Junction Tree, BP and Variational Methods

Adrian Weller

MLSALT4 Lecture
Feb 21, 2018

With thanks to David Sontag (MIT) and Tony Jebara (Columbia)
for use of many slides and illustrations

For more information, see
http://mlg.eng.cam.ac.uk/adrian/

High level overview of our 3 lectures

- 1. Directed and undirected graphical models (last Wed)
- 2. LP relaxations for MAP inference (last Friday)
- 3. Junction tree algorithm for exact inference, belief propagation, variational methods for approximate inference (today)

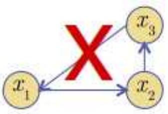
Further reading / viewing:

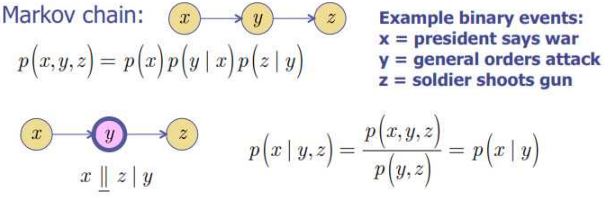
- Murphy, Machine Learning: a Probabilistic Perspective
- Barber, Bayesian Reasoning and Machine Learning
- Bishop, Pattern Recognition and Machine Learning
- Koller and Friedman, Probabilistic Graphical Models
  https://www.coursera.org/course/pgm
- Wainwright and Jordan, Graphical Models, Exponential Families, and Variational Inference

Review: directed graphical models = Bayesian networks

- A **Bayesian network** is specified by a directed *acyclic* graph DAG$= (V, \vec{E})$ with:
  1. One node $i \in V$ for each random variable X_i
  2. One conditional probability distribution (CPD) per node, $p(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$, specifying the variable's probability conditioned on its parents' values
- The DAG corresponds 1-1 with a particular factorization of the joint distribution:

$$p(x_1, \ldots x_n) = \prod_{i \in V} p(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$$





Review: undirected graphical models = MRFs

- As for directed models, we have one node for each random variable
- Rather than CPDs, we specify (non-negative) **potential functions** over sets of variables associated with (maximal) cliques C of the graph,

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

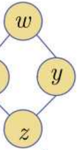
  Z is the **partition function** and normalizes the distribution:

$$Z = \sum_{\hat{x}_1, \ldots, \hat{x}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

- Like a CPD, $\phi_c(\mathbf{x}_c)$ can be represented as a table, but it is not normalized
- For both directed and undirected models, the joint probability is the product of sub-functions of (small) subsets of variables

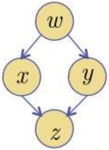
Directed and undirected models are different

• Example:



$x \perp\!\!\!\perp y \mid \{w, z\}$

$w \perp\!\!\!\perp z \mid \{x, y\}$

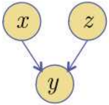


Directed can't do it!
Must be acyclic
Will have at least one V structure and ball goes through

$x \perp\!\!\!\perp y \mid \{w\}$

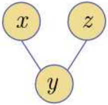
$x \not\perp\!\!\!\perp y \mid \{w, z\}$

• Example:



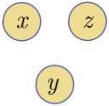
$x \perp\!\!\!\perp z$

$x \not\perp\!\!\!\perp z \mid y$



With <3 edges,
Undirected can't do it!

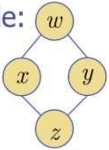
$x \perp\!\!\!\perp z \mid y$



$x \perp\!\!\!\perp z$

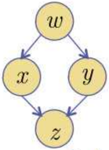
Directed and undirected models are different

- Example:



$x \perp\!\!\!\perp y \mid \{w, z\}$

$w \perp\!\!\!\perp z \mid \{x, y\}$

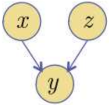


$x \perp\!\!\!\perp y \mid \{w\}$

$x \not\!\perp\!\!\!\perp y \mid \{w, z\}$

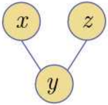
Directed can't do it!
Must be acyclic
Will have at least one V structure and ball goes through

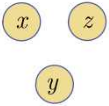
- Example:



$x \perp\!\!\!\perp z$

$x \not\!\perp\!\!\!\perp z \mid y$





With <3 edges,
Undirected can't do it!

