Variable Elimination
+
Belief Propagation
Q: Is Homework 1 representative of future assignments?

A: Not really...

Recall that the remaining assignments will involve a written and programming component, whereas HW1 has just a written section.
Reminders

• Homework 1: PGM Representation
  – Out: Mon, Feb. 15
  – Due: Mon, Feb. 22 at 11:59pm

• Homework 2: Exact inference and supervised learning (CRF+RNN)
  – Out: Mon, Feb. 22
  – Due: Mon, Mar. 08 at 11:59pm
Ex: Factor Graph over Binary Variables

\[
P(A=a, B=b, C=c) = p(a,b,c) = \frac{1}{Z} \psi_A(a) \psi_{AB}(a,b) \psi_{ABC}(a,b,c) = \sum_{a,b,c} s(a,b,c)\psi_A(a) \psi_{AB}(a,b) \psi_{ABC}(a,b,c)
\]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>\psi_A</th>
<th>\psi_{AB}</th>
<th>\psi_{ABC}</th>
<th>s(*)</th>
<th>p(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>48</td>
<td>48/Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>16/Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>6/Z</td>
</tr>
<tr>
<td>...</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ex: Marginal Inference

Marginal Probability:

\[ P(A=a) = p(a) = \sum_{b} \sum_{c} \sum_{d} \sum_{e} p(a, b, c, d, e) \]

\[ P(B=b) = p(b) = \sum_{a} \sum_{c} \sum_{d} \sum_{e} p(a, b, c, d, e) \]

\[ p(a, b, c) = \sum_{d} \sum_{e} p(a, b, c, d, e) \]

"marginalized out d and e"
BRUTE FORCE INFERENCE
Brute Force (Naïve) Inference

For all $i$, suppose the range of $X_i$ is $\{0, 1, 2\}$. Let $k=3$ denote the size of the range.

The distribution factorizes as:

\[ S(\mathbf{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\]
\[ \psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5) \]

Naively, we compute the partition function as:

\[ Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} S(\mathbf{x}) \]
Brute Force (Naïve) Inference

For all $i$, suppose the range of $X_i$ is $\{0, 1, 2\}$. Let $k=3$ denote the size of the range.

The distribution factorizes as:

$$S(\mathbf{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$

Naively, we compute the partition function as:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} S(\mathbf{x})$$

$s(\mathbf{x})$ can be represented as a joint probability table with $3^5$ entries:
Brute Force (Naïve) Inference

For all $i$, suppose the range of $X_i$ is $\{0, 1, 2\}$. Let $k=3$ denote the size of the range.

The distribution factorizes as:

$$S(x) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$

Naively, we compute the partition function as:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} S(x)$$

$s(x)$ can be represented as a joint probability table with $3^5$ entries:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$s(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.019517693</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.017090249</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.014885825</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.024117638</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.000925849</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0.028112576</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.028050205</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0.004812689</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0.007987737</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.028433687</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.037073469</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.013558227</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.019479016</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.012312901</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.023439775</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0.038206131</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.038996005</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0.041458783</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.044616806</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0.020846989</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0.030064750</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0.048436964</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.02854776</td>
</tr>
</tbody>
</table>

Naïve computation of $Z$ requires $3^5$ additions. Can we do better?
Simple and general exact inference for graphical models

VARIABLE ELIMINATION
The Variable Elimination Algorithm

Instead, capitalize on the factorization of \( s(x) \).

\[
Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

This “factor” is a much smaller table with \( 3^2 \) entries:

<table>
<thead>
<tr>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( s(x_4, x_5) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.019517693</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.017090249</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0.014885825</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.024117638</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.00925849</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.028112576</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.028050205</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.004812689</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.007987737</td>
</tr>
</tbody>
</table>
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(x)$.

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5)\psi_5(x_5)$$

Only $3^2$ additions are needed to marginalize out $x_5$.

We denote the marginal’s table by $m_5(x_4)$.

