Graphical Models

Lecture 16:

Maximum a Posteriori Inference

Andrew McCallum
mccallum@cs.umass.edu

Thanks to Noah Smith and Carlos Guestrin for some slide materials.

Probabilistic Inference

- Assume we are given a graphical model.
- Want:

$$
\begin{aligned}
P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e}) &= \frac{P(\boldsymbol{X}, \boldsymbol{E} = \boldsymbol{e})}{P(\boldsymbol{E} = \boldsymbol{e})} \\
&\propto P(\boldsymbol{X}, \boldsymbol{E} = \boldsymbol{e}) \\
&= \sum_{\boldsymbol{y} \in \mathrm{Val}(\boldsymbol{Y})} P(\boldsymbol{X}, \boldsymbol{E} = \boldsymbol{e}, \boldsymbol{Y} = \boldsymbol{y})
\end{aligned}
$$

Lecture 9

Inference: Where We Have Been

9. Variable elimination
10. Variable elimination, continued
11. Clique trees, sum-product message passing, calibration
12. Sum-product-divide (belief update) message passing

exact

13. Mean field variational inference
14. Cluster graphs, generalized loopy belief propagation
15. Sampling, Monte Carlo Markov chain

approximate

Probabilistic Inference: MAP

- Sometimes we are interested primarily in what is *most probable*:

$$\boldsymbol{x}^* \quad = \quad \arg \max_{\boldsymbol{x} \in \mathrm{Val}(\boldsymbol{X})} P(\boldsymbol{X} = \boldsymbol{x})$$

  – A single, coherent explanation.
  – "Decoding" metaphor
  – Note that constant factors do not matter, so unnormalized probabilities are okay!

$$\boldsymbol{x}^* \quad = \quad \arg \max_{\boldsymbol{x} \in \mathrm{Val}(\boldsymbol{X})} U(\boldsymbol{X} = \boldsymbol{x})$$

  – Evidence?

MAP Inference

- NP-hard in general.
- Sometimes called "max-product" problems:

$$\boldsymbol{x}^* = \arg \max_{\boldsymbol{x} \in \mathrm{Val}(\boldsymbol{X})} P(\boldsymbol{X} = \boldsymbol{x}) = \arg \max_{\boldsymbol{x} \in \mathrm{Val}(\boldsymbol{X})} \prod_{\phi_i \in \boldsymbol{\Phi}} \phi_i(\boldsymbol{x}_i)$$

Can also be understood as "max-sum" or "min-sum" (energy minimization):

$$= \arg \max_{\boldsymbol{x} \in \mathrm{Val}(\boldsymbol{X})} \sum_{\phi_i \in \boldsymbol{\Phi}} \log \phi(\boldsymbol{X}) = \arg \min_{\boldsymbol{x} \in \mathrm{Val}(\boldsymbol{X})} \sum_{\phi_i \in \boldsymbol{\Phi}} - \log \phi_i(\boldsymbol{x}_i)$$

Marginal MAP (A Generalization)

$$\begin{aligned} \boldsymbol{y}^* &= \arg \max_{\boldsymbol{y} \in \mathrm{Val}(\boldsymbol{Y})} P(\boldsymbol{Y} = \boldsymbol{y}) \\ &= \arg \max_{\boldsymbol{y} \in \mathrm{Val}(\boldsymbol{Y})} \sum_{\boldsymbol{z} \in \mathrm{Val}(\boldsymbol{X} \setminus \boldsymbol{Y})} P(\boldsymbol{X} = \langle \boldsymbol{y}, \boldsymbol{z} \rangle) \end{aligned}$$

- Find the most probable configuration of *some* random variables, marginalizing out others.
- Includes the case with evidence.
- Involves a max, a sum, and a product (hard).
  - Marginal MAP is in $\mathrm{NP}^{\mathrm{PP}}$ (contains the entire polynomial hierarchy, of which NP is only the first level).