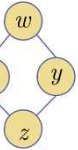
$x \perp\!\!\!\perp z \mid y$

$x \perp\!\!\!\perp z$

$$p(x, y, z) = p(x)p(z)p(y|x, z)$$

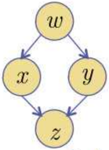
Directed and undirected models are different

- Example:



$x \perp\!\!\!\perp y \mid \{w, z\}$

$w \perp\!\!\!\perp z \mid \{x, y\}$

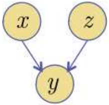


$x \perp\!\!\!\perp y \mid \{w\}$

$x \not\perp\!\!\!\perp y \mid \{w, z\}$

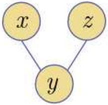
Directed can't do it!
Must be acyclic
Will have at least one V structure and ball goes through

- Example:



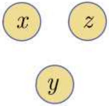
$x \perp\!\!\!\perp z$

$x \not\perp\!\!\!\perp z \mid y$



With <3 edges,
Undirected can't do it!

$x \perp\!\!\!\perp z \mid y$



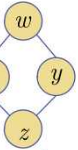
$x \perp\!\!\!\perp z$

$$p(x, y, z) = p(x)p(z)p(y|x, z) =: \phi_c(x, y, z), \; c = \{x, y, z\}$$

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

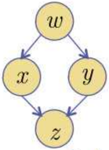
Directed and undirected models are different

- Example:



$x \perp\!\!\!\perp y \mid \{w, z\}$

$w \perp\!\!\!\perp z \mid \{x, y\}$

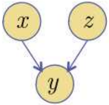


$x \perp\!\!\!\perp y \mid \{w\}$

$x \not\perp\!\!\!\perp y \mid \{w, z\}$

Directed can't do it!
Must be acyclic
Will have at least one V structure and ball goes through

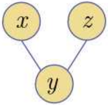
- Example:



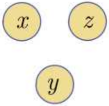
$x \perp\!\!\!\perp z$

$x \not\perp\!\!\!\perp z \mid y$

With <3 edges,
Undirected can't do it!



$x \perp\!\!\!\perp z \mid y$



$x \perp\!\!\!\perp z$

$$p(x, y, z) = p(x)p(z)p(y|x, z) =: \phi_c(x, y, z), \;\; c = \{x, y, z\}$$

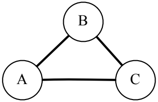
$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

What if we double ϕ_c?

Undirected graphical models / factor graphs

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c), \qquad Z = \sum_{\hat{x}_1, \ldots, \hat{x}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

Simple example (each edge potential function encourages its variables to take the same value):



$\phi_{A,B}(a,b) =$

A \ B	0	1
0	10	1
1	1	10

$\phi_{B,C}(b,c) =$

B \ C	0	1
0	10	1
1	1	10

$\phi_{A,C}(a,c) =$

A \ C	0	1
0	10	1
1	1	10

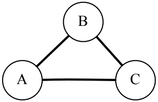
$$p(a, b, c) = \frac{1}{Z} \phi_{A,B}(a, b) \cdot \phi_{B,C}(b, c) \cdot \phi_{A,C}(a, c), \text{ where}$$

$$Z = \sum_{\hat{a}, \hat{b}, \hat{c} \in \{0,1\}^3} \phi_{A,B}(\hat{a}, \hat{b}) \cdot \phi_{B,C}(\hat{b}, \hat{c}) \cdot \phi_{A,C}(\hat{a}, \hat{c}) = 2 \cdot 1000 + 6 \cdot 10 = 2060.$$

Undirected graphical models / factor graphs

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c), \qquad Z = \sum_{\hat{x}_1, \ldots, \hat{x}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

Simple example (each edge potential function encourages its variables to take the same value):



$\phi_{A,B}(a,b) =$

A \ B	0	1
0	10	1
1	1	10

$\phi_{B,C}(b,c) =$

B \ C	0	1
0	10	1
1	1	10

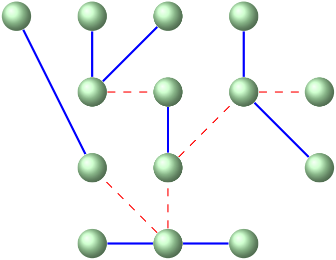
$\phi_{A,C}(a,c) =$

A \ C	0	1
0	10	1
1	1	10

$$p(a,b,c) = \frac{1}{Z} \phi_{A,B}(a,b) \cdot \phi_{B,C}(b,c) \cdot \phi_{A,C}(a,c), \text{ where}$$

With the max clique convention, this graph does not imply pairwise factorization: without further information, we must assume $p(a,b,c) = \frac{1}{Z}\phi_{A,B,C}(a,b,c)$

When is inference (relatively) easy?



