Lecture 14 - scribbles

today: inference - graph eliminate
- sum-product

graph elimination alg. (for inference)

- consider p ∈ $\mathcal{G}$, \( p(x) = \prod_{c \in C} \psi_c(x_c) \)
  undirected

say want to compute \( p(x_F) \) for \( F \in V \) "query nodes"

main trick: use distributivity of \( \oplus \) over \( \odot \) → \( \odot(a \oplus b) = \odot a + \odot b \)

\[ \frac{\sum_{\alpha_1, \alpha_2} f(x_1) g(x_2)}{\sum_{\alpha_1} f(x_1)} = \left( \sum_{\alpha_1} \frac{g(x_2)}{g(x_2)} \right) \left( \sum_{\alpha_2} \frac{f(x_1)}{f(x_1)} \right) \]

- convince yourself

more generally: \( \frac{\sum_{\alpha_1, \alpha_2} f(x_1) g(x_2)}{\sum_{\alpha_1} f(x_1)} = \sum_{i=1}^n \frac{g(x_2)}{g(x_2)} \)

[factor out \( \sum_{\alpha_1} f(x_1) \)]

\[ p(x_4) = \frac{\sum_{\alpha_1, \alpha_2} \psi_1(x_1, x_2) \psi_2(x_1, x_3) \psi_3(x_1, x_4) \psi_4(x_2, x_3) \psi_5(x_3, x_4)}{\sum_{\alpha_1} \psi_1(x_1, x_2) \psi_2(x_1, x_3) \psi_3(x_1, x_4) \psi_4(x_2, x_3) \psi_5(x_3, x_4)} \]

\[ \leq \frac{\psi_4(x_3, x_4) \psi_1(x_1, x_2)}{m_4(x_2, x_3)} \]

\[ \leq \frac{\psi_4(x_3, x_4)}{m_4(x_2, x_3)} \]

\[ = \frac{1}{2} m_3(x_4) \]

- last message is proportional to marginal \( p(x_4) \)

\[ \leq m_3(x_4) = \frac{1}{2} \]

general alg: Graph Eliminate

init: \[ a) \] choose an elimination ordering \( x_F \) are the last nodes

\[ b) \] put all \( \psi_c(x_c) \) on "active list"

"update": \[ c) \] repeat in order of variables to eliminate
C) repeat in order of variables to eliminate
(say \( x_i \) is variable to eliminate)

1) remove all factors from active list with \( x_i \) in it & take product
   \[ \prod_{\alpha \in S_i} \varphi_{\alpha} \]

2) sum \( x_i \) to get a new factor \( m_i (x_{S_i}) \)
   \[ m_i (x_{S_i}) = \sum_{x_i} \prod_{\alpha \in S_i} \varphi_{\alpha} \]
   new clique to
   \[ S_i = \left( \bigcup_{\alpha \in S_i} \right) \text{set 1} \]

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

'normalize' d) last factor left has only \( x_F \) \( \Rightarrow \) proportional to \( p(x_F) \)

memory needed? \( \approx 2^{|S_i|} \cdot \text{(size factors)} \)

computational cost \( \approx 2^{|S_i|+1} \cdot \text{n} \)

Later, related 'treewidth' of a graph

δ "augmented graph" \( \rightarrow \) graph obtained by running graph eliminate + keeping track of all edges added

"augmented graph" after graph Eliminate is always a **triangulated graph**

Def.: graph with no cycle of size 4 or more that cannot be broken by a "chord"
**Tree width of a graph** \( \leq \min \text{ size of largest clique in } \text{ overall elimination ordering} \)

\[ \text{tree width} \text{ (tree)} = 1 \]

Both memory and running time of graph elimination is dominated by \( 2^{\text{tree width}} \).

Best ordering gives \( 2^{\text{tree width} + 1} \).

Some ordering is good:

- Not all orderings are good.

Bad news:

1. NP hard to compute tree width (or find best ordering).
2. NP hard to do (exact) inference in general CSM.

\( \Rightarrow \) Need approximate methods.

