Today: max-product alg.,
- junction tree
- HMM

Getting started

\( p(x_i; x_E) \) or \( p(x_i; \overline{x_E}) \)

I keep this fixed during marginalization for each \( j \in E \)

(formal trick): redefine \( \tilde{\psi}_j(x_j) = \psi_j(x_j) \cdot \delta(x_j; \bar{x_j}) \)

\( \delta(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases} \)

computing \( M_{j \rightarrow i}(x_i) = \sum_{x_j} \tilde{\psi}_j(x_j) \text{ stuff } (x_i, x_j) \)

\( \rightarrow \psi_j(x_j) \text{ stuff } (x_i, x_j) \)

at the end, result of sum-product will give \( p(x_i; \bar{x_E}) = \prod_{k \in E_i} \tilde{\psi}_k(x_k) \)

renormalize over \( x_i \) to get \( p(x_i; \bar{x_E}) \)

max-product alg.

For sum-product, main property used was distributivity of \( \oplus \) over \( \odot \)

all need is that \( (\mathbb{R}, \oplus, \odot) \) is a semi-ring:

\[ a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c) \]

\( \odot \) can do "sum-product" on other semi-rings
(a) Can do "sum-product" on some semi-rings

\[(\mathbb{R} \cup \mathbb{R}^+, \max, +) \quad \max(a \circ b, a \circ c) = a + \max(b, c)\]

\[(\mathbb{R}^+, \max, \cdot) \quad \max(a \cdot c, b \cdot c) = a \cdot \max(b, c)\]

"max-product"

\[\max_i \prod_{i=1}^n f_i(x_i) = \prod_{i=1}^n \max_i f_i(x_i)\]

\[m_{i \rightarrow j}(x_j) = \max_{x_i} \left\{ \phi_i(x_i) \cdot \psi_{ij}(x_i, x_j) \prod_{k \in \mathcal{M}(i) \setminus \{j\}} m_{k \rightarrow i}(x_i) \right\}\]

- For getting argmax

Store argument of this max as a set of \(x_i\)

\[\max_{x_1, x_5} \prod_{k \in \mathcal{M}(x_5 \setminus \{x_1\})} \left[ m_{4 \rightarrow 5}(x_5) \cdot \psi_5(x_5) \right]\]

- To get argmax \(p(x_{1:n})\) "decoding":
  - Run max-product algo
  - Only forward messages
  - Backtrack the argmax pointers to get full argmax

Aka. Viterbi algo.

Property of tree WGM

\[p \in \mathcal{G}(\text{tree})\]

with non-zero marginals

\[p(x) = \prod_{i \in \mathcal{V}} p(x_i) \prod_{(i,j) \in \mathcal{E}} p(x_i, x_j) \prod_{i \in \mathcal{V}} p(x_i | x_{\text{pa}(i)}) \prod_{i \in \mathcal{V}} p(x_i)\]
\[ p(x_i | x_j) \]
\[ p(x_i) \]
\[ \phi \text{ local consistency property} \]
\[ \begin{align*}
    \begin{cases}
    \exists f_{ij}(x_i, x_j) = f_{ij} & \forall x_j \\
    \exists f_i(x_i) = f_i(x_i) & \forall x_i \\
    \exists f_j(x_i, x_j) = f_{ij}(x_j) & \forall x_j
    \end{cases}
\end{align*} \]

Then if define joint \( p(x) = \frac{1}{Z} \prod_{i} f_i(x_i) \prod_{j \in \mathcal{F}} f_{ij}(x_j) \).

(for tree) \( \rightarrow \) then can show that \( Z = 1 \)

and \( \prod_{i} f_i(x_i) \) are correct marginals.

eg. \( p(x_i) = \hat{f}_i(x_i) \) etc.

**Junction tree alg.:** generalization of sum-product to a **clique tree** (with J.T. property)

![Diagram of a junction tree](image)

Above is a clique tree with the "running intersection property" or "junction tree" property:

- If \( j \in C_1 \cap C_2 \), then \( j \in \mathcal{T}_1 \cap \mathcal{T}_2 \) along path from \( i \) to \( C_2 \)

To build a J.T. on a \( \Delta \)-graph:
- Use maximum weight spanning tree alg. on clique graph (with size of separator set as weight on edges)
  \( \Rightarrow \) has running intersection property
- A J.T. is triangulated/decomposable
  \( \Rightarrow \) a running intersection
If a J.T. is triangulated/decomposable graph

when have J.T., one can show

\[ P(x_v) = \sum_{C \in S} \frac{P(x_C)}{P(x_S)} \]

\[ S \subseteq \text{separator sets of J.T.} \]

