Belief Propagation, Junction Trees, and Factor Graphs

Daniel Lowd
CIS 410/510pm
Inference in a chain graph

\[ P(A) = \sum_{b,c,d} P(A,B = b,C = c,D = d) \]

\[ P(A) = \sum_{b,c,d} \frac{1}{Z} \phi_{AB}(A,b) \phi_{BC}(b,c) \phi_{CD}(c,d) \]

\[ P(A) = \alpha \sum_{b} \phi_{AB}(A,b) \sum_{c} \phi_{BC}(b,c) \sum_{d} \phi_{CD}(c,d) \]

\[ P(A) = \alpha \sum_{b} \phi_{AB}(A,b) \sum_{c} \phi_{BC}(b,c) m_{D \rightarrow C}(c) \]

\[ P(A) = \alpha \sum_{b} \phi_{AB}(A,b) m_{C \rightarrow B}(b) \]

\[ P(A) = \alpha m_{B \rightarrow A}(A) \]
Inference in a chain graph

\[ P(D) = \sum_{a,b,c} P(A = a, B = b, C = c, D) \]

\[ P(D) = \sum_{a,b,c} \frac{1}{Z} \phi_{AB}(a,b) \phi_{BC}(b,c) \phi_{CD}(c,D) \]

\[ P(D) = \alpha \sum_{c} \phi_{CD}(c,D) \sum_{b} \phi_{BC}(b,c) \sum_{a} \phi_{AB}(a,b) \]

\[ P(D) = \alpha \sum_{c} \phi_{CD}(c,D) \sum_{b} \phi_{BC}(b,c) m_{A \rightarrow B}(b) \]

\[ P(D) = \alpha \sum_{c} \phi_{CD}(c,D) m_{B \rightarrow C}(c) \]

\[ P(D) = \alpha m_{C \rightarrow D}(D) \]
Inference in a chain graph

\[ P(B) = \sum_{a,c,d} P(A = a,B,C = c,D = d) \]
\[ P(B) = \sum_{a,c,d} \frac{1}{Z} \phi_{AB}(a,b) \phi_{BC}(B,c) \phi_{CD}(c,d) \]
\[ P(B) = \alpha \left( \sum_a \phi_{AB}(a,B) \right) \left( \sum_c \phi_{BC}(B,c) \sum_d \phi_{CD}(c,d) \right) \]
\[ P(D) = \alpha m_{A\rightarrow B}(B) m_{C\rightarrow B}(B) \]

We already computed these messages when evaluating P(A) and P(D)!
What about a tree?

\[ P(B) = \sum_{a,c,d,e} P(A = a, B, C = c, D = d, E = e) \]

\[ P(B) = \sum_{a,c,d,e} \frac{1}{Z} \phi_{AB}(a,b) \phi_{BC}(B,c) \phi_{CD}(c,d) \phi_{CE}(c,e) \]

\[ P(B) = \alpha \left( \sum_a \phi_{AB}(a,B) \right) \left( \sum_c \phi_{BC}(B,c) \right) \left( \sum_d \phi_{CD}(c,d) \right) \left( \sum_e \phi_{CE}(c,e) \right) \]

\[ P(B) = \alpha m_{A \to B}(B) \left( \sum_c \phi_{BC}(B,c) m_{D \to C}(c) m_{E \to C}(c) \right) \]

\[ P(B) = \alpha m_{A \to B}(B) m_{C \to B}(B) \]
What about a tree?

\[
P(B) = \alpha m_{A\rightarrow B}(B) \left( \sum_c \phi_{BC}(B,c) m_{D\rightarrow C}(c)m_{E\rightarrow C}(c) \right)
\]

Therefore:

\[
m_{C\rightarrow B}(B) = \sum_c \phi_{BC}(B,c)m_{D\rightarrow C}(c)m_{E\rightarrow C}(c)
\]

General Pattern:

\[
m_{X\rightarrow Y}(Y) = \sum_x \phi_{XY}(x,Y) \prod_{W\in\text{neighbor}(X)/Y} m_{W\rightarrow X}(x)
\]
Overview of Belief Propagation

• We must receive incoming messages from all other neighbors in order to send an outgoing message to that neighbor:

\[ m_{X \to Y}(Y) = \sum_x \phi_{XY}(x,Y) \prod_{W \in \text{neighbor}(X)/Y} m_{W \to X}(x) \]

