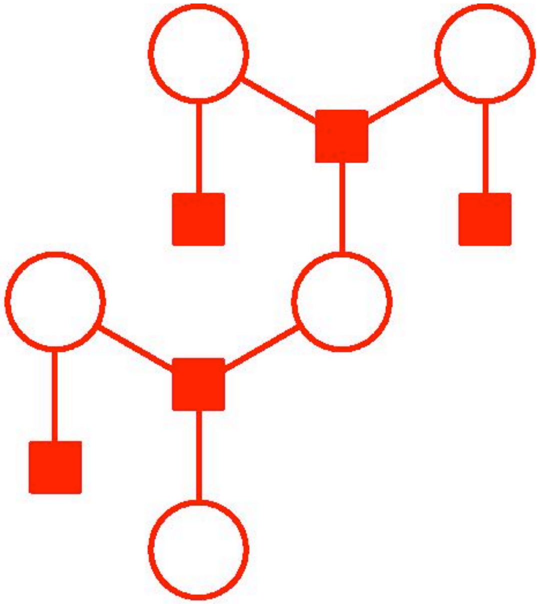
$$
\begin{aligned}
p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x}) &= \max_{x_1} \ldots \max_{x_M} p(\mathbf{x}) \\
&= \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} \left[\psi_{1,2}(x_1, x_2) \cdots \psi_{N-1,N}(x_{N-1}, x_N)\right] \\
&= \frac{1}{Z} \max_{x_1} \left[\max_{x_2} \left[\psi_{1,2}(x_1, x_2) \left[\cdots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots \right]\right]
\end{aligned}
$$

- As with calculation of marginals, we see that exchanging max and product operators result in a much more efficient algorithm.
- It can interpreted in terms of messages passed from x_N to node x_1.

Generalization to Trees

- It generalizes to tree-structured factor graphs.

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_n} \prod_{f_s \in \mathrm{ne}(x_n)} \max_{X_s} f_s(x_n, X_s)$$



- Compare to sum-product algorithms for computing margnals.

$$p(x) \quad = \quad \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right]$$

- Max-Product $\rightarrow$ Max-Sum:

For numerical reasons, it is easier to work with:

$$\ln \left(\max_{\mathbf{x}} p(\mathbf{x}) \right) = \max_{\mathbf{x}} \ln p(\mathbf{x}).$$

Again, we use distributive law:

$$\max(a + b, a + c) = a + \max(b, c).$$

Max Sum Algorithm

- It is now straightforward to derive the max-sum algorithm in terms of message passing: replace sum with max, and products with sum of logarithms.
- Initialization:

$$\mu_{x \to f}(x) = 0 \qquad\qquad \mu_{f \to x}(x) = \ln f(x)$$

- Recursion:

$$\mu_{f \to x}(x) \quad = \quad \max_{x_1, \ldots, x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

Replace sum with max

Replace product with sum of logs.

$$\mu_{x \to f}(x) \quad = \quad \sum_{l \in \mathrm{ne}(x) \setminus f} \mu_{f_l \to x}(x)$$

Exact Inference in General Graphs

• For many practical applications we have to deal with graphs with loops (e.g. see Restricted Boltzmann Machines).

• The message passing framework can be generalized to arbitrary graph topologies, giving an exact inference procedure known as a junction tree algorithm.

• However, computational cost grows exponentially in the treewidth of the graph.

• Treewidth is defined in terms of the number of the variables in the largest clique (minus one).

• So if the treewidth of the original graph is high, the junction tree algorithm becomes impractical (as is usually the case).

Loopy Belief Propagation

• For many practical problems, it will not be feasible to do exact inference, so we have to use approximations.

• One idea is to simply apply sum-product algorithm in graphs with loops.

• Initial unit messages passed across all links, after which messages are passed around until convergence (not guaranteed!).

• This is approach is known as loopy belief propagation. This is possible because the message passing of the sum-product algorithm are purely local.

