Cube Summing, Approximate Inference with Non-Local Features, and Dynamic Programming without Semirings

Kevin Gimpel and Noah A. Smith
Overview

- We introduce **cube summing**, which extends dynamic programming algorithms for summing with non-local features
  - Inspired by **cube pruning** (Chiang, 2007; Huang & Chiang, 2007)

- We relate cube summing to semiring-weighted logic programming
  - Without non-local features, cube summing is a novel semiring
  - Non-local features break some of the semiring properties
  - We propose an implementation based on **arithmetic circuits**
Outline

- Background
- Cube Pruning
- Cube Summing
- Semirings
- Implementation
- Conclusion
Fundamental Problems

- Consider an exponential probabilistic model

\[ p(y \mid x) \propto \prod_{m=1}^{M} \lambda_m h_m(x,y) \]

- Two fundamental problems we often need to solve

  - Decoding
  \[ \hat{y}(x) = \arg\max_{y \in Y} \prod_{m=1}^{M} \lambda_m h_m(x,y) \]

  - Summing
  \[ s(x) = \sum_{y \in Y} \prod_{m=1}^{M} \lambda_m h_m(x,y) \]
Fundamental Problems

Consider an exponential probabilistic model

$$p(y | x) \propto \prod_{m=1}^{M} \lambda_{m}^{h_{m}(x,y)}$$

example: HMM

$x$ is a sentence, $y$ is a tag sequence

Two fundamental problems we often need to solve

- Decoding

$$\hat{y}(x) = \arg\max_{y \in Y} \prod_{m=1}^{M} \lambda_{m}^{h_{m}(x,y)}$$

Viterbi algorithm

- Summing

$$s(x) = \sum_{y \in Y} \prod_{m=1}^{M} \lambda_{m}^{h_{m}(x,y)}$$

forward and backward algorithms
Fundamental Problems

- Consider an exponential probabilistic model
  \[
  p(y | x) \propto \prod_{m=1}^{M} \lambda_m h_m(x, y)
  \]
  example: PCFG
  \(x\) is a sentence, \(y\) is a parse tree

- Two fundamental problems we often need to solve
  - Decoding
    \[
    \hat{y}(x) = \arg\max_{y \in Y} \prod_{m=1}^{M} \lambda_m h_m(x, y)
    \]
    probabilistic CKY
  - Summing
    \[
    s(x) = \sum_{y \in Y} \prod_{m=1}^{M} \lambda_m h_m(x, y)
    \]
    inside algorithm
Fundamental Problems

- Consider an exponential probabilistic model

\[ p(y \mid x) \propto \prod_{m=1}^{M} \lambda_m h_m(x, y) \]

- Two fundamental problems we often need to solve

  - **Decoding**

    \[ \hat{y}(x) = \arg\max_{y \in y} \prod_{m=1}^{M} \lambda_m h_m(x, y) \]

  - **Summing**

    \[ s(x) = \sum_{y \in y} \prod_{m=1}^{M} \lambda_m h_m(x, y) \]
Dynamic Programming

Consider the probabilistic CKY algorithm

\[ C_{X,i-1,i} = \lambda_{X \rightarrow w_i} \]

\[ C_{X,i,k} = \max_{Y,Z \in \mathcal{N}; j \in \{i+1, \ldots, k-1\}} \lambda_{X \rightarrow YZ} \times C_{Y,i,j} \times C_{Z,j,k} \]

\[ \text{goal} = C_{S,0,n} \]
<table>
<thead>
<tr>
<th>Weighted Logic Programs</th>
<th>Probabilistic CKY</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>theorem</td>
<td>chart item</td>
<td>$C_{X,i,j}$</td>
</tr>
<tr>
<td>axiom</td>
<td>rule probability</td>
<td>$\lambda X \rightarrow Y Z$</td>
</tr>
<tr>
<td>proof</td>
<td>derivation</td>
<td>PP of the list</td>
</tr>
</tbody>
</table>
In semiring-weighted logic programming, theorem and axiom values come from a **semiring**

