Structured Belief Propagation for NLP

Matthew R. Gormley & Jason Eisner
ACL ‘15 Tutorial
July 26, 2015

For the latest version of these slides, please visit:
http://www.cs.jhu.edu/~mrg/bp-tutorial/
Language has a lot going on at once

Structured representations of utterances
Structured knowledge of the language

Many interacting parts for BP to reason about!
• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models**: Factor graphs can express interactions among linguistic structures.
2. **Algorithm**: BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions**: What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models**: Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm**: Finish in fewer steps and make the steps faster.
6. **Learning**: Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software**: Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.

2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.

3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?

4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.

5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.

6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.

7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!
Section 1: Introduction

Modeling with Factor Graphs
Sampling from a Joint Distribution

A joint distribution defines a probability $p(x)$ for each assignment of values $x$ to variables $X$. This gives the proportion of samples that will equal $x$.

Sample 1:

Sample 2:

Sample 3:

Sample 4:

Sample 5:

Sample 6:

$X_0$ $X_1$ $X_2$ $X_3$ $X_4$ $X_5$

<START> time flies like an arrow
A joint distribution defines a probability \( p(x) \) for each assignment of values \( x \) to variables \( X \). This gives the proportion of samples that will equal \( x \).
Sampling from a Joint Distribution

A **joint distribution** defines a probability $p(x)$ for each assignment of values $x$ to variables $X$. This gives the **proportion** of samples that will equal $x$. 

<table>
<thead>
<tr>
<th>Sample 1:</th>
<th>Sample 2:</th>
<th>Sample 3:</th>
<th>Sample 4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>p</td>
</tr>
<tr>
<td>time</td>
<td>flies</td>
<td>like</td>
<td>an</td>
</tr>
<tr>
<td>n</td>
<td>v</td>
<td>d</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>an</td>
<td>arrow</td>
<td></td>
</tr>
<tr>
<td>flies</td>
<td>like</td>
<td>an</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>flies</td>
<td>an</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>v</td>
<td>d</td>
<td>n</td>
</tr>
<tr>
<td>v</td>
<td>d</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

A joint distribution defines a probability $p(x)$ for each assignment of values $x$ to variables $X$. This gives the proportion of samples that will equal $x$. 

---

Sample 1:
- time
- flies
- like
- an
- arrow

Sample 2:
- time
- flies
- like
- an
- arrow

Sample 3:
- flies
- fly
- with
- their
- wings

Sample 4:
- with
- time
- you
- will
- see

---

$X_0$ $X_1$ $X_2$ $X_3$ $X_4$ $X_5$ $W_1$ $W_2$ $W_3$ $W_4$ $W_5$

<START>
Factors have local opinions (≥ 0)

Each black box looks at some of the tags $X_i$ and words $W_i$

Note: We chose to reuse the same factors at different positions in the sentence.
Factors have local opinions (≥ 0)

Each black box looks at some of the tags $X_i$ and words $W_i$

$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \ ?$
Global probability = product of local opinions

Each black box looks at some of the tags $X_i$ and words $W_i$

$$p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 * 8 * 5 * 3 * \ldots)$$

Uh-oh! The probabilities of the various assignments sum up to $Z > 1$.
So divide them all by $Z$. 
Markov Random Field (MRF)

Joint distribution over tags \(X_i\) and words \(W_i\). The individual factors aren’t necessarily probabilities.

\[
p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{\mathcal{Z}} (4 \times 8 \times 5 \times 3 \times \ldots)
\]
Hidden Markov Model

But sometimes we choose to make them probabilities. Constrain each row of a factor to sum to one. Now $Z = 1$.

$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \frac{1}{Z} (0.3 \times 0.8 \times 0.2 \times 0.5 \times \ldots)$$
Markov Random Field (MRF)

Joint distribution over tags $X_i$ and words $W_i$

$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \frac{1}{Z}(4 \times 8 \times 5 \times 3 \times \ldots)$$
Conditional Random Field (CRF)

Conditional distribution over tags $X_i$ given words $w_i$. The factors and $Z$ are now specific to the sentence $w$.

$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$
How General Are Factor Graphs?

- Factor graphs can be used to describe
  - Markov Random Fields (undirected graphical models)
    - i.e., log-linear models over a tuple of variables
  - Conditional Random Fields
  - Bayesian Networks (directed graphical models)

- Inference treats all of these interchangeably.
  - Convert your model to a factor graph first.
  - Pearl (1988) gave key strategies for exact inference:
    - Belief propagation, for inference on acyclic graphs
    - Junction tree algorithm, for making any graph acyclic
      (by merging variables and factors: blows up the runtime)
Object-Oriented Analogy

• What is a sample?
  A datum: an immutable object that describes a linguistic structure.

• What is the sample space?
  The class of all possible sample objects.

```java
class Tagging:
    int n; // length of sentence
    Word[] w; // array of n words (values w_i)
    Tag[] t; // array of n tags (values t_i)
```

• What is a random variable?
  An accessor method of the class, e.g., one that returns a certain field.
  – Will give different values when called on different random samples.

```java
Word W(int i) { return w[i]; } // random var W_i
Tag T(int i) { return t[i]; } // random var T_i

String S(int i) {
    return suffix(w[i], 3); // random var S_i
}
```

Random variable $W_5$ takes value $w_5$ == “arrow” in this sample
Object-Oriented Analogy

- **What is a sample?**
  A datum: an immutable object that describes a linguistic structure.

- **What is the sample space?**
  The class of all possible sample objects.

- **What is a random variable?**
  An accessor method of the class, e.g., one that returns a certain field.

- **A model is represented by a different object. What is a factor of the model?**
  A method of the model that computes a number $\geq 0$ from a sample, based on the sample’s values of a few random variables, and parameters stored in the model.

```java
class TaggingModel:
    float transition(Tagging tagging, int i) { // tag-tag bigram
        return tparam[tagging.t(i-1)][tagging.t(i)]; }
    float emission(Tagging tagging, int i) { // tag-word bigram
        return eparam[tagging.t(i)][tagging.w(i)]; }
    float uprob(Tagging tagging) { // unnormalized prob
        float p=1;
        for (i=1; i <= tagging.n; i++) {
            p *= transition(i) * emission(i); }
        return p; }
```

- **How do you find the scaling factor?**
  Add up the probabilities of all possible samples. If the result $Z \neq 1$, divide the probabilities by that $Z$. 

  ```java
  float uprob(Tagging tagging) {
      // unnormalized prob
      float p=1;
      for (i=1; i <= tagging.n; i++) {
         p *= transition(i) * emission(i); }
      return p; }
  ```
Modeling with Factor Graphs

• **Factor graphs** can be used to model many linguistic structures.

• Here we highlight a few example NLP tasks.
  – *People have used BP for all of these.*

• We’ll describe how **variables** and **factors** were used to describe structures and the interactions among their parts.
Annotating a Tree

Given: a **sentence** and unlabeled **parse** tree.

```
time   flies   like   an   arrow
```
Annotating a Tree

Given: a **sentence** and unlabeled **parse** tree.

Construct a factor graph which mimics the tree structure, to **predict** the tags / nonterminals.
Annotating a Tree

Given: a **sentence** and unlabeled **parse** tree.

Construct a factor graph which mimics the tree structure, to **predict** the tags / nonterminals.
Annotating a Tree

Given: a sentence and unlabeled parse tree.

Construct a factor graph which mimics the tree structure, to predict the tags / nonterminals.

We could add a linear chain structure between tags. (This creates cycles!)
Constituency Parsing

What if we needed to predict the tree structure too?

**Use more variables:**
Predict the nonterminal of each substring, or $\emptyset$ if it’s not a constituent.
Constituency Parsing

What if we needed to predict the tree structure too?

**Use more variables:**
Predict the nonterminal of each substring, or $\emptyset$ if it’s not a constituent.

But nothing prevents non-tree structures.
Constituency Parsing

What if we needed to predict the tree structure too?

**Use more variables:**
Predict the nonterminal of each substring, or $\emptyset$ if it’s not a constituent.

But nothing prevents non-tree structures.
Constituency Parsing

What if we needed to predict the tree structure too?

Use more variables:
Predict the nonterminal of each substring, or $\varnothing$ if it’s not a constituent.

But nothing prevents non-tree structures.

Add a factor which multiplies in 1 if the variables form a tree and 0 otherwise.
Constituency Parsing

What if we needed to predict the tree structure too?

**Use more variables:**
Predict the nonterminal of each substring, or $\emptyset$ if it’s not a constituent.

But nothing prevents non-tree structures.

Add a factor which multiplies in 1 if the variables form a tree and 0 otherwise.
Constituency Parsing

• **Variables:**
  – Constituent type (or \(\emptyset\)) for each of \(O(n^2)\) substrings

• **Interactions:**
  – Constituents must describe a binary tree
  – Tag bigrams
  – Nonterminal triples (parent, left-child, right-child)
    
    ![Diagram](https://via.placeholder.com/150)

    \[\text{these factors not shown}\]

Example Task: (Naradowsky, Vieira, & Smith, 2012)
Example Task:

Dependency Parsing

• Variables:
  – POS tag for each word
  – Syntactic label (or \(\emptyset\)) for each of \(O(n^2)\) possible directed arcs

• Interactions:
  – Arcs must form a tree
  – Discourage (or forbid) crossing edges
  – Features on edge pairs that share a vertex

• Learn to *discourage* a verb from having 2 objects, etc.

• Learn to *encourage* specific multi-arc constructions

(Smith & Eisner, 2008)
Joint CCG Parsing and Supertagging

- **Variables:**
  - Spans
  - Labels on non-terminals
  - Supertags on pre-terminals

- **Interactions:**
  - Spans must form a tree
  - Triples of labels: parent, left-child, and right-child
  - Adjacent tags

---

**Example Task:**

- Auli & Lopez (2011)
Example task: Transliteration or Back-Transliteration

• Variables (string):
  – English and Japanese orthographic strings
  – English and Japanese phonological strings

• Interactions:
  – All pairs of strings could be relevant
Example task: MORPHOLOGICAL PARADIGMS

- **Variables (string):**
  - Inflected forms of the same verb

- **Interactions:**
  - Between pairs of entries in the table (e.g. infinitive form affects present-singular)

(Dreyer & Eisner, 2009)
Word Alignment / Phrase Extraction

• **Variables (boolean):**
  – For each (Chinese phrase, English phrase) pair, are they linked?

• **Interactions:**
  – Word fertilities
  – Few “jumps” (discontinuities)
  – Syntactic reorderings
  – “ITG constraint” on alignment
  – Phrases are disjoint (?)

(Application: Burkett & Klein, 2012)
Congressional Voting

Application:

- Variables:
  - Representative’s vote
  - Text of all speeches of a representative
  - Local contexts of references between two representatives

- Interactions:
  - Words used by representative and their vote
  - Pairs of representatives and their local context

(Stoyanov & Eisner, 2012)
**Semantic Role Labeling with Latent Syntax**

**Variables:**
- Semantic predicate sense
- Semantic dependency arcs
- Labels of semantic arcs
- Latent syntactic dependency arcs

**Interactions:**
- Pairs of syntactic and semantic dependencies
- Syntactic dependency arcs must form a tree

(Naradowsky, Riedel, & Smith, 2012)
(Gormley, Mitchell, Van Durme, & Dredze, 2014)
Joint NER & Sentiment Analysis

• **Variables:**
  - Named entity spans
  - Sentiment directed toward each entity

• **Interactions:**
  - Words and entities
  - Entities and sentiment

(Application: love I Mark Twain PERSON POSITIVE)

(Mitchell, Aguilar, Wilson, & Van Durme, 2013)
Variable-centric view of the world

When we deeply understand language, what representations (type and token) does that understanding comprise?
To recover variables, model and exploit their correlations.
Section 2: Belief Propagation Basics
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.

2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.

3. **Intuitions:** What's going on here? Can we trust BP's estimates?

4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.

5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.

6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.

7. **Software:** Build the model you want!
Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years? Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.

2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.

3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?

4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.

5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.

6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.

7. **Software:** Build the model you want!
Factor Graph Notation

- Variables:
  \[ \mathcal{X} = \{ X_1, \ldots, X_i, \ldots, X_n \} \]

- Factors:
  \[ \psi_\alpha, \psi_\beta, \psi_\gamma, \ldots \]
  where \( \alpha, \beta, \gamma, \ldots \subseteq \{1, \ldots n\} \)

Joint Distribution

\[ p(x) = \frac{1}{Z} \prod_\alpha \psi_\alpha(x_\alpha) \]
Factors are Tensors

- Factors: $\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$
Inference

Given a factor graph, two common tasks ...

- Compute the most likely joint assignment,
  \[ x^* = \arg\max_x p(X=x) \]

- Compute the marginal distribution of variable \( X_i \):
  \[ p(X_i=x_i) \] for each value \( x_i \)

Both consider all joint assignments.