Max-Marginals

- A set of factors useful in intermediate steps of MAP inference algorithms.
- Let f : Val(**X**) → $\mathbb{R}$
- The max-marginal of f relative to variables **Y**⊆**X** is:

$$\forall \boldsymbol{y} \in \mathrm{Val}(\boldsymbol{Y}), \qquad \max_{\boldsymbol{z} \in \mathrm{Val}(\boldsymbol{X} \setminus \boldsymbol{Y})} f(\langle \boldsymbol{y}, \boldsymbol{z} \rangle)$$

  - Example: f = U, so that the max-marginal gives the unnormalized probability of the most likely configuration consistent with each **y**.

Exact MAP Inference

Products of Factors

- Given two factors with different scopes, we can calculate a new factor equal to their products.

$$\phi_{product}(\boldsymbol{x} \cup \boldsymbol{y}) \quad = \quad \phi_1(\boldsymbol{x}) \cdot \phi_2(\boldsymbol{y})$$

Lecture 9

Factor Marginalization

- Given **X** and Y (Y $\notin$ **X**), we can turn a factor ϕ(**X**, Y) into a factor ψ(**X**) via marginalization:

$$\psi(\boldsymbol{X}) = \sum_{y \in \mathrm{Val}(Y)} \phi(\boldsymbol{X}, y)$$

- We can refer to this new factor by $\sum_Y \phi$.

Lecture 9

Factor Maximization

- Given $\mathbf{X}$ and Y (Y $\notin$ $\mathbf{X}$), we can turn a factor $\phi(\mathbf{X}, Y)$ into a factor $\psi(\mathbf{X})$ via maximization:

$$\psi(\boldsymbol{X}) \quad = \quad \max_{Y} \phi(\boldsymbol{X}, Y)$$

- We can refer to this new factor by $\max_Y \phi$.

Factor Maximization

- Given **X** and Y (Y $\notin$ **X**), we can turn a factor ϕ(**X**, Y) into a factor ψ(**X**) via maximization:

$$\psi(\boldsymbol{X}) \quad = \quad \max_{Y} \phi(\boldsymbol{X}, Y)$$

A	B	C	ϕ (A, B, C)
0	0	0	0.9
0	0	1	0.3
0	1	0	1.1
0	1	1	1.7
1	0	0	0.4
1	0	1	0.7
1	1	0	1.1
1	1	1	0.2

"maximizing out" B

A	C	ψ(A, C)
0	0	1.1
0	1	1.7
1	0	1.1
1	1	0.7

Distributive Property

- A useful property we exploited in variable elimination:

$$X \notin \mathrm{Scope}(\phi_1) \quad \Rightarrow \quad \sum_X (\phi_1 \cdot \phi_2) = \phi_1 \cdot \sum_X \phi_2$$

- Under the same conditions, factor multiplication distributes over max, too:

$$\max_X (\phi_1 \cdot \phi_2) = \phi_1 \cdot \max_X \phi_2$$

Max-Product Variable Elimination

- Exactly like before, with two changes:
  - Replace sum with max
  - Traceback to recover the most likely assignment

Eliminating One Variable (Sum-Product Version)

Input: Set of factors $\mathbf{\Phi}$, variable Z to eliminate

Output: new set of factors $\mathbf{\Psi}$

1. Let $\mathbf{\Phi}' = \{\phi \in \mathbf{\Phi} \mid Z \in \text{Scope}(\phi)\}$
2. Let $\mathbf{\Psi} = \{\phi \in \mathbf{\Phi} \mid Z \notin \text{Scope}(\phi)\}$
3. Let τ be $\sum_Z \prod_{\phi \in \mathbf{\Phi}'} \phi$
4. Return $\mathbf{\Psi} \cup \{\tau\}$

Lecture 9

Eliminating One Variable (Max-Product Version)

Input: Set of factors $\mathbf{\Phi}$, variable Z to eliminate

Output: new set of factors $\mathbf{\Psi}$

1. Let $\mathbf{\Phi}' = \{\phi \in \mathbf{\Phi} \mid Z \in \text{Scope}(\phi)\}$
2. Let $\mathbf{\Psi} = \{\phi \in \mathbf{\Phi} \mid Z \notin \text{Scope}(\phi)\}$
3. Let τ be $\max_Z \prod_{\phi \in \mathbf{\Phi}'} \phi$
4. Return $\mathbf{\Psi} \cup \{\tau\}$