This “factor” is a much smaller table with 3 entries:

<table>
<thead>
<tr>
<th>$x_4$</th>
<th>$m_5(x_4)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.019517693</td>
</tr>
<tr>
<td>1</td>
<td>0.017090249</td>
</tr>
<tr>
<td>2</td>
<td>0.014885825</td>
</tr>
</tbody>
</table>
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(x)$.

\[
Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)
\]

\[
m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)
\]
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(x)$.

\[
Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_5(x_4)
\]

This “factor” is still a $3^4$ table so apply the same trick again.

\[
m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)
\]
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(x)$.

$$Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_{5}(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) m_3(x_1, x_2)$$

$$= \sum_{x_1} m_2(x_1)$$

Naïve solution requires $3^5 = 243$ additions.

Variable elimination only requires $3 + 3^2 + 3^3 + 3^3 + 3^2 = 75$ additions.
The Variable Elimination Algorithm

The same trick can be used to compute marginal probabilities. Just choose the variable elimination order such that the query variables are last.

\[
p(x_1) = \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

For directed graphs, \( Z = 1 \).
For undirected graphs, if we compute each (unnormalized) value on the LHS, we can sum them to get \( Z \).
The Variable Elimination Algorithm

\[ Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5) \]

\[ = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4) \]

\[ = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3) \]

In a factor graph, variable \textbf{elimination} corresponds to replacement of a subgraph with a factor.
The Variable Elimination Algorithm

In a factor graph, variable elimination corresponds to replacement of a subgraph with a factor.

\[ Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5) \]

\[ = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4) \]

\[ = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3) \]
In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.
The Variable Elimination Algorithm

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.
Variable Elimination for Marginal Inference

**Algorithm 1a: Variable Elimination for Marginal Inference**

**Input:** the factor graph and the query variable

**Output:** the marginal distribution for the query variable

a. Run a breadth-first-search starting at the query variable to obtain an ordering of the variable nodes
b. Reverse that ordering
c. Eliminate each variable in the reversed ordering using Algorithm 2

**Algorithm 2: Eliminate One Variable**

**Input:** the variable to be eliminated

**Output:** new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set
b. Replace the eliminated set with a new factor
   a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set
   b. The new factor should assign a score to each possible assignment of its neighboring variables
   c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable
Variable Elimination for Marginal Inference

**Algorithm 1b: Variable Elimination for the Partition Function**

**Input:** the factor graph  
**Output:** the partition function  

a. Run a breadth-first-search starting at an arbitrary variable to obtain an ordering of the variable nodes  
b. Eliminate each variable in the ordering using Algorithm 2

**Algorithm 2: Eliminate One Variable**

**Input:** the variable to be eliminated  
**Output:** new factor graph with the variable marginalized out  

a. Find the input variable and its neighboring factors -- call this set the eliminated set  
b. Replace the eliminated set with a new factor  
   a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set  
   b. The new factor should assign a score to each possible assignment of its neighboring variables  
   c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable
Variable Elimination

Whiteboard:

– Ex: Variable Elimination as factor replacement
Variable Elimination Complexity

Brute force, naïve, inference is $O(\text{____})$

Variable elimination is $O(\text{____})$

where $n = \#$ of variables
$k = \text{max } \#$ values a variable can take
$r = \#$ variables participating in largest “intermediate” table

In-Class Exercise: *Fill in the blank*
Exact Inference

Variable Elimination

- **Uses**
  - Computes the **partition function** of any factor graph
  - Computes the **marginal probability** of a query variable in any factor graph

- **Limitations**
  - Only computes the marginal for **one variable at a time** (i.e. need to re-run variable elimination for each variable if you need them all)
  - **Elimination order** affects runtime

Belief Propagation

- **Uses**
  - Computes the **partition function** of any acyclic factor graph
  - Computes all **marginal probabilities** of factors and variables at once, for any acyclic factor graph