Example: Tree width of a grid

\[ \sim \sqrt{1/4} \]

\( \text{size of grid} \sim \text{tree width} \)

Good news:

* Inference is linear time for trees (tree width = 1) (**sum-product algo**).
* Efficient for "small tree width graph" (HMM, Markov chain).

Use **junction tree algo**
Inference on trees

**graph Eliminate on a tree**

*good order: eliminate leaves first*

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

*get all messages cheaply by storing (caching) & re-using messages (dynamic programming)*

**root**

*called first*

**distribute phase**

*goal: \( \exists i, j \in E \), compute \( m_{ij}(x_j) \)

*rule: I can only send message to neighbor J when it has received all messages from other neighbors*

\[
p(x_i) = \prod_{E \ni (x_i, x_j)} \frac{p(x_i, x_j)}{p(x_i, x_j)} \]

\[
m_{ij}(x_j) = \prod_{E \ni (x_i, x_j)} \frac{p(x_i, x_j)}{p(x_i, x_j)} \]

\[
m_{j \rightarrow i}(x_i) = \prod_{E \ni (x_i, x_j)} \frac{p(x_i, x_j)}{p(x_i, x_j)} \]

at end

\[
p(x_i) = \prod_{\text{neighbors}} m_{j \rightarrow i}(x_i) \frac{p(x_i|x_j)}{p(x_i|x_j)}
\]
At end (node marginal)

\[
P(z_i) \propto \sum_{x_i} \prod_{j \in \mathcal{N}(i)} M_j \cdot p_i(z_i) \cdot \psi_i(x_i)
\]

Normalisation \( Z = \sum_x \prod_i x_i \)

\[
p(z_i, z_j) = \frac{\prod_{i \in \mathcal{E}} \prod_{j \in \mathcal{N}(i)} M_j(z_i, z_j) \cdot p_i(z_i) \cdot \psi_i(x_i)}{\prod_{j \in \mathcal{E}} \prod_{k \in \mathcal{N}(j)} K_{j}(z_j, z_k) \cdot \psi_j(x_j)}
\]

Sum-product schedule:

a) above, distribute/collect schedule

b) (floating) parallel schedule:

1) initialise all \( M_{i \rightarrow j}(z_j) \) messages to uniform dist. \( \psi_i(x_i) \) \( \forall i, j \in \mathcal{E} \) \( \forall (i, j) \in \mathcal{E} \)

2) at every step (in parallel) compute \( M_{i \rightarrow j}(z_j) \)

as if the neighbor messages were already correctly computed

- can prove that after "length of the tree" number of steps, all messages are correctly computed for a tree (and can fixate pt.)

Loopy Belief Propagation

[Loopy BP]: approximate inference for graph with cycles.

\[
M_{i \rightarrow j}(z_j) = \left( \frac{M_{i \rightarrow j}(z_j) \prod_{z_i} p_{i}(z_i) \prod_{j \in \mathcal{N}(i)} \phi_{i}(x_i)}{\prod_{i \in \mathcal{E}} \prod_{j \in \mathcal{N}(j)} K_{j}(z_j, z_k) \cdot \psi_j(x_j)} \right)
\]
\( \alpha \in [0, 1] \) \text{ "damping"}

\( \times \) this gives exact answers on trees (fixed pt. yields correct marginals)

\( \times \) on (not loop) graphs \( \rightarrow \) approximate solution

\[
p(x_i | x_E) \propto p(x_i, x_E)
\]

Getting conditions:

- Initially we have conditioning on
- keep this fixed during marginalization

(formal trick): redefine \( \hat{\gamma}_j(x_j) \triangleq \gamma_j | x_j \), \( S(x_j, x_E) \)

\[
\text{Kronecker delta: } S(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}
\]

Computing \( M_j \rightarrow (x_i) \)

\[
\leq \frac{\hat{\gamma}_j(x_j)}{S}
\]

\[
= \frac{\gamma_j | x_j \text{ shield } (x_j, x_E)}{S}
\]

At the end, result of sum-product will give

\[
p(x_i, x_E) = \prod_{j \in V}(M_j \rightarrow (x_j))
\]

renormalize over \( x_i \)

\[
p(x_i | x_E)
\]