Machine Learning - Lecture 17

Exact Inference & Belief Propagation

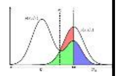
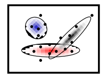
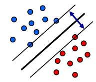
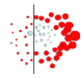
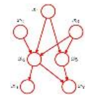
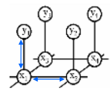
11.07.2016

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

Many slides adapted from C. Bishop, Z. Gharahmani

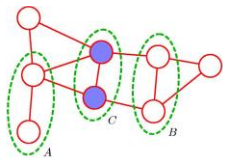
Course Outline

- Fundamentals (2 weeks)
  - Bayes Decision Theory
  - Probability Density Estimation
- Discriminative Approaches (5 weeks)
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees
- Generative Models (4 weeks)
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference

Recap: Undirected Graphical Models

- Undirected graphical models ("Markov Random Fields")
  - Given by undirected graph



- Conditional independence for undirected graphs
  - If every path from any node in set A to set B passes through at least one node in set C, then $A \perp\!\!\!\perp B | C$.
  - Simple Markov blanket:

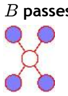


Image source: C. Bishop, 2006

Recap: Factorization in MRFs

- Joint distribution
  - Written as product of potential functions over maximal cliques in the graph:

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

  - The normalization constant Z is called the partition function.

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C)$$

- Remarks
  - BNs are automatically normalized. But for MRFs, we have to explicitly perform the normalization.
  - Presence of normalization constant is major limitation!
    - Evaluation of Z involves summing over $\mathcal{O}(K^M)$ terms for M nodes!

Recap: Factorization in MRFs

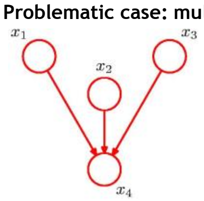
- Role of the potential functions
  - General interpretation
    - No restriction to potential functions that have a specific probabilistic interpretation as marginals or conditional distributions.
  - Convenient to express them as exponential functions ("Boltzmann distribution")

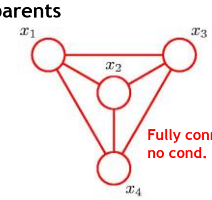
$$\psi_C(\mathbf{x}_C) = \exp\{-E(\mathbf{x}_C)\}$$

    - with an energy function E.
  - Why is this convenient?
    - Joint distribution is the product of potentials ⇒ sum of energies.
    - We can take the log and simply work with the sums…

Recap: Converting Directed to Undirected Graphs

- Problematic case: multiple parents





Fully connected, no cond. indep.!

$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)$$

Need a clique of $x_1,\ldots,x_4$ to represent this factor!

  - Need to introduce additional links ("marry the parents").

⇒ This process is called moralization. It results in the moral graph.

Slide adapted from Chris Bishop

Image source: C. Bishop, 2006

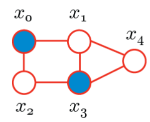
Recap: Conversion Algorithm

- General procedure to convert directed → undirected
  1. Add undirected links to marry the parents of each node.
  2. Drop the arrows on the original links ⇒ moral graph.
  3. Find maximal cliques for each node and initialize all clique potentials to 1.
  4. Take each conditional distribution factor of the original directed graph and multiply it into one clique potential.

- Restriction
  - Conditional independence properties are often lost!
  - Moralization results in additional connections and larger cliques.

Computing Marginals

- How do we apply graphical models?
  - Given some observed variables, we want to compute distributions of the unobserved variables.
  - In particular, we want to compute marginal distributions, for example $p(x_4)$.



- How can we compute marginals?
  - Classical technique: sum-product algorithm by Judea Pearl.
  - In the context of (loopy) undirected models, this is also called (loopy) belief propagation [Weiss, 1997].
  - Basic idea: message-passing.