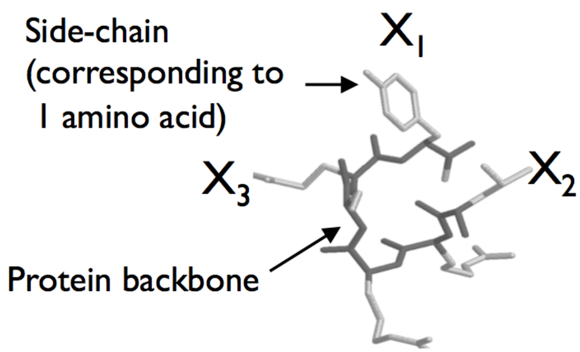
• *Approximate* but *tractable* for large graphs!

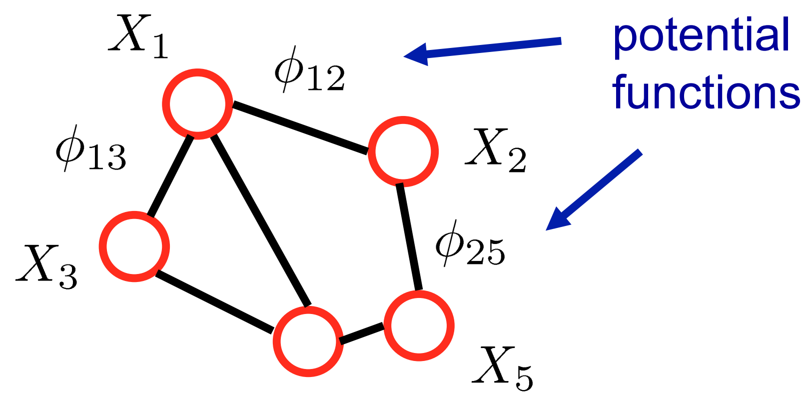
• Sometime works remarkably well, sometimes not at all.

• We will later look at other message passing algorithms such as expectation propagation.

Protein Design

- Given desired 3-D structure, choose amino acids that give the most stable folding:





- One variable per position (up to 180 positions).
- Each state specifies an amino-acid and discretized angles for the side-chain

X_i = 0 means position *i* is "tyrosine" at angle (20°, 10°, 60°)
= 1 … "tyrosine" … (40°, 10°, 80°)
= 2 … "cysteine … (20°, 10°, 60°)

~ 100 states

- Inference problem:

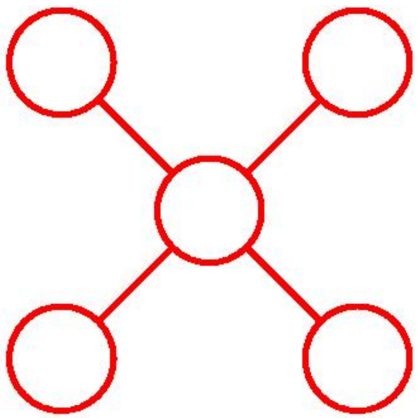
$$\mathbf{x}^{max} = \underset{\mathbf{x}}{\text{argmax}} \prod_C \phi_C(\mathbf{x}_c).$$

Hard search problem: 100^{180} possible configurations

Yanover et.al., Sontag et.al.

Learning in Undirected Graphs

- Consider binary pairwise tree structured undirected graphical model:



$$P_\theta(\mathbf{x}) = \frac{1}{\mathcal{Z}(\theta)} \exp\Big(\sum_{ij\in E} x_i x_j \theta_{ij} + \sum_{i\in V} x_i \theta_i\Big)$$

- Given a set of i.i.d training example vectors: $\mathcal{D} = \{\mathbf{x}^1, ..., \mathbf{x}^N\}$, we want to learning model parameters θ.