- **Limitations**
  - Only **exact** on acyclic factor graphs (though we’ll consider its “loopy” variant later)
  - **Message passing order** affects runtime (but the obvious topological ordering always works best)
MESSAGE PASSING
Great Ideas in ML: Message Passing

Count the soldiers

1 before you
2 before you
3 before you
4 before you
5 before you

5 behind you
4 behind you
3 behind you
2 behind you
1 behind you

there's 1 of me

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

Belief: Must be $2 + 1 + 3 = 6$ of us

2 before you

3 behind you

Only see my incoming messages

There's 1 of me

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

There's 1 of me

1 before you

Only see my incoming messages

4 behind you

Belief: Must be 1 + 1 + 4 = 6 of us

Belief: Must be 1 + 3 = 6 of us

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Each soldier receives reports from all branches of the tree.
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of the tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Belief: Must be 14 of us

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of the tree.

Belief: Must be 14 of us

wouldn't work correctly with a 'loopy' (cyclic) graph

adapted from MacKay (2003) textbook
Exact marginal inference for factor trees

SUM-PRODUCT BELIEF PROPAGATION
Both of these messages judge the possible values of variable $X$. Their product = belief at $X$ = product of all 3 messages to $X$. 

Message Passing in Belief Propagation
Sum-Product Belief Propagation

Beliefs

Factors

Variables

Messages
Sum-Product Belief Propagation

\[ b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i}(x_i) \]
Sum-Product Belief Propagation

\[ \mu_{i \rightarrow \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i) \]
Sum-Product Belief Propagation

Factor Belief

\[ b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in N(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]
Sum-Product Belief Propagation

Factor Belief

\[ b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]
Sum-Product Belief Propagation

\[ \mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in N(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i]) \]
Sum-Product Belief Propagation

$\mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in N(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i])$

matrix-vector product (for a binary factor)

$X_1 \quad \psi_1 \quad X_3$
Sum-Product Belief Propagation

**Input:** a factor graph with no cycles  
**Output:** exact marginals for each variable and factor

**Algorithm:**
1. Initialize the messages to the uniform distribution.
   \[ \mu_{i \rightarrow \alpha}(x_i) = 1 \quad \mu_{\alpha \rightarrow i}(x_i) = 1 \]
1. Choose a root node.
2. Send messages from the **leaves** to the **root**.  
   Send messages from the **root** to the **leaves**.
   \[ \mu_{i \rightarrow \alpha}(x_i) = \prod_{\alpha \in N(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i) \]
   \[ \mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_{\alpha}(x_\alpha) \prod_{j \in N(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i]) \]
1. Compute the beliefs (unnormalized marginals).
   \[ b_i(x_i) = \prod_{\alpha \in N(i)} \mu_{\alpha \rightarrow i}(x_i) \]
   \[ b_\alpha(x_\alpha) = \psi_{\alpha}(x_\alpha) \prod_{i \in N(\alpha)} \mu_{i \rightarrow \alpha}(x_\alpha[i]) \]
2. Normalize beliefs and return the **exact** marginals.
   \[ p_i(x_i) \propto b_i(x_i) \quad p_\alpha(x_\alpha) \propto b_\alpha(x_\alpha) \]
Sum-Product Belief Propagation

Variables

Factors

Beliefs

Messages

\[ b_i(x_i) = \prod_{\alpha \in N(i)} \mu_{\alpha 

\[ b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in N(\alpha)} \mu_{i \to \alpha}(x_\alpha[i]) \]
Sum-Product Belief Propagation

Variables

Factors

Beliefs

Messages

\[ \mu_{i \rightarrow \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i) \]

\[ \psi_{\alpha}(x_{\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_{\alpha}[i]) \]
FORWARD BACKWARD AS SUM-PRODUCT BP
CRF Tagging Model

$X_1$  $X_2$  $X_3$

find   preferred   tags

Could be verb or noun  Could be adjective or verb  Could be noun or verb
CRF Tagging by Belief Propagation

Forward algorithm = message passing (matrix-vector products)

Backward algorithm = message passing (matrix-vector products)

- Forward-backward is a message passing algorithm.
- It’s the simplest case of belief propagation.
So Let’s Review Forward-Backward …
So Let’s Review Forward-Backward ...