Inference on a Chain

- Chain graph

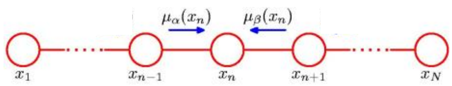
  - Joint probability

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

  - Marginalization

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

Inference on a Chain

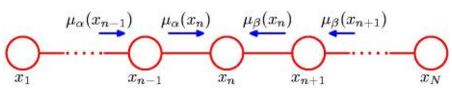


- Idea: Split the computation into two parts ("messages").

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

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

Inference on a Chain

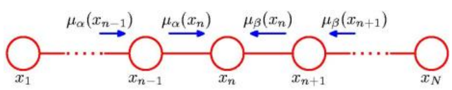


- We can define the messages recursively...

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

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

Inference on a Chain



- Until we reach the leaf nodes...

$$\mu_\alpha(x_2) = \sum_{x_1}\psi_{1,2}(x_1,x_2) \qquad \mu_\beta(x_{N-1}) = \sum_{x_N}\psi_{N-1,N}(x_{N-1},x_N)$$

- Interpretation
  - We pass messages from the two ends towards the query node x_n.
- We still need the normalization constant Z.
  - This can be easily obtained from the marginals:

$$Z = \sum_{x_n}\mu_\alpha(x_n)\mu_\beta(x_n)$$

Summary: Inference on a Chain

- To compute local marginals:
  - Compute and store all forward messages $\mu_\alpha(x_n)$.
  - Compute and store all backward messages $\mu_\beta(x_n)$.
  - Compute Z at any node x_m.
  - Compute

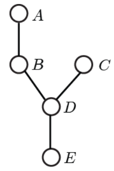
$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

    for all variables required.

- Inference through message passing
  - We have thus seen a first message passing algorithm.
  - How can we generalize this?

Inference on Trees

- Let's next assume a tree graph.
  - Example:



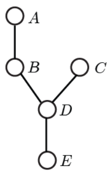
  - We are given the following joint distribution:

$$p(A,B,C,D,E) = \frac{1}{Z} f_1(A,B)\cdot f_2(B,D)\cdot f_3(C,D)\cdot f_4(D,E)$$

  - Assume we want to know the marginal $p(E)$...

Inference on Trees

- Strategy
  - Marginalize out all other variables by summing over them.



  - Then rearrange terms:

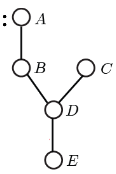
$$p(E) = \sum_A\sum_B\sum_C\sum_D p(A,B,C,D,E)$$

$$= \sum_A\sum_B\sum_C\sum_D \frac{1}{Z} f_1(A,B)\cdot f_2(B,D)\cdot f_3(C,D)\cdot f_4(D,E)$$

$$= \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot\left(\sum_B f_2(B,D)\cdot\left(\sum_A f_1(A,B)\right)\right)\right)$$

Marginalization with Messages

- Use messages to express the marginalization:



$$m_{A\to B} = \sum_A f_1(A,B) \qquad m_{C\to D} = \sum_C f_3(C,D)$$

$$m_{B\to D} = \sum_B f_2(B,D) m_{A\to B}(B)$$

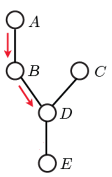
$$m_{D\to E} = \sum_D f_4(D,E) m_{B\to D}(D) m_{C\to D}(D)$$

$$p(E) = \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot\left(\sum_B f_2(B,D)\cdot\left(\sum_A f_1(A,B)\right)\right)\right)$$

$$= \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot\left(\sum_B f_2(B,D)\cdot m_{A\to B}(B)\right)\right)$$

Marginalization with Messages

- Use messages to express the marginalization:



$$m_{A\to B} = \sum_A f_1(A,B) \qquad m_{C\to D} = \sum_C f_3(C,D)$$

$$m_{B\to D} = \sum_B f_2(B,D) m_{A\to B}(B)$$

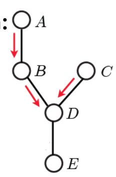
$$m_{D\to E} = \sum_D f_4(D,E) m_{B\to D}(D) m_{C\to D}(D)$$

$$p(E) = \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot\left(\sum_B f_2(B,D)\cdot\left(\sum_A f_1(A,B)\right)\right)\right)$$

$$= \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot m_{B\to D}(D)\right)$$