<table>
<thead>
<tr>
<th>Weighted Logic Programs</th>
<th>Probabilistic CKY</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>theorem</td>
<td>chart item</td>
<td>$C_{X,i,j}$</td>
</tr>
<tr>
<td>axiom</td>
<td>rule probability</td>
<td>$\lambda X \rightarrow Y Z$</td>
</tr>
<tr>
<td>proof</td>
<td>derivation</td>
<td><img src="image" alt="Derivation Diagram" /></td>
</tr>
</tbody>
</table>
Features

- Recall our model: \( p(y \mid x) \propto \prod_{m=1}^{M} \lambda_m^{h_m(x,y)} \)

- The \( h_m(x, y) \) are feature functions and the \( \lambda_m \) are nonnegative weights
Features

- Recall our model: \( p(y \mid x) \propto \prod_{m=1}^{M} \lambda_{m}^{h_{m}(x,y)} \)

- The \( h_{m}(x,y) \) are feature functions and the \( \lambda_{m} \) are nonnegative weights

- **Local features** depend *only* on theorems used in an equation (or any of the axioms), *not* on the proofs of those theorems

\[
C_{X,i,k} = \max_{Y,Z \in \mathcal{N} ; j \in \{i+1, \ldots, k-1\}} \lambda_{X \rightarrow YZ} \times C_{Y,i,j} \times C_{Z,j,k}
\]
There near the top of the list is quarterback Troy Aikman.
There near the top of the list is quarterback Troy Aikman.
Features

- Recall our model: \( p(y \mid x) \propto \prod_{m=1}^{M} \lambda_m^{h_m(x,y)} \)

- The \( h_m(x, y) \) are feature functions and the \( \lambda_m \) are nonnegative weights

- **Local features** depend *only* on theorems used in an equation (or any of the axioms), *not* on the proofs of those theorems

\[
C_{X,i,k} = \max_{Y,Z \in \mathcal{N}; j \in \{i+1, \ldots, k-1\}} \lambda_{X \rightarrow YZ} \times C_{Y,i,j} \times C_{Z,j,k}
\]

- **Non-local features** depend on theorem proofs
There near the top of the list is quarterback Troy Aikman

“NGramTree” feature
(Charniak & Johnson, 2005)
There near the top of the list is quarterback Troy Aikman.

“NgramTree” feature (Charniak & Johnson, 2005)

Non-local features break dynamic programming!
Other Algorithms for Approximate Inference

- Beam search (Lowerre, 1979)
- Reranking (Collins, 2000)
- Algorithms for graphical models
  - Variational methods (MacKay, 1997; Beal, 2003; Kurihara & Sato, 2006)
  - Belief propagation (Sutton & McCallum, 2004; Smith & Eisner, 2008)
  - MCMC (Finkel et al., 2005; Johnson et al., 2007)
  - Particle filtering (Levy et al., 2009)
- Integer linear programming (Roth & Yih, 2004)
- Stacked learning (Cohen & Carvalho, 2005; Martins et al., 2008)
- Cube pruning (Chiang, 2007; Huang & Chiang, 2007)
Other Algorithms for Approximate Inference

- Beam search (Lowerre, 1979)
- Reranking (Collins, 2000)
- Algorithms for graphical models
  - Variational methods (MacKay, 1997; Beal, 2003; Kurihara & Sato, 2006)
  - Belief propagation (Sutton & McCallum, 2004; Smith & Eisner, 2008)
  - MCMC (Finkel et al., 2005; Johnson et al., 2007)
  - Particle filtering (Levy et al., 2009)
- Integer linear programming (Roth & Yih, 2004)
- Stacked learning (Cohen & Carvalho, 2005; Martins et al., 2008)
- Cube pruning (Chiang, 2007; Huang & Chiang, 2007)

Why add one more?
- Cube pruning extends existing, widely-understood dynamic programming algorithms for decoding
- We want this for summing too
Outline

- Background
- Cube Pruning
- Cube Summing
- Semirings
- Implementation
- Conclusion
Cube Pruning
(Chiang, 2007; Huang & Chiang, 2007)

- Modification to dynamic programming algorithms for decoding to use non-local features approximately