- Show the possible values for each variable
So Let’s Review Forward-Backward …

- Let’s show the possible *values* for each variable
- One possible assignment
Let's show the possible values for each variable
• One possible assignment
• And what the 7 factors think of it ...
Viterbi Algorithm: Most Probable Assignment

• So $p(\mathbf{v} \ \mathbf{a} \ \mathbf{n}) = (1/Z) \ast \text{product of 7 numbers}$
• Numbers associated with edges and nodes of path
• Most probable assignment = path with highest product
Viterbi Algorithm: Most Probable Assignment

- So $p(v \ a \ n) = (1/Z) \ast \text{product weight of one path}$
Forward-Backward Algorithm: Finds Marginals

So $p(v \ a \ n) = (1/Z) \times \text{product weight of one path}

Marginal probability $p(X_2 = a) = (1/Z) \times \text{total weight of all paths through $a$} \forall X_2 \in X_1, X_2, X_3$
Forward-Backward Algorithm: Finds Marginals

- So \( p(v, a, n) = \frac{1}{Z} \) * product weight of one path
- Marginal probability \( p(X_2 = a) \)
  \[ = \frac{1}{Z} \) * total weight of all paths through \( n \)
Forward-Backward Algorithm: Finds Marginals

- So \( p(v a n) = (1/Z) \times \text{product weight of one path} \)
- Marginal probability \( p(X_2 = a) \)
  \( = (1/Z) \times \text{total weight of all paths through } v \)
Forward-Backward Algorithm: Finds Marginals

- So $p(v \ a \ n) = (1/Z) \times \text{product weight of one path}$
- Marginal probability $p(X_2 = a) = (1/Z) \times \text{total weight of all paths through all paths through } n
Forward-Backward Algorithm: Finds Marginals

\[ \alpha_2(n) = \text{total weight of these path prefixes} \]

(found by dynamic programming: matrix-vector products)
Forward-Backward Algorithm: Finds Marginals

\[ \beta_2(n) = \text{total weight of these path suffixes} \]

(found by dynamic programming: matrix-vector products)
Forward-Backward Algorithm: Finds Marginals

Product gives $ax+ay+az+bx+by+bz+cx+cy+cz = \alpha_2(n) \cdot \beta_2(n)$ = total weight of paths
Forward-Backward Algorithm: Finds Marginals

Oops! The weight of a path through a state also includes a weight at that state. So $\alpha(n) \cdot \beta(n)$ isn’t enough. The extra weight is the opinion of the unigram factor at this variable.

"belief that $X_2 = n$"

total weight of all paths through $n$:

$$= \alpha_2(n) \cdot \psi_2(n) \cdot \beta_2(n)$$
Forward-Backward Algorithm: Finds Marginals

"belief that $X_2 = v$"

"belief that $X_2 = n$"

preferred

total weight of all paths through $v$

\[= \alpha_2(v) \psi_{\{2\}}(v) \beta_2(v)\]
Forward-Backward Algorithm: Finds Marginals

```
<table>
<thead>
<tr>
<th></th>
<th>v</th>
<th>n</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>1.8</td>
<td>0</td>
<td>4.2</td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

divide by $Z=6$ to get marginal probs

“belief that $X_2 = v$”

“belief that $X_2 = n$”

“belief that $X_2 = a$”

sum = $Z$ (total probability of all paths)