J.T. alg: reconstruct the chain formulation

by starting with \( P(x_v) = \frac{1}{Z} \sum_{C \in S} \Psi_C(x_C) \)

where \( Z(C, x_0) = 1 \) at initialization

do message passing on J.T. to update

\[ \Psi_C \rightarrow P(x_C) \]

(see Miller's book ch. 17 for details)

---

**HMM** (Hidden Markov model)

\[ \begin{array}{cccccc}
Z_1 & Z_2 & \ldots & Z_k \\
X_1 & X_2 & \ldots & X_k \\
\end{array} \]

\( Z \in \{ 1, \ldots, k \} \) discrete

\( X \in \{ \text{ch. e.g. speech sound words} \} \)

**GMM**

HMM → generalization of mixture model

\[ \begin{array}{cccccc}
Z_1 & Z_2 & \ldots \end{array} \]

\[ \begin{array}{cccccc}
\psi_1 & \psi_2 & \ldots \end{array} \]

\[ DGM; \quad P(x_{1:T}, Z_{1:T}) = P(z_1) \prod_{t=1}^{T} P(x_t, z_t) \prod_{t=1}^{T} P(z_t | z_{t-1}) \]
DGM: \( p(x_{1:t}, z_{1:T}) = \prod_{t=1}^{T} p(x_t | x_{t-1}) \prod_{t=2}^{T} p(z_t | z_{t-1}) \)

Often, emission & transition prob are **homogeneous in time** (i.e. do not depend on \( t \))

\[ p_e(z_t | z_{t-1}) = f(z_{t-1}, z_t) \]

\[ p_e(z_t = i | z_{t-1} = j) = A_{ij} \]

(formal matrix)

**Inference tasks**: prediction \( p(z_{t+1} | x_{1:t}) \) "where next?"

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

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

\( T > t \)

**A-recurrsion**

Let's run sum-product algo to derive recursion to compute \( p(z_t | x_{1:t}) \)

\( p(z_t | x_{1:t}) = \frac{1}{Z} \prod_{k=1}^{t} \frac{\alpha_k(z_k) M_{z_k \rightarrow z_t}(z_t)}{\alpha_t(z_t)} \) (here \( Z = 1 \))

\( m_{z_k \rightarrow z_t}(z_t) = \sum_{z_k} p(z_k | z_{t-1}) S(x_t, z_t) = p(x_t | z_t) \)

\( m_{z_t \rightarrow z_k}(z_k) = \sum_{z_t} p(z_t | z_{t-1}) S(x_t, z_t) = p(x_t | z_t) \)

\( m_{z_t \rightarrow z_{t-1}}(z_{t-1}) = \sum_{z_t} p(z_t | z_{t-1}) S(x_t, z_t) = p(x_t | z_t) \)

\( \alpha_t(z_t) = \frac{p(x_t | z_t) \sum_{z_{t-1}} \frac{p(z_t | z_{t-1}) m_{z_{t-1} \rightarrow z_t}(z_t)}{\alpha_{t-1}(z_{t-1})}}{\alpha_{t-1}(z_{t-1})} \)
$\alpha^t(\mathbf{z}_t) = \frac{p(\mathbf{z}_t | \mathbf{z}_{t-1})}{\alpha_{t-1}(\mathbf{z}_{t-1})} \alpha_{t-1}(\mathbf{z}_{t-1})$

$\alpha$-recovery aka "forward recovery" like the "collect-phase" in sum-product Alg.

**Initialization:** $\alpha^1(\mathbf{z}_1) = p(\mathbf{z}_1, \bar{\mathbf{x}}_1) = p(\bar{\mathbf{x}}_1 | \mathbf{z}_1) p(\mathbf{z}_1)$

Let $\alpha^t(\mathbf{z}_t) \triangleq p(\mathbf{z}_t | \mathbf{z}_{1:t})$

$\alpha^t = \alpha^t \odot (A \alpha^{t-1})$

Hardwood product

$\alpha^t(\mathbf{z}_t) \triangleq p(\mathbf{z}_t | \bar{\mathbf{x}}_{t:t})$ "filtering distribution"

$= \frac{\alpha^t(\mathbf{z}_t)}{\sum_{\mathbf{z}_{1:t}} \alpha^t(\mathbf{z}_{1:t})} \leq p(\mathbf{z}_{t:t} | \bar{\mathbf{x}}_{t:t})$

"and then prob.

Space complexity: $O(tk^2)$ extra storage

Time complexity: $O(tk^2)$