Marginalization with Messages

- Use messages to express the marginalization:



$$m_{A\to B} = \sum_A f_1(A,B) \qquad m_{C\to D} = \sum_C f_3(C,D)$$

$$m_{B\to D} = \sum_B f_2(B,D) m_{A\to B}(B)$$

$$m_{D\to E} = \sum_D f_4(D,E) m_{B\to D}(D) m_{C\to D}(D)$$

$$p(E) = \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot\left(\sum_B f_2(B,D)\cdot\left(\sum_A f_1(A,B)\right)\right)\right)$$

$$= \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot m_{C\to D}(D)\cdot m_{B\to D}(D)\right)$$

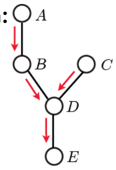
Marginalization with Messages

- Use **messages** to express the marginalization:

$$m_{A\to B} = \sum_A f_1(A,B) \qquad m_{C\to D} = \sum_C f_3(C,D)$$

$$m_{B\to D} = \sum_B f_2(B,D) m_{A\to B}(B)$$

$$m_{D\to E} = \sum_D f_4(D,E) m_{B\to D}(D) m_{C\to D}(D)$$

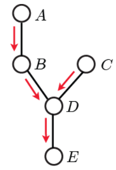


$$p(E) = \frac{1}{Z}\left(\sum_D f_4(D,E)\cdot\left(\sum_C f_3(C,D)\right)\cdot\left(\sum_B f_2(B,D)\cdot\left(\sum_A f_1(A,B)\right)\right)\right)$$

$$= \frac{1}{Z} m_{D\to E}(E)$$

Recap: Message Passing on Trees

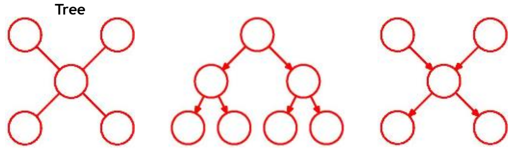
- **General procedure** for all tree graphs.
  - Root the tree at the variable that we want to compute the marginal of.
  - Start computing messages at the leaves.
  - Compute the messages for all nodes for which all incoming messages have already been computed.
  - Repeat until we reach the root.



- If we want to compute the marginals for all possible nodes (roots), we can reuse some of the messages.
  - Computational expense linear in the number of nodes.
- We already motivated message passing for inference.
  - How can we formalize this into a general algorithm?

How Can We Generalize This?

Undirected Tree | Directed Tree | Polytree



- Message passing algorithm motivated for trees.
  - Now: generalize this to directed polytrees.
  - We do this by introducing a common representation

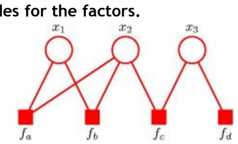
⇒ **Factor graphs**

Topics of This Lecture

- **Factor graphs**
  - Construction
  - Properties
- Sum-Product Algorithm for computing marginals
  - Key ideas
  - Derivation
  - Example
- Max-Sum Algorithm for finding most probable value
  - Key ideas
  - Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

Factor Graphs

- Motivation
  - Joint probabilities on both directed and undirected graphs can be expressed as a product of **factors over subsets of variables**.
  - **Factor graphs** make this decomposition explicit by introducing separate nodes for the factors.

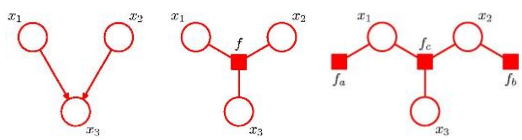


  - Joint probability

$$p(\mathbf{x}) = \frac{1}{Z} f_a(x_1,x_2) f_b(x_1,x_2) f_c(x_2,x_3) f_d(x_3)$$

$$= \frac{1}{Z}\prod_s f_s(\mathbf{x}_s)$$

Factor Graphs from Directed Graphs



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

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

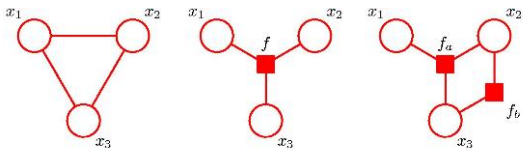
$$f_a(x_1) = p(x_1)$$
$$f_b(x_2) = p(x_2)$$
$$f_c(x_1,x_2,x_3) = p(x_3|x_1,x_2)$$

- Conversion procedure
  1. Take variable nodes from directed graph.
  2. Create factor nodes corresponding to conditional distributions.
  3. Add the appropriate links.