Both are NP-Hard in general.

So, we turn to approximations.

\[ p(X_i=x_i) = \text{sum of } p(X=x) \text{ over joint assignments with } X_i=x_i \]
Marginals by Sampling on Factor Graph

Suppose we took many samples from the distribution over taggings:

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

Sample 1:

\[ n \quad v \quad p \quad d \quad n \]

Sample 2:

\[ n \quad n \quad v \quad d \quad n \]

Sample 3:

\[ n \quad v \quad p \quad d \quad n \]

Sample 4:

\[ v \quad n \quad p \quad d \quad n \]

Sample 5:

\[ v \quad n \quad v \quad d \quad n \]

Sample 6:

\[ n \quad v \quad p \quad d \quad n \]

<START>
The marginal \( p(X_i = x_i) \) gives the probability that variable \( X_i \) takes value \( x_i \) in a random sample.
Marginals by Sampling on Factor Graph

Estimate the marginals as:

Sample 1:

Sample 2:

Sample 3:

Sample 4:

Sample 5:

Sample 6:

<START>
How do we get marginals without sampling?

That’s what Belief Propagation is all about!

Why not just sample?

• Sampling one joint assignment is also NP-hard in general.
  – In practice: Use MCMC (e.g., Gibbs sampling) as an anytime algorithm.
  – So draw an approximate sample fast, or run longer for a “good” sample.

• Sampling finds the high-probability values \( x_i \) efficiently.
  But it takes too many samples to see the low-probability ones.
  – How do you find \( p(“The quick brown fox ...”) \) under a language model?
    • Draw random sentences to see how often you get it? Takes a long time.
    • Or multiply factors (trigram probabilities)? That’s what BP would do.
Great Ideas in ML: Message Passing

Count the soldiers

there's 1 of me

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

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

Count the soldiers

there's 1 of me

2 before you

only see my incoming messages

Belief: Must be

2 + 1 + 3 = 6 of us

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

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 tree

Belief: Must be 14 of us

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

adapted from MacKay (2003) textbook
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

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

Variable Message

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

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

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

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

Matrix-vector product (for a binary factor)

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

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

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

Variables

Factors

Beliefs

Messages

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

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

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 ...

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

- START

- $X_1$
  - $v$
  - $n$
  - $a$

- $X_2$
  - $v$
  - $n$
  - $a$

- $X_3$
  - $v$
  - $n$
  - $a$

- END

- find
- preferred
- tags
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 = \text{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 \( p(X_2 = a) \)  
  \[ = (1/Z) \times \text{total weight of all paths through } a \]
Forward-Backward Algorithm: Finds Marginals

\[ p(\mathbf{v} \ a \ \mathbf{n}) = \frac{1}{Z} \] * product weight of one path

• Marginal probability \( p(X_2 = a) \)
  \[ = \frac{1}{Z} \] * total weight of \textit{all} paths through \( \mathbf{n} \)
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 \( p(X_2 = a) = (1/Z) \times \text{total weight of all paths through} \)
Forward-Backward Algorithm: Finds Marginals

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

(find 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.

“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

- **Preferred tags**: \(\psi\{2\}(v)\)
- **Belief that** \(X_2 = v\)**
- **Belief that** \(X_2 = n\)**

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

“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 $a$

$$= \alpha_2(a) \psi_{\{2\}}(a) \beta_2(a)$$

divide by $Z=6$ to get marginal probs
(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

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

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

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

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

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

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

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

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

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

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

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

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

**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

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

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

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

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

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
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.
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.
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.
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.
Loopy Belief Propagation

What if our graph has cycles?

- Messages from different subgraphs are no longer independent!
  - Dynamic programming can't help. It's now #P-hard in general to compute the exact marginals.

- But we can still run BP -- it's a local algorithm so it doesn't "see the cycles."
What can go wrong with loopy BP?

All 4 factors on cycle enforce equality

F

F

F

F
What can go wrong with loopy BP?

All 4 factors on cycle enforce equality

This factor says upper variable is twice as likely to be true as false (and that’s the true marginal!)
What can go wrong with loopy BP?

- Messages loop around and around ...
- 2, 4, 8, 16, 32, ... More and more convinced that these variables are T!
- So beliefs converge to marginal distribution (1, 0) rather than (2/3, 1/3).

- BP incorrectly treats this message as separate evidence that the variable is T.
- Multiplies these two messages as if they were independent.
  - But they don’t actually come from independent parts of the graph.
  - One influenced the other (via a cycle).

This is an extreme example. Often in practice, the cyclic influences are weak. (As cycles are long or include at least one weak correlation.)
A lie told often enough becomes truth. -- Lenin

What can go wrong with loopy BP?

Your prior doesn’t think Obama owns it. But everyone’s saying he does. Under a Naïve Bayes model, you therefore believe it.

A rumor is circulating that Obama secretly owns an insurance company. (Obamacare is actually designed to maximize his profit.)
What can go wrong with loopy BP?

Better model ... Rush can influence conversation.

- Now there are 2 ways to explain why everyone’s repeating the story: it’s true, or Rush said it was.
- The model favors one solution (probably Rush).
- Yet BP has 2 stable solutions. Each solution is self-reinforcing around cycles; no impetus to switch.

Actually 4 ways: but “both” has a low prior and “neither” has a low likelihood, so only 2 good ways.

If everyone blames Obama, then no one has to blame Rush. But if no one blames Rush, then everyone has to continue to blame Obama (to explain the gossip).

A lie told often enough becomes truth. -- Lenin
Loopy Belief Propagation Algorithm

- Run the BP update equations on a cyclic graph
  - Hope it “works” anyway (good approximation)
    - Though we multiply messages that aren’t independent
    - No interpretation as dynamic programming
  - If largest element of a message gets very big or small,
    - Divide the message by a constant to prevent over/underflow
- Can update messages in any order
  - Stop when the normalized messages converge
- Compute beliefs from final messages
  - Return normalized beliefs as approximate marginals

\[
p_i(x_i) \propto b_i(x_i) \quad p_\alpha(x_\alpha) \propto b_\alpha(x_\alpha)
\]

E.g., Murphy, Weiss & Jordan (1999)
Loopy Belief Propagation

**Input:** a factor graph with cycles  
**Output:** approximate marginals for each variable and factor

**Algorithm:**
1. Initialize the messages to the uniform distribution.
   \[
   \mu_{i\rightarrow\alpha}(x_i) = 1 \quad \mu_{\alpha\rightarrow i}(x_i) = 1
   \]
2. Send messages until convergence. Normalize them when they grow too large.
   \[
   \mu_{i\rightarrow\alpha}(x_i) = \prod_{\alpha \in N(i) \setminus \alpha} \mu_{\alpha\rightarrow i}(x_i) \\
   \mu_{\alpha\rightarrow i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in N(\alpha) \setminus i} \mu_{j\rightarrow\alpha}(x_\alpha[j])
   \]
3. Compute the beliefs (unnormalized marginals).
   \[
   b_i(x_i) = \prod_{\alpha \in N(i)} \mu_{\alpha\rightarrow i}(x_i) \\
   b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in N(\alpha)} \mu_{i\rightarrow\alpha}(x_\alpha[i])
   \]
4. Normalize beliefs and return the approximate marginals.
   \[
   p_i(x_i) \propto b_i(x_i) \\
   p_\alpha(x_\alpha) \propto b_\alpha(x_\alpha)
   \]
Section 2 Appendix

Tensor Notation for BP
In section 2, BP was introduced with a notation which defined messages and beliefs as functions.

This Appendix includes an alternate (and very concise) notation for the Belief Propagation algorithm using tensors.
Tensor Notation

• Tensor multiplication:

\[(A \otimes B)(W = w, X = x, Y = y) = A(W = w, X = x)B(X = x, Y = y)\]

• Tensor marginalization:

\[\left(\bigoplus^Y A\right)(Y = y) = \sum \sum A(W = w, X = x, Y = y)\]
Tensor Notation

A rank-r tensor is...

A real function with r keyword arguments

= Axis-labeled array with arbitrary indices

= Database with column headers

Tensor multiplication: (vector outer product)

\[(A \otimes B)(X = x, Y = y) = A(X = x)B(Y = y)\]
Tensor Notation

A rank-r tensor is…

- A real function with r keyword arguments
- Axis-labeled array with arbitrary indices
- Database with column headers

Tensor multiplication: (vector pointwise product)

\[(A \otimes B) (X = x) = A(X = x)B(X = x)\]
A rank-r tensor is... 

A real function with r keyword arguments = Axis-labeled array with arbitrary indices = Database with column headers

Tensor multiplication: (matrix-vector product)

\((A \otimes B)(X = x, Y = y) = A(X = x, Y = y)B(X = x)\)
Tensor Notation

A rank-\( r \) tensor is...

A real function with \( r \) keyword arguments = Axis-labeled array with arbitrary indices = Database with column headers

Tensor marginalization:

\[
\left( \bigoplus_Y A \right) \left( Y = y \right) = \sum_x A \left( X = x, Y = y \right)
\]
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.
   \[
   \mu_{i \rightarrow \alpha} = 1 \quad \mu_{\alpha \rightarrow i} = 1
   \]

2. Choose a root node.

3. Send messages from the **leaves** to the **root**.  
   Send messages from the **root** to the **leaves**.
   \[
   \mu_{i \rightarrow \alpha} = \bigotimes_{\beta \in \mathcal{N}(i) \setminus \alpha} \mu_{\beta \rightarrow i} \quad \mu_{\alpha \rightarrow i} = \bigoplus_{j \in \mathcal{N}(\alpha) \setminus i} \psi_{\alpha} \bigotimes_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}
   \]

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

5. 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**

- $X_1$
- $X_2$
- $X_3$

**Factors**

- $X_2$

**Beliefs**

- $\psi_1$
- $\psi_2$
- $\psi_3$

**Messages**

- $\psi_1$
- $\psi_2$
- $\psi_3$

Mathematical Formulas:

- For a variable $i$:
  \[ b_i = \bigotimes_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i} \]

- For a factor $\alpha$:
  \[ b_\alpha = \psi_\alpha \bigotimes_{i \in \mathcal{N}(\alpha)} \mu_{i \rightarrow \alpha} \]
Sum-Product Belief Propagation

Variables

Factors

Beliefs

Messages

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

\[ \mu_{\alpha \rightarrow i} = \bigoplus_i \psi_\alpha \otimes \bigotimes_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha} \]
Sum-Product Belief Propagation

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

Variable Message

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

Factor Belief

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

Factor Message

\[ \mu_{\alpha \rightarrow i} = \bigoplus_i^\alpha \psi_{\alpha} \otimes \bigotimes_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha} \]
Loopy Belief Propagation

**Input:** a factor graph with cycles

**Output:** approximate marginals for each variable and factor

**Algorithm:**

1. Initialize the messages to the uniform distribution.
   \[
   \mu_{i \rightarrow \alpha} = 1 \quad \mu_{\alpha \rightarrow i} = 1
   \]

2. Send messages until convergence. Normalize them when they grow too large.
   \[
   \mu_{i \rightarrow \alpha} = \bigotimes_{\beta \in N(i) \setminus \alpha} \mu_{\beta \rightarrow i} \quad \mu_{\alpha \rightarrow i} = \bigoplus_{i} \psi_{\alpha} \bigotimes_{j \in N(\alpha) \setminus i} \mu_{j \rightarrow \alpha}
   \]

3. Compute the beliefs (unnormalized marginals).
   \[
   b_{i} = \bigotimes_{\alpha \in N(i)} \mu_{\alpha \rightarrow i} \quad b_{\alpha} = \psi_{\alpha} \bigotimes_{i \in N(\alpha)} \mu_{i \rightarrow \alpha}
   \]

4. Normalize beliefs and return the approximate marginals.
   \[
   p_{i}(x_{i}) \propto b_{i}(x_{i}) \quad p_{\alpha}(x_{\alpha}) \propto b_{\alpha}(x_{\alpha})
   \]
Section 3:
Belief Propagation Q&A

Methods like BP and in what sense they work
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Q&A

Q: Forward-backward is to the Viterbi algorithm as sum-product BP is to __________?

A: max-product BP
Max-product Belief Propagation

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

• **Max-product BP** can be used to compute the most likely assignment, \( X^* = \operatorname{argmax}_X p(X) \)
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:

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

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>

4. Take resulting beliefs to power \( T \)
Q: This feels like **Arc Consistency**... Any relation?

A: Yes, BP is doing (with probabilities) what people were doing in AI long before.
From Arc Consistency to BP

Goal: Find a satisfying assignment
Algorithm: Arc Consistency
1. Pick a constraint
2. Reduce domains to satisfy the constraint
3. Repeat until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]
- \( X < Y \)
- \( Y = U \)
- \( T < U \)
- \( X < T \)

Note: These steps can occur in somewhat arbitrary order

Propagation completely solved the problem!
From Arc Consistency to BP

Goal: Find a satisfying assignment
Algorithm: Arc Consistency
1. Pick a constraint
2. Reduce domains to satisfy the constraint
3. Repeat until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]
\[ X < Y \]
\[ Y = U \]
\[ T < U \]
\[ X < T \]

Note: These steps can occur in somewhat arbitrary order

Arc Consistency is a special case of Belief Propagation.

