Exact Inference 3: Message Passing

Introduction

• We will cover the sum-product message passing algorithm
• Also known as belief propagation
Introduction

- Message passing is exact when the graph has no (undirected) loops eg.

- If there are loops, you need to use loopy belief propagation (which is approximate)

Message Passing

Intuition (using a chain as an example)
- Each node maintains its current marginal $P(X_i)$ (also called its belief).
- Initially, the marginal doesn’t take the influence of the neighbors into account
- Note that a Node’s belief is affected by its neighbors
- Neighboring nodes send messages to each other
Introduction

Intuition (using a chain as an example)

- Node $X_2$ receives a message $m_{1\rightarrow 2}$ from Node $X_1$
- The message tells Node $X_2$ what state Node $X_1$ thinks Node $X_2$ should be in
- The higher the value of the message, the more likely Node $X_1$ thinks Node $X_2$ should be in that state
- Node $X_2$ updates its belief about $P(X_2)$

\[
\text{\includegraphics[width=0.5\textwidth]{chain_diagram.png}}
\]

At convergence, the belief at a Node $X_i$ is the marginal probability $P(X_i)$

- This is equivalent to a dynamic programming approach (very efficient!)
Introduction

What if the graphical model isn’t a chain or a tree?
- Clump nodes into “mega-nodes” (ie. cliques) and treat the cliques like nodes
- This is where clique trees come in
Cluster Graph

In this section we are dealing with a product over factors:

\[ \tilde{P}_\Phi(X) = \prod_{\phi_i \in \Phi} \phi_i(X_i) \]

- Normalized distribution for Bayesian networks since factors are CPDs
- Unnormalized distribution for Gibbs distributions

Cluster Graph

A cluster graph \( \mathcal{U} \) for a set of factors \( \Phi \) over \( X \) is an undirected graph, each of whose nodes \( i \) is associated with a subset \( C_i \subseteq X \).

Example of a cluster graph

1: C, D
2: G, I, D
3: G, S, I
4: H, G, J
5: G, J, S, L
Cluster Graph

• Each factor $\phi \in \Phi$ must be associated with a cluster $C$, denoted $\alpha(\phi)$, such that $\text{Scope}[\phi] \subseteq C_i$.
• Each edge between a pair of clusters $C_i$ and $C_j$ is associated with a sepset $S_{i,j} \subseteq C_i \cap C_j$.
• A cluster graph is a generalization of a clique tree.

Example of a cluster graph

Example of a cluster graph

A new way to interpret variable elimination:
• (Recall: variable elimination defines a cluster graph)
• Factors $\psi_i$ accept messages $\tau_j$ from another factor $\psi_j$
• Factors $\psi_i$ also send their own messages $\tau_i$ to another factor
Note:
- Cluster graph produced by variable elimination is a tree
- Each original factor $\phi$ is used only once to create cluster $\psi$
- Execution of variable elimination causes messages to flow "up" to a "root" node
Cluster Graph

• T has the **running intersection property** if, whenever there is a variable X such that \( X \in C_i \) and \( X \in C_j \), then X is also in every cluster in the (unique) path in T between \( C_i \) and \( C_j \).

• Example: cluster tree below obeys the running intersection property (see G in \( C_2 \) and \( C_4 \)).

```
1: C, D
2: G, I, D
3: G, S, I
4: H, G, J
5: G, J, S, L
```

• Running intersection property implies sepset \( S_{i,j} = C_i \cap C_j \).
Clique Tree

• Let $\Phi$ be a set of factors over $X$. A cluster tree over $\Phi$ that satisfies the running intersection property is called a clique tree (aka junction tree or join tree).

• In the case of a clique tree, the clusters are also called cliques.

Message Passing: Sum Product
Message Passing: Sum Product

• Assume we are given a clique tree
• Note: can use the same clique tree to cache computations for multiple executions of variable elimination
• Cheaper than performing each variable elimination separately

Example: Simplified Extended Student Clique tree

- First step: generate a set of initial potentials $\psi_i(C_i)$ with each clique eg. by multiplying the initial factors
  - For instance, $\psi_5(J,L,G,S) = \phi_L(L,G) \cdot \phi_J(J,L,S)$
- Suppose we have to compute $P(J)$:
  - Select a root clique that does contain $J$ eg. $C_5$. 
Message Passing: Sum Product

Execute the following:

- In $C_1$: Eliminate $C$ by $\sum_C \psi_1(C,D)$. Resulting factor has scope $D$. Send message $\delta_{1 \rightarrow 2}(D)$ to $C_2$.
- In $C_2$: Define $\beta_2(G,I,D) = \delta_{1 \rightarrow 2}(D) \cdot \psi_2(G,I,D)$. Eliminate $D$ to get a factor $\delta_{2 \rightarrow 3}(G,I)$ which is sent to $C_3$.
- In $C_3$: Define $\beta_3(G,S,I) = \delta_{2 \rightarrow 3}(G,I) \cdot \psi_3(G,S,I)$. Eliminate $I$ to get a factor $\delta_{3 \rightarrow 5}(G,S)$ which is sent to $C_5$.

