Message Passing and Junction Tree Algorithms

Kayhan Batmanghelich
\[ P(C, D, I, G, S, L, J, H) \]
\[ = P(C)P(D|C)P(I)P(G|I, D)P(S|I)P(L|G)P(J|L, S)P(H|G, J) \]
\[ p(C, D, I, G, S, L, J, H) \]
\[ = \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D)\psi_S(S, I)\psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J) \]
Review

\((C, D, I, \hat{H}, G, S, L)\)

Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Slides adapted from Matt Gormley (2016)

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Slides adapted from Matt Gormley (2016)
adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Belief:
Must be 14 of us

Slides adapted from Matt Gormley (2016)
adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Belief:
Must be 14 of us

wouldn't work correctly with a 'loopy' (cyclic) graph

adapted from MacKay (2003) textbook
Message from one C1 to C2: Multiply all incoming messages with the local factor and sum over variables that are not shared

\[
m_e(a, c, d) = \sum_e p(e | c, d)m_g(e)m_f(a, e)
\]
Message passing (Belief Propagation) on singly connected graph
Remember this: Factor Graph?

A factor graph is a graphical model representation that unifies directed and undirected models.

It is an undirected bipartite graph with two kinds of nodes.
- **Round** nodes represent variables,
- **Square** nodes represent factors.

And there is an edge from each variable to every factor that mentions it.

We are going to study messages passing between nodes.
How General Are Factor Graphs?

• Factor graphs can be used to describe
  – Markov Random Fields (undirected graphical models)
    • i.e., log-linear models over a tuple of variables
  – Conditional Random Fields
  – Bayesian Networks (directed graphical models)

• Inference treats all of these interchangeably.
  – Convert your model to a factor graph first.
  – Pearl (1988) gave key strategies for exact inference:
    • Belief propagation, for inference on acyclic graphs
    • Junction tree algorithm, for making any graph acyclic
      (by merging variables and factors: blows up the runtime)
Factor Graph Notation

• Variables:

$$\mathcal{X} = \{X_1, \ldots, X_i, \ldots, X_n\}$$

• Factors:

$$\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$$

where $$\alpha, \beta, \gamma, \ldots \subseteq \{1, \ldots n\}$$

Joint Distribution

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\mathbf{x}_\alpha)$$

where $$\alpha$$ is a subset of $$\{1, \ldots n\}$$.
Factors are Tensors

- Factors:

  \[ \psi_\alpha, \psi_\beta, \psi_\gamma, \ldots \]
An Inference Example

\[ p(a, b) = f_1(a, b) \sum_{c, d} f_2(b, c, d) f_3(c) f_5(d) \sum_{e} f_4(d, e) \]

\[ \mu_{f_2 \rightarrow b}(b) \]

\[ p(a) = \sum_{b} f_1(a, b) \mu_{f_2 \rightarrow b}(b) \]

\[ \mu_{f_1 \rightarrow a}(a) \]

\[ f_X(a, b) = p(a|b) \]

\[ p(a|b)p(b|c, d)p(c)p(d)p(e|d) \]

\[ f_1(a, b) f_2(b, c, d) f_3(c) f_4(d, e) f_5(d) \]
Both of these messages judge the possible values of variable $X$. Their product = belief at $X$ = product of all 3 messages to $X$. 

Slides adapted from Matt Gormley (2016)
Sum-Product Belief Propagation

Variables

Factors

Beliefs

Messages

Slides adapted from Matt Gormley (2016)
Sum-Product Belief Propagation

Variable Belief

\[ b_i(x_i) = \prod_{\alpha \in N(i)} \mu_{\alpha \rightarrow i}(x_i) \]
Sum-Product Belief Propagation

Variable Message

\[
\mu_i \rightarrow \alpha(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i)
\]

Slides adapted from Matt Gormley (2016)
Sum-Product Belief Propagation

Factor Belief

\[ b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]
Sum-Product Belief Propagation

Factor Belief

\[ b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in N(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]
Sum-Product Belief Propagation

\[ \mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i]) \]

\[ \sum_{x_3} \psi_1(x_3) \{ x_3 \rightarrow \psi \} \]
Sum-Product Belief Propagation

Factor Message

matrix-vector product
(for a binary factor)

\[
\mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in N(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i])
\]

Slides adapted from Matt Gormley (2016)
Summary of the Messages

Variable to Factor message

\[ \mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x) \]

Factor to Variable message

\[ \mu_{f \rightarrow x}(x) = \max_{\mathcal{A}_f} \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f}(y) \]

Marginal

\[ p(x) \propto \prod_{f \in \text{ne}(x)} \mu_{f \rightarrow x}(x) \]
Sum-Product Belief Propagation