Variable Elimination (Sum-Product Version)

Input: Set of factors $\mathbf{\Phi}$, ordered list of variables $\mathbf{Z}$ to eliminate

Output: new factor

1. For each $Z_i \in \mathbf{Z}$ (in order):
   – Let $\mathbf{\Phi}$ = Eliminate-One($\mathbf{\Phi}$, Z_i)
2. Return $\prod_{\phi \in \mathbf{\Phi}} \phi$

   (unnormalized marginal probabilities of remaining variables)

Lecture 9

Variable Elimination (Max-Product Version)

Input: Set of factors $\mathbf{\Phi}$, ordered list of variables $\mathbf{Z}$ to eliminate

Output: new factor

1. For each $Z_i \in \mathbf{Z}$ (in order):
   – Let $\mathbf{\Phi}$ = Eliminate-One($\mathbf{\Phi}$, Z_i)
2. Return $\prod_{\phi \in \mathbf{\Phi}} \phi$

   (unnormalized max-marginal probabilities of remaining variables)

Recovering the MAP Assignment

- Need to "trace back" and find values for all of the variables that were eliminated.
  - Requires us to remember the intermediate factors.
- Connection to dynamic programming: you do not know the "answer" until you have completed the process; your intermediate calculations let you recover the answer *at the end*.

Eliminating One Variable (Max-Product Version with Bookkeeping)

Input: Set of factors $\mathbf{\Phi}$, variable Z to eliminate

Output: new set of factors $\mathbf{\Psi}$

1. Let $\mathbf{\Phi'} = \{\phi \in \mathbf{\Phi} \mid Z \in \text{Scope}(\phi)\}$
2. Let $\mathbf{\Psi} = \{\phi \in \mathbf{\Phi} \mid Z \notin \text{Scope}(\phi)\}$
3. Let τ be $\max_Z \prod_{\phi \in \mathbf{\Phi'}} \phi$
   – Let ψ be $\prod_{\phi \in \mathbf{\Phi'}} \phi$ (bookkeeping)
4. Return $\mathbf{\Psi} \cup \{\tau\}, \psi$

Variable Elimination (Max-Product Version with Decoding)

Input: Set of factors $\mathbf{\Phi}$, ordered list of variables $\mathbf{Z}$ to eliminate

Output: new factor

1. For each $Z_i \in \mathbf{Z}$ (in order):
   - Let $(\mathbf{\Phi}, \psi_{Z_i})$ = Eliminate-One$(\mathbf{\Phi}, Z_i)$
2. Return $\prod_{\phi \in \mathbf{\Phi}} \phi$, Traceback$(\{\psi_{Z_i}\})$

Traceback

Input: Sequence of factors with associated variables: $(\psi_{Z_1}, \ldots, \psi_{Z_k})$

Output: $\mathbf{z}^*$

- Each ψ_Z is a factor with scope including Z and variables eliminated *after* Z.
- Work backwards from $i = k$ to 1:
  - Let $z_i = \arg\max_z \psi_{Z_i}(z, z_{i+1}, z_{i+2}, \ldots, z_k)$
- Return $\mathbf{z}$

About the Traceback

- No extra (asymptotic) expense.
  - Linear traversal over the intermediate factors.
- The factor operations for both sum-product VE and max-product VE can be generalized.
  - Example: get the K most likely assignments

Variable Elimination for Marginal MAP

$$\begin{aligned}
\boldsymbol{y}^* &= \arg \max_{\boldsymbol{y} \in \mathrm{Val}(\boldsymbol{Y})} P(\boldsymbol{Y} = \boldsymbol{y}) \\
&= \arg \max_{\boldsymbol{y} \in \mathrm{Val}(\boldsymbol{Y})} \sum_{\boldsymbol{z} \in \mathrm{Val}(\boldsymbol{X} \setminus \boldsymbol{Y})} P(\boldsymbol{X} = \langle \boldsymbol{y}, \boldsymbol{z} \rangle)
\end{aligned}$$

- Use sum-product to marginalize out **X \ Y**.
- Use max-product to maximize over **Y**.
- For correctness, we must sum all variables in **X \ Y** first, *before* maximizing over **Y**.
  - Restricts the variable elimination ordering; effects on runtime?