- Maximize log-likelihood objective: $L(\theta) = \frac{1}{N}\sum_{n=1}^{N} \log P_\theta(\mathbf{x}^{(n)})$
- Derivative of the log-likilihood:

$$\frac{\partial L(\theta)}{\partial \theta_{ij}} = \frac{1}{N}\sum_n [x_i^{(n)} x_j^{(n)}] - \sum_{\mathbf{x}} [x_i x_j P_\theta(\mathbf{x})] = \mathrm{E}_{P_{data}}[x_i x_j] - \mathrm{E}_{P_\theta}[x_i x_j]$$

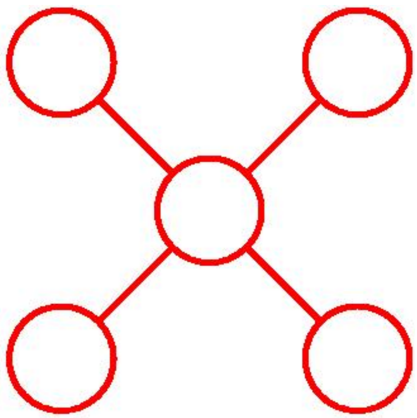
Compute from the data.

Inference: Compute pairwise marginals: Run sum-product algorithm.

Learning in Undirected Graphs

• At the maximum likelihood solution we have:

$$\mathrm{E}_{P_\theta}[x_i x_j] = \frac{1}{N} \sum_n x_i^{(n)} x_j^{(n)}.$$



$$P_\theta(\mathbf{x}) = \frac{1}{\mathcal{Z}(\theta)} \exp\Big(\sum_{ij \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \Big)$$

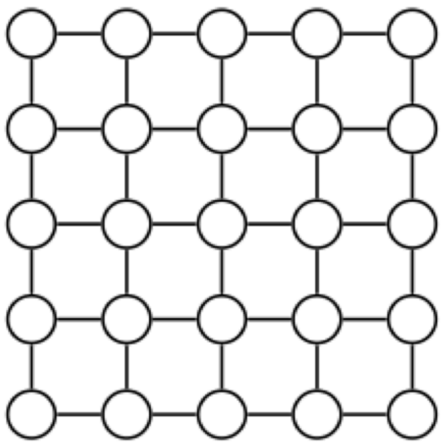
• The maximum likelihood estimates simply match the estimated inner products between the nodes to their observed inner products.

• This is a standard form for the gradient equation for exponential family models.

• Sufficient statistics are set equal to their expectations under the model.

Learning in Undirected Graphs

- Consider parameter learning in graphs with loops, e.g. Ising model:



$$P_\theta(\mathbf{x}) = \frac{1}{\mathcal{Z}(\theta)} \exp\Big(\sum_{ij \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \Big)$$

- Given a set of i.i.d training example vectors: $\mathcal{D} = \{\mathbf{x}^1, ..., \mathbf{x}^N\}$, we want to learning model parameters θ.

- As before, maximize log-likelihood objective: $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\mathbf{x}^{(n)})$
- Derivative of the log-likelihood:

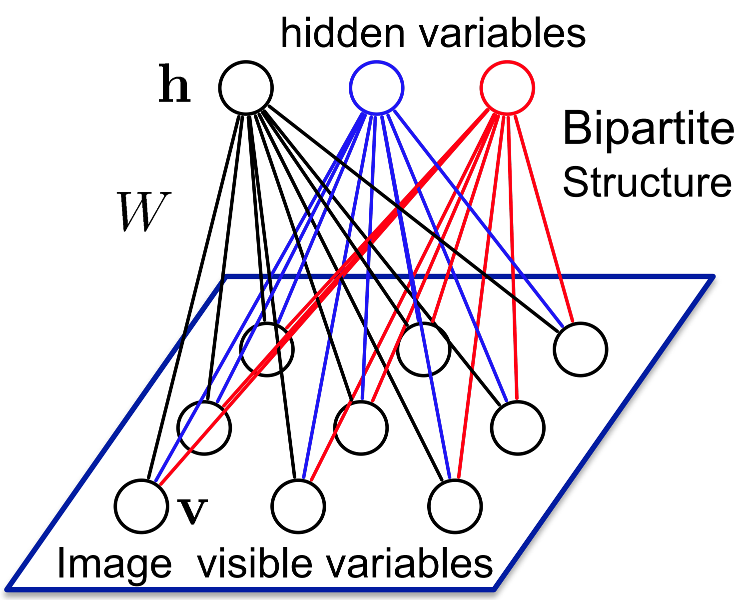
$$\frac{\partial L(\theta)}{\partial \theta_{ij}} = \frac{1}{N} \sum_n [x_i^{(n)} x_j^{(n)}] - \sum_{\mathbf{x}} [x_i x_j P_\theta(\mathbf{x})] = \mathrm{E}_{P_{data}}[x_i x_j] - \mathrm{E}_{P_\theta}[x_i x_j]$$

Compute from the data.