```
total weight of all paths through $a$
= $\alpha_2(a) \psi_{\{2\}}(a) \beta_2(a)$
```
BP AS DYNAMIC PROGRAMMING
(Acyclic) Belief Propagation

In a factor graph with no cycles:
1. Pick any node to serve as the root.
2. Send messages from the leaves to the root.
3. Send messages from the root to the leaves.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.
(Acyclic) Belief Propagation

In a factor graph with no cycles:
1. Pick any node to serve as the root.
2. Send messages from the leaves to the root.
3. Send messages from the root to the leaves.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.
Acyclic BP as Dynamic Programming

\[ p(X_i = x_i) \propto b(x_i) = \sum_{x:x_{[i]}=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) \]

\[ = \left( \sum_{x:x_{[i]}=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) \right) \left( \sum_{x:x_{[i]}=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) \right) \left( \sum_{x:x_{[i]}=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) \right) \]

Subproblem: Inference using just the factors in subgraph \( H \)

Figure adapted from Burkett & Klein (2012)
Acyclic BP as Dynamic Programming

\[ p(X_i = x_i) \propto b_i(x_i) = \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_\alpha(x_\alpha) \]

\[ = \left( \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_\alpha(x_\alpha) \right) \left( \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_\alpha(x_\alpha) \right) \left( \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_\alpha(x_\alpha) \right) \]

Subproblem:
Inference using just the factors in subgraph \( H \)

The marginal of \( X_i \) in that smaller model is the message sent to \( X_i \) from subgraph \( H \)

Message to a variable
Acyclic BP as Dynamic Programming

\[ p(X_i = x_i) \propto b_i(x_i) = \sum_{x : x[i] = x_i} \prod_{\alpha \subseteq F} \psi_{\alpha}(x_{\alpha}) \]

\[ = \left( \sum_{x : x[i] = x_i} \prod_{\alpha \subseteq F} \psi_{\alpha}(x_{\alpha}) \right) \left( \sum_{x : x[i] = x_i} \prod_{\alpha \subseteq G} \psi_{\alpha}(x_{\alpha}) \right) \left( \sum_{x : x[i] = x_i} \prod_{\alpha \subseteq H} \psi_{\alpha}(x_{\alpha}) \right) \]

**Subproblem:**
Inference using just the factors in subgraph \( H \)

The marginal of \( X_i \) in that smaller model is the message sent to \( X_i \) from subgraph \( H \)

**Message to a variable**
Acyclic BP as Dynamic Programming

\[ p(X_i = x_i) \propto b_i(x_i) = \sum_{x: x[i] = x_i} \prod_{\alpha} \psi_\alpha(x_{\alpha}) = \left( \sum_{x: x[i] = x_i} \prod_{\alpha} \psi_\alpha(x_{\alpha}) \right) \left( \sum_{x: x[i] = x_i} \prod_{\alpha} \psi_\alpha(x_{\alpha}) \right) \left( \sum_{x: x[i] = x_i} \prod_{\alpha} \psi_\alpha(x_{\alpha}) \right) \]

Subproblem:
Inference using just the factors in subgraph \( H \)

The marginal of \( X_i \) in that smaller model is the message sent to \( X_i \) from subgraph \( H \)

Message to a variable
Acyclic BP as Dynamic Programming

\[ p(X_i = x_i) \propto b_i(x_i) = \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) = \left( \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) \right) \left( \sum_{x:x[i]=x_i} \prod_{\alpha} \psi_{\alpha}(x_{\alpha}) \right) \left( \sum_{x:x[i]=x_i} \prod_{\alpha \subseteq H} \psi_{\alpha}(x_{\alpha}) \right) \]

Subproblem:
Inference using just the factors in subgraph \( F \cup H \)

The marginal of \( X_i \) in that smaller model is the message sent by \( X_i \) out of subgraph \( F \cup H \)

Message from a variable
• If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a product of $k$ marginals computed on smaller subgraphs.
• Each subgraph is obtained by cutting some edge of the tree.
• The message-passing algorithm uses dynamic programming to compute the marginals on all such subgraphs, working from smaller to bigger. So you can compute all the marginals.
Acyclic BP as Dynamic Programming

• If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a product of $k$ marginals computed on smaller subgraphs.
