Lecture 14 - scribbles
Friday, October 19, 2018 13:24

Inference on trees

- Graph: Eliminate on a tree
  - Good order: Eliminate leaves first

\[ p(x) = \frac{1}{Z} \prod_i \psi_i(x_i) \prod_{ij} \psi_{ij}(x_i, x_j) \]

Order: Make a directed tree by using \( x_F \) as a root

- Start from leaves; go up towards root

\[ m_i \rightarrow_j (x_j) = \sum_{x_i} \psi_i(x_i) \psi_{ij}(x_i, x_j) \prod_{k \in \text{children}(i)} m_k \rightarrow_i (x_i) \]
**Sum-product algo. (for trees)**

- Get all marginals cheaply by storing (caching) & re-using messages (dynamic programming).

**Root**

- "Collect phase"  
- "Distribute phase"

**Goal:** For all $i, j \in E$, compute $m_{i \rightarrow j}(x_j)$

- $m_{i \rightarrow j}(x_j) = \sum_{i}^{\gamma_i(x_i)} \gamma_{ij}(x_i, x_j) \prod_{k \in \text{children}(i)} m_{k \rightarrow i}(x_i)$

**Rule:** $i$ can only send message to neighbor $j$

- When it has received all messages from other neighbors.

$K \subseteq \text{children}(i)$

$\exists j \in K$
at end: \( p(x_i) \propto \prod_{j \in \mathcal{N}(i)} m_j \psi_i(x_i) \)

normalization \[ \sum_{x_i} \]

(energy marginal)
\[
p(x_i, z_j) = \frac{\prod_i \psi_i(x_i) \prod_j \psi_j(x_i, z_j) \prod_{k \in \mathcal{N}(i)} m_k \psi_i(x_i)}{\sum_{x_i, z_j} \prod_{k \in \mathcal{N}(i)} m_k \psi_i(x_i) \prod_{k \in \mathcal{N}(i)} m_k \psi_j(x_i, z_j)}
\]

sum-product schedules

a) above, distribute/collect schedule

b) (flooding) parallel schedule:
   1) initialize \( m_{i \rightarrow j}(z_j) \) messages to uniform dist. \( \psi(z_j) \) s.t. \( \sum_{z_j} \psi(z_j) = 1 \)
   2) at every step (in parallel) compute \( m_{i \rightarrow j}(z_j)^{\text{new}} \)

as if the neighbor messages were already correctly computed

→ can prove that after "diameter of the tree" # of steps, all messages are correctly computed

for a tree (and all fixed points)
Loopy Belief Propagation (loopy BP): approximate inference

\[ m_{i \rightarrow j}^{(n+1)}(x_j) = \alpha \sum_{x_i} m_{i \rightarrow j}^{(n)}(x_j) \prod_{k \neq i \neq j} \frac{m_{k \rightarrow i}(x_i)}{\sum_{x_i} m_{k \rightarrow i}(x_i)} \]

\[ \alpha \in [0,1] \quad \text{"damping"} \]

- This gives exact ansuere on tree (fixed pt. yields correct marginals)
- on (not too loopy) graphs \(\rightarrow\) approximate solution

getting conditionals:

\[ p(x_i | \overline{x_E}) \propto p(x_i, \overline{x_E}) \]

indicate values we are conditioning on

keep this fixed during marginalization

for each \( j \in E \): (formal trick) \(\hat{\Psi}_j(x_j) \equiv \Psi_j(x_j \mid \overline{x_j}) \cdot S(x_j, \overline{x_j}) \)
computing \( M_j \rightarrow \delta (x_i) \),

\[
\underbrace{\sum_{x_j}^{\sim} \psi_j (x_j)}_{x_j} \cdot \text{shuff}(x_j, x_i)
\]

\[
= \sum_{x_j}^{\sim} \psi_j (x_j) \cdot \text{shuff}(x_j, x_i)
\]

at the end, result of sum-product

will give

\[
p(x_i | \overline{x}_E) = \frac{1}{Z} \prod_{i \in E} \prod_{k \in M_j} \psi_j (x_i)
\]

\[
p(x_i | \overline{x}_E) \; \text{renormalize over } x_i
\]

\( (4h3) \)

\text{Max-product algorithm:}

\text{for sum-product, main property used was distributivity of } \oplus \text{ over } \odot

\( (\mathbb{R}, +, \odot) \) is a semi-ring

\( b \) don't need additive inverse

can do "sum-product" on other semi rings
\[(\mathbb{R}, \max, \oplus) \quad \max(a \oplus b, a \oplus c) = a \oplus \max(b, c)\]
\[(\mathbb{R}_+, \max, \odot) \quad \max(a \odot b, a \odot c) = a \odot \max(b, c)\]

"max-product"

\[
\max_{x_i \in i} \left( \prod_{j \in J_i} \lambda_{ij}(x_i, x_j) \right) = \prod_{j \in J_i} \max_{x_i} \left( \lambda_{ij}(x_i, x_j) \right)
\]

\[M_{i \rightarrow j}(x_j) = \max_{x_i} \left[ \prod_{j \in J_i} \lambda_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus j} \lambda_{ki}(x_i) \right]
\]

For getting argmax, store argument of this max as feat. of \(x_j\)

To get argmax \(Q(x_1:n)\) "decoding"

- run max-product algo (only forward messages)
- backtrack the argmax pointers to get full argmax

Aka, viterbi algorithm

Property: \(p \in \mathcal{F}(\text{tree})\) with non-zero marginals

\[p(x) = \prod_{i \in V} p(x_i) \prod_{\text{parents}(x_i)} \frac{p(x_i; x_j)}{p(x_i)}\]
with non-zero marginals

(proof: simple exercise)

similar to DGMs for any set of factors $\exists f_{ij}(x_i, x_j) \neq 0$.

"local consistency property"

\[
\begin{align*}
\forall x_i \quad f_i(x_i) &= f_i(x_i) \\
\forall x_i, x_j \quad f_{ij}(x_i, x_j) &= f_{ij}(x_i, x_j) \\
\forall x_i \quad f_i(x_i) &= 1
\end{align*}
\]

Then if define joint (for tree)

\[
p(x) = \prod_i f_i(x_i) \prod_{ij} \frac{f_{ij}(x_i, x_j)}{f_i(x_i) f_j(x_j)}
\]

we get correct marginals i.e. $p(x_i) = f_i(x_i)$ etc...

Junction tree algo:

generalization of sum-product to a clique tree

[Diagram of a junction tree with nodes A, B, C, and D, and edges connecting them.]
above is a clique tree with the "running intersection property"

"junction tree"

to build JT: use max weighted spanning tree algo. (with size of separator sets as weight on clique graph from a D-graph)

JT \rightarrow \text{triangulated graph / decomposable and running graph / Eliminate}

when have JT, you can show

\[
p(x_v) = \frac{\prod p(x_c)}{\sum_{\mathcal{S}} p(x_S)}
\]

\[
\forall x_S, \text{ separator set in any JT}
\]

JT tree alg.: reconstruct the above formulation

\[
p(x_v) = \frac{\prod p(x_c)}{\sum_{\mathcal{S}} p(x_S)}
\]

where \( s(x_S) = 1 \) at beginning

do message passing to update new and old p(x_c)