Clique Trees and Max-Product

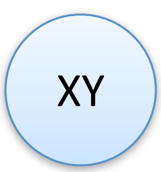
- Recall that, after discussing VE, we reinterpreted it as message passing in clique trees.
- We can do the same thing here.
  - Passing "max messages" instead of sum messages.
  - Upward/downward passes
  - Max-calibration: $\max_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i = \max_{\boldsymbol{C}_j \setminus \boldsymbol{S}_{i,j}} \beta_j = \mu_{i,j}(\boldsymbol{S}_{i,j})$
  - Re-parameterization and invariant
  - Max-product and max-product-divide

Clique Trees and Max-Product

- How to decode?
- Choose value of each random variable based on local beliefs?

Clique Trees and Max-Product

- How to decode?
- Choose value of each random variable based on local beliefs?
  - No! Might give an inconsistent assignment with overall low probability.
  - Example: P(X, Y) = 0.1 if X = Y, 0.4 otherwise.



0	0	0.1
0	1	0.4
1	0	0.4
1	1	0.1

max-marginal for X:

0	0.4
1	0.4

max-marginal for Y:

0	0.4
1	0.4

Clique Trees and Max-Product

- How to decode?
- Choose value of each random variable based on local beliefs?
  - This is okay if the calibrated node beliefs are *unambiguous* (no ties).

Clique Trees and Max-Product

- Local optimality of a (complete) configuration:

$$\boldsymbol{x}[\boldsymbol{C}_i] \in \arg\max_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i)$$

- Local optimality is satisfied for all clique tree node beliefs if and only if **x** is globally optimal (global MAP configuration).
  - Use a traceback to get a consistent assignment that is locally optimal everywhere.

Exact MAP

- Sometimes you can do it.
- Often, the structure of your problem gives you a specialized algorithm.
  - Examples I have seen: dynamic programming (really just VE); maximum weighted bipartite matching, minimum spanning tree, max flow, …

Approximate MAP Inference

Approximate MAP Inference

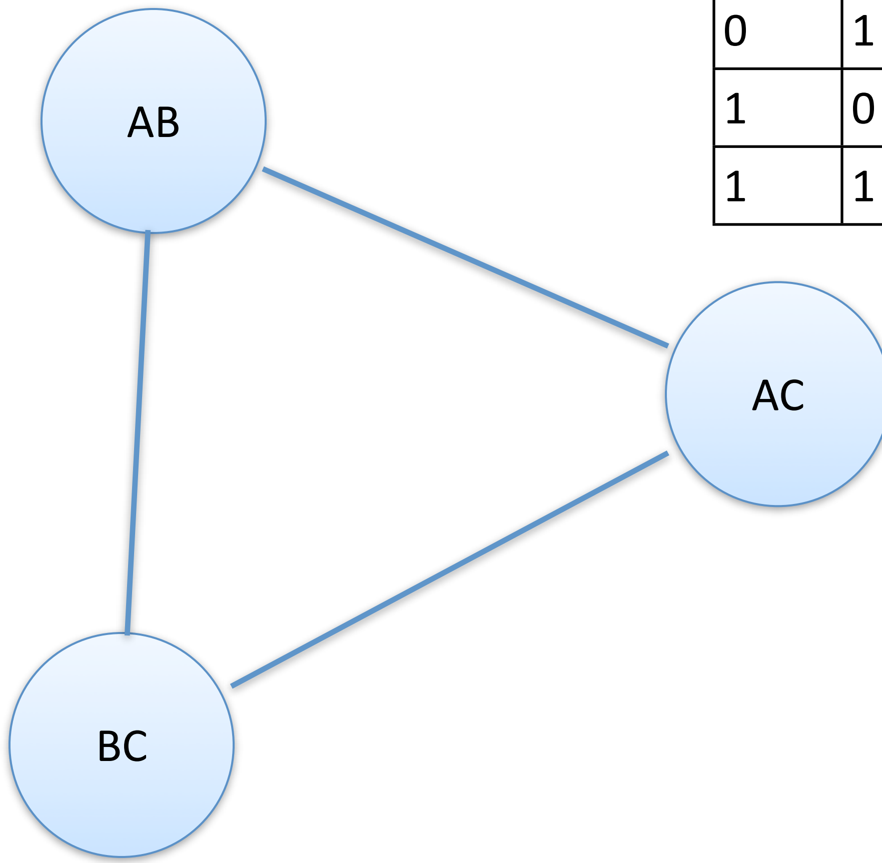
- Huge topic, getting a lot of attention.
- Key techniques:
  - Max-product belief propagation in loopy cluster graphs
  - Linear programming formulations

