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 $A \perp B | C$.
  - Simple Markov blanket:

Recap: Factorization in MRFs

- Joint distribution
  - Written as product of potential functions over maximal cliques in the graph:
    \[ p(x) = \frac{1}{Z} \prod_C \psi_C(x_C) \]
  - The normalization constant $Z$ is called the partition function.

- 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 $O(K^M)$ 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")
    \[ \psi_C(x_C) = \exp(-E(x_C)) \]
    - with an energy function $E$.
  - Why is this convenient?
    - Joint distribution is the product of potentials $\Rightarrow$ sum of energies.
    - We can take the log and simply work with the sums..

Recap: Converting Directed to Undirected Graphs

- Problematic case: multiple parents
  - Need a clique of $x_1, \ldots, x_4$ to represent this factor!
  - Need to introduce additional links ("marry the parents").
    $\Rightarrow$ This process is called moralization. It results in the moral graph.

\[ p(x) = p(x_1)p(x_2)p(x_3)p(x_1|x_2, x_3) \]

Fully connected, no cond. indep!
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 \( p(x_i) \).

- 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
    \[ p(x) = \frac{1}{Z} \psi(2^x \cdot \cdot \cdot \cdot \psi N^{-1}N(x_{N-1} x_N) \]
  - Marginalization
    \[ p(x_i) = \sum_{x_{i-1}} \sum_{x_{i+1}} \sum_{x_N} p(x) \]

- Until we reach the leaf nodes...
  - We can define the messages recursively...
    \[ \mu_a(x_a) = \sum_{x_{a-1}} \psi_{a-1,a}(x_{a-1}, x_a) \left( \sum_{x_{a+1}} \psi_{a,a+1}(x_{a+1}) \right) \]
    \[ \mu_b(x_b) = \sum_{x_{b+1}} \psi_{b+1,b}(x_{b+1}) \left( \sum_{x_{b-1}} \psi_{b,b-1}(x_{b-1}) \right) \]

- 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:
    \[ Z = \sum_{x_n} \mu_a(x_a) \mu_b(x_b) \]
To compute local marginals:

- Compute and store all forward messages $\mu_1(x_n)$.
- Compute and store all backward messages $\mu_2(x_n)$.
- Compute $Z$ at any node $x_n$.
- Compute
  \[
  p(x_n) = \frac{1}{Z} \mu_1(x_n) \mu_2(x_n)
  \]
  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

- Strategy
  - Marginalize out all other variables by summing over them.
  - Then rearrange terms:
    \[
    p(E) = \sum_A \sum_B \sum_C \sum_D \sum_E p(A, B, C, D, E)
    = \sum_A \sum_B \sum_C \sum_D \sum_E \frac{1}{Z} f_1(A, B) \cdot f_2(B, D) \cdot f_3(C, D) \cdot f_4(D, E)
    = \frac{1}{Z} \left( \sum_D f_1(D, E) \left( \sum_C f_3(C, D) \right) \left( \sum_B f_2(B, D) \left( \sum_A f_1(A, B) \right) \right) \right)
    \]

Marginalization with Messages

- Use messages to express the marginalization:
  \[
  m_{A \rightarrow B} = \sum_A f_1(A, B) \quad m_{C \rightarrow D} = \sum_C f_3(C, D)
  
  m_{B \rightarrow D} = \sum_B f_2(B, D) m_{A \rightarrow B}(B) 
  
  m_{D \rightarrow E} = \sum_D f_4(D, E) m_{B \rightarrow D}(D) m_{C \rightarrow D}(D)
  \]
  \[
  p(E) = \frac{1}{Z} \sum_D f_1(D, E) \left( \sum_C f_3(C, D) \right) \left( \sum_B f_2(B, D) \left( \sum_A f_1(A, B) \right) \right)
  = \frac{1}{Z} \sum_D f_1(D, E) \left( \sum_C f_3(C, D) \right) \left( \sum_B f_2(B, D) \cdot m_{A \rightarrow B}(B) \right)
  \]

