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:

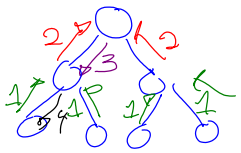
1) initialize $m_{i \to j}(x_j)$ messages to uniform dist. $\forall (i,j)$ s.t. $\{i,j\} \in E$

2) at every step (in parallel) compute $m^{new}_{i \to j}(x_j)$

as if neighbor messages were correctly computed

→ can prove that often "diameter of the tree" steps, all messages are correctly computed for a tree

↳ since longest path in the tree



Loopy Belief Propagation (loopy BP): approximate inference

$$m^{new}_{i \to j}(x_j) = \left(m^{old}_{i \to j}(x_j)\right)^{\alpha} \left(\sum_{x_i} \psi(x_i)\psi_{ij}(x_i,x_j) \prod_{k \in N(i) \setminus \{j\}} m^{old}_{k \to i}(x_i)\right)^{1-\alpha}$$

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

· this gives exact answer on tree (fixed point is exact)

- this gives exact answer on tree (fixed point is exact)
- on (not too loopy) graphs, → approximate soln

getting conditionals:

indicate values we are conditioning on

we have $p(x_i \mid \bar{x}_E) \propto p(x_i, \bar{x}_E)$

keep it fixed during product

(formal trick):

redefine $\tilde{\psi}_j(x_j) \triangleq \psi_j(x_j) \cdot \delta(x_j, \bar{x}_j)$ for $j \in E$

Kronecker-Delta function $\delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{o.w.} \end{cases}$

$m_{j \to i}(x_i)$

→ will have $\sum_{x_j} \tilde{\psi}_j(x_j) \cdot \text{stuff}(x_j, x_i)$

$= \psi_j(\bar{x}_j) \cdot \text{stuff}(\bar{x}_j, x_i)$

at end of day, result of sum-product will give $p(x_i, \bar{x}_E) = \frac{1}{Z} \psi_i(x_i) \prod_{k \in N(i)} m_{k \to i}(x_i)$

$p(x_i \mid \bar{x}_E)$ ▷ renormalize

Max-product alg:

for sum-product, main property used was distributivity of $\oplus$ over $\otimes$

$(\mathbb{R}, \oplus, \odot)$ is semi-ring

↳ don't need additive inverse

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

$(\mathbb{R}, \max, \oplus)$ $\quad \max(a \oplus b, a \oplus c) = a \oplus \max(b, c)$

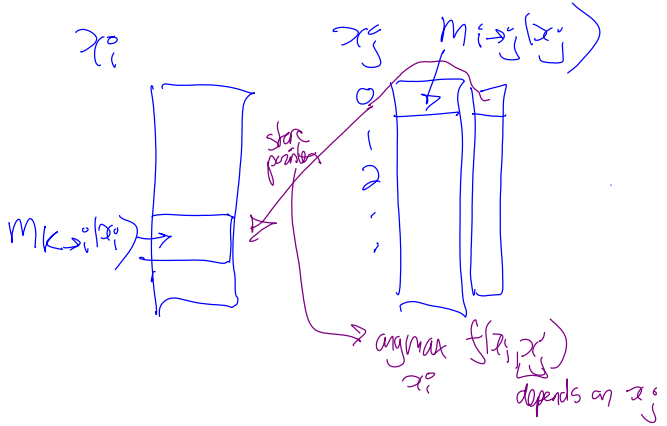
$(\mathbb{R}_+, \max, \odot)$ $\quad \max(a \odot b, a \odot c) = a \odot \max(b, c)$

↳ "max-product"

$$\max_{x_{1:n}} \prod_i f_i(x_i) = \prod_i \max_{x_i} f_i(x_i)$$

$$m_{i \to j}(x_j) = \max_{x_i} \left[\psi_i(x_i) \psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus \{j\}} m_{k \to i}(x_i) \right]$$

↳ for argmax, store argument of this max as function of x_j



max-product alg aka Viterbi alg

→ "decoding" $\operatorname{argmax}_{x_{1:n}} p(x_{1:n})$

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

$$p(x) = \prod_i p(x_i) \prod_{i,j \in E} \frac{p(x_i, x_j)}{p(x_i) p(x_j)}$$

proof:

Fact: if $p \in \mathcal{F}$ (undirected tree)

$\Rightarrow p \in \mathcal{F}$ (any orientation of tree)

$\Rightarrow p(x) = \prod_i p(x_i | x_{\pi_i})$ for some orientation of tree

$$= \left(\prod_{i<n} p(x_i | x_{\pi_i})\right) p(x_n) \quad (p(x_n|\emptyset))$$

$$= \left(\prod_{i<n} \frac{p(x_i, x_{\pi_i}) p(x_i)}{p(x_{\pi_i}) p(x_i)}\right) p(x_n)$$

$$= \prod_i p(x_i) \prod_{\{i,j\} \in E} \frac{p(x_i, x_j)}{p(x_i) p(x_j)}$$

as before, for any set of factors $\{f_{ij}(x_i, x_j), f_i(x_i)\}$ $f_{ij} \geq 0$ $f_i \geq 0$

s.t. ("local consistency property")

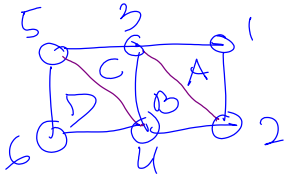
$$\begin{cases} \sum_{x_j} f_{ij}(x_i, x_j) = f_i(x_i) & \forall x_i \\ \sum_{x_i} f_{ij}(x_i, x_j) = f_j(x_j) \\ \sum_{x_i} f_i(x_i) = 1 \end{cases}$$