Propagation completely solved the problem!

Slide thanks to Rina Dechter (modified)
From Arc Consistency to BP

Solve the same problem with BP
• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]
\[ X < Y \]
\[ Y = U \]
\[ T < U \]
\[ X < T \]
From Arc Consistency to BP

Solve the same problem with BP
• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]
- \[ X < Y \]
- \[ Y = U \]
- \[ T < U \]
- \[ X < T \]
From Arc Consistency to BP

Solve the same problem with BP
• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]
\[ X < Y \]
\[ Y = U \]
\[ T < U \]
\[ X < T \]
From Arc Consistency to BP

Solve the same problem with BP
• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]
\[ X < Y \]
\[ Y = U \]
\[ T < U \]
\[ X < T \]
From Arc Consistency to BP

Solve the same problem with BP
- Constraints become “hard” factors with only 1’s or 0’s
- Send messages until convergence

\[
X, Y, U, T \in \{1, 2, 3\}
\]

- \(X < Y\)
- \(Y = U\)
- \(T < U\)
- \(X < T\)

Slide thanks to Rina Dechter (modified)
From Arc Consistency to BP

Solve the same problem with BP

• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

$X, Y, U, T \in \{1, 2, 3\}$

$X < Y$
$Y = U$
$T < U$
$X < T$

Slide thanks to Rina Dechter (modified)
From Arc Consistency to BP

Solve the same problem with BP

• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]

\[ X < Y \]
\[ Y = U \]
\[ T < U \]
\[ X < T \]
From Arc Consistency to BP

Solve the same problem with BP

• Constraints become “hard” factors with only 1’s or 0’s
• Send messages until convergence

\[ X, Y, U, T \in \{1, 2, 3\} \]

\[ X < Y \]
\[ Y = U \]
\[ T < U \]
\[ X < T \]

Loopy BP will converge to the equivalent solution!
From Arc Consistency to BP

Takeaways:

- Arc Consistency is a special case of Belief Propagation.
- Arc Consistency will only rule out impossible values.
- BP rules out those same values (belief = 0).

Loopy BP will converge to the equivalent solution!

Slide thanks to Rina Dechter (modified)
Q: Is BP totally divorced from sampling?

A: Gibbs Sampling is also a kind of message passing algorithm.
From Gibbs Sampling to Particle BP to BP

Message Representation:

A. Belief Propagation: full distribution
B. Gibbs sampling: single particle
C. Particle BP: multiple particles
From Gibbs Sampling to Particle BP to BP
From Gibbs Sampling to Particle BP to BP

Approach 1: Gibbs Sampling

- For each variable, resample the value by conditioning on all the other variables
  - Called the “full conditional” distribution
  - Computationally easy because we really only need to condition on the Markov Blanket
- We can view the computation of the full conditional in terms of message passing
  - Message puts all its probability mass on the current particle (i.e. current value)
From Gibbs Sampling to Particle BP to BP

Approach 1:
- For each variable, resample its value by conditioning on all the other variables
  - Called the "full conditional" distribution
  - Computationally easy because we only need to condition on the Markov Blanket
- We can view the computation of the full conditional in terms of message passing
  - Message puts all its probability mass on the current particle (i.e. current value)
From Gibbs Sampling to Particle BP to BP

Approach 1:

- For each variable in the model, 
  - For each variable, resample the value by conditioning on all the other variables
  - Called the "full conditional" distribution
  - Computationally easy because we really only need to condition on the Markov Blanket
- We can view the computation of the full conditional in terms of message passing
  - Message puts all its probability mass on the current particle (i.e. current value)
From Gibbs Sampling to Particle BP to BP
From Gibbs Sampling to Particle BP to BP

Approach 2: Multiple Gibbs Samplers

- Run each Gibbs Sampler independently
- Full conditionals computed independently
  - \( k \) separate messages that are each a pointmass distribution
From Gibbs Sampling to Particle BP to BP

Approach 3: Gibbs Sampling w/Averaging

- Keep k samples for each variable
- Resample from the **average of the full conditionals** for each possible pair of variables
  - Message is a uniform distribution over current particles
### From Gibbs Sampling to Particle BP to BP

**Approach 3: Gibbs Sampling w/Averaging**

- Keep $k$ samples for each variable.
- Resample from the **average of the full conditionals** for each possible pair of variables.
  - Message is a uniform distribution over current particles.
From Gibbs Sampling to Particle BP to BP

Approach 3: Gibbs Sampling w/Averaging
• Keep $k$ samples for each variable
• Resample from the average of the full conditionals for each possible pair of variables
  – Message is a uniform distribution over current particles
From Gibbs Sampling to Particle BP to BP

Approach 4: Particle BP
- Similar in spirit to Gibbs Sampling w/Averaging
- Messages are a weighted distribution over \( k \) particles

(Ihler & McAllester, 2009)
From Gibbs Sampling to Particle BP to BP

Approach 5: BP
- In Particle BP, as the number of particles goes to $+\infty$, the estimated messages approach the true BP messages.
- Belief propagation represents messages as the full distribution.
  - This assumes we can store the whole distribution compactly.

(Ihler & McAllester, 2009)
From Gibbs Sampling to Particle BP to BP

Message Representation:

A. Belief Propagation: full distribution
B. Gibbs sampling: single particle
C. Particle BP: multiple particles
From Gibbs Sampling to Particle BP to BP

Tension between approaches...

**Sampling values or combinations of values:**
- quickly get a good estimate of the frequent cases
- may take a long time to estimate probabilities of infrequent cases
- may take a long time to draw a sample (mixing time)
- exact if you run forever

**Enumerating each value and computing its probability exactly:**
- have to spend time on all values
- but only spend $O(1)$ time on each value (don’t sample frequent values over and over while waiting for infrequent ones)
- runtime is more predictable
- lets you tradeoff exactness for greater speed (brute force exactly enumerates exponentially many assignments, BP approximates this by enumerating local configurations)
Background: Convergence

When BP is run on a tree-shaped factor graph, the beliefs converge to the marginals of the distribution after two passes.
Q: How long does loopy BP take to converge?

Q: When loopy BP converges, does it always get the same answer?

A: No. Sensitive to initialization and update order.
Q: Are there convergent variants of loopy BP?

A: Yes. It's actually trying to minimize a certain differentiable function of the beliefs, so you could just minimize that function directly.
Q: But does that function have a unique minimum?

A: No, and you'll only be able to find a local minimum in practice. So you're still dependent on initialization.
Q: If you could find the global minimum, would its beliefs give the marginals of the true distribution?

A: No.
Q: Is it finding the marginals of some other distribution (as mean field would)?

A: No, just a collection of beliefs.

Might not be globally consistent in the sense of all being views of the same elephant.

*Cartoon by G. Renee Guzlas*
Q: Does the global minimum give beliefs that are at least **locally consistent**?

A: Yes.

A variable belief and a factor belief are **locally consistent** if the marginal of the factor’s belief equals the variable’s belief.

\[ b_i(x_i) = \sum_{x_\alpha \setminus x_i} b_\alpha(x_\alpha), \quad \forall i, \alpha \in \mathcal{N}(i) \]
Q: In what sense are the beliefs at the global minimum any good?

A: They are the global minimum of the Bethe Free Energy.
Q: When loopy BP **converges**, in what sense are the **beliefs** any good?

A: They are a **local minimum** of the Bethe Free Energy.
Q&A

**Q:** Why would you want to minimize the Bethe Free Energy?

**A:**

1) It’s easy to minimize* because it’s a sum of functions on the individual beliefs.

2) On an *acyclic* factor graph, it measures KL divergence between beliefs and true marginals, and so is minimized when beliefs = marginals. (For a *loopy* graph, we close our eyes and hope it still works.)

[*] Though we can’t just minimize each function separately – we need message passing to keep the beliefs locally consistent.
Section 3: Appendix

BP as an Optimization Algorithm
BP as an Optimization Algorithm

This Appendix provides a more in-depth study of BP as an optimization algorithm.


We also include a discussion of the convergence properties of max-product BP.
KL and Free Energies

Kullback–Leibler (KL) divergence

$$KL(b||p) = \sum_x b(x) \log \left[ \frac{b(x)}{p(x)} \right]$$

$$= \sum_x b(x) \log \left[ \frac{b(x)}{\prod_\alpha \psi_\alpha(x_\alpha)} \right] + \log Z$$

Gibbs Free Energy

$$F(b) = KL(b||p) - \log Z = \sum_x b(x) \log \left[ \frac{b(x)}{\prod_\alpha \psi_\alpha(x_\alpha)} \right]$$

Helmholtz Free Energy

$$F_H = -\log Z = \min_b F(b)$$
Minimizing KL Divergence

• If we find the distribution $b$ that minimizes the KL divergence, then $b = p$

$$p(x) = \arg\min_b \text{KL}(b||p)$$

$$= \arg\min_b \sum_x b(x) \log \left[ \frac{b(x)}{\prod_\alpha \psi_\alpha(x_\alpha)} \right]$$

• Also, true of the minimum of the Gibbs Free Energy

• But what if $b$ is not (necessarily) a probability distribution?
BP on a 2 Variable Chain

True distribution:

$$p(x, y) = \frac{\psi_1(x, y)}{Z}$$

Beliefs at the end of BP:

$$b(x, y) = \frac{\psi_1(x, y)}{Z}$$

$$b(x) \propto \sum_y \psi_1(x, y)$$

$$b(y) \propto \sum_x \psi_1(x, y)$$

We successfully minimized the KL divergence!

$$p(x) = \arg\min_b KL(b||p)$$

*where U(x) is the uniform distribution*
BP on a 3 Variable Chain

True distribution:

$$p(w, x, y) = \frac{\psi_1(w, x)\psi_2(x, y)}{Z}$$

The true distribution can be expressed in terms of its **marginals**:

$$p(w, x, y) = p(w|x)p(x, y) = \frac{p(w, x)p(x, y)}{p(x)}$$

Define the **joint belief** to have the same form:

$$b(w, x, y) := \frac{b(w, x)b(x, y)}{b(x)}$$

KL decomposes over the marginals:

$$\text{KL}(b||p) = \sum_{w,x,y} b(w, x, y) \log \left[ \frac{b(w, x, y)}{p(w, x, y)} \right]$$

$$= \sum_{w,x} b(w, x) \log \left[ \frac{b(w, x)}{\psi_1(w, x)} \right]$$

$$+ \sum_{x,y} b(x, y) \log \left[ \frac{b(x, y)}{\psi_2(x, y)} \right]$$

$$- \sum_x b(x) \log b(x) + \log Z$$
BP on a 3 Variable Chain

True distribution:

\[ p(w, x, y) = \frac{\psi_1(w, x)\psi_2(x, y)}{Z} \]

The true distribution can be expressed in terms of its **marginals**:

\[ p(w, x, y) = p(w|x)p(x, y) = \frac{p(w, x)p(x, y)}{p(x)} \]

Define the **joint belief** to have the same form:

\[ b(w, x, y) := \frac{b(w, x)b(x, y)}{b(x)} \]

\[ F(b) = \sum_{w,x} b(w, x) \log \left( \frac{b(w, x)}{\psi_1(w, x)} \right) + \sum_{x,y} b(x, y) \log \left( \frac{b(x, y)}{\psi_2(x, y)} \right) - \sum_x b(x) \log b(x) \]

**Gibbs Free Energy** decomposes over the marginals.
BP on an Acyclic Graph

True distribution:

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

The true distribution can be expressed in terms of its marginals:

\[ p(x) = \frac{\prod_{\alpha} p(x_{\alpha})}{\prod_{i} p(x_i)^{N_i-1}} \]

Define the joint belief to have the same form:

\[ b(x) := \frac{\prod_{\alpha} b_{\alpha}(x_{\alpha})}{\prod_{i} b_i(x_i)^{N_i-1}} \]

\[ \text{KL}(b|p) = \sum_x b(x) \log \left[ \frac{b(x)}{p(x)} \right] \]

\[ = \sum_\alpha \sum_{x_\alpha} b_\alpha(x_\alpha) \log \left[ \frac{b_\alpha(x_\alpha)}{\psi_\alpha(x_\alpha)} \right] \]

\[ - \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i) + \log Z \]

KL decomposes over the marginals
BP on an Acyclic Graph

True distribution:

\[ p(\mathbf{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\mathbf{x}_\alpha) \]

The true distribution can be expressed in terms of its **marginals**:

\[ p(\mathbf{x}) = \frac{\prod_{\alpha} p(\mathbf{x}_\alpha)}{\prod_i p(x_i)^{N_i-1}} \]

Define the **joint belief** to have the same form:

\[ b(\mathbf{x}) := \frac{\prod_{\alpha} b_\alpha(\mathbf{x}_\alpha)}{\prod_i b_i(x_i)^{N_i-1}} \]

Gibbs Free Energy decomposes over the marginals:

\[ F(b) = \sum_{\alpha} \sum_{\mathbf{x}_\alpha} b_\alpha(\mathbf{x}_\alpha) \log \left[ \frac{b_\alpha(\mathbf{x}_\alpha)}{\psi_\alpha(\mathbf{x}_\alpha)} \right] \]
\[ - \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i) \]
BP on a Loopy Graph

True distribution:
\[ p(x) = \frac{1}{Z} \prod_{\alpha} \psi_{\alpha}(x_\alpha) \]

Construct the **joint belief** as before:
\[ b(x) := \frac{\prod_{\alpha} b_{\alpha}(x_\alpha)}{\prod_i b_i(x_i)^{N_i-1}} \]

This might **not** be a distribution!