Marginalization with Messages

- Use messages to express the marginalization:
  \[
  m_{A \rightarrow B} = \sum_A f_1(A, B) \quad m_{C \rightarrow D} = \sum_C f_3(C, D)
  
  m_{B \rightarrow D} = \sum_B f_2(B, D) m_{A \rightarrow B}(B) 
  
  m_{D \rightarrow E} = \sum_D f_4(D, E) m_{B \rightarrow D}(D) m_{C \rightarrow D}(D)
  \]
  \[
  p(E) = \frac{1}{Z} \sum_D f_1(D, E) \left( \sum_C f_3(C, D) \right) \left( \sum_B f_2(B, D) \left( \sum_A f_1(A, B) \right) \right)
  = \frac{1}{Z} \sum_D f_1(D, E) \left( \sum_C f_3(C, D) \right) \left( \sum_B f_2(B, D) \cdot m_{A \rightarrow B}(B) \right)
  \]
Marginalization with Messages

- Use messages to express the marginalization:

\[
m_{A \rightarrow B} = \sum_f f(A, B)
m_{C \rightarrow D} = \sum_f f(C, D)
m_{B \rightarrow D} = \sum_f f(B, D)m_{A \rightarrow B}(B)
m_{D \rightarrow E} = \sum_f f(D, E)m_{B \rightarrow D}(D)m_{C \rightarrow D}(D)
\]

\[
p(E) = \frac{1}{Z} \prod_{A} f_1(A; B)\prod_{B} f_2(B; D)\prod_{C} f_3(C; D)\prod_{D} f_4(D; E)
\]

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?

- 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.

\[
p(x) = p(x_1)p(x_2) = p(x_1|x_2)p(x_2) = \prod_x f_x(x)
\]

Factor Graphs from Directed Graphs

- **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:

\[ \psi(x_1, x_2, x_3) \]

\[ f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3) \]

\[ f_j(x_1, x_2, x_3) f_k(x_2, x_3) \]

⇒ 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).
Evaluating the messages:

- Each factor $F_i(x, X_i)$ is again described by a factor (sub-)graph. Can itself be factorized:
  \[
  F_i(x, X_i) = f_i(x, x_1, \ldots, x_M)G_1(x_1, X_{s1}) \cdots G_M(x_M, X_{sM})
  \]

- Recursive definition:
  \[
  \mu_{f_i \rightarrow x_i}(x_i) = \sum_{X_i} F_i(x, X_i)
  \]

  Each term $G_i(x_{s_i}, X_{s_i})$ is again given by a product
  \[
  G_i(x_{s_i}, X_{s_i}) = \prod_{l \in \text{in}(f_i) \setminus i} F_l(x_{s_l}, X_{s_l})
  \]

  
  Recursive message evaluation:
- Exchanging sum and product, we again get
  \[
  \mu_{x_i \leftarrow f_i}(x_i) = \sum_{X_i} G_i(x_i, X_i) = \sum_{X_i \in \text{in}(f_i)} \prod_{l \in \text{in}(f_i) \setminus i} F_l(x_{s_l}, X_{s_l})
  \]

  \[
  = \prod_{l \in \text{in}(f_i)} \mu_{x_i \leftarrow f_l}(x_i)
  \]

- Two kinds of messages:
  - Message from factor node to variable node:
    - Sum of factor contributions
      \[
      \mu_{f_i \rightarrow x_i}(x) = \sum_{X_i} f_i(x, X_i)
      \]
      \[
      = \sum_{X_i} f_i(x) \prod_{l \in \text{in}(f_i) \setminus i} \mu_{x_l \rightarrow f_i}(x_i)
      \]
    - Product of incoming messages
      \[
      \mu_{x_i \rightarrow f_i}(x_i) = \prod_{l \in \text{in}(f_i)} \mu_{f_l \rightarrow x_i}(x_i)
      \]
      \[
      \Rightarrow \text{Simple propagation scheme.}
      \]
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:
    \[ p(x) \propto \prod_{e \in \text{tree}(x)} \mu_f \rightarrow_x(e) \]
  - We want to compute the values of all marginals...

