**The Belief Propagation (Sum-Product) Algorithm**

Probabilistic Graphical Models (10-708)

Lecture 5, Sep 31, 2007

Eric Xing

Reading: J-Chap 4

---

**From Elimination to Belief Propagation**

- Recall that induced dependency during marginalization is captured in elimination cliques
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

\[
P(a)P(b)P(c|b)P(d|a)cP(e|c,d)P(f|a)eP(g|e)P(h|e,f) \\
\Rightarrow P(a)P(b)P(c|b)P(d|a)cP(e|c,d)P(f|a)eP(g|e)\phi_0(e,f) \\
\Rightarrow P(a)P(b)P(c|b)P(d|a)cP(e|c,d)P(f|a)\phi_0(e)\phi_0(e,f) \\
\Rightarrow P(a)P(b)P(c|b)P(d|a)cP(e|c,d)\phi_2(a,c,e) \\
\Rightarrow P(a)P(b)P(c|b)P(d|a)\phi_2(a,c,e,d) \\
\Rightarrow P(a)P(b)P(c|b)\phi_2(a,c) \\
\Rightarrow P(a)\phi_0(a,b) \\
\Rightarrow P(a)\phi_0(a) \\
\Rightarrow \phi(a)
\]

- Can this lead to a generic inference algorithm?
Tree GMs

- **Undirected tree:** a unique path between any pair of nodes.
- **Directed tree:** all nodes except the root have exactly one parent.
- **Poly tree:** can have multiple parents.

Equivalence of directed and undirected trees

- Any undirected tree can be converted to a directed tree by choosing a root node and directing all edges away from it.
- A directed tree and the corresponding undirected tree make the same conditional independence assertions.
- Parameterizations are essentially the same.
  - **Undirected tree:** \( p(x) = \frac{1}{Z} \prod_{i \in V} \psi(x_i) \prod_{\langle i,j \rangle \in E} \psi(x_i, x_j) \)
  - **Directed tree:** \( p(x) = p(x_r) \prod_{\langle i,r \rangle \in E} p(x_j|x_i) \)
  - **Equivalence:** \( \psi(x_r) = p(x_r); \quad \psi(x_i, x_j) = p(x_j|x_i); \quad Z = 1, \quad \psi(x_i) = 1 \)
  - **Evidence:**
From elimination to message passing

- Recall ELIMINATION algorithm:
  - Choose an ordering \( Z \) in which query node \( f \) is the final node
  - Place all potentials on an active list
  - Eliminate node \( i \) by removing all potentials containing \( i \), take sum/product over \( x_i \)
  - Place the resultant factor back on the list

- For a TREE graph:
  - Choose query node \( f \) as the root of the tree
  - View tree as a directed tree with edges pointing towards from \( f \)
  - Elimination ordering based on depth-first traversal
  - Elimination of each node can be considered as message-passing (or Belief Propagation) directly along tree branches, rather than on some transformed graphs
  - thus, we can use the tree itself as a data-structure to do general inference!!

The elimination algorithm

**Procedure Initialize** \((G, Z)\)
1. Let \( Z_1, \ldots, Z_k \) be an ordering of \( Z \) such that \( Z_i < Z_j \) iff \( i < j \)
2. Initialize \( \mathcal{F} \) with the full the set of factors

**Procedure Evidence** \((E)\)
1. for each \( i \in E \),
   \[ \mathcal{F} = \mathcal{F} \cup \delta(E_i, e_i) \]

**Procedure Sum-Product-Variable-Elimination** \((\mathcal{F}, Z, <)\)
1. for \( i = 1, \ldots, k \)
   \[ \mathcal{F} \leftarrow \text{Sum-Product-Eliminate-Var}(\mathcal{F}, Z_i) \]
2. \( \phi^* \leftarrow \prod_{\phi \in \mathcal{F}} \phi \)
3. return \( \phi^* \)
4. Normalization \((\phi^*)\)