So add **constraints**...

1. The beliefs are distributions: are non-negative and sum-to-one.
2. The beliefs are locally consistent:
\[ b_i(x_i) = \sum_{x_\alpha \setminus x_i} b_{\alpha}(x_\alpha), \quad \forall i, \alpha \in \mathcal{N}(i) \]

**KL is no longer well defined**, because the **joint belief** is not a proper distribution.

**KL**
\[ KL(b||p) = \sum_{w,x,y} b(w,x,y) \log \left[ \frac{b(w,x,y)}{p(w,x,y)} \right] \]
BP on a Loopy Graph

True distribution:

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

Construct the joint belief as before:

\[ b(x) := \frac{\prod_{\alpha} b_{\alpha}(x_{\alpha})}{\prod_i b_i(x_i)^{N_i - 1}} \]

This might **not** be a distribution!

So add constraints...

1. The beliefs are distributions: are non-negative and sum-to-one.
2. The beliefs are locally consistent:

\[ b_i(x_i) = \sum_{x_{\alpha} \setminus x_i} b_{\alpha}(x_{\alpha}), \quad \forall i, \alpha \in \mathcal{N}(i) \]

But we can still optimize the same objective as before, subject to our belief constraints:

\[
F_{\text{Bethe}}(b) = \sum_{\alpha} \sum_{x_{\alpha}} b_{\alpha}(x_{\alpha}) \log \left[ \frac{b_{\alpha}(x_{\alpha})}{\psi_{\alpha}(x_{\alpha})} \right] - \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)
\]

This is called the **Bethe Free Energy** and decomposes over the marginals
BP as an Optimization Algorithm

• The **Bethe Free Energy**, a function of the beliefs:

\[
F_{\text{Bethe}}(b) = \sum_{\alpha} \sum_{x_{\alpha}} b_{\alpha}(x_{\alpha}) \log \left[ \frac{b_{\alpha}(x_{\alpha})}{\psi_{\alpha}(x_{\alpha})} \right] - \sum_{i} (N_{i} - 1) \sum_{x_{i}} b_{i}(x_{i}) \log b_{i}(x_{i})
\]

• BP minimizes a **constrained** version of the Bethe Free Energy
  – BP is just one local optimization algorithm: fast but not guaranteed to converge
  – If BP converges, the beliefs are called **fixed points**
  – The **stationary points** of a function have a gradient of zero

The **fixed points** of BP are local **stationary points** of the Bethe Free Energy (Yedidia, Freeman, & Weiss, 2000)
BP as an Optimization Algorithm

• The **Bethe Free Energy**, a function of the beliefs:

\[
F_{\text{Bethe}}(b) = \sum_{\alpha} \sum_{x_\alpha} b_{\alpha}(x_\alpha) \log \left[ \frac{b_{\alpha}(x_\alpha)}{\psi_\alpha(x_\alpha)} \right] \\
- \sum_{i} (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)
\]

• BP minimizes a **constrained** version of the Bethe Free Energy
  – BP is just one local optimization algorithm: fast but not guaranteed to converge
  – If BP converges, the beliefs are called **fixed points**
  – The **stationary points** of a function have a gradient of zero

The **stable fixed points** of BP are local **minima** of the Bethe Free Energy (Heskes, 2003)
BP as an Optimization Algorithm

For graphs with no cycles:
- The minimizing beliefs = the true marginals
- BP finds the global minimum of the Bethe Free Energy
- This **global minimum** is $-\log Z$ (the “Helmholtz Free Energy”)

For graphs with cycles:
- The minimizing beliefs only **approximate** the true marginals
- Attempting to minimize may get stuck at **local minimum** or other critical point
- Even the global minimum only approximates $-\log Z$
Convergence of Sum-product BP

- The fixed point beliefs:
  - Do not necessarily correspond to marginals of any joint distribution over all the variables (Mackay, Yedidia, Freeman, & Weiss, 2001; Yedidia, Freeman, & Weiss, 2005)

- Unbelievable probabilities
  - Conversely, the true marginals for many joint distributions cannot be reached by BP (Pitkow, Ahmadian, & Miller, 2011)

The figure shows a two-dimensional slice of the Bethe Free Energy for a binary graphical model with pairwise interactions

Figure adapted from (Pitkow, Ahmadian, & Miller, 2011)
Convergence of Max-product BP

If the max-marginals $b_i(x_i)$ are a fixed point of BP, and $x^*$ is the corresponding assignment (assumed unique), then $p(x^*) > p(x)$ for every $x \neq x^*$ in a rather large neighborhood around $x^*$ (Weiss & Freeman, 2001).

The **neighbors of $x^*$** are constructed as follows: For any set of vars $S$ of **disconnected trees and single loops**, set the variables in $S$ to arbitrary values, and the rest to $x^*$.

Informally: If you take the fixed-point solution $x^*$ and arbitrarily change the values of the dark nodes in the figure, the overall probability of the configuration will decrease.

Figure from (Weiss & Freeman, 2001)
Convergence of Max-product BP

If the max-marginals $b_i(x_i)$ are a fixed point of BP, and $x^*$ is the corresponding assignment (assumed unique), then $p(x^*) > p(x)$ for every $x \neq x^*$ in a rather large neighborhood around $x^*$ (Weiss & Freeman, 2001).

The neighbors of $x^*$ are constructed as follows: For any set of vars $S$ of disconnected trees and single loops, set the variables in $S$ to arbitrary values, and the rest to $x^*$.

Informally: If you take the fixed-point solution $x^*$ and arbitrarily change the values of the dark nodes in the figure, the overall probability of the configuration will decrease.

Figure from (Weiss & Freeman, 2001)
Convergence of Max-product BP

If the max-marginals $b_i(x_i)$ are a fixed point of BP, and $x^*$ is the corresponding assignment (assumed unique), then $p(x^*) > p(x)$ for every $x \neq x^*$ in a rather large neighborhood around $x^*$ (Weiss & Freeman, 2001).

The neighbors of $x^*$ are constructed as follows: For any set of vars $S$ of disconnected trees and single loops, set the variables in $S$ to arbitrary values, and the rest to $x^*$.

Informally: If you take the fixed-point solution $x^*$ and arbitrarily change the values of the dark nodes in the figure, the overall probability of the configuration will decrease.

Figure from (Weiss & Freeman, 2001)
Section 4:
Incorporating Structure into Factors and Variables
Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What's going on here? Can we trust BP's estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?

• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.

2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.

3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?

4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.

5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.

6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.

7. **Software:** Build the model you want!
BP for Coordination of Algorithms

• Each factor is tractable by dynamic programming
• Overall model is no longer tractable, but BP lets us pretend it is

parse tagger

the white house

blanca
casa
la

aligner

tagger

parser
BP for Coordination of Algorithms

• Each factor is tractable by dynamic programming
• Overall model is no longer tractable, but BP lets us pretend it is
Sending Messages: Computational Complexity

From Variables

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

To Variables

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

**O(d * k)**
- \( d \) = # of neighboring factors
- \( k \) = # possible values for \( X_i \)

**O(d * k^d)**
- \( d \) = # of neighboring variables
- \( k \) = maximum # possible values for a neighboring variable
Sending Messages: Computational Complexity

From Variables

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

\( O(d*k) \)
- \( d \) = # of neighboring factors
- \( k \) = # possible values for \( X_i \)

To Variables

\( O(d*k^d) \)
- \( d \) = # of neighboring variables
- \( k \) = maximum # possible values for a neighboring variable
INCORPORATING STRUCTURE INTO FACTORS
Unlabeled Constituency Parsing

**Given:** a sentence.

**Predict:** unlabeled parse.

We could predict whether each span is present T or not F.

(time flies like an arrow)

(Naradowsky, Vieira, & Smith, 2012)
Unlabeled Constituency Parsing

**Given:** a sentence.

**Predict:** unlabeled parse.

We could predict whether each span is present T or not F.

(Naradowsky, Vieira, & Smith, 2012)
Unlabeled Constituency Parsing

**Given:** a sentence.
**Predict:** unlabeled parse.

We could predict whether each span is present $T$ or not $F$.

(Naradowsky, Vieira, & Smith, 2012)
**Unlabeled Constituency Parsing**

**Given:** a sentence.

**Predict:** unlabeled parse.

We could predict whether each span is present T or not F.

(Naradowsky, Vieira, & Smith, 2012)
Unlabeled Constituency Parsing

**Given:** a sentence.
**Predict:** unlabeled parse.

We could predict whether each span is present \( T \) or not \( F \).

(Naradowsky, Vieira, & Smith, 2012)
Given: a sentence.
Predict: unlabeled parse.

We could predict whether each span is present $T$ or not $F$.

(Naradowsky, Vieira, & Smith, 2012)
Given: a sentence.
Predict: unlabeled parse.

We could predict whether each span is present $T$ or not $F$.

Sending a message to a variable from its unary factors takes only $O(d^*k^d)$ time where $k=2$ and $d=1$.

(Naradowsky, Vieira, & Smith, 2012)
Unlabeled Constituency Parsing

*Given:* a sentence.

*Predict:* unlabeled parse.

We could predict whether each span is present $T$ or not $F$.

But nothing prevents non-tree structures.

Sending a message to a variable from its unary factors takes only $O(d^*k^d)$ time where $k=2$ and $d=1$.

(Naradowsky, Vieira, & Smith, 2012)
Unlabeled Constituency Parsing

**Given:** a sentence.
**Predict:** unlabeled parse.

We could predict whether each span is present $T$ or not $F$.

But nothing prevents non-tree structures.

Add a $CKYTree$ factor which multiplies in $1$ if the variables form a tree and $0$ otherwise.

(Naradowsky, Vieira, & Smith, 2012)
Given: a sentence.
Predict: unlabeled parse.

We could predict whether each span is present T or not F.

But nothing prevents non-tree structures.

Add a CKYTree factor which multiplies in 1 if the variables form a tree and 0 otherwise.

(Naradowsky, Vieira, & Smith, 2012)
Unlabeled Constituency Parsing

How long does it take to send a message to a variable from the the CKYTree factor?

For the given sentence, $O(d^*k^d)$ time where $k=2$ and $d=15$.

For a length $n$ sentence, this will be $O(2^{n^*})$.

But we know an algorithm (inside-outside) to compute all the marginals in $O(n^3)$.

So can’t we do better?

Add a CKYTree factor which multiplies in 1 if the variables form a tree and 0 otherwise.

