Lecture 15 - scribbles

Tuesday, October 29, 2019 14:30

today: max-product algo.
- junction tree

max-product algo:

for sum-product, main property used was distributivity of $+$ over $\cdot$

all need $\varepsilon: (\mathbb{R}, +, \cdot)$ is a semi-ring

$\Rightarrow$ don't need additive noise

$\Rightarrow$ can do "sum-product" on other semi-rings

$(\mathbb{R}, \max, +)$: $\max (a + b, a + c) = a + \max (b, c)$

$(\mathbb{R}, \max, \cdot)$: $\max (a \cdot b, a \cdot c) = a \cdot \max (b, c)$

$\Rightarrow$ "max-product"

$max \max_i \{ f_i (x_i) \} = \max_i \{ \max_j \{ f_j (z_i) \} \}$

$m_{i,j} (x_j) = \max_{x_j} \{ \sum_{x_i} m_i (x_i) \cdot \eta_j (x_i, x_j) \}$

$\Rightarrow$ for getting argmax, choose argument of this max as a set of $x_j$

$\Rightarrow$ to get argmax $p (x_{1:n})$, "decoding"

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

aka: Viterbi algorithm

property of tree USM
property of tree UGM

\[ p \in S(\text{free}) \quad \text{with non-zero marginals} \quad \Rightarrow \quad p(x) = \prod_{i \in v} p(x_i) \prod_{i,j \in E} \frac{p(x_i, x_j)}{p(x_i)p(x_j)} \]

(proof: simple exercise)

\[ \sum_{x_j} S_{ij}(x_i, x_j) S_j(x_j) \geq S_i(x_i) \]

\[ \text{local consistency (property)} \]

\[ \begin{align*}
& \sum_{x_j} S_{ij}(x_i, x_j) = S_i(x_i) \quad \forall x_i \\
& \sum_{x_i} S_{ij}(x_i, x_j) = S_j(x_j) \quad \forall x_j \\
& \sum_{x_i} S_i(x_i) = 1
\end{align*} \]

then if define joint

\[ p(x) = \prod_{i} S_i(x_i) \prod_{i,j \in E} \frac{S_i(x_i)S_j(x_j)}{S_i(x_i)S_j(x_j)} \]

then we get constant marginals i.e. \( p(x) = S_i(x_i) \)

\[ \text{generalization: sum-product to a clique tree} \]

\[ \text{with JT property} \]


Junction tree algorithm

generalization: sum-product to a clique tree

(with JT property)

Diagram showing a clique tree with the "running intersection" property.

To build a JT: use max-weight spanning tree e.g.
on a \( A \)-graph

on a clique graph (with size of separator sets as
weight on the edges)

\[ FJT \leftarrow \text{triangulated graph/} \quad \alpha \rightarrow \text{running graph} \]

\[ \text{decomposable graph} \]
when have J.T., one can show

\[ p(x_r) = \frac{\sum_T p(x_c)}{\sum_T p(x_s)} \]

\[ \text{separators set in the J.T.} \]

J.T. tree alg.: reconstruct the above formulation

by starting with

\[ p(x_r) = \frac{1}{Z} \frac{\prod_T p(x_c)}{\sum_T p(x_s)} \]

where \( p(x_r) = 1 \) at ley.

Do message passing on J.T. to update \( \pi^\text{new}_c \) at the end:

\[ \begin{aligned}
& \pi^\text{new}_c \\
& \pi^\text{old}_c \\
& p(x_c) \\
& p(x_s)
\end{aligned} \]

15h35

[\text{Hidden Markov model}]

\[ Z_1 \rightarrow Z_2 \rightarrow \ldots \rightarrow Z_t \rightarrow \ldots \rightarrow Z_T \]

\[ x_t \in \Omega, \ldots, x_T \text{ discrete} \]

\[ x_t \sim \text{dist}, \quad \text{e.g. speech signal} \]

\[ Z_t \sim \text{dist}, \quad \text{e.g. HMM sequence} \]

\[ \text{Emission prob} \]

\[ \text{Transition prob} \]

\[ \text{HMM = generalization of mixture model} \]

\[ GMM \]

\[ \begin{array}{c}
\begin{aligned}
& \text{add dependency on } Z_t \\
& \text{on } Z_t
\end{aligned}
\end{array} \]

\[ \text{DCM} : \quad p(x_r, x_t, z_{1:t}) = p(z_t) \prod_{t=1}^T \left( \frac{p(x_t | z_t)}{\sum_T p(x_t | z_T)} \right) \prod_{t=2}^T \left( \frac{p(z_t | z_{t-1})}{p(z_t | z_{t-1})} \right) \]

often, emission prob \& transition prob are homogenous in time (i.e. do not depend on \( t \))

\[ p(x_t | z_t) = \xi(x_t | z_t) \]

\[ \xi \]
\[
\begin{align*}
p_t(X_t | x_t) &= \mathcal{N}(X_t | x_t, \Sigma) \\
p_t(Z_t = i | Z_{t-1} = j) &= A_{ij}
\end{align*}
\]