**Procedure Normalization** \((\phi^*)\)
1. \( P(\lambda|E) = \phi^*(\lambda) \sum_{\phi^*} \phi^*(\lambda) \)

**Procedure Sum-Product-Eliminate-Var** \((\mathcal{F}, Z, \prec)\)
1. \( \mathcal{F}' \leftarrow \{ \phi \in \mathcal{F} : Z \in \text{Scope}[\phi] \} \)
2. \( \mathcal{F}'' \leftarrow \mathcal{F} \setminus \mathcal{F}' \)
3. \( \psi \leftarrow \prod_{\phi \in \mathcal{F}''} \phi \)
4. \( \tau \leftarrow \sum_{\phi \in \mathcal{F}''} \psi \)
5. return \( \mathcal{F}'' \cup \{ \tau \} \)
Message passing for trees

Let $m(x_i)$ denote the factor resulting from eliminating variables from below up to $i$, which is a function of $x_i$:

$$m(x_i) = \sum_{x_j} \left( \psi(x_j) \psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_k(x_j) \right)$$

This is reminiscent of a message sent from $j$ to $i$.

$$m_{ji}(x_i) = \sum_{x_j} \left( \psi(x_j) \psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_k(x_j) \right)$$

$$p(x_f) \propto \psi(x_f) \prod_{e \in E} m_{ef}(x_f)$$

$m_j(x_i)$ represents a "belief" of $x_i$ from $x_j$.

- Elimination on trees is equivalent to message passing along tree branches!
The message passing protocol:

- A node can send a message to its neighbors when (and only when) it has received messages from all its *other* neighbors.
- Computing node marginals:
  - Naive approach: consider each node as the root and execute the message passing algorithm.

$$
\begin{align*}
\text{Computing } P(X_1) \\
X_1 & \quad m_{21}(x_1) \\
X_2 & \quad m_{32}(x_2) \quad m_{42}(x_2) \\
X_3 & \\
X_4 &
\end{align*}
$$

The message passing protocol:

- A node can send a message to its neighbors when (and only when) it has received messages from all its *other* neighbors.
- Computing node marginals:
  - Naive approach: consider each node as the root and execute the message passing algorithm.

$$
\begin{align*}
\text{Computing } P(X_2) \\
X_1 & \quad m_{21}(x_2) \\
X_2 & \quad m_{12}(x_2) \quad m_{42}(x_2) \\
X_3 & \\
X_4 &
\end{align*}
$$
The message passing protocol:

- A node can send a message to its neighbors when (and only when) it has received messages from all its \textit{other} neighbors.

Computing node marginals:
- Naive approach: consider each node as the root and execute the message passing algorithm

\[ \text{Computing } P(X_3) \]

Computing node marginals

- Naive approach:
  - Complexity: \text{NC}
  - \text{N} is the number of nodes
  - \text{C} is the complexity of a complete message passing

- Alternative dynamic programming approach
  - 2-Pass algorithm (next slide ➔)
  - Complexity: \text{2C!}
The message passing protocol:

- A two-pass algorithm:

Belief Propagation (SP-algorithm): Sequential implementation
Belief Propagation (SP-algorithm): Parallel synchronous implementation

- For a node of degree d, whenever messages have arrived on any subset of d-1 node, compute the message for the remaining edge and send!
- A pair of messages have been computed for each edge, one for each direction
- All incoming messages are eventually computed for each node

Correctness of BP on tree

- Collorary: the synchronous implementation is "non-blocking"

- Thm: The Message Passage Guarantees obtaining all marginals in the tree

\[ m_{i:j}(x_i) = \sum_{x_j} \psi(x_j) \psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{k:j}(x_j) \]

- What about non-tree?
Another view of SP: Factor Graph

- Example 1

\[
P(X_1) \quad P(X_2) \quad P(X_3|X_1,X_2) \quad P(X_5|X_1,X_3) \quad P(X_4|X_2,X_3)
\]

\[
fa(X_1) \quad fb(X_2) \quad fc(X_3,X_1,X_2) \quad fd(X_5,X_1,X_3) \quad fe(X_4,X_2,X_3)
\]

Another view of SP: Factor Graph

- Example 2

\[
\psi(x_1,x_2,x_3) = fa(x_1,x_2)fb(x_2,x_3)fc(x_3,x_1)
\]

- Example 3

\[
\psi(x_1,x_2,x_3) = fa(x_1,x_2,x_3)
\]
**Factor Tree**

- A Factor graph is a **Factor Tree** if the undirected graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree.