(Naradowsky, Vieira, & Smith, 2012)
Example: The *Exactly1* Factor

**Variables:** $d$ binary variables $X_1, \ldots, X_d$

**Global Factor:** $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is on}, \\ 0 & \text{otherwise} \end{cases}$

(Smith & Eisner, 2008)
Example: The *Exactly 1* Factor

**Variables:** $d$ binary variables $X_1, \ldots, X_d$

**Global Factor:**

$$\text{Exactly 1}(X_1, \ldots, X_d) = \begin{cases} 
1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is on}, \\
0 & \text{otherwise}
\end{cases}$$

(Smith & Eisner, 2008)
Example: The *Exactly1* Factor

**Variables:** $d$ binary variables $X_1, \ldots, X_d$

**Global Factor:** $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is on,} \\ 0 & \text{otherwise} \end{cases}$

(Smith & Eisner, 2008)
Example: The *Exactly*1 Factor

**Variables:** $d$ binary variables $X_1, \ldots, X_d$

**Global Factor:** $Exactly 1(X_1, \ldots, X_d) = \begin{cases} 1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is on,} \\ 0 & \text{otherwise} \end{cases}$

(Smith & Eisner, 2008)
Example: The *Exactly1* Factor

**Variables:** $d$ binary variables $X_1, \ldots, X_d$

**Global Factor:** $\text{Exactly1}(X_1, \ldots, X_d) = \begin{cases} 
1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is on,} \\
0 & \text{otherwise}
\end{cases}$
Example: The *Exactly1* Factor

**Variables:** $d$ binary variables $X_1, \ldots, X_d$

**Global Factor:** $\text{Exactly1}(X_1, \ldots, X_d) = \begin{cases} 
1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is on,} \\
0 & \text{otherwise}
\end{cases}$

(Smith & Eisner, 2008)
Messages: The *Exactly1* Factor

From Variables

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

\(O(d*2)\)

d = # of neighboring factors

To Variables

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

\(O(d*2^d)\)

d = # of neighboring variables
Messages: The *Exactly1* Factor

From Variables

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

\[ O(d^*2) \]
\( d = \# \text{ of neighboring factors} \)

To Variables

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

\[ O(d^*2^d) \]
\( d = \# \text{ of neighboring variables} \)

Fast!
Messages: The *Exactly1* Factor

To Variables

\[ \psi_{E1} \]

\[ X_1 \quad X_2 \quad X_3 \quad X_4 \]

\[ \psi_1 \quad \psi_2 \quad \psi_3 \quad \psi_4 \]

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

\[ O(d*2^d) \]

\[ d = \# \text{ of neighboring variables} \]
Messages: The Exactly1 Factor

But the **outgoing** messages from the Exactly1 factor are defined as a sum over the $2^d$ possible assignments to $X_1, \ldots, X_d$.

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

Conveniently, $\psi_{E1}(x_a)$ is 0 for all but $d$ values – so **the sum is sparse**!

So we can compute all the outgoing messages from $\psi_{E1}$ in $O(d)$ time!
Messages: The *Exactly1* Factor

But the **outgoing** messages from the *Exactly1* factor are defined as a sum over the $2^d$ possible assignments to $X_1, \ldots, X_d$.

\[
\mu_{\alpha \rightarrow i}(x_i) = \sum_{x_\alpha : x_i = 1} \psi_\alpha(x_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \rightarrow \alpha}(x_\alpha[i])
\]

Conveniently, $\psi_{E1}(x_a)$ is 0 for all but $d$ values – so the sum is sparse!

So we can compute all the outgoing messages from $\psi_{E1}$ in $O(d)$ time!

Fast!

$O(d \times 2^d)$

$d = \# \text{ of neighboring variables}$
Messages: The *Exactly1* Factor

A factor has a belief about each of its variables.

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

An outgoing message from a factor is the factor's belief with the incoming message divided out.

\[
\mu_{\alpha \rightarrow i}(v) = \frac{b_\alpha(x_i)}{\mu_{i \rightarrow \alpha}(v)}
\]

We can compute the Exactly1 factor’s beliefs about each of its variables efficiently. (Each of the parenthesized terms needs to be computed only once for all the variables.)

\[
b_\alpha(X_i = 1) = \left( \prod_{j \in \mathcal{N}(\alpha)} \mu_{j \rightarrow \alpha}(0) \right) \frac{\mu_{i \rightarrow \alpha}(1)}{\mu_{i \rightarrow \alpha}(0)}
\]

\[
b_\alpha(X_i = 0) = \left( \sum_{j=1}^{n} b_\alpha(X_j = 1) \right) - b_\alpha(X_i = 1)
\]

*(Smith & Eisner, 2008)*
Example: The CKYTree Factor

Variables: $O(n^2)$ binary variables $S_{ij}$

Global Factor: $CKYTree(S_{01}, S_{12}, \ldots, S_{04}) = \begin{cases} 1 & \text{if the span variables form a constituency tree,} \\ 0 & \text{otherwise} \end{cases}$

(Naradowsky, Vieira, & Smith, 2012)
Messages: The CKYTree Factor

From Variables

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

$$O(d*2)$$

$d = \# \text{ of neighboring factors}$

To Variables

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

$$O(d*2^d)$$

$d = \# \text{ of neighboring variables}$
Messages: The CKYTree Factor

From Variables

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

Fast!

\[ O(d \times 2) \]
\[ d = \# \text{ of neighboring factors} \]

To Variables

\[ O(d \times 2^d) \]
\[ d = \# \text{ of neighboring variables} \]
Messages: The CKYTree Factor

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

\[ O(d \times 2^d) \]

\( d = \# \) of neighboring variables
Messages: The *CKYTree* Factor

But the **outgoing** messages from the *CKYTree* factor are defined as a sum over the $O(2^{n^2})$ possible assignments to $\{S_{ij}\}$.

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

$\psi_{CKYTree}(x_\alpha)$ is 1 for exponentially many values in the sum – **but they all correspond to trees**!

With inside-outside we can compute all the outgoing messages from *CKYTree* in $O(n^3)$ time!

$d = \# \text{ of neighboring variables}$

$O(d \times 2^d)$
But the **outgoing** messages from the CKYTree factor are defined as a sum over the $O(2^{n \times n})$ possible assignments to $\{S_{ij}\}$.

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

$\psi_{CKYTree}(x_a)$ is 1 for exponentially many values in the sum – **but they all correspond to trees**!

With inside-outside we can compute all the outgoing messages from CKYTree in $O(n^3)$ time!

$O(d \times 2^d)$

$d = \# \text{ of neighboring variables}$
Example: The CKYTree Factor

For a length $n$ sentence, define an anchored weighted context free grammar (WCFG).

Each span is weighted by the ratio of the incoming messages from the corresponding span variable.

$$w(iX_j \rightarrow iX_k \ kX_j) = \frac{\mu_{S_{ij} \rightarrow \psi}(1)}{\mu_{S_{ij} \rightarrow \psi}(0)}$$

$$w(iX_{i+1} \rightarrow a_{i+1}) = \frac{\mu_{S_{i,i+1} \rightarrow \psi}(1)}{\mu_{S_{i,i+1} \rightarrow \psi}(0)}$$

Run the inside-outside algorithm on the sentence $a_1, a_1, \ldots, a_n$ with the anchored WCFG.

$$\mu_{S_{ij} \rightarrow \psi}(1) = \frac{\text{outside}(iX_j)}{\text{inside}(0X_n)}$$

$$\mu_{S_{ij} \rightarrow \psi}(0) = 1 - w(iX_j \rightarrow iX_k \ kX_j) \frac{\text{outside}(iX_j)}{\text{inside}(0X_n)}$$

(Naradowsky, Vieira, & Smith, 2012)
Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.
Consider the joint distribution of a trigram HMM over 5 variables:

– It’s traditionally defined as a Bayes Network

(Smith & Eisner, 2008)
Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.
Consider the joint distribution of a trigram HMM over 5 variables:
- It’s traditionally defined as a Bayes Network
- But we can represent it as a (loopy) factor graph

(Smith & Eisner, 2008)
Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.

Consider the joint distribution of a trigram HMM over 5 variables:
- It’s traditionally defined as a Bayes Network
- But we can represent it as a (loopy) factor graph
- We could even pack all those factors into a single *TrigramHMM* factor

(Smith & Eisner, 2008)
Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.

Consider the joint distribution of a trigram HMM over 5 variables:
- It’s traditionally defined as a Bayes Network
- But we can represent it as a (loopy) factor graph
- We could even pack all those factors into a single *TrigramHMM* factor

(Smith & Eisner, 2008)
Example: The *TrigramHMM* Factor

**Variables:** $d$ discrete variables $X_1, \ldots, X_d$

**Global Factor:** $\text{TrigramHMM} (X_1, \ldots, X_d) = p(X_1, \ldots, X_d)$ according to a trigram HMM model

\[ X_1 \leadsto X_2 \leadsto X_3 \leadsto X_4 \leadsto X_5 \]

- $X_1$: time
- $X_2$: flies
- $X_3$: like
- $X_4$: an
- $X_5$: arrow

(Source: Smith & Eisner, 2008)
**Example: The *TrigramHMM* Factor**

**Variables:** $d$ discrete variables $X_1, \ldots, X_d$

**Global Factor:** $TrigramHMM (X_1, \ldots, X_d) = p(X_1, \ldots, X_d)$ according to a trigram HMM model

Compute outgoing messages **efficiently** with the standard trigram HMM dynamic programming algorithm (junction tree)!

(Smith & Eisner, 2008)
Combinatorial Factors

• Usually, it takes $O(k^d)$ time to compute outgoing messages from a factor over $d$ variables with $k$ possible values each.

• But not always:
  1. Factors like $\text{Exactly1}$ with only polynomially many nonzeroes in the potential table
  2. Factors like $\text{CKYTree}$ with exponentially many nonzeroes but in a special pattern
  3. Factors like $\text{TrigramHMM}$ with all nonzeroes but which factor further
Combinatorial Factors

Factor graphs can encode structural constraints on many variables via constraint factors.

Example NLP constraint factors:

- Projective and non-projective dependency parse constraint (Smith & Eisner, 2008)
- **CCG parse** constraint (Auli & Lopez, 2011)
- Labeled and unlabeled constituency parse constraint (Naradowsky, Vieira, & Smith, 2012)
- Inversion transduction grammar (ITG) constraint (Burkett & Klein, 2012)
Combinatorial Optimization within Max-Product

• **Max-product BP** computes **max-marginals**.
  – The max-marginal $b_i(x_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $X_i = x_i$.

• Duchi et al. (2006) define factors, over many variables, for which efficient combinatorial optimization algorithms exist.
  – **Bipartite matching**: max-marginals can be computed with standard max-flow algorithm and the Floyd-Warshall all-pairs shortest-paths algorithm.
  – **Minimum cuts**: max-marginals can be computed with a min-cut algorithm.

• Similar to sum-product case: the combinatorial algorithms are **embedded** within the standard loopy BP algorithm.

(Duchi, Tarlow, Elidan, & Koller, 2006)
Structured BP vs. Dual Decomposition

<table>
<thead>
<tr>
<th></th>
<th>Sum-product BP</th>
<th>Max-product BP</th>
<th>Dual Decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td>Approximate marginals</td>
<td>Approximate MAP assignment</td>
<td>True MAP assignment (with branch-and-bound)</td>
</tr>
<tr>
<td><strong>Structured Variant</strong></td>
<td>Coordinates marginal inference algorithms</td>
<td>Coordinates MAP inference algorithms</td>
<td>Coordinates MAP inference algorithms</td>
</tr>
<tr>
<td><strong>Example Embedded Algorithms</strong></td>
<td>- Inside-outside - Forward-backward</td>
<td>- CKY - Viterbi algorithm</td>
<td>- CKY - Viterbi algorithm</td>
</tr>
</tbody>
</table>

(Koo et al., 2010; Rush et al., 2010)

(Duchi, Tarlow, Elidan, & Koller, 2006)
Additional Resources

See **NAACL 2012 / ACL 2013 tutorial** by Burkett & Klein “Variational Inference in Structured NLP Models” for...

- An alternative approach to efficient marginal inference for NLP: **Structured Mean Field**
- Also, includes **Structured BP**

Sending Messages: Computational Complexity

From Variables

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

To Variables

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

\( \mathcal{O}(d \times k) \)
- \( d = \# \) of neighboring factors
- \( k = \# \) possible values for \( X_i \)

\( \mathcal{O}(d \times k^d) \)
- \( d = \# \) of neighboring variables
- \( k = \) maximum \( \# \) possible values for a neighboring variable
Sending Messages: Computational Complexity

From Variables

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

$O(d^*k)$

$d = \# \text{ of neighboring factors}$

$k = \# \text{ possible values for } X_i$

To Variables

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

$O(d^*k^d)$

$d = \# \text{ of neighboring variables}$

$k = \text{ maximum } \# \text{ possible values for a neighboring variable}$
INCORPORATING STRUCTURE INTO VARIABLES
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is
BP for Coordination of Algorithms

• Each factor is tractable by dynamic programming
• Overall model is no longer tractable, but BP lets us pretend it is
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is
String-Valued Variables

Consider two examples from Section 1:

- **Variables (string):**
  - English and Japanese orthographic strings
  - English and Japanese phonological strings

- **Interactions:**
  - All pairs of strings could be relevant

- **Variables (string):**
  - Inflected forms of the same verb

- **Interactions:**
  - Between pairs of entries in the table (e.g. infinitive form affects present-singular)
Graphical Models over Strings

• Most of our problems so far:
  – Used **discrete** variables
  – Over a small finite set of **string values**
  – Examples:
    • POS tagging
    • Labeled constituency parsing
    • Dependency parsing

• We use **tensors** (e.g. vectors, matrices) to represent the messages and factors
Graphical Models over Strings

Time Complexity:
var. $\rightarrow$ fac. $O(d^d k^d)$
fac. $\rightarrow$ var. $O(d^d k)$

What happens as the # of possible values for a variable, $k$, increases?