Tree

Basic idea: marginal inference for a chain

- Suppose we have a simple chain $A \to B \to C \to D$, and we want to compute $p(D)$, a **set** of values, $\{p(D = d), d \in \mathrm{Val}(D)\}$
- The joint distribution factorizes as

$$p(A, B, C, D) = p(A)p(B \mid A)p(C \mid B)p(D \mid C)$$

- In order to compute $p(D)$, we have to marginalize over A, B, C:

$$p(D) = \sum_{a,b,c} p(A = a, B = b, C = c, D)$$

How can we perform the sum efficiently?

- Our goal is to compute

$$p(D) = \sum_{a,b,c} p(a,b,c,D) = \sum_{a,b,c} p(a)p(b \mid a)p(c \mid b)p(D \mid c)$$
$$= \sum_c \sum_b \sum_a p(D \mid c)p(c \mid b)p(b \mid a)p(a)$$

- We can push the summations inside to obtain:

$$p(D) = \sum_c p(D \mid c) \sum_b p(c \mid b) \underbrace{\sum_a \underbrace{p(b \mid a)p(a)}_{\psi_1(a,b)}}_{\tau_1(b) \text{ 'message about } b\text{'}}$$

- Let's call $\psi_1(A,B) = P(A)P(B|A)$. Then, $\tau_1(B) = \sum_a \psi_1(a,B)$
- Similarly, let $\psi_2(B,C) = \tau_1(B)P(C|B)$. Then, $\tau_2(C) = \sum_b \psi_2(b,C)$
- This procedure is dynamic programming: efficient 'inside out' computation instead of 'outside in'

Marginal inference in a chain

- Generalizing the previous example, suppose we have a chain $X_1 \to X_2 \to \cdots \to X_n$, where each variable has k states
- For $i = 1$ up to $n - 1$, compute (and cache)

$$p(X_{i+1}) = \sum_{x_i} p(X_{i+1} \mid x_i) p(x_i)$$

- Each update takes $\mathcal{O}(k^2)$ time
- The total running time is $\mathcal{O}(nk^2)$
- In comparison, naively marginalizing over all latent variables has time complexity $\mathcal{O}(k^n)$
- Great! We performed marginal inference over the joint distribution without ever explicitly constructing it

How far can we extend the chain approach?

Can we extend the chain idea to do something similar for:

- More complex graphs with many branches?
- Can we get marginals of **all** variables efficiently?
- With cycles?

How far can we extend the chain approach?

Can we extend the chain idea to do something similar for:

- More complex graphs with many branches?
- Can we get marginals of **all** variables efficiently?
- With cycles?

- The junction tree algorithm does all these
- But it's not magic: in the worst case, the problem is NP-hard (even to approximate)
- Junction tree achieves time linear in the number of bags = maximal cliques, exponential in the treewidth ← **key point**

Idea: a junction tree, treewidth

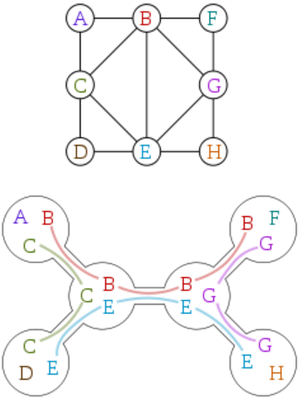


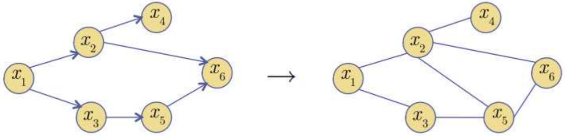
Figure from Wikipedia: Treewidth

- 8 nodes, all $|$maximal clique$| = 3$
- Form a tree where each maximal clique or bag becomes a 'super-node'
- Key properties:
  - Each edge of the original graph is in some bag
  - Each node of the original graph features in contiguous bags: running intersection property
  - Loosely, this will ensure that local consistency $\Rightarrow$ global consistency
- This is called a tree decomposition (graph theory) or junction tree (ML)
- It has treewidth $= \max |$ bag size $| - 1$ (so a tree has treewidth $= 1$)

How can we build a junction tree?