• Each subgraph is obtained by cutting some edge of the tree.
• The message-passing algorithm uses dynamic programming to compute the marginals on all such subgraphs, working from smaller to bigger. So you can compute all the marginals.
Acyclic BP as Dynamic Programming

- If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.

![Diagram of a tree with nodes and messages]

- Time
- Flies
- Like
- An
- Arrow
If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a product of $k$ marginals computed on smaller subgraphs.

Each subgraph is obtained by cutting some edge of the tree.

The message-passing algorithm uses dynamic programming to compute the marginals on all such subgraphs, working from smaller to bigger. So you can compute all the marginals.
Acyclic BP as Dynamic Programming

- If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a product of $k$ marginals computed on smaller subgraphs.
- Each subgraph is obtained by cutting some edge of the tree.
- The message-passing algorithm uses dynamic programming to compute the marginals on all such subgraphs, working from smaller to bigger. So you can compute all the marginals.
If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a product of $k$ marginals computed on smaller subgraphs.

Each subgraph is obtained by cutting some edge of the tree.

The message-passing algorithm uses dynamic programming to compute the marginals on all such subgraphs, working from smaller to bigger. So you can compute all the marginals.
Acyclic BP as Dynamic Programming

- If you want the marginal $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.
Exact MAP inference for factor trees

MAX-PRODUCT BELIEF PROPAGATION
Max-product Belief Propagation

• **Sum-product BP** can be used to compute the marginals, \( p_i(X_i) \)
• compute the partition function, \( Z \)

• **Max-product BP** can be used to compute the most likely assignment, 
  \( X^* = \arg\max_X p(X) \)
Max-product Belief Propagation

• Change the sum to a max:

\[
\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)
\]

\[
\mu_{\alpha \to i}(x_i) = \sum_{x_{\alpha}: x_{\alpha}[i] = x_i} \psi_{\alpha}(x_{\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(x_{\alpha}[i])
\]

• **Max-product BP** computes max-marginals
  – The max-marginal \( b_i(x_i) \) is the (unnormalized) probability of the MAP assignment under the constraint \( X_i = x_i \).
  – For an acyclic graph, the MAP assignment (assuming there are no ties) is given by:

\[
x_i^* = \arg \max_{x_i} b_i(x_i)
\]
Max-product Belief Propagation

- Change the sum to a max:

\[
\mu_{i \rightarrow \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \rightarrow i}(x_i)
\]

\[
\mu_{\alpha \rightarrow i}(x_i) = \max_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i])
\]

- **Max-product BP computes max-marginals**
  - The max-marginal \( b_i(x_i) \) is the (unnormalized) probability of the MAP assignment under the constraint \( X_i = x_i \).
  - For an acyclic graph, the MAP assignment (assuming there are no ties) is given by:

\[
x_i^* = \arg \max_{x_i} b_i(x_i)
\]
Deterministic Annealing

**Motivation:** Smoothly transition from sum-product to max-product

1. Incorporate inverse temperature parameter into each factor:

   \[
   p(x) = \frac{1}{Z} \prod_{\alpha} \psi_{\alpha}(x_{\alpha})^{\frac{1}{T}}
   \]

   **Annealed Joint Distribution**

2. Send messages as usual for sum-product BP

3. Anneal \( T \) from 1 to 0:

   \[
   \begin{array}{|c|c|}
   \hline
   T = 1 & \text{Sum-product} \\
   T \to 0 & \text{Max-product} \\
   \hline
   \end{array}
   \]

4. Take resulting beliefs to power \( T \)
Semirings

- Sum-product $+/*$ and max-product $\max/*$ are commutative semirings.
- We can run BP with any such commutative semiring.

\[
\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)
\]

\[
\mu_{\alpha \to i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(x_\alpha[i])
\]

- In practice, multiplying many small numbers together can yield underflow.