⇒ Different factor graphs possible for same directed graph.

Factor Graphs from Undirected Graphs

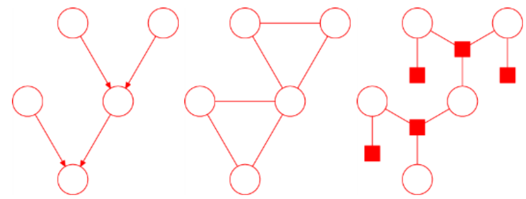
- Some factor graphs for the same undirected graph:



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

⇒ The factor graph keeps the factors explicit and can thus convey more detailed information about the underlying factorization!

Factor Graphs - Why Are They Needed?



- Converting a directed or undirected tree to factor graph
  - The result will again be a tree.
- Converting a directed polytree
  - Conversion to undirected tree creates loops due to moralization!
  - Conversion to a factor graph again results in a tree.

Topics of This Lecture

- Factor graphs
  - Construction
  - Properties
- Sum-Product Algorithm for computing marginals
  - Key ideas
  - Derivation
  - Example
- Max-Sum Algorithm for finding most probable value
  - Key ideas
  - Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

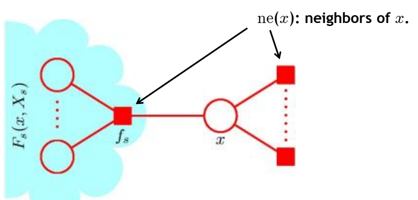
Sum-Product Algorithm

- Objectives
  - Efficient, exact inference algorithm for finding marginals.
  - In situations where several marginals are required, allow computations to be shared efficiently.
- General form of message-passing idea
  - Applicable to tree-structured factor graphs.
  - ⇒ Original graph can be undirected tree or directed tree/polytree.
- Key idea: Distributive Law

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

  - ⇒ Exchange summations and products exploiting the tree structure of the factor graph.
  - Let's assume first that all nodes are hidden (no observations).

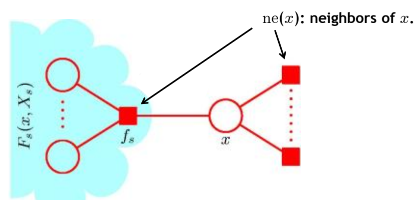
Sum-Product Algorithm



- Goal:
  - Compute marginal for x: $p(x) = \sum_{\mathbf{x} \backslash x} p(\mathbf{x})$
  - Tree structure of graph allows us to partition the joint distrib. into groups associated with each neighboring factor node:

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

Sum-Product Algorithm



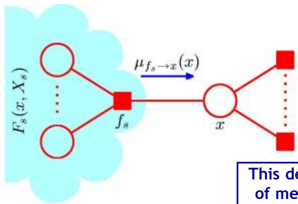
- Marginal:

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

  - Exchanging products and sums:

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

Sum-Product Algorithm



- **Marginal:**

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

This defines a first type of message $\mu_{f_s \to x}(x)$:

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

  - Exchanging products and sums:

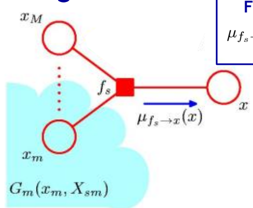
$$p(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right] = \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x)$$

Image source: C. Bishop, 2006

Sum-Product Algorithm

First message type:

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



- **Evaluating the messages:**
  - Each factor $F_s(x, X_s)$ is again described by a factor (sub-)graph.

