Belief Propagation
Q: What if I already answered a homework question using different assumptions than what was clarified in a Piazza note?

A: Just write down the assumptions you made.

We will usually give credit so long as your assumptions are clear in the writeup and your answer correct under those assumptions.

(Obviously, this only applies to underspecified / ambiguous questions. You can’t just add arbitrary assumptions!)
Reminders

• Homework 1: DAgger for seq2seq
  – Out: Thu, Sep. 12
  – Due: Thu, Sep. 26 at 11:59pm

• Homework 2: Labeling Syntax Trees
  – Out: Thu, Sep. 26
  – Due: Thu, Oct. 10 at 11:59pm
Instead, capitalize on the factorization of $p(x)$.

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

Brute force, naïve, inference is $O(\underline{\quad})$  
Variable elimination is $O(\underline{\quad})$

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

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

there's 1 of me

2 before you

only see my incoming messages

3 behind you

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
Great Ideas in ML: Message Passing

*Each soldier receives reports from all branches of tree*

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
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

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
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 $X$. 
Sum-Product Belief Propagation

Variables

Factors

Beliefs

Messages
Sum-Product Belief Propagation

Variable Belief

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

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

Factor Message

\[ \mu_{\alpha \rightarrow i}(x_i) = \sum_{\mathbf{x}_\alpha : \mathbf{x}_\alpha[i] = x_i} \psi_\alpha(\mathbf{x}_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(\mathbf{x}_\alpha[i]) \]
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

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

1. Compute the beliefs (unnormalized marginals).
   \[
   b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i}(x_i) \quad b_{\alpha}(x_{\alpha}) = \psi_{\alpha}(x_{\alpha}) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \rightarrow \alpha}(x_{\alpha}[i])
   \]

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

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

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

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

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

Find preferred tags

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

Could be verb or noun

Could be adjective or verb

Could be noun or verb
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
So Let’s Review Forward-Backward ...

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

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

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

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

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

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

\[ \alpha_2(n) \] = 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

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

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

Product gives \[ ax+ay+az+bx+by+bz+cx+cy+cz \] = total weight of paths
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.

Forward-Backward Algorithm: Finds Marginals

"belief that $X_2 = n$"

total weight of all paths through $n$

$$= \alpha_2(n) \psi_{\{2\}}(n) \beta_2(n)$$
Forward-Backward Algorithm: Finds Marginals

\[ \text{total weight of } \textit{all paths through} \triangle v = \alpha_2(v) \psi_{\{2\}}(v) \beta_2(v) \]

"belief that \( X_2 = v \)"

"belief that \( X_2 = n \)"

Preferred
Forward-Backward Algorithm: Finds Marginals

- **v** | 1.8
- **n** | 0
- **a** | 4.2

divide by Z=6 to get marginal probs

```
| v | 0.3 |
| n | 0   |
| a | 0.7 |
```

“belief that \(X_2 = v\)”

“belief that \(X_2 = n\)”

“belief that \(X_2 = a\)”

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

```
\[ \alpha_2(a) \psi_{\{2\}}(a) \beta_2(a) \]
```

total weight of *all* paths through \(a\)
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

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

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

\[
\mu_{F \rightarrow i}(x_i) \quad \mu_{G \rightarrow i}(x_i) \quad \mu_{H \rightarrow i}(x_i)
\]

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

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

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

**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

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

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

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

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

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.
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.
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:

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

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

• **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:

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

2. Send messages as usual for sum-product BP

3. Anneal $T$ from 1 to 0:

<table>
<thead>
<tr>
<th>$T = 1$</th>
<th>Sum-product</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T \to 0$</td>
<td>Max-product</td>
</tr>
</tbody>
</table>

3. 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

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

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

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

\[ X_1 \xrightarrow{\psi_1} X_2 \xrightarrow{\psi_2} X_3 \xrightarrow{\psi_3} X_4 \xrightarrow{\psi_4} X_5 \]

time  flies  like  an  arrow
Forward-Backward Algorithm

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

![Diagram of factor graph]

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

- Replace each variable domain with its cross product e.g. \{B,I,O\} → \{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 -\(∞\) 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.