- **Total number of messages**
  - \( f \) number of links in the graph.
  - Maximal parallel runtime \( = 2 \cdot \text{number of links in the graph} \)

- **Computational effort**
  - \( = 2 \cdot \text{tree height} \)

Slide adapted from Chris Bishop

---

Sum-Product: Example

Picking \( x_i \) as root...
\( \Rightarrow x_i \) and \( x_i \) are leaves.

Unnormalized joint distribution:
\[ p(x) = f_0(x_1, x_2) f_1(x_2, x_3) f_2(x_2, x_4) \]

Slide adapted from Chris Bishop

---

Message definitions:

- \( \mu_{f \rightarrow_x(e)}(x) = \sum_{\hat{x}} f_\hat{x}(\hat{x}_1, x_2) \prod_{e \in \text{tree}(x)} \mu_e \rightarrow_{\hat{x}}(e) \)
- \( \mu_{e \rightarrow_x(e)}(x) \)
- \( \mu_f \rightarrow_x(x) \)
- \( \mu_{f \rightarrow_y(x)}(x) \)

\( e \) from the root to the leaf
\( x \) from the leaf nodes to the root

\( x \) to \( y \)
\( y \) to \( x \)

- \( \text{Message definitions:} \)

Slide adapted from Chris Bishop
So far, we have assumed that we are dealing with discrete variables. We are interested in the maximum value of the joint distribution.

Key ideas:
- Sum
- Loopy Belief Propagation
- Partition

For numerical reasons, use the logarithm.

Key idea 2:
- Value of

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

Further generalizations:
- So far, we assumed 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.

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 x into hidden variables h and observed variables $v = \hat{v}$.
    - Simply multiply the joint distribution $p(x)$ by
      \[
      \prod_i I(v_i, \hat{v}_i) \text{ where } I(v_i, \hat{v}_i) = \begin{cases} 
        1, & \text{if } v_i = \hat{v}_i \\
        0, & \text{else}.
      \end{cases}
      \]
      \[\Rightarrow\] Any summation over variables in $v$ collapses into a single term.

- Max-Sum Algorithm
  - Objective: an efficient algorithm for finding
    - Value $x^{\text{max}}$ that maximises $p(x)$;
    - Value of $p(x^{\text{max}})$.
    \[\Rightarrow\] Application of dynamic programming in graphical models.

In general, maximum marginals $\neq$ joint maximum.

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

\[\arg \max p(x, y) = 1 \quad \arg \max p(x) = 0\]

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 - Key Ideas
- Key idea 1: Distributive Law (again)
  \[\max(ab, ac) = a \max(b, c)\]
  \[\max(a + b, a + c) = a + \max(b, c)\]
  \[\Rightarrow\] Exchange products/summations and max operations exploiting the tree structure of the factor graph.

- Key idea 2: Max-Product $\rightarrow$ Max-Sum
  - We are interested in the maximum value of the joint distribution $p(x^{\text{max}}) = \max x p(x)$
  \[\Rightarrow\] Maximize the product $p(x)$.
  - For numerical reasons, use the logarithm.
    \[\ln \left( \max p(x) \right) = \max \ln p(x)\]
  \[\Rightarrow\] Maximize the sum (of log-probabilities).
Max-Sum Algorithm

- Maximizing over a chain (max-product)
  \[
  \text{Maximize over } x_1, x_2, \ldots, x_{N-1}, x_N
  \]

- Exchange max and product operators
  \[
  p(x^{\text{max}}) = \max_x p(x) = \max_{x_1} \cdots \max_{x_N} p(x)
  \]
  \[
  = \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} \left[ \prod_{i=1}^{N-1} \psi_i(x_i, x_{i+1}) \right]
  \]
- Generalizes to tree-structured factor graph
  \[
  \max_{x} p(x) = \max_{x \in \text{factor}} \prod_{x_i} \max_{x_i} f_i(x_i, x_{\text{parents}(x)})
  \]