  ⇒ Can itself be factorized:

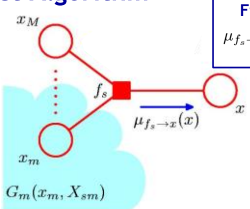
$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \ldots G_M(x_M, X_{sM})$$

Image source: C. Bishop, 2006

Sum-Product Algorithm

First message type:

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



- **Evaluating the messages:**
  - Thus, we can write

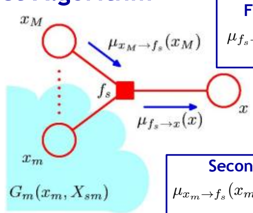
$$\mu_{f_s \to x}(x) = \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right]$$
$$= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

Image source: C. Bishop, 2006

Sum-Product Algorithm

First message type:

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



Second message type:

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

- **Evaluating the messages:**
  - Thus, we can write

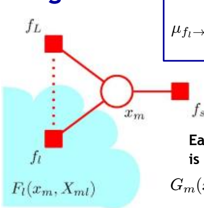
$$\mu_{f_s \to x}(x) = \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right]$$
$$= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

Image source: C. Bishop, 2006

Sum-Product Algorithm

Recursive definition:

$$\mu_{f_l \to x_m}(x_m) \equiv \sum_{X_{sm}} F_l(x_m, X_{sm})$$



Each term $G_m(x_m, X_{sm})$ is again given by a product

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

- **Recursive message evaluation:**
  - Exchanging sum and product, we again get

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

Image source: C. Bishop, 2006

Sum-Product Algorithm - Summary

- **Two kinds of messages**
  - Message from factor node to variable nodes:
    - Sum of factor contributions

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$
$$= \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

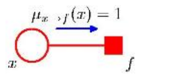
  - Message from variable node to factor node:
    - Product of incoming messages

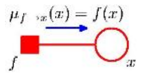
$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

⇒ Simple propagation scheme.

Sum-Product Algorithm

- **Initialization**
  - Start the recursion by sending out messages from the leaf nodes





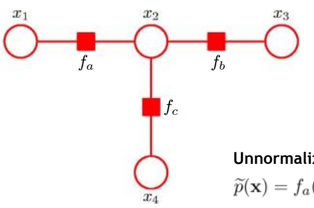
- **Propagation procedure**
  - A node can send out a message once it has received incoming messages from all other neighboring nodes.
  - Once a variable node has received all messages from its neighboring factor nodes, we can compute its marginal by multiplying all messages and renormalizing:

$$p(x) \propto \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x)$$

Sum-Product Algorithm - Summary

- **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.
- **Computational effort**
  - Total number of messages = 2 · number of links in the graph.
  - Maximal parallel runtime = 2 · tree height.

Sum-Product: Example



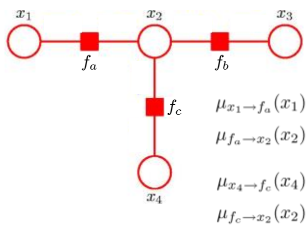
Picking x_3 as root…
⇒ x_1 and x_4 are leaves.

Unnormalized joint distribution:

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

- We want to compute the values of all marginals…

Sum-Product: Example



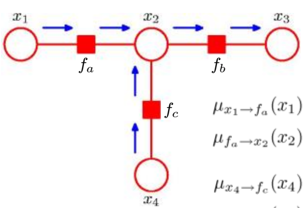
Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

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

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

Sum-Product: Example



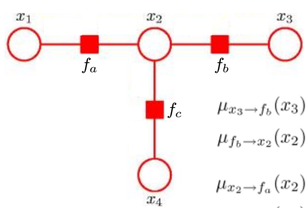
Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

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

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

Sum-Product: Example



Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

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

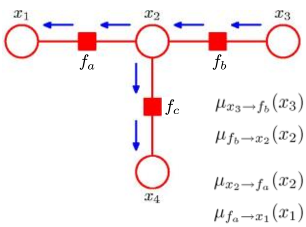
$$\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2) \mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2)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