**Input:** a factor graph with no cycles  
**Output:** exact marginals for each variable and factor

**Algorithm:**
1. Initialize the messages to the uniform distribution.
   \[ \mu_{i \rightarrow \alpha}(x_i) = 1 \quad \mu_{\alpha \rightarrow i}(x_i) = 1 \]
1. Choose a root node.
2. Send messages from the **leaves** to the **root**.
   Send messages from the **root** to the **leaves**.
   \[ \mu_{i \rightarrow \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i) \quad \mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[j]) \]
1. Compute the beliefs (unnormalized marginals).
   \[ b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i}(x_i) \quad b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]
2. Normalize beliefs and return the **exact** marginals.
   \[ p_i(x_i) \propto b_i(x_i) \quad p_\alpha(x_\alpha) \propto b_\alpha(x_\alpha) \]
Sum-Product Belief Propagation

\[ b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i}(x_i) \]

\[ b_{\alpha}(x_\alpha) = \psi_{\alpha}(x_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]

Slides adapted from Matt Gormley (2016)
Sum-Product Belief Propagation

Variables

Beliefs

Factors

Messages

\[ \mu_{i \rightarrow \alpha}(x_i) = \prod_{\alpha \in N(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i) \]

\[ \mu_{\alpha \rightarrow i}(x_i) = \sum_{x_{\alpha_i} : x_{\alpha_i}[i] = x_i} \psi_{\alpha}(x_{\alpha}) \prod_{j \in N(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_{\alpha}[i]) \]

Slides adapted from Matt Gormley (2016)
In a factor graph with no cycles:
1. Pick any node to serve as the root.
2. Send messages from the leaves to the root.
3. Send messages from the root to the leaves.
A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.

Slides adapted from Matt Gormley (2016)
(Acyclic) Belief Propagation

In a factor graph with no cycles:
1. Pick any node to serve as the root.
2. Send messages from the leaves to the root.
3. Send messages from the root to the leaves.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.
A note on the implementation

To avoid numerical precision issue, use log message ($\lambda = \log \mu$):

Variable to Factor message

$$\mu_{x \rightarrow f} (x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x} (x)$$

$$\lambda_{x \rightarrow f} (x) = \sum_{g \in \{\text{ne}(x) \setminus f\}} \lambda_{g \rightarrow x} (x)$$

Factor to Variable message

$$\mu_{f \rightarrow x} (x) = \max_{\{x' \in \mathcal{X}_f \setminus x\}} \phi_f (x') \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f} (y)$$

$$\lambda_{f \rightarrow x} (x) = \log \left( \sum_{\mathcal{X}_f \setminus x} \phi_f (x') \exp \left( \sum_{y \in \{\text{ne}(f) \setminus x\}} \lambda_{y \rightarrow f} (y) \right) \right)$$
How about other queries?
(MPA, Evidence)
Example

\[
\max_x f(x) = \max_{x_1, x_2, x_3, x_4} \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4) = \max_{x_1, x_2, x_3} \phi(x_1, x_2)\phi(x_2, x_3) \max_{x_4} \phi(x_3, x_4)
\]

\[
\max_{x_1, x_2, x_3} \phi(x_1, x_2)\phi(x_2, x_3) \gamma_3(x_3)
\]

\[
\max_{x_1, x_2} \phi(x_1, x_2)\gamma_3(x_2)
\]

\[
\max_{x_1} \gamma_2(x_1)
\]
The Max Product Algorithm

Variable to Factor message

\[
\mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x)
\]

Factor to Variable message

\[
\mu_{f \rightarrow x}(x) = \max_{\mathcal{X}_f \setminus x} \phi_f(\mathcal{X}_f) \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f}(y)
\]

Maximal State

\[
P(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5)
\]

\[
x^* = \arg\max_x \prod_{f \in \text{ne}(x)} \mu_{f \rightarrow x}(x)
\]
Can I use BP in a multiply connected graph?
Loops the trouble makers

- One needs to account for the fact that the structure of the graph changes.
- The junction tree algorithm deals with this by combining variables to make a new singly connected graph for which the graph structure remains singly connected under variable elimination.
Clique Graph

- **Def (Clique Graph):** A clique graph consists of a set of potentials, $\phi_1(\chi_1), \ldots, \phi_n(\chi_n)$ each defined on a set of variables $\chi_1$. For neighboring cliques on the graph, defined on sets of variables $\chi_1$ and $\chi_j$, the intersection $\chi_s = \chi_i \cap \chi_j$ is called the separator and has a corresponding potential $\phi_s(\chi_s)$. A clique graph represents the function

$$\frac{\prod_c \phi_c(\chi^c)}{\prod_s \phi_s(\chi^s)}$$

**Example**

$$\Psi(A,B,C) \cdot \phi(B,C,D) \cdot \phi(\chi_1) \cdot \phi(\chi_2) / \phi(\chi_1 \cap \chi_2)$$

$$\phi(B,C)$$
Clique Graph

• Def (Clique Graph): A clique graph consists of a set of potentials, \( \phi_1(\chi_1), \ldots, \phi_n(\chi_n) \) each defined on a set of variables \( \chi_1 \). For neighboring cliques on the graph, defined on sets of variables \( \chi_1 \) and \( \chi_j \), the intersection \( \chi_s = \chi_i \cap \chi_j \) is called the separator and has a corresponding potential \( \phi_s(\chi_s) \).