Recipe guaranteed to build a junction tree

1. Moralize (if directed)

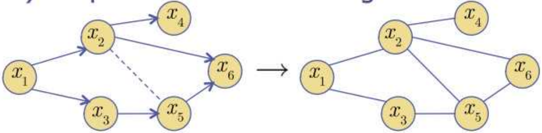


2. Triangulate
3. Identify maximal cliques
4. Build a max weight spanning tree

Then we can propagate probabilities: junction tree algorithm

Moralize

- Converts directed graph into undirected graph
- By moralization, marrying the parents:
  1) Connect nodes that have common children
  2) Drop the arrow heads to get undirected

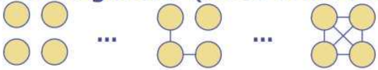


$$p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2)p(x_5 \mid x_3)p(x_6 \mid x_2, x_5)$$
$$\to \tfrac{1}{Z}\psi(x_1, x_2)\psi(x_1, x_3)\psi(x_2, x_4)\psi(x_3, x_5)\psi(x_2, x_5, x_6)$$

$$p(x_1)p(x_2 \mid x_1) \to \psi(x_1, x_2)$$
$$p(x_4 \mid x_2) \to \psi(x_2, x_4)$$
$$Z \to 1$$

- Note: moralization resolves *coupling* due to marginalizing
- moral graph is more general (loses some independencies)

most specific



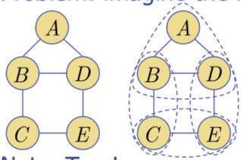
most general

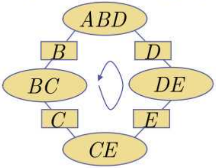
- Each ψ is different based on its arguments, don't get confused
- Ok to put the $p(x_1)$ term into either $\psi_{12}(x_1, x_2)$ or $\psi_{13}(x_1, x_3)$

Triangulate

- We want to build a tree of maximal cliques = bags
- Notation here: an oval is a maximal clique,
  a rectangle is a separator

•Problem: imagine the following undirected graph





•Not a Tree!

•To ensure Junction Tree is a tree (no loops, etc.)
before forming it must first Triangulate moral graph
before finding the cliques...

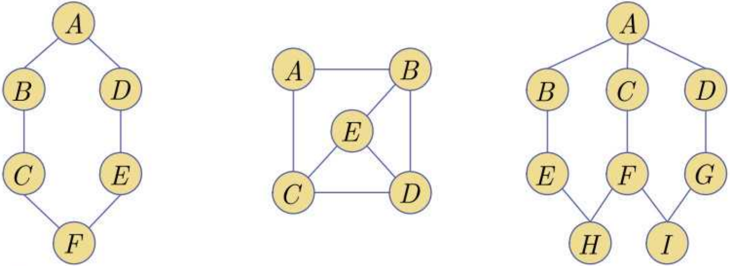
•Triangulating gives more general graph (like moralization)

•Adds links to get rid of cycles or loops

•Triangulation: Connect nodes in moral graph until
no chordless cycle of 4 or more nodes remains in the graph

Actually, often we enforce a tree, in which case triangulation and other steps ⇒ running intersection property

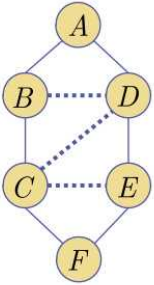
Triangulate

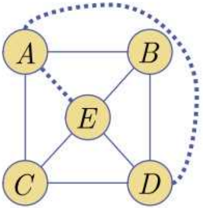


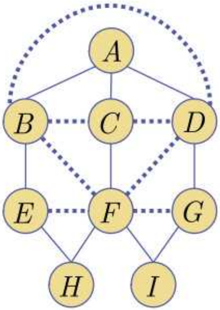
•Cycle: A closed (simple) path, with no repeated vertices other than the starting and ending vertices

•Chordless Cycle: a cycle where no two non-adjacent vertices on the cycle are joined by an edge.

•Triangulated Graph: a graph that contains no chordless cycle of four or more vertices (aka a Chordal Graph).