We can still keep the computational complexity down by including only low arity factors (i.e. small $d$).

(Dreyer & Eisner, 2009)
Graphical Models over Strings

But what if the domain of a variable is $\Sigma^*$, the infinite set of all possible strings?

How can we represent a distribution over one or more infinite sets?

(Dreyer & Eisner, 2009)
Graphical Models over Strings

Finite State Machines let us represent something infinite in finite space!

(Dreyer & Eisner, 2009)
Graphical Models over Strings

messages and beliefs are Weighted Finite State Acceptors (WFSA)

factors are Weighted Finite State Transducers (WFST)

Finite State Machines let us represent something infinite in finite space!

(Dreyer & Eisner, 2009)
Graphical Models over Strings

That solves the problem of representation.

But how do we manage the problem of computation? (We still need to compute messages and beliefs.)

Finite State Machines let us represent something infinite in finite space!

mes\text{sages}\text{ and beliefs} are Weighted Finite State Acceptors (WFSA)

factors are Weighted Finite State Transducers (WFST)

(Dreyer & Eisner, 2009)
Graphical Models over Strings

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

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

All the message and belief computations simply reuse standard FSM dynamic programming algorithms.

(Dreyer & Eisner, 2009)
Graphical Models over Strings

The pointwise product of two WFSAs is...

... their intersection.

Compute the product of (possibly many) messages $\mu_{\alpha \rightarrow i}$ (each of which is a WSFA) via WFSA intersection

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

(Dreyer & Eisner, 2009)
Graphical Models over Strings

Compute marginalized product of WFSA message $\mu_{k \rightarrow \alpha}$ and WFST factor $\psi_\alpha$, with:

- domain-compose($\psi_\alpha, \mu_{k \rightarrow \alpha}$))
- compose: produces a new WFST with a distribution over $(X_i, X_j)$
- domain: marginalizes over $X_j$ to obtain a WFSA over $X_i$ only

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

(Dreyer & Eisner, 2009)
Graphical Models over Strings

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

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

All the message and belief computations simply reuse standard FSM dynamic programming algorithms.

(Dreyer & Eisner, 2009)
The usual NLP toolbox

- **WFSA**: weighted finite state automata
- **WFST**: weighted finite state transducer
- **k-tape WFSM**: weighted finite state machine jointly mapping between k strings

They each assign a score to a set of strings.

We can interpret them as factors in a graphical model.

The only difference is the **arity** of the factor.
WFSA as a Factor Graph

- **WFSA**: weighted finite state automata
- **WFST**: weighted finite state transducer
- **k-tape WFSM**: weighted finite state machine jointly mapping between k strings

\[ \psi_1(x_1) = 4.25 \]

A **WFSA** is a function which maps a string to a score.
WFST as a Factor Graph

- **WFSA**: weighted finite state automata
- **WFST**: weighted finite state transducer
- **$k$-tape WFSM**: weighted finite state machine jointly mapping between $k$ strings

\[
\psi_1(x_1, x_2) = 13.26
\]

A WFST is a function that maps a pair of strings to a score.

(Dreyer, Smith, & Eisner, 2008)
$k$-tape WFSM as a Factor Graph

- **WFSA**: weighted finite state automata
- **WFST**: weighted finite state transducer
- **$k$-tape WFSM**: weighted finite state machine jointly mapping between $k$ strings

What's wrong with a 100-tape WFSM for jointly modeling the 100 distinct forms of a Polish verb?
- Each arc represents a 100-way edit operation
- Too many arcs!
Factor Graphs over Multiple Strings

\[ P(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_1(x_1, x_2) \psi_2(x_1, x_3) \psi_3(x_1, x_4) \psi_4(x_2, x_3) \psi_5(x_3, x_4) \]

Instead, just build factor graphs with WFST factors (i.e. factors of arity 2)

(Dreyer & Eisner, 2009)
Factor Graphs over Multiple Strings

\[ P(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_1(x_1, x_2) \psi_2(x_1, x_3) \psi_3(x_1, x_4) \psi_4(x_2, x_3) \psi_5(x_3, x_4) \]

Instead, just build factor graphs with WFST factors (i.e. factors of arity 2)

(Dreyer & Eisner, 2009)
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming.
- Overall model is no longer tractable, but BP lets us pretend it is.
Section 5:
What if even BP is slow?

Computing fewer message updates
Computing them faster
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Loopy Belief Propagation Algorithm

1. For every directed edge, initialize its message to the uniform distribution.

2. Repeat until all normalized beliefs converge:
   a. Pick a directed edge $u \rightarrow v$.
   b. Update its message: recompute $u \rightarrow v$ from its “parent” messages $v' \rightarrow u$ for $v' \neq v$.

Or if $u$ has high degree, can share work for speed:
• Compute all outgoing messages $u \rightarrow \ldots$ at once, based on all incoming messages $\ldots \rightarrow u$. 
Loopy Belief Propagation Algorithm

1. For every directed edge, initialize its message to the uniform distribution.

2. Repeat until all normalized beliefs converge:
   a. **Pick** a directed edge $u \rightarrow v$.
   b. **Update** its message: recompute $u \rightarrow v$ from its “parent” messages $v' \rightarrow u$ for $v' \neq v$.

Which edge do we pick and recompute? A “stale” edge?
Message Passing in Belief Propagation

My other factors think I’m a noun

But my other variables and I think you’re a verb
Stale Messages

We update this message from its antecedents. Now it’s “fresh.” Don’t need to update it again.
Stale Messages

We update this message from its antecedents. Now it’s “fresh.” Don’t need to update it again.

But it again becomes “stale” – out of sync with its antecedents – if they change. Then we do need to revisit.

The edge is very stale if its antecedents have changed a lot since its last update. Especially in a way that might make this edge change a lot.
For a high-degree node that likes to update all its outgoing messages at once ...
We say that the whole node is very stale if its incoming messages have changed a lot.
Stale Messages

For a high-degree node that likes to update all its outgoing messages at once ...
We say that the whole node is very stale if its incoming messages have changed a lot.
Maintain an Queue of Stale Messages to Update

Initially all messages are uniform.

Messages from factors are stale. Messages from variables are actually fresh (in sync with their uniform antecedents).
Maintain an Queue of Stale Messages to Update
Maintain an Queue of Stale Messages to Update
Maintain a *queue* of stale messages to update a priority queue! (heap)

- **Residual BP:** *Always update the message that is most stale* (would be *most changed* by an update).
- Maintain a priority queue of stale edges (& perhaps variables).
  - Each step of residual BP: “Pop and update.”
  - Prioritize by degree of staleness.
  - When something becomes stale, put it on the queue.
  - If it becomes staler, move it earlier on the queue.
  - Need a measure of staleness.
- **So, process biggest updates first.**
- Dramatically improves speed of convergence.
  - And chance of converging at all. 😊

(Elidan et al., 2006)
But what about the topology?

In a graph with no cycles:
1. Send messages from the leaves to the root.
2. Send messages from the root to the leaves.
Each outgoing message is sent only after all its incoming messages have been received.
A bad update order for residual BP!
Try updating an entire acyclic subgraph

Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash
Try updating an entire acyclic subgraph

Pick this subgraph; update leaves to root, then root to leaves

Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash
Try updating an entire acyclic subgraph

Another subgraph; update leaves to root, then root to leaves

Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash
Try updating an entire acyclic subgraph

Another subgraph; update leaves to root, then root to leaves

At every step, pick a spanning tree (or spanning forest) that covers many stale edges

As we update messages in the tree, it affects staleness of messages outside the tree
In a graph with no cycles:
1. Send messages from the leaves to the root.
2. Send messages from the root to the leaves.
Each outgoing message is sent only after all its incoming messages have been received.
Summary of Update Ordering

In what order do we send messages for Loopy BP?

- **Asynchronous**
  - Pick a directed edge: update its message
  - Or, pick a vertex: update *all* its outgoing messages at once

**Wait for your antecedents**

Don’t update a message if its antecedents will get a big update.

Otherwise, will have to re-update.

**Size.** Send big updates first.

- Forces other messages to wait for them.

**Topology.** Use graph structure.

- E.g., in an acyclic graph, a message can wait for *all* updates before sending.
The order in which messages are sent has a significant effect on convergence.

- **Synchronous (bad idea)**
  - Compute all the messages
  - Send all the messages
- **Asynchronous**
  - Pick an edge: compute and send that message
- **Tree-based Reparameterization**
  - Successively update embedded spanning trees
  - Choose spanning trees such that each edge is included in at least one
- **Residual BP**
  - Pick the edge whose message would change the most if sent: compute and send that message

![Graph showing the percentage of runs converged vs. time in seconds for different methods.](image)
Message Scheduling

Even better dynamic scheduling may be possible by reinforcement learning of a problem-specific heuristic for choosing which edge to update next.

(Jiang, Moon, Daumé III, & Eisner, 2013)
Section 5: What if even BP is slow?

Computing fewer message updates

Computing them faster

A variable has $k$ possible values. What if $k$ is large or infinite?
Computing Variable Beliefs

Suppose...

– \( X_i \) is a discrete variable
– Each incoming messages is a Multinomial

Pointwise product is easy when the variable’s domain is small and discrete

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

Suppose...

– $X_i$ is a real-valued variable
– Each incoming message is a Gaussian

The pointwise product of $n$ Gaussians is...

...a Gaussian!

$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i}(x_i)$$
Computing Variable Beliefs

Suppose...

- \( X_i \) is a real-valued variable
- Each incoming messages is a mixture of \( k \) Gaussians

The pointwise product explodes!

\[
p(x) = p_1(x) \ p_2(x) \ldots p_n(x)
\]

\[
(0.3 \ q_{1,1}(x) + 0.7 \ q_{1,2}(x)) \ (0.5 \ q_{2,1}(x) + 0.5 \ q_{2,2}(x))
\]

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

Suppose...

- $X_i$ is a string-valued variable (i.e. its domain is the set of all strings)
- Each incoming message is a FSA

The pointwise product explodes!

$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \rightarrow i}(x_i)$$
Example: String-valued Variables

- Messages can **grow larger** when sent through a transducer factor.
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)
Example: String-valued Variables

• Messages can **grow larger** when sent through a transducer factor

• Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)
Example: String-valued Variables

- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)
Example: String-valued Variables

- Messages can **grow larger** when sent through a transducer factor.
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)
Example: String-valued Variables

- Messages can **grow larger** when sent through a transducer factor.
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)
Example: String-valued Variables

• The domain of these variables is infinite (i.e. $\Sigma^*$);
• WSFA’s representation is finite – but the size of the representation can grow;
• In cases where the domain of each variable is small and finite this is not an issue.

• Messages can grow larger when sent through a transducer factor;
• Repeatedly sending messages through a transducer can cause them to grow to unbounded size!

(Dreyer & Eisner, 2009)
Message Approximations

Three approaches to dealing with complex messages:

1. Particle Belief Propagation (see Section 3)
2. Message pruning
3. Expectation propagation
Message Pruning

• **Problem:** Product of $d$ messages = complex distribution.
  – **Solution:** Approximate with a simpler distribution.
  – For speed, compute approximation *without* computing full product.

For **real variables**, try a mixture of $K$ Gaussians:
  – E.g., true product is a mixture of $K^d$ Gaussians
    – **Prune back:** Randomly keep just $K$ of them
    – Chosen in proportion to weight in full mixture
    – Gibbs sampling to efficiently choose them

  – What if incoming messages are not Gaussian mixtures?
    – Could be anything sent by the factors ...
    – Can extend technique to this case.

(Sudderth et al., 2002 –“Nonparametric BP”)
Message Pruning

- **Problem:** Product of $d$ messages = complex distribution.
  - **Solution:** Approximate with a simpler distribution.
  - For speed, compute approximation *without* computing full product.

**For string variables, use a small finite set:**
- Each message $\mu_i$ gives positive probability to ...
  - ... every word in a 50,000 word vocabulary
  - ... every string in $\Sigma^*$ (using a weighted FSA)
- **Prune back** to a list $L$ of a few “good” strings
  - Each message adds its own $K$ best strings to $L$
  - For each $x \in L$, let $\mu(x) = \prod_i \mu_i(x)$ – each message scores $x$
  - For each $x \notin L$, let $\mu(x) = 0$

(Dreyer & Eisner, 2009)
Expectation Propagation (EP)

- **Problem:** Product of $d$ messages = complex distribution.
  - **Solution:** Approximate with a simpler distribution.
  - For speed, compute approximation *without* computing full product.

EP provides four special advantages over pruning:

1. **General recipe** that can be used in many settings.
2. **Efficient.** Uses approximations that are very fast.
3. **Conservative.** Unlike pruning, never forces $b(x)$ to 0.
   - Never kills off a value $x$ that had been possible.
4. **Adaptive.** Approximates $\mu(x)$ more carefully if $x$ is favored by the other messages.
   - Tries to be accurate on the most “plausible” values.