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

**Message Passing on a Factor Tree**

- Two kinds of messages:
  1. \(\nu\): from variables to factors
  2. \(\mu\): from factors to variables

\[
\nu_x(x_i) = \prod_{t \in \mathcal{N}(i) \setminus o} \mu_t(x_t) \\
\mu_x(x_i) = \sum_{x_{\mathcal{N}(s) \setminus i}} \left( f_s(x_{\mathcal{N}(s) \setminus i}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_j(x_j) \right)
\]
Message Passing on a Factor Tree, con'd

- Message passing protocol:
  - A node can send a message to a neighboring node only when it has received messages from all its other neighbors
- Marginal probability of nodes:

\[
P(x_i) \propto \prod_{s \in N(i)} \mu_s(x_i) \\
\propto \nu_{i_a}(x_i) \nu_{i_b}(x_i)
\]

BP on a Factor Tree
Why factor graph?

- Tree-like graphs to Factor trees

Poly-trees to Factor trees
Why factor graph?

- Because FG turns tree-like graphs to factor trees,
- and trees are a data-structure that guarantees correctness of BP!

Max-product algorithm: computing MAP probabilities

\[
\max_{x} p(x) = \max_{x_f} \psi(x_f) m_{i_f}(x_f)
\]

\[
m_{i_j}(x_i) = \max_{x_j} \left( \psi(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) - i} m_{k_j}(x_j) \right)
\]

\[
m_{i_j}(x_j)
\]
Max-product algorithm:
computing MAP configurations using a final bookkeeping backward pass

\[
x_f^* = \arg \max_{x_f} (\psi(x_f)m_{i_f}(x_f))
\]

\[
x_i^* = \arg \max_{x_i} (\psi(x_i)\psi(x_f^*, x_i)m_{ij}(x_i))
\]

\[
x_j^* = \arg \max_{x_j} (\psi(x_j)\psi(x_i^*, x_j)m_{kj}(x_j)m_{lj}(x_j))
\]

\[
x_k^* = \arg \max_{x_k} (\psi(x_k)\psi(x_k, x_j^*))
\]

Summary

- Sum-Product algorithm computes singleton marginal probabilities on:
  - Trees
  - Tree-like graphs
  - Poly-trees

- Maximum a posteriori configurations can be computed by replacing sum with max in the sum-product algorithm
  - Extra bookkeeping required
Inference on general GM

- Now, what if the GM is not a tree-like graph?
- Can we still directly run message
  message-passing protocol along its edges?
- For non-trees, we do not have the guarantee that message-passing
  will be consistent!
- Then what?
  - Construct a graph data-structure from \( P \) that has a tree structure, and run
    message-passing on it!

\[ \rightarrow \] Junction tree algorithm

Elimination Clique

- Recall that Induced dependency during marginalization is captured in elimination cliques
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

\[
\begin{align*}
P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|c)P(h|e,f) & \Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|c)\phi_1(e,f) \\
P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|c)\phi_2(e) & \Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)\phi_2(e) \\
P(a)P(b)P(c|b)P(d|a)P(e|c,d)\phi_3(a,e,c) & \Rightarrow P(a)P(b)P(c|b)P(d|a)\phi_3(a,e,c,d) \\
P(a)P(b)P(c|b)\phi_4(a,e) & \Rightarrow P(a)P(b)\phi_4(a,b) \\
P(a)\phi_5(a) & \Rightarrow \phi_5(a)
\end{align*}
\]

- Can this lead to an generic inference algorithm?
A Clique Tree

\[
m_{(a,c,d)} = \sum_e p(e|c,d)m_g(e)m_f(a,e)
\]

From Elimination to Message Passing

- Elimination \(\equiv\) message passing on a clique tree

\[
m_{(a,c,d)} = \sum_e p(e|c,d)m_g(e)m_f(a,e)
\]

- Messages can be reused
From Elimination to Message Passing

- Elimination $\equiv$ message passing on a clique tree
  - Another query ...

- Messages $m_y$ and $m_h$ are reused, others need to be recomputed