- Keeps a $k$-best list of proofs for each theorem

- Applies non-local feature functions on these proofs when proving new theorems
There near the top of the list is quarter back Troy Aikman.
\[ C_{NP,0.7} = C_{NP,0.1} \times C_{PP,1.7} \times \lambda_{NP \rightarrow NP \, PP} \]

\[ C_{NP,0.1} = \begin{array}{ccc}
0.4 & 0.3 & 0.02 \\
\end{array} \]

\[ C_{PP,1.7} = \begin{array}{ccc}
0.2 & 0.1 & 0.05 \\
\end{array} \]
\[ C_{NP,0.7} = C_{NP,0.1} \times C_{PP,1.7} \times \lambda_{NP \rightarrow NP \ PP} \]
\[ C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP \ PP} \]

\[ \lambda_{NP \rightarrow NP \ PP} = 0.5 \]
\[ C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP PP} \]
\[ \lambda_{\text{EX NP NP PP IN near}} = 0.2 \]

\[
\begin{array}{c|c|c|c}
\text{C}_{\text{PP,1,7}} & 0.2 & 0.1 & 0.05 \\
\hline
0.4 & 0.04 \times 0.2 & 0.02 \times 0.2 & 0.01 \\
0.3 & 0.03 & 0.015 & 0.0075 \\
0.02 & 0.002 & 0.001 & 0.0005 \\
\end{array}
\]
\begin{align*}
\lambda_{\text{There EX NP NP PP IN near}} &= 0.2 \\
\lambda_{\text{There RB NP NP PP IN near}} &= 0.6 \\
\lambda_{\text{There NNP NP NP PP IN near}} &= 0.1 \\
\lambda_{\text{There EX NP NP PP RB near}} &= 0.1 \\
\lambda_{\text{There RB NP NP PP RB near}} &= 0.4 \\
\lambda_{\text{There NNP NP NP PP RB near}} &= 0.2 \\
\end{align*}
There were a few... near the top...
There near the top ...

There near the top ...

There near the top ...

C_{NP,0,1} = |
| 0.2 | 0.1 | 0.05 |
| 0.4 | 0.008 | 0.004 | 0.001 |
| 0.3 | 0.018 | 0.009 | 0.003 |
| 0.02 | 0.0002 | 0.0001 | 0.0001 |

C_{NP,0,7} = |
| 0.018 | 0.009 | 0.008 |
Clarification

- Cube pruning does not actually expand all $k^2$ proofs as this example showed.
- It uses an approximation that only looks at $O(k)$ proofs.
- But since we are summing, we want to look at as many proofs as possible.
- We use the algorithm that we just showed as the basis for cube summing (we call it **cube decoding** – details in paper).
Outline

- Background
- Cube Pruning
- Cube Summing
- Semirings
- Implementation
- Conclusion
There \( C_{NP,0,1} = \begin{bmatrix} 0.4 & 0.3 & 0.02 \end{bmatrix} \)

There \( C_{PP,1,7} = \begin{bmatrix} 0.2 & 0.1 & 0.05 \end{bmatrix} \)
There (NP) = There (NP) = There (NP) =

<table>
<thead>
<tr>
<th></th>
<th>NP</th>
<th>PP</th>
<th>NP</th>
<th>IN</th>
<th>DT</th>
<th>NN</th>
<th>NP</th>
<th>PP</th>
<th>NP</th>
<th>IN</th>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>There</td>
<td>Near the top</td>
<td>...</td>
<td>Near the top</td>
<td>...</td>
<td>Near the top</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$C_{NP,0,1} =$

| 0.4 | 0.3 | 0.02 | 0.05 |

$C_{PP,1,7} =$

| 0.2 | 0.1 | 0.05 | 0.03 |

“residual”
- Computation of local and non-local features is same as before
- Only difference is computing the residual for the result

<table>
<thead>
<tr>
<th></th>
<th>$C_{NP,0,1}$</th>
<th>$C_{PP,1,7}$</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.008</td>
<td>0.004</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
<td>0.009</td>
<td>0.003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>0.0002</td>
<td>0.0001</td>
<td>0.0001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.03</td>
</tr>
</tbody>
</table>