Visualization of the Back-Tracking Procedure

- Example: Markov chain
  \[
  \Rightarrow \text{Same idea as in Viterbi algorithm for HMMs...}
  \]

Max-Sum Algorithm

- Initialization (leaf nodes)
  \[
  \mu_{f \rightarrow x}(x) = 0 \quad \mu_{x \rightarrow f}(x) = \ln f(x)
  \]
- Recursion
  \[
  \mu_{f \rightarrow x}(x) = \max_{x \in \text{domain}(x)} \left[ \ln f(x) + \sum_{m \in \text{np}(f \rightarrow x)} \mu_{x \rightarrow f}(x_m) \right]
  \]
  \[
  \mu_{x \rightarrow f}(x) = \sum_{i \in \text{np}(x \rightarrow f)} \mu_{f \rightarrow x}(x_i)
  \]
  - For each node, keep a record of which values of the variables gave rise to the maximum state:
  \[
  \phi(x) = \arg \max_{x \in \text{domain}(x)} \ln f(x) + \sum_{m \in \text{np}(f \rightarrow x)} \mu_{x \rightarrow f}(x_m)
  \]

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

Junction Tree Algorithm

- Motivation
  - Exact inference on general graphs.
  - Works by turning the initial graph into a junction tree and then running a sum-product-like algorithm.
  - Intractable on graphs with large cliques.
- Main steps
  1. If starting from directed graph, first convert it to an undirected graph by moralization.
  2. Introduce additional links by triangulation in order to reduce the size of cycles.
  3. Find cliques of the moralized, triangulated graph.
  4. Construct a new graph from the maximal cliques.
  5. Remove minimal links to break cycles and get a junction tree.
  \[
  \Rightarrow \text{Apply regular message passing to perform inference.}
  \]
Junction Tree Algorithm

1. Convert to an undirected graph through moralization.
   - Marry the parents of each node.
   - Remove edge directions.

2. Triangulate
   - Such that there is no loop of length > 3 without a chord.
   - This is necessary so that the final junction tree satisfies the "running intersection" property (explained later).

3. Find cliques of the moralized, triangulated graph.

4. Construct a new junction graph from maximal cliques.
   - Create a node from each clique.
   - Each link carries a list of all variables in the intersection.
   - Drawn in a "separator" box.

5. Remove links to break cycles \(\Rightarrow\) junction tree.
   - For each cycle, remove the link(s) with the minimal number of shared nodes until all cycles are broken.
   - Result is a maximal spanning tree, the junction tree.
Junction Tree - Properties

- Running intersection property
  - "If a variable appears in more than one clique, it also appears in all intermediate cliques in the tree".
  - This ensures that neighboring cliques have consistent probability distributions.
  - Local consistency → global consistency

Junction Tree: Example 1

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
  3. Find cliques
  4. Construct junction graph

Junction Tree: Example 2

- Without triangulation step
  - The final graph will contain cycles that we cannot break without losing the running intersection property!

- When applying the triangulation
  - Only small cycles remain that are easy to break.
  - Running intersection property is maintained.
Junction Tree Algorithm

- **Good news**
  - The junction tree algorithm is efficient in the sense that for a given graph there does not exist a computationally cheaper approach.

- **Bad news**
  - This may still be too costly.
  - Effort determined by number of variables in the largest clique.
  - Grows exponentially with this number (for discrete variables).
  - Algorithm becomes impractical if the graph contains large cliques!

Loopy Belief Propagation

- Alternative algorithm for loopy graphs
  - Sum-Product on general graphs.
  - Strategy: simply ignore the problem.
  - Initial unit messages passed across all links, after which messages are passed around until convergence
    - Convergence is not guaranteed!
    - Typically break off after fixed number of iterations.
  - Approximate but tractable for large graphs.
  - Sometime works well, sometimes not at all.

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