A
B
D
C
E
F

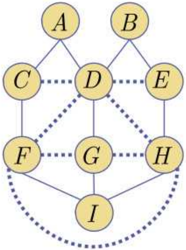

A
B
E
C
D

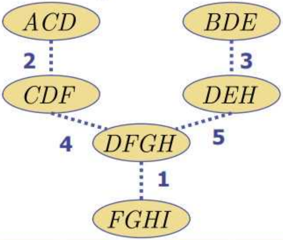

A
B
C
D
E
F
G
H
I

Identify maximal cliques, build a max weight spanning tree

- For edge weights, use |*separator*|
- For max weight spanning tree, several algorithms e.g. Kruskal's

•Start with unconnected cliques (after triangulation)





	ACD	BDE	CDF	DEH	DFGH	FGHI
ACD	-	1	2	1	1	0
BDE		-	1	2	1	0
CDF			-	1	2	1
DEH				-	2	1
DFGH					-	3
FGHI						-

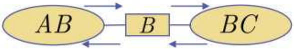
We now have a valid junction tree!

- We had $p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_c \psi_c(x_c)$
- Think of our junction tree as composed of maximal cliques $c =$ bags with $\psi_c(x_c)$ terms
- And separators s with $\phi_s(x_s)$ terms, initialize all $\phi_s(x_s) = 1$
- Write $p(x_1, \ldots, x_n) = \frac{1}{Z} \frac{\prod_c \psi_c(x_c)}{\prod_s \phi_s(x_s)}$

- Now let the message passing begin!
- At every step, we update some $\psi'_c(x_c)$ and $\phi'_s(x_s)$ functions but we always preserve $p(x_1, \ldots, x_n) = \frac{1}{Z} \frac{\prod_c \psi'_c(x_c)}{\prod_s \phi'_s(x_s)}$
- This is called Hugin propagation, can interpret updates as reparameterizations 'moving score around between functions' (may be used as a theoretical proof technique)

Message passing for just 2 maximal cliques (Hugin)

- Send message from each clique *to* its separators of what it thinks the submarginal on the separator is.
- Normalize each clique by incoming message *from* its separators so it agrees with them



$V = \{A, B\} \quad S = \{B\} \quad W = \{B, C\}$

If agree: $\sum_{V \setminus S} \psi_V = \phi_S = p(S) = \phi_S = \sum_{W \setminus S} \psi_W$ **...Done!**

Else:

Send message From V to W...

$$\phi_S^* = \sum_{V \setminus S} \psi_V$$
$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W$$
$$\psi_V^* = \psi_V$$

Send message From W to V...

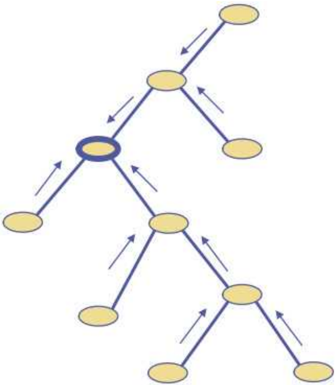
$$\phi_S^{**} = \sum_{W \setminus S} \psi_W^*$$
$$\psi_V^{**} = \frac{\phi_S^{**}}{\phi_S^*} \psi_V^*$$
$$\psi_W^{**} = \psi_W^*$$

Now they Agree...Done!

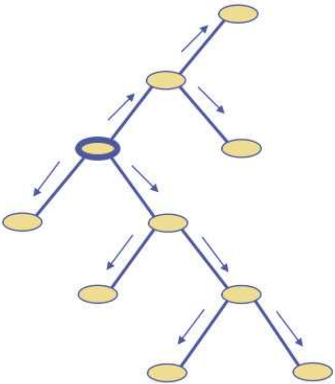
$$\sum_{V \setminus S} \psi_V^{**} = \sum_{V \setminus S} \frac{\phi_S^{**}}{\phi_S^*} \psi_V^* = \frac{\phi_S^{**}}{\phi_S^*} \sum_{V \setminus S} \psi_V^* = \phi_S^{**} = \sum_{W \setminus S} \psi_W^{**}$$

Message passing for a general junction tree

1. Collect



2. Distribute



Then done!

(may need to normalize)

A different idea: belief propagation (Pearl)

- If the initial graph is a tree, inference is simple
- If there are cycles, we can form a junction tree of maximal cliques 'super-nodes'...
- Or just pretend the graph is a tree! Pass messages until convergence (we hope)
- This is loopy belief propagation (LBP), an approximate method
- Perhaps surprisingly, it is often very accurate (e.g. error correcting codes, see McEliece, MacKay and Cheng, 1998, *Turbo Decoding as an Instance of Pearl's "Belief Propagation" Algorithm*)

- Prompted much work to try to understand why
- First we need some background on variational inference (you should know: almost all approximate marginal inference approaches are either variational or sampling methods)

Variational approach for marginal inference

- We want to find the true distribution p but this is hard
- Idea: Approximate p by q for which computation is easy, with q 'close' to p
- How should we measure 'closeness' of probability distributions?