(Minka, 2001; Heskes & Zoeter, 2002)
Expectation Propagation (EP)

Belief at $X_3$ will be simple!
Messages to and from $X_3$ will be simple!

Exponential-family approximations inside

$X_1$, $X_2$, $X_3$, $X_4$, $X_5$, $X_6$, $X_7$
Expectation Propagation (EP)

**Key idea:** Approximate variable $X$’s incoming messages $\mu$. We force them to have a simple parametric form:

$$\mu(x) = \exp(\theta \cdot f(x))$$

“log-linear model” (unnormalized)

where $f(x)$ extracts a feature vector from the value $x$.

For each variable $X$, we’ll choose a feature function $f$.

So by storing a few parameters $\theta$, we’ve defined $\mu(x)$ for all $x$. Now the messages are super-easy to multiply:

$$\mu_1(x) \mu_2(x) = \exp(\theta \cdot f(x)) \exp(\theta \cdot f(x)) = \exp((\theta_1 + \theta_2) \cdot f(x))$$

Represent a message by its parameter vector $\theta$.

To multiply messages, just add their $\theta$ vectors!

So beliefs and outgoing messages also have this simple form.
Expectation Propagation

- Form of messages/beliefs at $X_3$?
  - Always $\mu(x) = \exp(\theta \cdot f(x))$
- If $x$ is real:
  - Gaussian: Take $f(x) = (x, x^2)$
- If $x$ is string:
  - Globally normalized trigram model: Take $f(x) = (\text{count of aaa, count of aab, ... count of zzz})$
- If $x$ is discrete:
  - Arbitrary discrete distribution (can exactly represent original message, so we get ordinary BP)
  - Coarsened discrete distribution, based on features of $x$
- Can’t use mixture models, or other models that use latent variables to define $\mu(x) = \sum_y p(x, y)$
Expectation Propagation

- Each message to $X_3$ is
  \[ \mu(x) = \exp(\theta \cdot f(x)) \]
  for some $\theta$. We only store $\theta$.

- To take a product of such messages, just add their $\theta$
  - Easily compute belief at $X_3$
    (sum of incoming $\theta$ vectors)
  - Then easily compute each outgoing message
    (belief minus one incoming $\theta$)
- All very easy …
Expectation Propagation

- But what about messages from factors?
  - Like the message $M_4$.
  - This is not exponential family! Uh-oh!
  - It’s just whatever the factor happens to send.

- This is where we need to approximate, by $\mu_4$. 
Expectation Propagation

- blue = arbitrary distribution, green = simple distribution $\exp(\theta \cdot f(x))$

- The belief at $x$ “should” be
  $$p(x) = \mu_1(x) \cdot \mu_2(x) \cdot \mu_3(x) \cdot M_4(x)$$
- But we’ll be using
  $$b(x) = \mu_1(x) \cdot \mu_2(x) \cdot \mu_3(x) \cdot \mu_4(x)$$

- Choose the simple distribution $b$ that minimizes $KL(p \parallel b)$.
- Then, work backward from belief $b$ to message $\mu_4$.
  - Take $\theta$ vector of $b$ and subtract off the $\theta$ vectors of $\mu_1, \mu_2, \mu_3$.
  - Chooses $\mu_4$ to preserve belief well.

That is, choose $b$ that assigns high probability to samples from $p$.
Find $b$’s params $\theta$ in closed form – or follow gradient:
$$E_{x \sim p}[f(x)] - E_{x \sim b}[f(x)]$$
ML Estimation = Moment Matching

<table>
<thead>
<tr>
<th></th>
<th>fo</th>
<th>foo</th>
<th>bar</th>
<th>az</th>
<th>xy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Predicted Counts</td>
<td>2.6</td>
<td>-0.5</td>
<td>1.2</td>
<td>3.1</td>
<td>-6.0</td>
</tr>
</tbody>
</table>

Broadcast n-gram counts

Fit model that predicts same counts
FSA Approx. = Moment Matching

A distribution over strings

\[
\begin{align*}
\text{fo} &= 3.1 \\
\text{foo} &= 0.9 \\
\text{bar} &= 2.2 \\
\text{zz} &= 0.1 \\
\text{az} &= 4.1 \\
\text{xx} &= 0.1
\end{align*}
\]

(can compute with forward-backward)

Fit model that predicts same fractional counts
FSA Approx. = Moment Matching

\[
\min_\theta \text{KL}(\cdot | | \cdot)
\]

Finds parameters \( \theta \) that minimize KL “error” in belief
How to approximate a message?

\[ KL(\cdot) \]

\[
\begin{array}{c|c|c}
\text{fo} & 1.2 & \text{fo} & 0.2 \\
\text{bar} & 0.5 & \text{bar} & 1.1 \\
\text{az} & 4.3 & \text{az} & -0.3 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{fo} & 0.6 & \text{fo} & 2.0 \\
\text{bar} & 0.6 & \text{bar} & -1.0 \\
\text{az} & -1.0 & \text{az} & 3.0 \\
\end{array}
\]

Wisely, KL doesn’t insist on good approximations for values that are low-probability in the belief.

Finds message parameters \( \theta \) that minimize KL “error” of resulting belief.
Analogy: Scheduling by approximate email messages

Wisely, KL doesn’t insist on good approximations for values that are low-probability in the belief

“This is an approximation to my true schedule. I’m not actually free on all Tue/Thu, but the bad Tue/Thu dates have already been ruled out by messages from other folks.”
Example: Factored PCFGs

**Expectation Propagation**

- **Task:** Constituency parsing, with factored annotations
  - Lexical annotations
  - Parent annotations
  - Latent annotations

- **Approach:**
  - Sentence specific approximation is an anchored grammar:
    \[ q(A \rightarrow B C, i, j, k) \]
  - Sending messages is equivalent to marginalizing out the annotations

(Hall & Klein, 2012)
Section 6:
Approximation-aware Training
Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!
Modern NLP

Mathematical Modeling

Linguistics

Combinatorial Optimization

NLP

Machine Learning
Linguistics inspires the structures we want to predict.

No semantic interpretation.
Machine Learning for NLP

Our model defines a score for each structure:

\[ p_{\theta}(\text{time flies like an arrow}) = 0.50 \]

\[ p_{\theta}(\text{time flies like an arrow}) = 0.25 \]

\[ p_{\theta}(\text{time flies like an arrow}) = 0.10 \]

\[ \ldots \]

\[ p_{\theta}(\text{time flies like an arrow}) = 0.01 \]
Our **model** defines a score for each structure. It also tells us what to optimize.

\[
p_\theta(\text{time flies like an arrow}) = 0.50
\]

\[
p_\theta(\text{time flies like an arrow}) = 0.25
\]

\[
p_\theta(\text{time flies like an arrow}) = 0.10
\]

\[
p_\theta(\text{time flies like an arrow}) = 0.01
\]
Machine Learning for NLP

Given training instances \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)

Find the best model parameters, \( \theta \)

\[
\theta^* = \arg\max_{\theta} \prod_{i=1}^{n} \rho_\theta(y_i \mid x_i)
\]
Machine Learning for NLP

Given training instances \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)

Find the best model parameters, \( \theta \)
Machine Learning for NLP

Given training instances \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)

Find the best model parameters, \( \theta \)
Machine Learning for NLP

- Given a **new sentence**, $x_{\text{new}}$
- Search over the **set of all possible structures** (often exponential in size of $x_{\text{new}}$)
- Return the **highest scoring** structure, $y^*$

$$y^* = \arg\max_y p_\theta(y \mid x_{\text{new}})$$

**Inference** finds the best structure for a new sentence

(Inference is usually called as a **subroutine** in learning)
Machine Learning for NLP

- Given a **new sentence**, $x_{\text{new}}$
- Search over the **set of all possible structures** (often exponential in size of $x_{\text{new}}$)
- Return the **Minimum Bayes Risk (MBR)** structure, $y^*$

\[
y^* = \arg\min_y \mathbb{E}_{p_{\theta}}(y' | x) [\ell(y, y')]\]

*Inference finds the best structure for a new sentence*

(Inference is usually called as a **subroutine** in learning)
Machine Learning for NLP

- Given a **new sentence**, $x_{\text{new}}$
- Search over the **set of all possible structures** (often exponential in size of $x_{\text{new}}$)
- Return the **Minimum Bayes Risk (MBR)** structure, $y^*$

$$y^* = \arg\min_y \mathbb{E}_{p_\theta(y'|x)} [\ell(y, y')]$$

**Inference** finds the best structure for a new sentence

(Inferrference is usually called as a **subroutine** in learning)
Modern NLP

**Linguistics** inspires the structures we want to predict.

**Inference** finds the best structure for a new sentence. *(Inference is usually called as a subroutine in learning)*

Our **model** defines a score for each structure. It also tells us what to optimize.

**Learning** tunes the parameters of the model.

- Linguistics
- Mathematical Modeling
- Combinatorial Optimization
- Machine Learning

Inference is usually called as a subroutine in learning.
An Abstraction for Modeling

Now we can work at this level of abstraction.

\[ p_\theta(y) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(y_\alpha) \]
Training

Thus far, we’ve seen how to compute (approximate) marginals, given a factor graph...

...but where do the potential tables $\psi_\alpha$ come from?

– Some have a fixed structure (e.g. Exactly1, CKYTree)
– Others could be trained ahead of time (e.g. TrigramHMM)
– For the rest, we define them parametrically and learn the parameters!

Two ways to learn:

1. **Standard CRF Training**
   (very simple; often yields state-of-the-art results)
2. **ERMA**
   (less simple; but takes approximations and loss function into account)
Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(x_\alpha, y_\alpha; \theta) = \exp(\theta \cdot f_\alpha(x_\alpha, y_\alpha))$$

- **Observed variables**
- **Predicted variables**
Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$
\psi_\alpha(x_\alpha, y_\alpha; \theta) = \exp(\theta \cdot f_\alpha(x_\alpha, y_\alpha))
$$
Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(x_\alpha, y_\alpha; \theta) = \exp(\theta \cdot f_\alpha(x_\alpha, y_\alpha))$$
What is Training?

That’s easy:

Training = picking good model parameters!

But how do we know if the model parameters are any “good”? 
Conditional Log-likelihood Training

1. Choose **model**

\[ p_\theta(y) = \frac{1}{Z} \prod_\alpha \psi_\alpha(y_\alpha) \]

2. Choose **objective:**
Assign high probability to the things we observe and low probability to everything else

\[ L(\theta) = \sum_{y \in D} \log p_\theta(y) \]

3. Compute derivative **by hand** using the chain rule

\[
\frac{dL(\theta)}{d\theta_j} = \sum_{y \in D} \left( \sum_\alpha \left[ f_{\alpha,j}(y_\alpha) - \sum_{y'} p_\theta(y'_\alpha) f_{\alpha,j}(y'_\alpha) \right] \right)
\]
Conditional Log-likelihood Training

1. Choose **model**
   Such that derivative in #3 is easy

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else

3. Compute derivative **by hand** using the chain rule

4. Replace **exact inference** by approximate inference

   \[ p_\theta(y) = \frac{1}{Z} \prod_\alpha \exp(\theta \cdot f_\alpha(y_\alpha)) \]

   \[ L(\theta) = \sum_{y \in \mathcal{D}} \log p_\theta(y) \]

   \[ \frac{dL(\theta)}{d\theta_j} = \sum_{y \in \mathcal{D}} \left( \sum_\alpha \left[ f_{\alpha,j}(y_\alpha) - \sum_{y'} \frac{p_\theta(y')}{p_\theta(y_\alpha)} f_{\alpha,j}(y'_\alpha) \right] \right) \]

   \[ \approx \sum_{y \in \mathcal{D}} \left( \sum_\alpha \left[ f_{\alpha,j}(y_\alpha) - \sum_{y'} b_\theta(y'_\alpha) f_{\alpha,j}(y'_\alpha) \right] \right) \]

   We can **approximate** the **factor marginals** by the (normalized) **factor beliefs** from BP!
Stochastic Gradient Descent

**Input:**
- Training data, \( \{(x^{(i)}, y^{(i)}) : 1 \leq i \leq N \} \)
- Initial model parameters, \( \theta \)

**Output:**
- Trained model parameters, \( \theta \).

**Algorithm:**

While not converged:
- Sample a training example \( (x^{(i)}, y^{(i)}) \)
- Compute the gradient of \( \log(p_\theta(y^{(i)} | x^{(i)})) \) with respect to our model parameters \( \theta \).
- Take a (small) step in the direction of the gradient.
What’s wrong with the usual approach?

If you add too many factors, your predictions might get worse!

• The model might be richer, but we replace the true marginals with approximate marginals (e.g. beliefs computed by BP)

• Approximate inference can cause gradients for structured learning to go awry! (Kulesza & Pereira, 2008).
What’s wrong with the usual approach?