Cannot compute exactly: Approximate inference, e.g. loopy belief propagation

Restricted Boltzmann Machines

- For many real-world problems, we need to introduce hidden variables.
- Our random variables will contain visible and hidden variables x=(v,h).



Stochastic binary visible variables $\mathbf{v} \in \{0,1\}^D$ are connected to stochastic binary hidden variables $\mathbf{h} \in \{0,1\}^F$.

The energy of the joint configuration:

$$E(\mathbf{v},\mathbf{h};\theta) = -\sum_{ij} W_{ij} v_i h_j - \sum_i b_i v_i - \sum_j a_j h_j$$

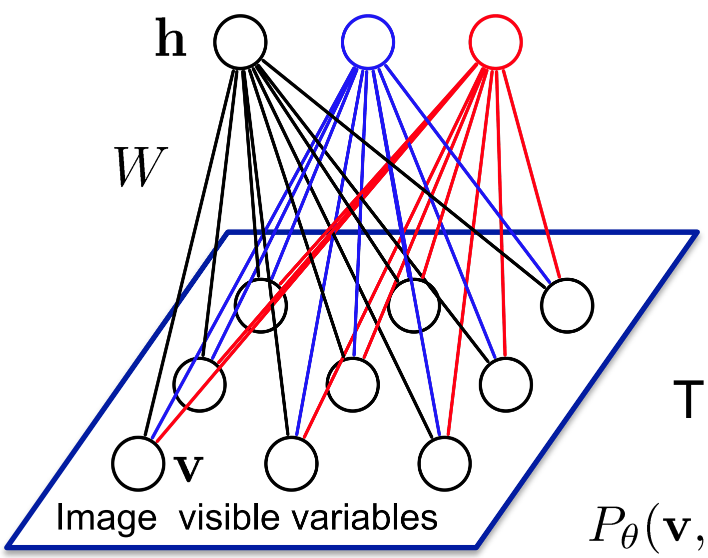
$\theta = \{W, a, b\}$ model parameters.

Probability of the joint configuration is given by the Boltzmann distribution:

$$P_\theta(\mathbf{v},\mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \exp\big(-E(\mathbf{v},\mathbf{h};\theta)\big) = \underbrace{\frac{1}{\mathcal{Z}(\theta)}}_{\text{partition function}} \prod_{ij} \underbrace{e^{W_{ij} v_i h_j}}_{\text{potential functions}} \prod_i e^{b_i v_i} \prod_j e^{a_j h_j}$$

$$\mathcal{Z}(\theta) = \sum_{\mathbf{h},\mathbf{v}} \exp\big(-E(\mathbf{v},\mathbf{h};\theta)\big)$$

Product of Experts



Product of Experts formulation.

The joint distribution is given by:

$$P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \exp\Big(\sum_{ij} W_{ij} v_i h_j + \sum_i b_i v_i + \sum_j a_j h_j\Big)$$