Variational approach for marginal inference

- We want to find the true distribution p but this is hard
- Idea: Approximate p by q for which computation is easy, with q 'close' to p
- How should we measure 'closeness' of probability distributions?
- A very common approach: Kullback-Leibler (KL) divergence
- The 'qp' KL-divergence between two probability distributions q and p is defined as

$$D(q \| p) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}$$

  (measures the expected number of extra bits required to describe *samples from* $q(\mathbf{x})$ using a code based on p instead of q)
- $D(q \parallel p) \geq 0$ for all q, p, with equality iff $q = p$ (a.e.)
- KL-divergence is not symmetric

Variational approach for marginal inference

- Suppose that we have an arbitrary graphical model:

$$p(\mathbf{x};\theta) = \frac{1}{Z(\theta)} \prod_{\mathbf{c}\in C} \psi_c(\mathbf{x_c}) = \exp\left(\sum_{\mathbf{c}\in C} \theta_c(\mathbf{x_c}) - \log Z(\theta)\right)$$

- Rewrite the KL-divergence as follows:

$$\begin{aligned}
D(q\|p) &= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \\
&= -\sum_{\mathbf{x}} q(\mathbf{x}) \log p(\mathbf{x}) - \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{1}{q(\mathbf{x})} \\
&= -\sum_{\mathbf{x}} q(\mathbf{x}) \left(\sum_{\mathbf{c}\in C} \theta_c(\mathbf{x_c}) - \log Z(\theta)\right) - H(q(\mathbf{x})) \\
&= -\sum_{\mathbf{c}\in C} \sum_{\mathbf{x}} q(\mathbf{x}) \theta_c(\mathbf{x_c}) + \sum_{\mathbf{x}} q(\mathbf{x}) \log Z(\theta) - H(q(\mathbf{x})) \\
&= \underbrace{-\sum_{\mathbf{c}\in C} E_q[\theta_c(\mathbf{x_c})]}_{\text{expected score}} + \log Z(\theta) - \underbrace{H(q(\mathbf{x}))}_{\text{entropy}}
\end{aligned}$$

The log-partition function $\log Z$

- Since $D(q \| p) \geq 0$, we have

$$-\sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + \log Z(\theta) - H(q(\mathbf{x})) \geq 0,$$

which implies that

$$\log Z(\theta) \geq \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))$$

- Thus, any approximating distribution $q(\mathbf{x})$ gives a lower bound on the log-partition function (for a Bayesian network, this is the probability of the evidence)
- Recall that $D(q \| p) = 0$ iff $q = p$. Thus, if we optimize over **all** distributions, we have:

$$\log Z(\theta) = \max_q \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))$$

Variational inference: Naive Mean Field

$$\log Z(\theta) = \max_{q \in \mathbb{M}} \underbrace{\sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))}_{\text{concave}} \leftarrow H \text{ of global distn}$$

- The space of **all** valid marginals for q is the marginal polytope
- The naive mean field approximation restricts q to a simple factorized distribution:

$$q(\mathbf{x}) = \prod_{i \in V} q_i(x_i)$$

- Corresponds to optimizing over a non-convex inner bound on the marginal polytope $\Rightarrow$ global optimum hard to find

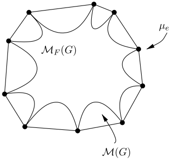
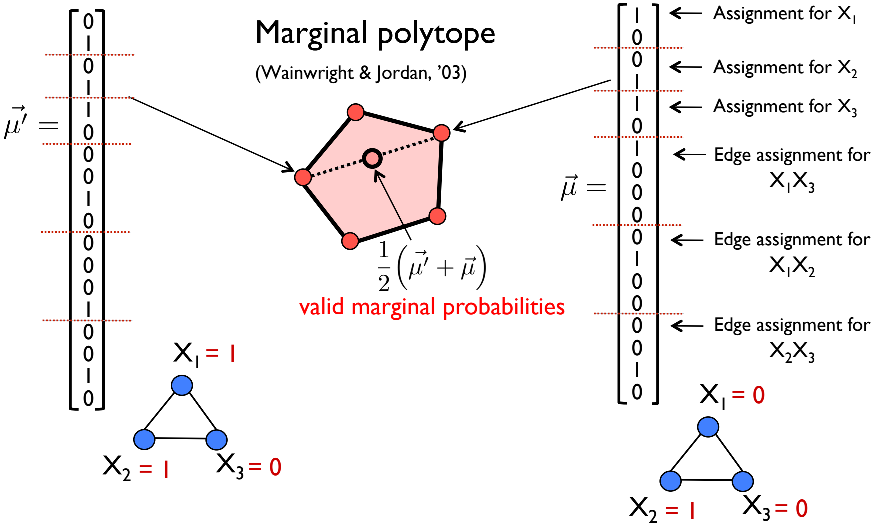