Mistakes made by Standard CRF Training:
1. Using BP (approximate)
2. Not taking loss function into account
3. Should be doing MBR decoding

Big pile of approximations…
... which has tunable parameters.

Treat it like a neural net, and run backprop!
Modern NLP

Linguistics inspires the structures we want to predict

Our model defines a score for each structure

It also tells us what to optimize

Inference finds the best structure for a new sentence

(Inference is usually called as a subroutine in learning)

Learning tunes the parameters of the model
Empirical Risk Minimization

1. Given training data:
   \[ \{ \mathbf{x}_i, y_i \}^{N}_{i=1} \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_{\theta}(\mathbf{x}_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy
Empirical Risk Minimization

1. Given training data:
   \[ \{x_i, y_i\}_{i=1}^{N} \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_{\theta}(x_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:
   \[ \theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \ell(f_{\theta}(x_i), y_i) \]

4. Train with SGD:
   (take small steps opposite the gradient)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i) \]
Empirical Risk Minimization

1. Given training data:
   \[ \{x_i, y_i\}_{i=1}^{N} \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_{\theta}(x_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:
   \[ \theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \ell(f_{\theta}(x_i), y_i) \]

4. Train with SGD:
   (take small steps opposite the gradient)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i) \]
Conditional Log-likelihood Training

1. Choose **model**
   Such that derivative in #3 is easy
   \[ p_\theta(y) = \frac{1}{Z} \prod_\alpha \exp(\theta \cdot f_\alpha(y_\alpha)) \]

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else
   \[ L(\theta) = \sum_{y \in D} \log p_\theta(y) \]

3. Compute derivative by **hand** using the chain rule
   \[ \frac{dL(\theta)}{d\theta_j} = \sum_{y \in D} \left( \sum_\alpha \left[ f_{\alpha,j}(y_\alpha) - \sum_{y'} p_\theta(y'_\alpha) f_{\alpha,j}(y'_\alpha) \right] \right) \]

4. Replace true inference by **approximate** inference
   \[ \approx \sum_{y \in D} \left( \sum_\alpha \left[ f_{\alpha,j}(y_\alpha) - \sum_{y'} b_\theta(y'_\alpha) f_{\alpha,j}(y'_\alpha) \right] \right) \]
What went wrong?

How did we compute these approximate marginal probabilities anyway?

By Belief Propagation of course!

A) Compute Potentials
\( \psi_\alpha(x_\alpha) = \exp(\theta \cdot f(x_\alpha)) \)

B) Initial Messages
\( m^{(0)}_{i \to \alpha}(x_i) = 1 \)
\( m^{(0)}_{\alpha \to i}(x_i) = 1 \)

C) Messages at time \( t = 1 \)
\( \hat{m}^{(1)}_{i \to \alpha}(x_i) = \ldots, \hat{m}^{(1)}_{\alpha \to i}(x_i) = \ldots \)

C) Messages at time \( T \)
\( \hat{m}^{(T)}_{i \to \alpha}(x_i) = \ldots, \hat{m}^{(T)}_{\alpha \to i}(x_i) = \ldots \)

D) Beliefs
\( \hat{b}_1(x_i) = \ldots, \hat{b}_\alpha(x_\alpha) = \ldots \)

E) Decode and Loss
\( J = \ldots \)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation
Error Back-Propagation

\[ P(y_3 = \text{noun} | x) \]

\[ \mu(y_1 \rightarrow y_2) = \mu(y_3 \rightarrow y_1) \cdot \mu(y_4 \rightarrow y_1) \]

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

• Applying the chain rule of differentiation over and over.

• Forward pass:
  – Regular computation (inference + decoding) in the model (+ remember intermediate quantities).

• Backward pass:
  – Replay the forward pass in reverse, computing gradients.
Background: Backprop through time

Recurrent neural network:

\[ y_{t+1} \]
\[ x_{t+1} \]
\[ b_t \]
\[ a \]
\[ x_t \]

BPTT:
1. Unroll the computation over time

2. Run backprop through the resulting feed-forward network

What went wrong?

How did we compute these approximate marginal probabilities anyway?

By Belief Propagation of course!

A) Compute Potentials
\[ \psi_\alpha(x_\alpha) = \exp(\theta \cdot f(x_\alpha)) \]

B) Initial Messages
\[ m_{i \to \alpha}(x_i) = 1 \]
\[ m_{\alpha \to i}(x_i) = 1 \]

C) Messages at time \( t = 1 \)
\[ \tilde{m}_{i \to \alpha}^{(1)}(x_i) = \ldots, \tilde{m}_{\alpha \to i}^{(1)}(x_i) = \ldots \]

C) Messages at time \( t \)
\[ \tilde{m}_{i \to \alpha}^{(t)}(x_i) = \ldots, \tilde{m}_{\alpha \to i}^{(t)}(x_i) = \ldots \]

C) Messages at time \( T \)
\[ \tilde{m}_{i \to \alpha}^{(T)}(x_i) = \ldots, \tilde{m}_{\alpha \to i}^{(T)}(x_i) = \ldots \]

D) Beliefs
\[ \tilde{b}_i(x_i) = \ldots, \tilde{b}_\alpha(x_\alpha) = \ldots \]

E) Decode and Loss
\[ J = \ldots \]
ERMA

Empirical Risk Minimization under Approximations (ERMA)

- Apply Backprop through time to Loopy BP
- Unrolls the BP computation graph
- Includes inference, decoding, loss and all the approximations along the way

(A) Compute Potentials
\[ \psi_{\alpha}(x_{\alpha}) = \exp(\theta \cdot f(x_{\alpha})) \]

(B) Initial Messages
\[ m^{(0)}_{i \to \alpha}(x_i) = 1 \]
\[ m^{(0)}_{\alpha \to i}(x_i) = 1 \]

(C) Messages at time \( t = 1 \)
\[ \tilde{m}_{i \to \alpha}^{(1)}(x_i) = \ldots, \tilde{m}_{\alpha \to i}^{(1)}(x_i) = \ldots \]

(C) Messages at time \( t \)
\[ \tilde{m}_{i \to \alpha}^{(t)}(x_i) = \ldots, \tilde{m}_{\alpha \to i}^{(t)}(x_i) = \ldots \]

(D) Beliefs
\[ \tilde{b}_i(x_i) = \ldots, \tilde{b}_{\alpha}(x_{\alpha}) = \ldots \]

(E) Decode and Loss
\[ J = \ldots \]

(Stoyanov, Ropson, & Eisner, 2011)
1. Choose **model** to be the computation with all its approximations

2. Choose **objective** to likewise include the approximations

3. Compute **derivative** by backpropagation (treating the entire computation as if it were a neural network)

4. Make no approximations! (Our gradient is exact)

---

**ERMA**

\[ p_{\theta}(y) = \]  
\[ L(\theta) = \]

---

**Key idea: Open up the black box!**

- **E) Decode and Loss**  
  \[ J = \ldots \]

- **D) Beliefs**  
  \[ \tilde{b}_i(x_i) = \ldots, \tilde{b}_\alpha(x_\alpha) = \ldots \]

- **C) Messages at time \( T \)**  
  \[ \tilde{m}^{(T)}_{i \rightarrow \alpha}(x_i) = \ldots, \tilde{m}^{(T)}_{\alpha \rightarrow i}(x_i) = \ldots \]

- **C) Messages at time \( t \)**  
  \[ \tilde{m}^{(t)}_{i \rightarrow \alpha}(x_i) = \ldots, \tilde{m}^{(t)}_{\alpha \rightarrow i}(x_i) = \ldots \]

- **C) Messages at time \( t = 1 \)**  
  \[ \tilde{m}^{(1)}_{i \rightarrow \alpha}(x_i) = \ldots, \tilde{m}^{(1)}_{\alpha \rightarrow i}(x_i) = \ldots \]

- **A) Compute Potentials**  
  \[ \psi_\alpha(x_\alpha) = \exp(\theta \cdot f(x_\alpha)) \]

- **B) Initial Messages**  
  \[ m_{i \rightarrow \alpha}^{(0)}(x_i) = 1 \]
  \[ m_{\alpha \rightarrow i}^{(0)}(x_i) = 1 \]

---

(Stoyanov, Ropson, & Eisner, 2011)
Empirical Risk Minimization

\[ \theta^* = \arg\min_{\theta} \frac{1}{D} \sum_{d=1}^{D} \ell(h_{\theta}(x^{(d)}), y^{(d)}) \]

Minimum Bayes Risk (MBR) Decoder

\[ h_{\theta}(x) = \arg\min_{y} \mathbb{E}_{p_{\theta}}(y' \mid x) [\ell(y, y')] \]

(E) Decode and Loss

\[ J = \ldots \]

(D) Beliefs

\[ \tilde{b}_i(x_i) = \ldots, \tilde{b}_\alpha(x_\alpha) = \ldots \]

(C) Messages at time \( T \)

\[ \tilde{m}^{(T)}_{i \rightarrow \alpha}(x_i) = \ldots, \tilde{m}^{(T)}_{\alpha \rightarrow i}(x_i) = \ldots \]

(Stoyanov, Ropson, & Eisner, 2011)
Approximation-aware Learning

What if we’re using Structured BP instead of regular BP?

- No problem, the same approach still applies!
- The only difference is that we embed dynamic programming algorithms inside our computation graph.

\[ p_{\theta}(y) = \]

\[ L(\theta) = \]

(Gormley, Dredze, & Eisner, 2015)
Connection to Deep Learning

\[ \exp(\Theta_y \cdot f(x)) \]

(Gormley, Yu, & Dredze, In submission)
Empirical Risk Minimization under Approximations (ERMA)

<table>
<thead>
<tr>
<th>Loss Aware</th>
<th>Approximation Aware</th>
<th>SVM $^\text{struct}$ [Finley and Joachims, 2008]</th>
<th>M$^3$N [Taskar et al., 2003]</th>
<th>Softmax-margin [Gimpel &amp; Smith, 2010]</th>
<th>MLE</th>
<th>ERMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure from (Stoyanov & Eisner, 2012)
Section 7: Software
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.

2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.

3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?

4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.

5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.

6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.

7. **Software:** Build the model you want!
Outline

• Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
• Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What’s going on here? Can we trust BP’s estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!
Pacaya

Features:
- Structured Loopy BP over factor graphs with:
  - Discrete variables
  - Structured constraint factors (e.g. projective dependency tree constraint factor)
  - ERMA training with backpropagation
  - Backprop through structured factors (Gormley, Dredze, & Eisner, 2015)

Language: Java
Authors: Gormley, Mitchell, & Wolfe
URL: http://www.cs.jhu.edu/~mrg/software/
ERMA

Features:
ERMA performs inference and training on CRFs and MRFs with arbitrary model structure over discrete variables. The training regime, Empirical Risk Minimization under Approximations is loss-aware and approximation-aware. ERMA can optimize several loss functions such as Accuracy, MSE and F-score.

Language: Java
Authors: Stoyanov
URL: [https://sites.google.com/site/ermasoftware/](https://sites.google.com/site/ermasoftware/)
Graphical Models Libraries

- **Factorie** (McCallum, Shultz, & Singh, 2012) is a Scala library allowing modular specification of inference, learning, and optimization methods. Inference algorithms include belief propagation and MCMC. Learning settings include maximum likelihood learning, maximum margin learning, learning with approximate inference, SampleRank, pseudo-likelihood. [http://factorie.cs.umass.edu/](http://factorie.cs.umass.edu/)

- **LibDAI** (Mooij, 2010) is a C++ library that supports inference, but not learning, via Loopy BP, Fractional BP, Tree-Reweighted BP, (Double-loop) Generalized BP, variants of Loop Corrected Belief Propagation, Conditioned Belief Propagation, and Tree Expectation Propagation. [http://www.libdai.org](http://www.libdai.org)

- **OpenGM2** (Andres, Beier, & Kappes, 2012) provides a C++ template library for discrete factor graphs with support for learning and inference (including tie-ins to all LibDAI inference algorithms). [http://hci.iwr.uni-heidelberg.de/opengm2/](http://hci.iwr.uni-heidelberg.de/opengm2/)

- **FastInf** (Jaimovich, Meshi, Mcgraw, Elidan) is an efficient Approximate Inference Library in C++. [http://compbio.cs.huji.ac.il/FastInf/fastInf/FastInf_Homepage.html](http://compbio.cs.huji.ac.il/FastInf/fastInf/FastInf_Homepage.html)

References

• M. Dreyer, “A non-parametric model for the discovery of inflectional paradigms from plain text using graphical models over strings,” Johns Hopkins University, Baltimore, MD, USA, 2011.


• A. T. Ihler and D. A. McAllester, “Particle belief propagation,” in International Conference on Artificial Intelligence and Statistics, 2009, pp. 256–263.