Message Passing: Sum Product

Execute the following:

- In $C_4$: Eliminate $H$ by $\sum_H \psi_4(H,G,J)$. Send factor $\delta_{4 \rightarrow 5}(G,J)$ to $C_5$.
- In $C_5$: Define $\beta_5(G,J,S,L) = \delta_{3 \rightarrow 5}(G,S) \cdot \delta_{4 \rightarrow 5}(G,J) \cdot \psi_5(G,J,S,L)$.
- Sum out $G$, $L$, and $S$ from $\beta_5$ to get $P(J)$.
Message Passing: Sum Product

1: (C,D) → 2: (G, I, D) → 3: (G, S, I) → 5: (G, J, S, L) → 4: (H, G, J)

• Clique is ready when it has received all of its incoming messages eg.
  – C₄ ready at the start
  – C₂ ready only after getting message from C₁

• C₁, C₄, C₂, C₃, C₅ is a legal execution ordering for the tree rooted at C₅

• C₂, C₁, C₄, C₃, C₅ is not a legal execution ordering

Message Passing: Sum Product

1: (C,D) → 2: (G, I, D) → 3: (G, S, I) → 5: (G, J, S, L) → 4: (H, G, J)

δ₁→₂(D): Σᵣψ₁(C₁)
δ₂→₅(G,I): Σᵣψ₂(C₂) × δ₁→₂
δ₅→₆(G,S): Σᵣψ₅(C₅) × δ₂→₅

Could also define C₄ as the root

• In C₁: computation and message unchanged
• In C₂: computation and message unchanged
• In C₃: computation and message unchanged
Message Passing: Sum Product

C\textsubscript{4} as the root
- In C\textsubscript{5}: Define $\beta_5(G,J,S,L) = \delta_{3\rightarrow5}(G,S) \cdot \psi_5(G,J,S,L)$. Eliminate S and L. Send out factor $\delta_{5\rightarrow4}(G,J)$ to C\textsubscript{4}.
- In C\textsubscript{4}: Define $\beta_4(H,G,J) = \delta_{5\rightarrow4}(G,S) \cdot \psi_4(H,G,J)$.
- Eliminate H and G from $\beta_4(H,G,J)$ to get P(J)

Message Passing: Sum Product

Clique-Tree Message Passing
1. Set initial potentials
2. Pass messages to neighboring cliques, sending to root clique
Message Passing: Sum Product

1. Initial potentials
   – Each factor $\phi \in \Phi$ is assigned to some clique $\alpha(\phi)$
   – The initial potential of $C_j$ is:
     \[ \psi_j(C_j) = \prod_{\phi: \alpha(\phi) = j} \phi \]
   – Since each factor is assigned to exactly one clique, we have:
     \[ \prod_{\phi} \phi = \prod_j \psi_j \]

Message Passing: Sum Product

2. Message passing
   – Definitions:
     • $C_r$ = root clique
     • $\text{Nb}_i$ = indices of cliques that are neighbors of $C_i$
     • $p_r(i)$ = upstream neighbor of $i$ (the one on the path to the root clique $r$)
   – Start with the leaves of the clique tree and move inward
   – Each clique $C_i$ (except for the root) performs a message passing computation and sends message to upstream neighbor $C_{p_r(i)}$
Message Passing: Sum Product

Message from $C_i$ to $C_j$:
\[
\delta_{i\rightarrow j} = \sum_{C_i \in S_{i,j}} \psi_i \cdot \prod_{k \in (\text{Nh}_i \setminus \{j\})} \delta_{k\rightarrow i}
\]

- Clique $C_i$ multiplies incoming messages from its neighbors (except $j$) with its initial clique potential.
- Sums out all variables except those in the sepset between $C_i$ and $C_j$.
- Sends resulting factor to $C_j$.

At the root, once all messages are received, it multiplies them with its own initial potential.

Result is a factor called the beliefs $\beta_r(C_r)$, which represents:
\[
\tilde{P}_\phi(C_r) = \sum_{X \setminus C_r} \prod_{\phi} \phi
\]
Message Passing: Sum Product

**Procedure** CT-Tree-SP-Upward (Φ, // Set of factors
\(\mathcal{T}\), // Clique tree over \(\Phi\)
\(\alpha\), // Initial assignment of factors to cliques
\(\mathcal{C}_r\) // Some selected root clique
)

1. Initialize-Cliques()
2. while \(\mathcal{C}_r\) is not ready
3. Let \(\mathcal{C}_i\) be a ready clique
4. \(\delta_{i\rightarrow pr(i)}(S_{i,pr(i)}) \leftarrow SP\text{-}Message(i, pr(i))\)
5. \(\beta_r \leftarrow \psi_r \cdot \prod_{k \in \text{Nh}_r} \delta_{k\rightarrow r}\)
6. return \(\beta_r\)

---

Message Passing: Sum Product

**Procedure** Initialize-Cliques ()
1. for each clique \(\mathcal{C}_i\)
2. \(\psi_i(\mathcal{C}_i) \leftarrow \prod_{\phi : \alpha(\phi) = i} \phi\)

**Procedure** SP-Message (i, // sending clique
j // receiving clique
)
1. \(\psi(\mathcal{C}_i) \leftarrow \psi_i \cdot \prod_{k \in \text{Nh}_r \sim j} \delta_{k\rightarrow i}\)
2. \(\tau(S_{i,j}) \leftarrow \sum_{\mathcal{C}_i \in S_{i,j}} \psi(\mathcal{C}_i)\)
3. return \(\tau(S_{i,j})\)