Lecture 14 - scribbles
Friday, October 20, 2017
13:31

Today:
- Max-product
- HMM

**Sum-product schedules**

(a) before, we saw distribute/collect schedule

(b) (flooding) parallel schedule:

1) initialize $m_{i \rightarrow j}(x_j)$ messages to uniform dist. $\forall i,j \ s.t. \ E(i,j) \in E$

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

$\Rightarrow$ can prove that after "diameter of the tree" steps, all messages are correctly computed for a tree

**Loopy Belief Propagation** (loopy BP): approximate inference

$$m_{i \rightarrow j}^{\text{new}}(x_j) = \left( m_{i \rightarrow j}^{\text{old}}(x_j) \right) \left( \prod_{i \in N(j) \setminus \{i\}} m_{k \rightarrow i}^{\text{old}}(x_i) \right)$$

$\alpha \in [0,1]$ "damping"

This gives exact answer on tree (fixed point is exact)
this gives exact answer on tree (fund point is exact)
on (not too large) graphs, \(\Rightarrow\) approximate soln

\[ p(x_i | \bar{x}_E) \times p(x_i, \bar{x}_E) \]

indicate values we are conditioning on

keep it fixed during product

(formal trick)

redesign \( \tilde{\psi}_j(x_j) = \psi_j(x_j) \cdot \delta(x_j, \bar{x}_j) \) for \( j \in E \)

Kronecker delta function \( \delta(a, b) = 1 \) if \( a = b \),
\( 0 \) o.w.

\( \text{M}_x \) on \( x_i \)

\[ \implies \text{will have} \quad \sum_{\bar{x}_j} \tilde{\psi}_j(x_j) \cdot \text{stuff}(\bar{x}_j, x_i) \]
\[ = \psi_j(x_j) \cdot \text{stuff}(\bar{x}_j, x_i) \]

at end of day, result of sum-product
will give
\[ p(x_i, \bar{x}_E) = \frac{1}{Z} \sum_{x_i} \psi_i(x_i) \prod_{k \in \Gamma(i) \setminus E} \text{M}_{x_i} (x_i) \]
\[ p(x_i | \bar{x}_E) \text{ renorm} \]

Max-product alg

for sum-product, main property used was distributivity of \( \circ \) over \( \oplus \)
\((\mathbb{R}, \circ, \circ)\) is semi-ring

\[\text{L: don't need additive unity}\]

can do "sum-product" on other semi-rings

\((\mathbb{R}, \max, \oplus)\)

\[\max(\alpha \odot b, \alpha \odot c) = \alpha \oplus \max(b, c)\]

\((\mathbb{R}^+, \max, \odot)\)

\[\max(\alpha \odot b, \alpha \odot c) = \alpha \odot \max(b, c)\]

\(\Rightarrow \text{"max-product"}\)

\[\max \bigg( \bigg\{ \sum_i f_i(x_i) \bigg\} \bigg) = \bigg\{ \max_{x_i} \bigg[ \sum_i f_i(x_i) \bigg] \bigg\} \bigg\}

\[\Rightarrow \text{arg-max, store argument of this max as function of } x_i\]

\[\text{max-product alg aka Viterbi alg}\]

\[\Rightarrow \text{"decoding" argmax } P(x_{i:n})\]

**property:**

\(p \in \mathcal{L}(\text{tree})\) with non-zero marginals

\[\Rightarrow P(x) = \bigg( \sum_{y \in \text{tree}} P(y) \bigg) \bigg( \sum_{z \in \text{tree}} P(z) \bigg) \cdot P(x)\]
if \( p \in \mathcal{P}(\text{standard tree}) \) 
\[ \Rightarrow p \in \mathcal{P}(\text{any orientation of tree}) \]

\[ p(z) = \prod_{i} p_{x_i}(x_{ni}) \text{ for some orientation of tree} \]

\[ = \left( \prod_{i} \frac{p(x_{ni} \mid x_{ni})}{p(z_{ni})} \right) p(z_{ni}) \]

\[ = \left( \prod_{i} \frac{p(x_{ni} \mid x_{ni})}{p(z_{ni})} p(z_{ni}) \right) \]

\[ = \prod_{z_{ni}} \left( \prod_{i} \frac{p(x_{ni} \mid x_{ni})}{p(z_{ni})} p(z_{ni}) \right) \]

as before, for any set of factors \( \xi_{ij}(x_{ni}, x_{nj}), f_{ij}(x_{ni}) \) \( \xi_{ij} \geq 0, f_{ij} \geq 0 \)

such that \( \sum_{x_{ni}} \xi_{ij}(x_{ni}, x_{nj}) = \xi_{ij}(x_{ni}) \) for all \( x_{ni} \)

subject to

\[ \sum_{x_{ni}} \xi_{ij}(x_{ni}, x_{nj}) = \xi_{ij}(x_{ni}) \]

\[ \frac{\xi_{ij}(x_{ni}, x_{nj})}{\xi_{ij}(x_{ni})} = 1 \]

Then it defines a joint \( p(z) = \prod_{i} \xi_{ij}(x_{ni}) \prod_{i} \frac{f_{ij}(x_{ni})}{\xi_{ij}(x_{ni})} \)

We get correct marginals i.e., \( p(x_{ni}) = \xi_{ij}(x_{ni}) \) etc...