Figure from Martin Wainwright

- Hence, always attains a lower bound on $\log Z$

Background: *the marginal polytope* $\mathbb{M}$ (all valid marginals)



Entropy?

Variational inference: Tree-reweighted (TRW)

$$\log Z(\theta) = \max_{q \in \mathbb{M}} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))$$

- TRW makes 2 pairwise approximations:
  - Relaxes marginal polytope $\mathbb{M}$ to local polytope $\mathbb{L}$, convex outer bound
  - Uses a tree-reweighted upper bound $H_T(q(\mathbf{x})) \geq H(q(\mathbf{x}))$
    The exact entropy on any spanning tree is easily computed from single and pairwise marginals, and yields an upper bound on the true entropy, then H_T takes a convex combination

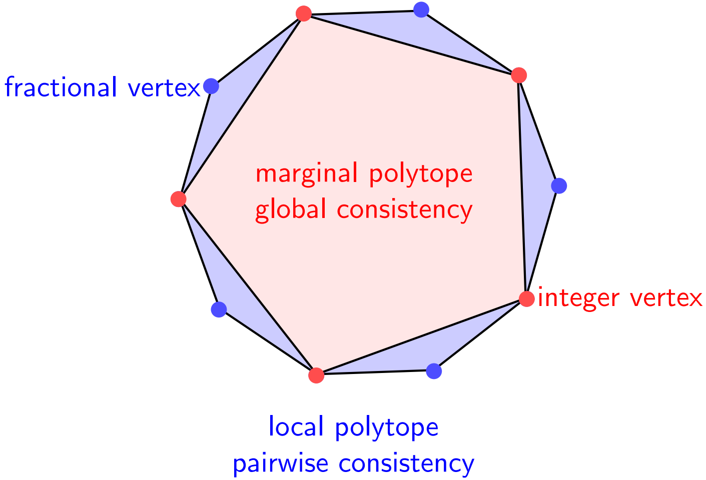
$$\log Z_T(\theta) = \max_{q \in \mathbb{L}} \underbrace{\sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H_T(q(\mathbf{x}))}_{\text{concave}}$$

- Hence, always attains an upper bound on $\log Z$

$$Z_{MF} \leq Z \leq Z_T$$

The local polytope $\mathbb{L}$ has extra fractional vertices

The local polytope is a **convex outer bound** on the marginal polytope



Variational inference: Bethe 

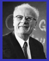
$$\log Z(\theta) = \max_{q \in \mathbb{M}} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))$$

- Bethe makes 2 pairwise approximations:
  - Relaxes marginal polytope $\mathbb{M}$ to local polytope $\mathbb{L}$
  - Uses the Bethe entropy approximation $H_B(q(\mathbf{x})) \approx H(q(\mathbf{x}))$
    The Bethe entropy is exact for a tree. Loosely, it calculates an approximation **pretending** the model is a tree.

$$\log Z_B(\theta) = \max_{q \in \mathbb{L}} \underbrace{\sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H_B(q(\mathbf{x}))}_{\text{not concave in general}}$$

- In general, is neither an upper nor a lower bound on $\log Z$, though is often very accurate (bounds are known for some cases)
- There is a neat relationship between the approximate methods

$$Z_{MF} \leq Z_B \leq Z_T$$

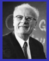
Variational inference: Bethe 

$$\log Z(\theta) = \max_{q \in \mathbb{M}} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))$$

- Bethe makes 2 pairwise approximations:
  - Relaxes marginal polytope $\mathbb{M}$ to local polytope $\mathbb{L}$
  - Uses the Bethe entropy approximation $H_B(q(\mathbf{x})) \approx H(q(\mathbf{x}))$
    The Bethe entropy is exact for a tree. Loosely, it calculates an approximation **pretending** the model is a tree.

$$\log Z_B(\theta) = \max_{q \in \mathbb{L}} \underbrace{\sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H_B(q(\mathbf{x}))}_{\text{not concave in general}}$$

- In general, is neither an upper nor a lower bound on $\log Z$, though is often very accurate (bounds are known for some cases)
- Does this remind you of anything?