  - Instead of using $+/*$, we use log-add$+/$
  - Instead of using $\max/*$, we use $\max/+$. 

88
Exact inference for linear chain models

FORWARD-BACKWARD AND VITERBI ALGORITHMS
Forward-Backward Algorithm

• Sum-product BP on an HMM is called the forward-backward algorithm
• Max-product BP on an HMM is called the Viterbi algorithm
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph

time  flies  like  an  arrow
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph

Trick: (See also Sha & Pereira (2003))

• Replace each variable domain with its cross product
e.g. \{B, I, O\} \rightarrow \{BB, BI, BO, IB, II, IO, OB, OI, OO\}

• Replace each pair of variables with a single one. For all \(i, y_{i,i+1} = (x_i, x_{i+1})\)

• Add features with weight \(-\infty\) that disallow illegal configurations
between pairs of the new variables
e.g. **legal** = BI and IO **illegal** = II and OO

• This is effectively a special case of the junction tree algorithm
Summary

1. **Factor Graphs**
   - Alternative representation of directed / undirected graphical models
   - Make the cliques of an undirected GM explicit

2. **Variable Elimination**
   - Simple and general approach to exact inference
   - Just a matter of being clever when computing sum-products

3. **Sum-product Belief Propagation**
   - Computes all the marginals and the partition function in only twice the work of Variable Elimination

4. **Max-product Belief Propagation**
   - Identical to sum-product BP, but changes the semiring
   - Computes: max-marginals, probability of MAP assignment, and (with backpointers) the MAP assignment itself.
An example of why we need approximate inference

EXACT INFERENCE ON GRID CRF
Application: Pose Estimation

\[ \phi_i(y_i, x) \in \mathbb{R}^{\approx 1000} : \text{local image representation, e.g. HoG} \]

\[ \langle w_i, \phi_i(y_i, x) \rangle : \text{local confidence map} \]

\[ \phi_{i,j}(y_i, y_j) = \text{good fit}(y_i, y_j) \in \mathbb{R}^1 : \text{test for geometric fit} \]

\[ \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle : \text{penalizer for unrealistic poses} \]

Together: \[ \operatorname{argmax}_y p(y|x) \] is sanitized version of local cues

![original](image1)

local classification

local + geometry
Feature Functions for CRF in Vision

$$\phi_i(y_i, x): \text{local representation, high-dimensional}$$
$$\rightarrow \langle w_i, \phi_i(y_i, x) \rangle: \text{local classifier}$$

$$\phi_{i,j}(y_i, y_j): \text{prior knowledge, low-dimensional}$$
$$\rightarrow \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle: \text{penalize outliers}$$

learning adjusts parameters:

- unary $w_i$: learn local classifiers and their importance
- binary $w_{ij}$: learn importance of smoothing/penalization

$$\operatorname*{argmax}_y p(y|x)$$ is cleaned up version of local prediction
Case Study: Image Segmentation

- Image segmentation (FG/BG) by modeling of interactions btw RVs
  - Images are noisy.
  - Objects occupy continuous regions in an image.

\[ Y^* = \arg \max_{y \in \{0,1\}^n} \left[ \sum_{i \in S} V_i(y_i, X) + \sum_{i \in S} \sum_{j \in N_i} V_{i,j}(y_i, y_j) \right]. \]

Nowozin, Lampert 2012

© Eric Xing @ CMU, 2005-2015
• Suppose we want to image segmentation using a grid model
Grid CRF

- Suppose we want to image segmentation using a grid model

Assuming we divide into foreground/background, each factor is a table with $2^2$ entries.
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

• Suppose we want to image segmentation using a grid model
• What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?

This new factor has $2^5$ entries
Grid CRF

• Suppose we want to image segmentation using a grid model
• What happens when we run variable elimination?

For an MxM grid the new factor has $2^M$ entries
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?

For an $M \times M$ grid the new factor has $2^M$ entries

In general, for high treewidth graphs like this, we turn to approximate inference (which we’ll cover soon!)