Lecture 14 - Inference, GraphEliminate; sum-product

today: inference is graphEliminate

sum-product alg.

graph elimination alg (for inference in generic VGM)

consider $p \in f(x)$, $p(x) = \frac{1}{Z} \prod_{C \in \mathcal{G}} \psi_C(x_C)$

undirected

say want to compute $p(x_F)$ for some $F \subseteq V$ "query nodes"

main trick: use distributivity of $\oplus$ over $\otimes$ \rightarrow $\otimes (\alpha \oplus \beta) = \otimes \alpha + \otimes \beta$

\[ x_1, x_2 \checkmark \]

\[ f(x_1) g(x_2) = \left( \frac{1}{x_1} f(x_1) \right) \left( \frac{1}{x_2} g(x_2) \right) \]

\[ \Rightarrow \]

more generally

\[ \frac{n}{x_1, n} \Rightarrow \frac{n}{x_1, n} \]

\[ 0(k^n n) \Rightarrow O(k^n n) \]

\[ x_i \text{ takes } k \text{ values} \]

\[ p(x_4) = \frac{1}{Z} \sum_{x_{1,2,3}} \psi(\overline{x_2}) \psi(x_0, x_4) \psi(\overline{x_1, x_2}) \psi(x_3, x_4) \]

\[ = \frac{1}{Z} \sum_{x_3} \psi(x_3, x_4) \left( \prod_{x_1} \psi(x_1, x_2, x_3) \right) \]

\[ \Rightarrow \]

\[ M_{1}(x_2, x_3) \text{ "message"} \]

\[ \Leftrightarrow \]

\[ M_{2}(x_3, x_4) \]

\[ = \frac{1}{Z} m_3(x_4) \]

\[ \frac{1}{Z} m_3(x_4) \Rightarrow \]

\[ \frac{1}{Z} m_3(x_4) = Z \]
general alg: graph Eliminate

init: a) choose an elimination ordering so F are the last nodes
   b) put all \( V_r(x_i) \) on "active list"

update: c) repeat, in order of variables to eliminate

1) (say \( x_i \) is variable to eliminate)
   remove all factors from active list with \( x_i \) in it \& take their product
   i.e. \( \prod_{a \in \text{set}_r \cap \text{rel}_a} V_{a,i}(x_a) \)

2) sum over \( x_i \) to get a new factor \( M_i(x_{S_i}) \) (think as \( \Psi_{S_i}(x_S) \))
   i.e. \( S_i \) are all variables in these factors except \( i \)
   get \( M_i(x_{S_i}) \leq \prod_{a \in \text{set}_r \cap \text{rel}_a} V_{a,i}(x_a) \)
   \( \Psi_i \triangleq (\prod_{a \in \text{set}_r \cap \text{rel}_a} V_{a,i}(x_a)) \), new clique to remove \( S_i \cup V_S, i \)

3) put back \( M_i(x_{S_i}) \) in active list \( \blacksquare \)

"normalise": d) last product of factors has only \( x_F \) \( \Rightarrow \) proportional \( p(x_F) \)

\[ \text{Suppose } x_i \in \{0,1\}, \text{ memory needed } \approx 2^{\max |S_i|} \]
\[ \text{computational } = (2^{\max |S_i| + 1}) \cdot n \]

\( \text{(max size of active list)} \)

\( \text{related to "treewidth" of a graph} \)

\( \text{"augmented graph" } \rightarrow \text{ graph obtained by running graph Eliminate + keeping track of all edges added} \)
"augmented graph" → graph obtained by running graph Eliminate + keeping track of all edges added for specific ordering

*note: augmented graph obtained after graph Eliminate is always a triangulated graph

 Definition: graph with no cycle of size 4 or more that cannot be broken by a chord

an edge between two non-neighborhood nodes in a cycle

A graph is not triangulated

A graph is a triangulated graph

Both memory and running time of graph Eliminate is dominated by the size of the largest clique, on augmented graph

Not all orderings are good

Bad news:

a) NP hard to compute treewidth of a general graph
   (or find best ordering)

b) NP hard to do (exact) inference in general U64

Example: approximate methods
Example 1: treewidth of a grid
\[ \approx \sqrt{1+1} \]

\[ \Rightarrow \text{approximate methods} \]

**Good News:***

* Inference is linear time for tree (treewidth=1) ("sum-product alg.")

\[ \Rightarrow \text{IV} + \text{LE} \] (HMM, Markov chain)

* Efficient for "small treewidth graph"

\[ \Rightarrow \text{use junction tree alg.} \]

15h39

**Inference on trees**

* Graph: Eliminate on a tree

\[ \Rightarrow \text{good order is to eliminate leaves first} \]

\[ F \]

\[ m_t \rightarrow \prod_{F} (x_t) \]

\[ m_t \rightarrow \prod_{F} (x_t) \]

\[ \rho(x) = \frac{1}{Z} \prod_{i} \sum_{x_i} \prod_{j \text{ child}(i)} \psi(x_i, x_j) \]

(assuming \( F \) is a given order. Make a directed tree by using \( x_F \) as a root)

**Sum-product alg. (for trees)**

Alg. to get all node/edge marginals cheaply by storing (caching) & re-using messages
alg. to get all node/edge marginals cleanly by storage (caching) & re-using messages (dynamic programming)

goal: \( i, j \in E \), compute \( M_{ij}(x_j) \)

\[ M_{ij}(x_j) = \sum_{x_i} \psi_i(x_i) \psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus \{j\}} m_{k \rightarrow i}(x_i) \]

Rule: \( i \) can only send message to neighbor \( j \) when it has received all messages from other neighbors

at end:

(node marginal)
\[ p(x_i) \propto \varphi_i(x_i) \prod_{j \in N(i)} M_{ij}(x_i) \]

normalize \( Z = \sum_{x_i} \)

(edge marginal)
\[ p(x_i, x_j) = \frac{1}{Z} \varphi_i(x_i) \varphi_j(x_j) \psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus \{j\}} m_{k \rightarrow i}(x_i) \prod_{k' \in N(j) \setminus \{i\}} m_{k' \rightarrow j}(x_j) \]

for this marginal need to use graph eliminate

sum-product schedules:

a) online, collect/distribute schedule
b) (floating) parallel schedule
1) initialize all $m_{i \rightarrow j}(x_j)$ messages to uniform dist $\mathcal{U}(\ell_{ij}, \ell_{ji})$ for all $i, j \in E$

2) at every step (in parallel) compute $m_{i \rightarrow j}^{\text{new}}(x_j)$ as if the neighbors were correctly computed

one can prove that after "diameter of tree" # of steps, all messages are correctly computed (for a tree) (and are fixed to a fixed point)

**Loopy belief propagation** (Loopy BP): approximate inference for graphs with cycles

\[
m_{j \rightarrow i}^{\text{new}}(x_i) = \prod_{x_j \neq x_i} m_{i \rightarrow j}(x_j) \prod_{k \in \text{nei}(i)} m_{k \rightarrow i}(x_i)
\]

\(\alpha \in [0, 1] \)  

"damping"

this gives exact answer on trees  
(fixed pt. \(\Rightarrow\) yields correct marginals)

\(\star\) on (not too large) graphs \(\Rightarrow\) approximate solution