Inference tasks:
- prediction \( p(Z_t | x_{1:t-1}) \) "where next?"
- filtering \( p(Z_t | x_{1:t}) \) "where now?"
- smoothing \( p(Z_t | x_{1:T}) \) "where in the past?"

\( T > t \)

\( \alpha \)-recursion:

Let's run sum-product here to derive recursions to compute \( \alpha_t \).

\[
\alpha_t(z_t) = \frac{1}{Z} \prod_{z_{t-1}} p(z_{t-1} | z_t) \mathcal{M}_{z_{t-1}}(z_t) \quad \text{(here } t = 1) \]

\[
\mathcal{M}_{z_{t-1}}(z_t) = \sum_{z_{t-1}} p(z_{t-1}) \delta(z_t | z_{t-1}) = p(z_t | z_{t-1})
\]

\[
\mathcal{M}_{z_{t-1}}(z_t) = \sum_{z_{t-1}} p(z_{t-1}) \mathcal{M}_{z_{t-2}}(z_{t-1}) \mathcal{M}_{z_{t-1}}(z_{t-1}) m_{z_{t-1}, z_t}(z_{t-1}) m_{z_{t-1}, z_t}(z_{t-1})
\]

\[
\alpha_t(z_t) = p(z_{t-1} | Z_{t-1}) \mathcal{M}_{z_{t-1}}(z_t) \quad \text{vector}[z_t] \quad \text{matrix} \quad \text{vector}
\]

\( \alpha_t(z_t) \) is also called "forward recursion" like the "collect phase" in sum-product.
let \( \alpha_t(z_t) \triangleq p(x_t \mid z_t) \)

\[
\alpha_t = \alpha_0 \circ (A \alpha_{t-1})
\]

```
backward product
```

\[
\alpha_t(z_t) = p(z_t \mid x_{1:t}) \quad \text{"forward distribution"}
\]

**time complexity:** \( O(tk^2) \)

**space complexity:** \( O(k) \) extra storage

\[
\sum_{z_t} \frac{p(z_t, \bar{x}_{1:t})}{q_t(z_t)} = \frac{p(\bar{x}_{1:t})}{q_t(z_t)} \quad \text{"evidence probability"}
\]

**\( \beta \)-recursion (smoothing)**

\[
p(z_t, \bar{x}_{1:T}) = \frac{1}{Z} \frac{p(z_t \mid x_t)}{q_t(z_t)} m_{z_{t+1} \rightarrow z_t}(z_t) = \beta_t(z_t)
\]

\[
m_{z_{t+1} \rightarrow z_t}(z_t) = \sum_{z_{t+1}} p(z_{t+1} \mid z_t) p(x_{t+1} \mid z_{t+1}) m_{z_{t+2} \rightarrow z_{t+1}}(z_{t+1})
\]

\[
\beta_t(z_t) = \sum_{z_{t+1}} p(z_{t+1} \mid z_t) p(x_{t+1} \mid z_{t+1}) \beta_{t+1}(z_{t+1})
\]

**\( \beta \)-recursion (aka backward recursion)**

```
turns out that \( \beta_t(z_t) \triangleq p(\bar{x}_{t+1:T} \mid z_t) \)
```

**initialization:** \( \beta_t(z_t) = 1 \quad \forall z_t \)

\[
\beta_t(z_t) = p(\bar{x}_{t+1:T} \mid z_t)
\]

**Smoothing works**

\[
p(\bar{x}_t, z_t) = p(x_t \mid z_t) p(z_t)
\]

\[
(\forall z_t) \quad \beta_t(z_t) = p(z_t)
\]

\[
(\forall z_t) \quad \beta_t(z_t) = p(z_t)
\]
edge marginal:

\[ p(z_t, z_{t+1}, x_{1:T}) = \frac{A_t(z_t)}{A_{t+1}(z_{t+1})} \frac{p(z_{t+1} | z_t)}{p(x_{t+1} | z_{t+1})} \]