\textbf{Junction treealg} & \textbf{generalization of sum-product to a clique tree}
Junction tree deg.

![Graph](image)

A generalization of semi-product to a clique tree

This is a clique tree with the "running intersection property" "junction tree"

To build JT, use max weighted spanning tree with size of separator sets as weight

on clique graph from A-graph

\[ \mathcal{J}T \Leftrightarrow \text{triangulated graph/decomposable graph} \Leftrightarrow \text{running graph/eliminate} \]

When have JT, we can show

\[
\begin{align*}
p(x_v) &= \frac{\prod_{C \in \text{separator sets}} p(x_C)}{\prod_{C} p(x_C)} \\
&= \frac{\prod_{C} p(x_C)}{\prod_{S} p(x_S)} \\
&= \frac{\prod_{S} p(x_S)}{\prod_{S} \mathcal{R}_S(x_S)}
\end{align*}
\]

\[
\mathcal{J}T \text{ log. } p(x_v) = \frac{1}{2} \frac{\prod_{S} \mathcal{R}_S(x_S)}{\prod_{S} \mathcal{R}_S(x_S)} \quad \text{where } \mathcal{R}_S(x_S) = 1 \text{ at beginning}
\]

HMM: (hidden Markov model)

\[ Z_1, Z_2, \ldots \]
$x_t \in \mathbb{E}_t, \ldots, k_t$ discrete

$x_t \rightarrow c_t \rightarrow o_t$, speech signal

discrete \rightarrow DNA sequence

speech recognition

$x_t \rightarrow$ speech signal

$z_t \rightarrow$ phonemes

HMM \rightarrow generalization of mixture model

$\text{GMM}$

$\text{GMM}$

$\text{GMM}$

$DGMs$:

$p(x_{1:T} \mid z_{1:T}) = \prod_{t=1}^{T} \left( \frac{1}{Z_t} \sum_{z_t} p(z_t \mid z_{t-1}) \right)$

Often, the emission prob and trans. probs are homogeneous, i.e., do not depend on $t$

$p_t(z_t \mid z_{t-1}) = \sum_{z_t} p(z_t \mid z_{t-1})$

$p_t(z_t = i \mid z_{t-1} = j) = A_{ij}$

$A \left( \begin{bmatrix} \text{dist. over} \ z_t \end{bmatrix} \right)$

"Stochastic matrix"

$\sum_{i} A_{ij} = 1$
prediction \( p(Z_t \mid x_{1:t-1}) \) "where next?"

falling \( p(Z_t \mid x_{1:t}) \) "where now?"

smoothing \( p(Z_t \mid x_{1:T}) \) "where in the past?"

Let's run some

product here

\[
p(Z_t, x_{1:t}) = \frac{1}{Z} \cdot M_{z_{t-1} \rightarrow z_t}(z_t) \cdot M_{z_{t:} \rightarrow x_t}(z_t) \\
\text{here } Z = 1
\]

\[
m_{z_{t-1} \rightarrow z_t}(z_t) = \sum_{z_t} p(x_t \mid z_t) \delta(x_t, z_t) = p(x_t \mid z_t)
\]

\[
m_{z_{t:} \rightarrow x_t}(z_t) = \sum_{z_t} p(z_t \mid z_{t-1}) M_{z_{t-1} \rightarrow z_t}(z_{t-1}) M_{z_{t-1} \rightarrow z_t}(z_{t-1}) p(z_{t-1} \mid x_{1:t-1})
\]

define: \( \alpha_t(z_t) \triangleq p(z_t, x_{1:t}) \)

\[
\alpha_t(z_t) = p(x_t \mid z_t) \sum_{z_{t-1}} p(z_t \mid z_{t-1}) \alpha_{t-1}(z_{t-1})
\]
\[ \alpha_t(z_t) = \frac{p(z_t | z_{t-1}) \prod_{z_{t-1}} p(z_{t-1}) \alpha_{t-1}(z_{t-1})}{\text{vector} \times \text{matrix} \times \text{vector}} \]

\( \alpha_t(z_t) \) is the \( \alpha \)-recursion or a forward recursion. Like the "recognition phase" in sum-product algo with \( z_t \) as the test.

\[ \alpha_t(z_t) = p(z_t, z_{t-1}) = p(z_t) p(z_{t-1}) \]  
"filtering"

Time complexity: \( O(t \cdot k^2) \)
Space complexity: \( O(k) \) extra for storing \( A \)

\[ \sum_{z_t} \alpha_t(z_t) = p(z_{t+1}) \]  "evidence prob."