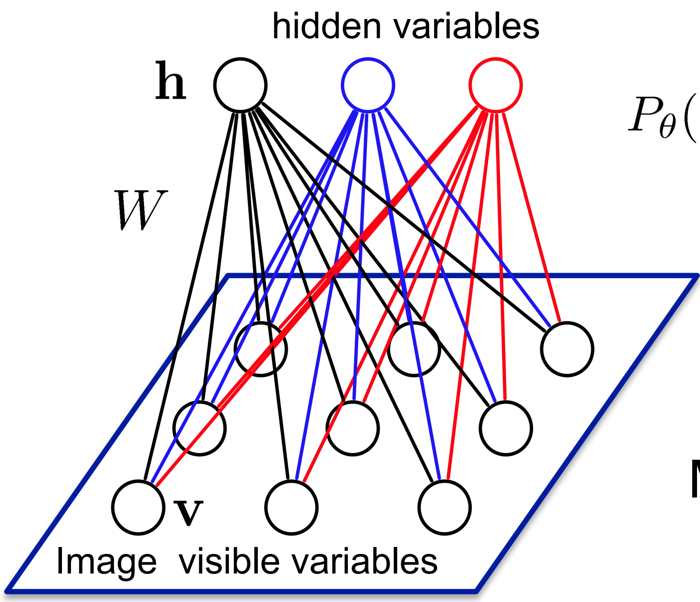
where the undirected edges in the graphical model represent $\{W_{ij}\}$.

Marginalizing over the states of hidden variables:

$$P_\theta(\mathbf{v}) = \sum_{\mathbf{h}} P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \prod_i \exp(b_i v_i) \prod_j \Big(1 + \exp(a_j + \sum_i W_{ij} v_i)\Big)$$

Markov random fields, Boltzmann machines, log-linear models.

Model Learning



$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left[\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}\right]$$

Given a set of i.i.d. training examples $\mathcal{D} = \{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, ..., \mathbf{v}^{(N)}\}$ we want to learn model parameters $\theta = \{W, a, b\}$

Maximize log-likelihood objective:

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\mathbf{v}^{(n)})$$

Derivative of the log-likelihood:

$$\frac{\partial L(\theta)}{\partial W_{ij}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial W_{ij}} \log\left(\sum_{\mathbf{h}} \exp\left[\mathbf{v}^{(n)\top} W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}^{(n)}\right]\right) - \frac{\partial}{\partial W_{ij}} \log \mathcal{Z}(\theta)$$

$$= \mathrm{E}_{P_{data}}[v_i h_j] - \underbrace{\mathrm{E}_{P_\theta}[v_i h_j]}_{\text{Difficult to compute: exponentially many configurations}}$$

Approximate maximum likelihood learning

$$P_{data}(\mathbf{v}, \mathbf{h}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta) P_{data}(\mathbf{v})$$

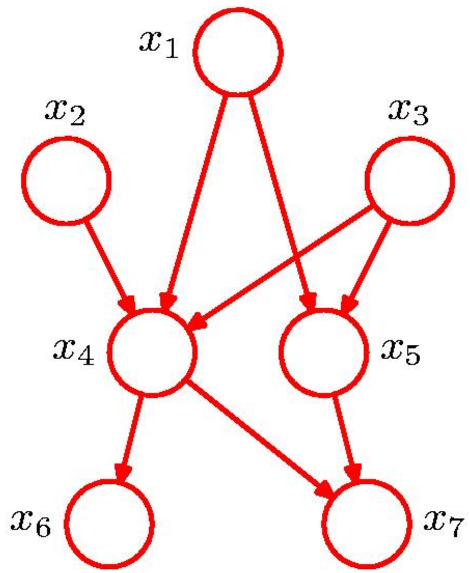
$$P_{data}(\mathbf{v}) = \frac{1}{N} \sum_n \delta(\mathbf{v} - \mathbf{v}^{(n)})$$

Difficult to compute: exponentially many configurations

Later, we will see learning in directed graphs with hidden variables.

Learning the Graph Structure

- So far we assumed that the graph structure is given and fixed.
- We might be interested in learning the graph structure itself from data.



- From a Bayesian viewpoint, we would ideally like to compute the posterior distribution over graph structures:

$$p(m|\mathcal{D}) \propto p(\mathcal{D}|m)p(m),$$

where D is the observed data and p(m) represent a prior over graphs, indexed by m.

- However, the model evidence p(D|m) requires marginalization over latent variables and present computational problems.

- For undirected graphs, the problem is much worse, as computing the likelihood function requires computing the normalizing constant.