<p>|       | $C_{NP,0,7}$ |       |       |       |
|-------|--------------|-------|-------|
| 0.018 | 0.009        | 0.008 | 0.0287|</p>
<table>
<thead>
<tr>
<th>$C_{NP,0.1}$</th>
<th>0.4</th>
<th>0.3</th>
<th>0.02</th>
<th>0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.008</td>
<td>0.018</td>
<td>0.018</td>
<td>0.0084</td>
</tr>
<tr>
<td>$C_{PP,1,7}$</td>
<td>0.2</td>
<td>0.1</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>0.008</td>
<td>0.009</td>
<td>0.0084</td>
<td></td>
</tr>
<tr>
<td>$C_{NP,0.7}$</td>
<td>0.018</td>
<td>0.009</td>
<td>0.008</td>
<td>0.0287</td>
</tr>
<tr>
<td></td>
<td>$C_{NP,1,7}$</td>
<td>$C_{NP,0,1}$</td>
<td>$C_{PP,1,7}$</td>
<td>$C_{PP,0,7}$</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>0.4</td>
<td>0.008</td>
<td>0.02</td>
<td>0.0084</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
<td>0.009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td>0.01</td>
<td>0.005</td>
<td>0.0025</td>
<td></td>
</tr>
</tbody>
</table>

$C_{NP,0,7}$ | 0.018 | 0.009 | 0.008 | 0.0287 |
\[ C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP PP} \]

\[ \lambda_{NP \rightarrow NP PP} = 0.5 \]

<table>
<thead>
<tr>
<th>( C_{NP,0,1} )</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
<td>0.009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td></td>
<td></td>
<td>0.0084</td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td>0.01 \times 0.5</td>
<td>0.005 \times 0.5</td>
<td>0.0025 \times 0.5</td>
<td></td>
</tr>
</tbody>
</table>

<p>| ( C_{NP,0,7} ) | 0.018 | 0.009 | 0.008 | 0.0287 |</p>
<table>
<thead>
<tr>
<th>( C_{NP,0.1} )</th>
<th>( C_{PP,1,7} )</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.008</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td></td>
<td>0.0084</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td>0.005</td>
<td>0.0025</td>
<td>0.00125</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| ( C_{NP,0.7} ) | 0.018 | 0.009 | 0.008 | 0.0287 |</p>
<table>
<thead>
<tr>
<th>$C_{NP,0.1}$</th>
<th>0.4</th>
<th>0.3</th>
<th>0.02</th>
<th>0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.008</td>
<td>0.018</td>
<td>0.0084</td>
<td>0.00875</td>
</tr>
<tr>
<td>$C_{NP,0.7}$</td>
<td>0.018</td>
<td>0.009</td>
<td>0.008</td>
<td>0.0287</td>
</tr>
</tbody>
</table>

$C_{PP,1.7}$

$C_{NP,0.1}$ and $C_{NP,0.7}$ are represented in a color-coded format.
<table>
<thead>
<tr>
<th>$C_{PP,0.1}$</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.008</td>
<td></td>
<td>0.012×0.5</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
<td>0.009</td>
<td>0.009×0.5</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td></td>
<td></td>
<td>0.0006×0.5</td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
<td>0.00875</td>
</tr>
</tbody>
</table>

| $C_{NP,0.7}$ | 0.018 | 0.009 | 0.008 | 0.0287 |

**Note:** The table represents the values of $C_{PP,0.1}$ and $C_{NP,0.7}$ with corresponding operations and results.
<table>
<thead>
<tr>
<th>$C_{NP,0.1}$</th>
<th>0.4</th>
<th>0.3</th>
<th>0.02</th>
<th>0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.008</td>
<td>0.018</td>
<td>0.0084</td>
<td>0.00875</td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| $C_{NP,0.7}$ | 0.018 | 0.009 | 0.008 | 0.0287 |
| $C_{NP,0.7}$ | 0.018 | 0.009 | 0.008 | 0.0287 |
| $C_{PP,1.7}$ | 0.2   | 0.1   | 0.05  | 0.03   |
| 0.4          | 0.008 |       |       |        |
| 0.3          | 0.018 | 0.009 |       | 0.0108 |
| 0.02         |       |       | 0.0084|        |
| 0.05         |       | 0.00875|      | 0.0015 $\times$ 0.5 |</p>
<table>
<thead>
<tr>
<th>$C_{NP,0.1}$</th>
<th>$C_{PP,1.7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.00875</td>
</tr>
<tr>
<td>0.02</td>
<td>0.0084</td>
</tr>
<tr>
<td>0.4</td>
<td>0.008</td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_{NP,0.7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.018</td>
</tr>
<tr>
<td>0.009</td>
</tr>
<tr>
<td>0.008</td>
</tr>
<tr>
<td>0.0287</td>
</tr>
</tbody>
</table>
\[ 0.0287 = 0.0084 + 0.00875 + 0.0108 + 0.00075 \]