Max-Product Belief Propagation in Loopy Cluster Graphs

- Exactly the same, only use a max instead of a sum when calculating the messages.
- No guarantees of convergence.
  - Anecdotally, seems to converge less often than sum-product.
  - Calibration at convergence: **pseudo-max-marginals**.
  - How to decode?

Frustrated Loops

0	0	1
0	1	2
1	0	2
1	1	1



0	0	1
0	1	2
1	0	2
1	1	1

0	0	1
0	1	2
1	0	2
1	1	1

Max-Product Belief Propagation in Loopy Cluster Graphs: Decoding

- When all node beliefs are unambiguous (no ties), there is a unique maximizing assignment to the local clusters that is consistent.
- It's possible to have ambiguous node beliefs and a locally optimal joint assignment!
- In general, finding the locally optimal assignments that are consistent is a **constraint satisfaction problem**.
  - NP hard.

MAP as Optimization

- We got some traction out of treating marginal inference as optimization (lecture 15 on mean field variational inference).
- We can do the same thing for MAP inference.
  - Special cases for exact inference I mentioned earlier.
  - General formulation: **integer linear programming**.

Linear Objective

- For each factor ϕ_r with scope $\mathbf{C}_r$, and for each value of its random variables $\mathbf{c}$, let there be a free variable

$$z_{r,\mathbf{c}} = 1 \text{ iff } \mathbf{C}_r = \mathbf{c}, 0 \text{ otherwise}$$

- One binary variable* for each row of each factor.
- Optimization problem:

$$\max_{\{z_{r,\boldsymbol{c}}\}} \prod_r \prod_{\boldsymbol{c} \in \mathrm{Val}(\boldsymbol{C}_r)} \phi_r(\boldsymbol{c})^{z_{r,\boldsymbol{c}}} \quad = \quad \max_{\mathbf{z}} \mathbf{z}^\top \boldsymbol{\eta}$$

*Do not confuse with the random variables!

Constraints

- Each $z_{r,\mathbf{c}}$ must be in {0, 1}.
  - Integer constraints.
- Exactly one of the $\mathbf{z}_r$ is equal to 1.
  - Linear constraints.
- Factors must agree on their shared variables.
  - Linear constraints; see assignment 5.

Integer Linear Programming

- Optimizing a linear function with respect to a set of integer-valued variables (perhaps with linear constraints) is called an **integer linear programming** problem.
  – NP-hard in general.
  – Some special cases can be solved efficiently.
  – There are some really good solvers for ILPs that make this not as scary as it used to be.

Relaxation

- Relaxing the integer constraints from {0, 1} to [0, 1] has useful effects:
  - ILP becomes an LP; solvable in polynomial time.
  - Feasible region of the LP is a polytope.
  - Solve the relaxed LP; if solution is integer, you are done. If not, go greedy, randomized rounding, etc.
- Can add more constraints to the LP, perhaps getting a better approximation.

General Solvers

- General solvers are always tempting, but algorithms that "know" about the special structure of your problem are usually faster and/or more accurate.
- My advice: formulate the problem first, understand the landscape of specialized optimization techniques that might apply, and resort to general techniques if you can't find anything.
  - And be on the lookout for ways to improve the general technique using your problem's structure!

Final Note

- Finding the best consistent configuration is an *old* problem; old solutions exist.
  - Branch and bound, A*
  - Local search methods (e.g., beam search, tabu)
  - Randomized methods (e.g., simulated annealing)
- Some of the above can be better understood or generalized using data structures developed for inference (e.g., clique trees and cluster graphs).