A clique graph represents the function

\[
\frac{\prod_c \phi_c(\chi^c)}{\prod_c (\phi(\chi^c)/\phi(\chi^c_{\text{separator}}))}
\]

Don’t confuse it with Factor Graph!

Example

\[
\begin{array}{c}
\chi^1 \\
\chi^1 \cap \chi^2 \\
\chi^2
\end{array}
\]

\[
\phi(\chi^1)\phi(\chi^2)/\phi(\chi^1 \cap \chi^2)
\]
Example: probability density

\[ Z p(a, b, c) = \sum_{d} \phi(a, b, c) \phi(b, c, d) \]
\[ Z p(b, c, d) = \sum_{a} \phi(a, b, c) \phi(b, c, d) \]

\[ p(a, b, c, d) = \frac{\phi(a, b, c) \phi(b, c, d)}{Z} \]

\[ p(a, b, c, d) = \frac{\phi(a, b, c) \phi(b, c, d)}{Z} \]

\[ Z = \sum_{a, d} \phi(a, b, c) \]

\[ \psi(b, c) \]

\[ \sum_{a, d} \phi(a, b, c) \]

\[ \frac{\phi(a, b, c) \phi(b, c, d)}{Z} \]

\[ \psi(b, c) \]
Junction Tree

• **Idea**: form a new representation of the graph in which variables are clustered together, resulting in a singly-connected graph in the cluster variables.

• **Insight**: distribution can be written as product of marginal distributions, divided by a product of the intersection of the marginal distributions.

• Not a remedy to the intractability.
Let’s learn by an example....
Let’s learn by an example....

Let’s pick an ordering for the variable elimination

C, D, I, H, G, S, L
Let’s learn by an example....

Let’s pick an ordering for the variable elimination

C, D, I, H, G, S, L
Let’s learn by an example....

Let’s pick an ordering for the variable elimination: C, D, I, H, G, S, L.
Let’s learn by an example....

Let’s pick an ordering for the variable elimination
C, D, I, H, G, S, L
Let’s learn by an example....

Let’s pick an ordering for the variable elimination:

C, D, I, H, G, S, L
Let’s learn by an example....

Let’s pick an ordering for the variable elimination C, D, I, H, G, S, L

The rest is obvious
OK, what we got so far?

We started with

Moralized and Triangulated
**Def:** An undirected graph is triangulated if every loop of length 4 or more has a *chord*. An equivalent term is that the graph is *decomposable* or *chordal*. From this definition, one may show that an undirected graph is triangulated if and only if its clique graph has a junction tree.
Let’s build the Junction Tree

The ordering
C, D, I, H, G, S, L
Pass the messages on the JT

The ordering  C, D, I, H, G, S, L

Can you figure out the directionality?
The message passing protocol

Cluster B is allowed to send a message to a neighbor C only after it has received messages from all neighbors except C.

```python
def COLLECT(C):
    for B in children (C):
        COLLECT(B)
        send message to C

def DISTRIBUT(E):  
    for B in children (C):
        send message to B
        DISTRIBUT(E)
```
Message from Clique to another (The Shafer-Shenoy Algorithm)

\[ \mu_{B\to C}(u) = \sum_{v \in B \setminus C} \psi_B(u \cup v) \prod_{(A,B) \in \mathcal{E}, A \neq C} \mu_{A\to B}(u_A \cup v_A) \]
Formal Algorithm

- **Moralisation**: Marry the parents (only for directed distributions).
- **Triangulation**: Ensure that every loop of length 4 or more has a chord.
- **Junction Tree**: Form a junction tree from cliques of the triangulated graph, removing any unnecessary links in a loop on the cluster graph. Algorithmically, this can be achieved by finding a tree with *maximal spanning weight* with weight given by the number of variables in the separator between cliques. Alternatively, given a clique *elimination order* (with the lowest cliques eliminated first), one may connect each clique to the single neighboring clique.
- **Potential Assignment**: Assign potentials to junction tree cliques and set the separator potentials to unity.
- **Message Propagation**
Some Facts about BP

• BP is exact on trees.

• If BP converges it has reached a local minimum of an objective function
  • (the Bethe free energy Yedidia et.al ‘00, Heskes ’02) → often good approximation

• If it converges, convergence is fast near the fixed point.

• Many exciting applications:
  - error correcting decoding (MacKay, Yedidia, McEliece, Frey)
  - vision (Freeman, Weiss)
  - bioinformatics (Weiss)
  - constraint satisfaction problems (Dechter)
  - game theory (Kearns)
  - ...

\[ f(x) = x \]
Summary of the Network Zoo

**UGM and DGM**
- Use to represent family of probability distributions
- Clear definition of arrows and circles

**Factor Graph**
- A way to present factorization for both UGM and DGM
- It is bipartite graph
- More like a data structure
- Not to read the independencies

**Clique graph or Junction Tree**
- A data structure used for exact inference and message passing
- Nodes are cluster of variables
- Not to read the independencies