<table>
<thead>
<tr>
<th></th>
<th>$C_{PP,1,7}$</th>
<th>$C_{NP,0,1}$</th>
<th>$C_{NP,0,7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{PP,1,7}$</td>
<td>0.2</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>0.4</td>
<td>0.008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.018</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td></td>
<td>0.0084</td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td></td>
<td>0.00875</td>
<td>0.00075</td>
</tr>
</tbody>
</table>

$0.018$ $0.009$ $0.008$ $0.0287$
Summary

- Maintain residual sum of all proofs not in $k$-best list
- Redefine operations to update the residual as necessary
- Result is approximate $k$-best proof list for $goal$ and approximate sum of all other proofs of $goal$
- When $k = \infty$, result is exact
Outline

- Background
- Cube Pruning
- Cube Summing
- **Semirings**
- Implementation
- Conclusion
Semirings

- A **semiring** is a tuple \( \langle A, \oplus, \otimes, 0, 1 \rangle \) such that:
  - \( \oplus : A \times A \rightarrow A \) is associative and commutative
  - \( \otimes : A \times A \rightarrow A \) is associative and distributes over \( \oplus \)
  - \( \forall a \in A, a \oplus 0 = a, \quad a \otimes 1 = a, \quad a \otimes 0 = 0 \otimes a = 0 \)

<table>
<thead>
<tr>
<th>Semiring</th>
<th>( A )</th>
<th>( \oplus )</th>
<th>( \otimes )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inside</td>
<td>( \mathbb{R}_{\geq 0} )</td>
<td>( a + b )</td>
<td>( ab )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Viterbi</td>
<td>( \mathbb{R}_{\geq 0} )</td>
<td>( \max(a, b) )</td>
<td>( ab )</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Non-local features break some of the semiring properties!
(see paper for details)
“Generalized” Semirings

Semirings

k-best + residual

k = 0

inside

Baum et al., 1970

k-best proof

Goodman, 1999

k = 1

Viterbi proof

Goodman, 1999

k = \infty

all proof

Goodman, 1999

ignore residual

ignore proof

Viterbi

Viterbi, 1967

cube summing

cube decoding

local features only

local features only
Implementation

- Several implementation tools exist for dynamic programming
  - Dyna (Eisner et al., 2005) and Goodman (1999) assume semirings
  - Hypergraphs (Klein & Manning, 2001; Huang, 2008) do not require semirings but are aimed at decoding

- These could be extended for cube summing, but we instead use a lower-level formalism: arithmetic circuits
Arithmetic Circuits

- Explicitly represent computations to be performed using a directed graph
  - Operators and operands are nodes in the graph
  - A value is associated with each node
  - Operators point to their operands

- Allow automatic differentiation in the reverse mode (Griewank & Corliss, 1991) for efficient gradient computation
Example

\[ C_{NP,0,1}, 0.5 \]

\[ C_{NP,0,7} \rightarrow NP PP \]

\[ C_{PP,1,7}, \lambda_{NP \rightarrow NP PP}, 0.5 \]

\[ ... \]
Outline

- Background
- Cube Pruning
- Cube Summing
- Semirings
- Implementation
- Conclusion
Conclusion and Ongoing Work

- We have described cube summing, a technique for approximate summing using dynamic programming with non-local features.

- With only local features, cube summing is a semiring that generalizes those in common use.

- Some semiring properties are broken by non-local features but an implementation based on arithmetic circuits can be used.

- We are currently using cube summing to train a log-linear syntactic translation model with hidden variables.
Thanks!

Cube Summing, Approximate Inference with Non-Local Features, and Dynamic Programming without Semirings

Kevin Gimpel and Noah A. Smith