Variational inference: Bethe  a remarkable connection

$$\log Z(\theta) = \max_{q \in \mathbb{M}} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H(q(\mathbf{x}))$$

- Bethe makes 2 pairwise approximations:
  - Relaxes marginal polytope $\mathbb{M}$ to local polytope $\mathbb{L}$
  - Uses the Bethe entropy approximation $H_B(q(\mathbf{x})) \approx H(q(\mathbf{x}))$
    The Bethe entropy is exact for a tree. Loosely, it calculates an approximation **pretending** the model is a tree.

$$\log Z_B(\theta) = \underbrace{\max_{q \in \mathbb{L}} \sum_{\mathbf{c} \in C} E_q[\theta_c(\mathbf{x_c})] + H_B(q(\mathbf{x}))}_{\text{stationary points correspond 1-1 with fixed points of LBP!}}$$

- Hence, LBP may be considered a heuristic to optimize the Bethe approximation
- This connection was revealed by Yedidia, Freeman and Weiss, NIPS 2000, *Generalized Belief Propagation*

Software packages

1. libDAI
   - `http://www.libdai.org`
   - Mean-field, loopy sum-product BP, tree-reweighted BP, double-loop GBP
2. Infer.NET
   - `http://research.microsoft.com/en-us/um/cambridge/projects/infernet/`
   - Mean-field, loopy sum-product BP
   - Also handles continuous variables

Extra slides for questions or further explanation

ML learning in Bayesian networks

- Maximum likelihood learning: $\max_\theta \ell(\theta; \mathcal{D})$, where

$$\begin{aligned} \ell(\theta; \mathcal{D}) = \log p(\mathcal{D}; \theta) &= \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) \\ &= \sum_i \sum_{\hat{\mathbf{x}}_{pa(i)}} \sum_{\substack{\mathbf{x} \in \mathcal{D}: \\ \mathbf{x}_{pa(i)} = \hat{\mathbf{x}}_{pa(i)}}} \log p(x_i \mid \hat{\mathbf{x}}_{pa(i)}) \end{aligned}$$

- In Bayesian networks, we have the closed form ML solution:

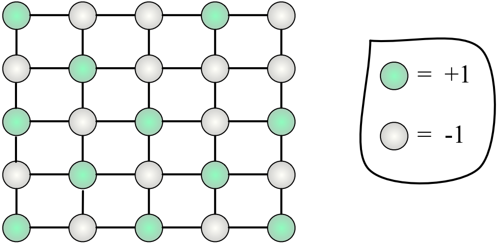
$$\theta^{ML}_{x_i | \mathbf{x}_{pa(i)}} = \frac{N_{x_i, \mathbf{x}_{pa(i)}}}{\sum_{\hat{x}_i} N_{\hat{x}_i, \mathbf{x}_{pa(i)}}}$$

  where $N_{x_i, \mathbf{x}_{pa(i)}}$ is the number of times that the (partial) assignment $x_i, \mathbf{x}_{pa(i)}$ is observed in the training data

- We can estimate each CPD independently because the objective **decomposes** by variable and parent assignment

Parameter learning in Markov networks

- How do we learn the parameters of an Ising model?



$$p(x_1, \cdots, x_n) = \frac{1}{\color{red}{Z}} \exp\left(\sum_{i<j} w_{i,j} x_i x_j + \sum_i \theta_i x_i\right)$$

Bad news for Markov networks

- The global normalization constant $Z(\theta)$ kills decomposability:

$$\begin{aligned}
\theta^{ML} &= \arg\max_{\theta} \ \log \prod_{\mathbf{x}\in\mathcal{D}} p(\mathbf{x};\theta) \\
&= \arg\max_{\theta} \sum_{\mathbf{x}\in\mathcal{D}} \left(\sum_{c} \log \phi_c(\mathbf{x}_c;\theta) - \log Z(\theta) \right) \\
&= \arg\max_{\theta} \left(\sum_{\mathbf{x}\in\mathcal{D}} \sum_{c} \log \phi_c(\mathbf{x}_c;\theta) \right) - |\mathcal{D}| \log Z(\theta)
\end{aligned}$$

- The log-partition function prevents us from decomposing the objective into a sum over terms for each potential
- Solving for the parameters becomes much more complicated