• Marginal probability of a variable is the produce of all messages addressed to that variable:

\[ P(X) = \prod_{W \in \text{neighbor}(X)} m_{W \to X}(x) \]

• In a tree, we can compute all marginals by:
  – Choosing a root
  – Sending messages from the leaves to the root
  – Sending messages from the root to the leaves
Belief Propagation in a Tree

\[ m_{X \rightarrow Y}(Y) = \sum_{x} \phi_{XY}(x,Y) \prod_{W \in \text{neighbor}(X)/Y} m_{W \rightarrow X}(x) \]

1. Upward Pass
Belief Propagation in a Tree

\[ m_{X \to Y}(Y) = \sum_x \phi_{XY}(x,Y) \prod_{W \in \text{neighbor}(X)/Y} m_{W \to X}(x) \]

2. Downward Pass
Junction Tree Algorithm

• How do we handle loops?
• Create a junction tree or clique tree in which each node is labeled with a set of variables
• Running intersection property: If variable $X$ appears in cliques $S$ and $T$, it must appear on every node along the path between them.
• Each potential function must be assigned to exactly one clique in the tree
• You can use variable elimination to generate the tree!
Example

\[ P(A,B,C,D,E) = \frac{1}{Z} \phi_{AB}(A,B) \phi_{BC}(B,C) \phi_{CD}(C,D) \phi_{DE}(D,E) \phi_{AE}(A,E) \]

1. Start with a Markov network
2. Triangulate, form clique graph
3. Form a tree and assign potentials
4. Run belief propagation!
   One twist: Only sum out variables not in target clique.
Junction Tree Savings

• Avoids redundancy in repeated variable elimination

• Need to build junction tree only once ever

• Need to repeat belief propagation only when new evidence is received
Loopy Belief Propagation

- Inference is efficient if graph is tree
- Inference cost is exponential in treewidth (size of largest clique in graph – 1)
- What if treewidth is too high?
- Solution: Do belief prop. on original graph
- May not converge, or converge to bad approx.
- In practice, often fast and good approximation
Markov networks: Different factorizations, same graph

\[ P(A,B,C) = \frac{1}{Z} \phi_{AB}(ABC) \]

\[ P(A,B,C) = \frac{1}{Z} \phi_{AB}(AB) \phi_{AC}(AC) \phi_{BC}(BC) \]

\[ P(A,B,C) = \frac{1}{Z} \phi_{ABC}(ABC) \phi_{AB}(AB) \phi_{C}(C) \]
Factor graphs:
Different factorizations, different graphs

\[ P(A,B,C) = \frac{1}{Z} \phi_{AB}(ABC) \]

\[ P(A,B,C) = \frac{1}{Z} \phi_{AB}(AB) \phi_{AC}(AC) \phi_{BC}(BC) \]

\[ P(A,B,C) = \frac{1}{Z} \phi_{ABC}(ABC) \phi_{AB}(AB) \phi_{C}(C) \]
A factor graph is a bipartite graph with a node for each random variable and each factor. There is an edge between a factor and each variable that participates in that factor.
Factor Graphs

A factor graph is a bipartite graph with a node for each random variable and each factor. There is an edge between a factor and each variable that participates in that factor.
Belief Propagation in Factor Graphs

Original rule for trees:

\[ m_{X \rightarrow Y}(Y) = \sum_x \phi_{XY}(x,Y) \prod_{W \in \text{neighbor}(X) \setminus Y} m_{W \rightarrow X}(x) \]

Break message passing into two steps:

1. Messages from variables to factors

\[ m_{x \rightarrow f}(X) = \prod_{h \in n(x) \setminus \{f\}} m_{h \rightarrow x}(X) \]

   Multiply incoming messages from all other neighboring factors.

2. Messages from factors to variables.

\[ m_{f \rightarrow x}(X) = \sum_{\sim \{x\}} \left( f(n(f)) \prod_{y \in n(f) \setminus \{x\}} m_{y \rightarrow f}(Y) \right) \]

   Multiply other incoming messages and sum out other variables.
Loopy Belief Propagation

• **Incorporate evidence:** For each evidence variable, select a potential that includes that variable and change the potential values to zero for everything that contradicts the evidence.

• **Initialize:** Set all messages to one

• **Run:** Pass messages from variables to factors, then factors to variables

• **Stop:** When messages change by very little, loopy belief propagation has converged (may not always happen!)