Sum-Product: Example

Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

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



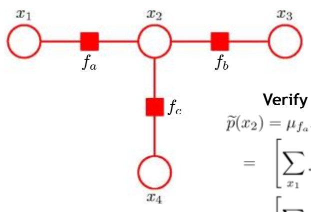
$$\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b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

Sum-Product: Example

Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

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



Verify that marginal is correct:

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

Sum-Product Algorithm - Extensions

- **Dealing with observed nodes**
  - Until now we had assumed that all nodes were hidden…
  - Observed nodes can easily be incorporated:
    - Partition $\mathbf{x}$ into hidden variables $\mathbf{h}$ and observed variables $\mathbf{v} = \hat{\mathbf{v}}$.
    - Simply multiply the joint distribution $p(\mathbf{x})$ by

$$\prod_i I(v_i, \hat{v}_i) \quad \text{where} \quad I(v_i, \hat{v}_i) = \begin{cases} 1, & \text{if } v_i = \hat{v}_i \\ 0, & \text{else.} \end{cases}$$

⇒ Any summation over variables in $\mathbf{v}$ collapses into a single term.

- **Further generalizations**
  - So far, assumption that we are dealing with discrete variables.
  - But the sum-product algorithm can also be generalized to simple continuous variable distributions, e.g. linear-Gaussian variables.

Topics of This Lecture

- Factor graphs
  - Construction
  - Properties
- Sum-Product Algorithm for computing marginals
  - Key ideas
  - Derivation
  - Example
- **Max-Sum Algorithm for finding most probable value**
  - Key ideas
  - Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

Max-Sum Algorithm

- **Objective: an efficient algorithm for finding**
  - Value $\mathbf{x}^{\max}$ that maximises $p(\mathbf{x})$;
  - Value of $p(\mathbf{x}^{\max})$.

  ⇒ Application of dynamic programming in graphical models.

- **In general, maximum marginals ≠ joint maximum.**
  - Example:

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

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

Max-Sum Algorithm - Key Ideas

- **Key idea 1: Distributive Law (again)**

$$\begin{aligned}
\max(ab, ac) &= a\max(b, c) \\
\max(a + b, a + c) &= a + \max(b, c)
\end{aligned}$$

⇒ Exchange products/summations and max operations exploiting the tree structure of the factor graph.

- **Key idea 2: Max-Product → Max-Sum**
  - We are interested in the maximum value of the joint distribution

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

⇒ Maximize the product $p(\mathbf{x})$.

  - For numerical reasons, use the logarithm.

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

⇒ Maximize the sum (of log-probabilities).

Max-Sum Algorithm

- Maximizing over a chain (max-product)

- Exchange max and product operators

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

- Generalizes to tree-structured factor graph

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

Max-Sum Algorithm

- Initialization (leaf nodes)

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

- Recursion
  - Messages

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

  - For each node, keep a record of which values of the variables gave rise to the maximum state:

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

Max-Sum Algorithm

- Termination (root node)
  - Score of maximal configuration

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

  - Value of root node variable giving rise to that maximum

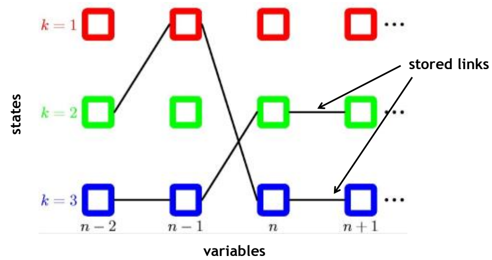
$$x^{\max} = \arg\max_{x} \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]$$

  - Back-track to get the remaining variable values

$$x_{n-1}^{\max} = \phi(x_n^{\max})$$

Visualization of the Back-Tracking Procedure

- Example: Markov chain



⇒ Same idea as in Viterbi algorithm for HMMs…

References and Further Reading

- A thorough introduction to Graphical Models in general and Bayesian Networks in particular can be found in Chapter 8 of Bishop's book.

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006