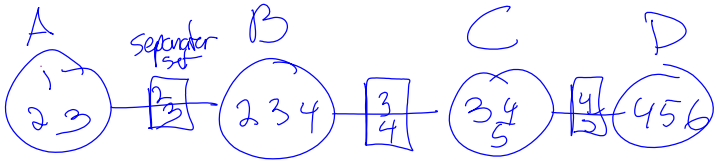
then if define a joint $p(x) = \prod_i f_i(x_i) \prod_{\{i,j\} \in E} \frac{f_{ij}(x_i, x_j)}{f_i(x_i) f_j(x_j)}$

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

Junction tree alg: generalization of sum-product to a clique tree

Junction tree alg: generalization of sum-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

JT ⟷ triangulated graph / decomposable graph ⟷ running graph Eliminate

when have JT, you can show

$$p(x_V) = \frac{\prod_C p(x_C)}{\prod_S p(x_S)}$$

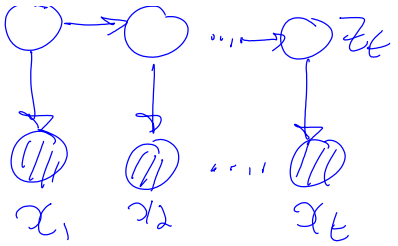
S = separator sets

JT alg. $p(x_V) = \frac{1}{Z} \frac{\prod_C \psi_C(x_C)}{\prod_S \varphi_S(x_S)}$ where $\varphi_S(x_S) = 1$ at beginning

---

HMM: (hidden Markov model)

z_1 z_2 ...



$z_t \in \{1, \dots, k\}$ discrete (later $z_t \sim$ Gaussian in class → Kalman filter)

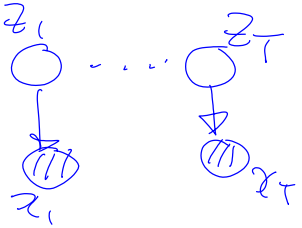
x_t — cts → e.g. speech signal
discrete → e.g. DNA sequence

speech recognition: x_t → speech signal, z_t → phonemes

HMM → generalization of mixture model

GMM 

add temporal dep. on z_t



DGM:

$$p(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=1}^{T} p(x_t|z_t) \prod_{t=2}^{T} p(z_t|z_{t-1})$$

$p(x_t|z_t)$: emission prob.; $p(z_t|z_{t-1})$: transition prob.

often, the emission prob and tran. prob are homogeneous i.e. do not depend on t

$$p_t(x_t|z_t) = f(x_t|z_t)$$

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

"stochastic matrix"

A (j ▯ dist. over z_t)

$$\sum_i A_{ij} = 1$$

Inference tasks

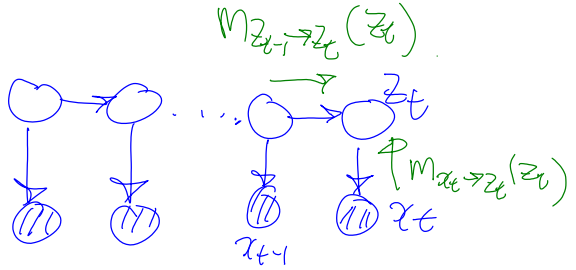
inference tasks:

prediction $p(z_t \mid x_{1:t-1})$ "where next"?

filtering $p(z_t \mid x_{1:t})$ "where now?"

smoothing $p(z_t \mid x_{1:T})$, $T > t$ "where in the past?"

let's run sum product here



to compute $p(z_t, \bar{x}_{1:t})$

$$\underbrace{p(z_t, \bar{x}_{1:t})}_{q_t(z_t)} = \frac{1}{Z} 1 \cdot m_{z_{t-1} \to z_t}(z_t) \cdot m_{x_t \to z_t}(z_t) \qquad \text{here } Z = 1$$

$$m_{x_t \to z_t}(z_t) = \sum_{x_t} p(x_t \mid z_t)\, \delta(x_t, \bar{x}_t) = p(\bar{x}_t \mid z_t)$$

$$m_{z_{t-1} \to z_t}(z_t) = \sum_{z_{t-1}} p(z_t \mid z_{t-1}) \underbrace{m_{z_{t-2} \to z_{t-1}}(z_{t-1})\, m_{x_{t-1} \to z_{t-1}}(z_{t-1})}_{p(z_{t-1}, \bar{x}_{1:t-1})}$$

$$\underbrace{p(z_{t-1}, \bar{x}_{1:t-1})}_{\alpha_{t-1}(z_{t-1})}$$

define: $\alpha_t(z_t) \triangleq p(z_t, \bar{x}_{1:t})$

$$\alpha_t(z_t) = \underbrace{p(\bar{x}_t \mid z_t)}_{\text{vector}} \sum_{z_{t-1}} \underbrace{p(z_t \mid z_{t-1})}_{\text{matrix}} \underbrace{\alpha_{t-1}(z_{t-1})}_{\text{vector}}$$

Hadamard product

$$\alpha_t(z_t) = p(\bar{x}_t \mid z_t) \sum_{z_{t-1}} p(z_t \mid z_{t-1}) \alpha_{t-1}(z_{t-1})$$

vector — Hadamard product — matrix — vector

α-recursion aka forward recursion

like the "collect phase" in sum-product alg with z_t as the root

$$\alpha_1(z_1) = p(z_1, \bar{x}_1) = p(z_1)\, p(\bar{x}_1 \mid z_1)$$

← "filtering"

time complexity: $O(t \cdot K^2)$

space complexity: $O(K)$ extra for α storage ($O(K^2)$ for storing A)

$$\sum_{z_t} \alpha_t(z_t) = p(\bar{x}_{1:t})$$

"evidence prob."