Approximation-aware Dependency Parsing by Belief Propagation

Matt Gormley
Mark Dredze
Jason Eisner

September 19, 2015
TACL at EMNLP
Motivation #1: Approximation-unaware Learning

**Problem:** Approximate inference causes standard learning algorithms to go awry (Kulesza & Pereira, 2008)

Can we take our approximations into account?
Graphical models let you encode domain knowledge.

Neural nets are really good at fitting the data discriminatively to make good predictions.

Could we define a neural net that incorporates domain knowledge?
Our Solution

**Key idea:** Treat your unrolled approximate inference algorithm as a deep network

![Diagram of a chart parser with symbols and equations related to inference and deep networks.](chart-parser-diagram.png)
Eaton et al., 2011)

Loopy BP + Dynamic Prog. = Structured BP

Loopy BP + Backprop. = ERMA / Back-BP

Loopy BP + Dynamic Prog. + Backprop. = This Talk
Loopy BP + Dynamic Prog. + Backprop. = This Talk

Graphical + Hypergraphs + Neural Networks = The models that interest me

• If you’re thinking, “This sounds like a great direction!”
• Then you’re in good company
• And have been since before 1995
Loopy BP + Dynamic Prog. + Backprop. = This Talk

Graphical + Hypergraphs + Neural Networks = The models that interest me

• So what’s new since 1995?
• Two new emphases:
  1. Learning under approximate inference
  2. Structural constraints
An Abstraction for Modeling

Factor Graph (bipartite graph)
- variables (circles)
- factors (squares)

\[
\begin{align*}
\psi_2 & \quad \psi_{12} \\
y_1 & \\
y_2
\end{align*}
\]
Factor Graph for Dependency Parsing
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

Juan Carlos
abdica
reino

Left arc
Right arc
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

Unary: local opinion about one edge

 Unary	
  local	
  opinion	
  about	
  one	
  edge

Juan_Carlos
abdica
reino

0
1
2
3
4

Juan Carlos
abdica
su
reino

(Smith & Eisner, 2008)
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

Unary: local opinion about one edge

Juan Carlos
abdica
reino

0
1
2
3
4

(WALL)
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

PTree: Hard constraint, multiplying in 1 if the variables form a tree and 0 otherwise.

Unary: local opinion about one edge
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

PTree: Hard constraint, multiplying in 1 if the variables form a tree and 0 otherwise.

Unary: local opinion about one edge
Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

- PTree: Hard constraint, multiplying in 1 if the variables form a tree and 0 otherwise.
- Unary: local opinion about one edge
- Grandparent: local opinion about grandparent, head, and modifier

Juan Carlos suabdica reino

PTree:
- Hard constraint, multiplying in 1 if the variables form a tree and 0 otherwise.

Unary:
- local opinion about one edge

Grandparent:
- local opinion about grandparent, head, and modifier

0 <WALL> 1 JuanCarlos 2 abdica 3 su 4 reino
Factor Graph for Dependency Parsing

(Riedel and Smith, 2010)
(Martins et al., 2010)

PTree: Hard constraint, multiplying in 1 if the variables form a tree and 0 otherwise.

Unary: local opinion about one edge

Grandparent: local opinion about grandparent, head, and modifier

Sibling: local opinion about pair of arbitrary siblings
Now we can work at this level of abstraction.

\[ p_\theta(y) = \frac{1}{Z} \prod_\alpha \psi_\alpha(y_\alpha) \]
Why *dependency parsing*?

1. Simplest example for Structured BP
2. Exhibits both polytime and NP-hard problems
The Impact of Approximations

Linguistics

(time flies like an arrow)

Model

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

Inference

(NP-hard)

(Inference is usually called as a subroutine in learning)
The Impact of Approximations

Linguistics

Inference

Model

$p_0(\text{time, flies, like, an, arrow}) = 0.50$

Poly-time approximation!

(Inference is usually called as a subroutine in learning)
The Impact of Approximations

**Linguistics**

- time
- flies
- like
- an
- arrow

**Model**

- $p_0(\text{time flies like an arrow}) = 0.50$

**Inference**

- Poly-time approximation!

- (Inference is usually called as a subroutine in learning)

**Learning**

- Does learning know inference is approximate?
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

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

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

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

1. Choose **model**
   (3. comes from log-linear factors)
   \[ p_\theta(y) = \frac{1}{Z} \prod_\alpha \exp(\theta \cdot f_\alpha(y_\alpha)) \]

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

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

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

How did we compute these *approximate* marginal probabilities anyway?

By *Structured Belief Propagation* of course!
Everything you need to know about: Structured BP

1. It’s a message passing algorithm
2. The message computations are just multiplication, addition, and division
3. Those computations are differentiable
Structured Belief Propagation

This is just another factor graph, so we can run Loopy BP

What goes wrong?
• Naïve computation is inefficient
• We can embed the inside-outside algorithm within the structured factor

Juan Carlos
abdica
reino

(Wall & Eisner, 2008)
Algorithmic Differentiation

• Backprop works on more than just neural networks
• You can apply the chain rule to any arbitrary differentiable algorithm

That’s the key (old) idea behind this talk.

• Alternatively: could estimate a gradient by finite-difference approximations – but algorithmic differentiation is much more efficient!
Feed-forward Topology of Inference, Decoding and Loss

- Unary factor: vector with 2 entries
- Binary factor: (flattened) matrix with 4 entries

\[ \psi_\alpha(y_\alpha) \]

\[ \theta \]

Factors
Model parameters
Feedforward Topology of Inference, Decoding and Loss

- Messages from neighbors used to compute next message
- Leads to sparsity in layerwise connections

\[
\psi_\alpha(y_\alpha) \quad \theta
\]

Messages at time \(t=1\)
Messages at time \(t=0\)
Factors
Model parameters
Arrows in Neural Net:
Linear combination, then a sigmoid

\[ a_i = \sigma \left( \sum_j \theta_j b_j \right) \]
**Arrows in Neural Net:**
Linear combination, then a sigmoid

\[ a_i = \sigma \left( \sum_j \theta_j b_j \right) \]

**Arrows in This Diagram:**
A different semantics given by the algorithm

\[ m_{\alpha \rightarrow i}(y_i) = \frac{1}{\kappa_{\alpha \rightarrow i}} \sum_{y_{\alpha \sim y_i}} \psi_\alpha(y_\alpha) \prod_{j \in N(\alpha) \setminus i} m_{j \rightarrow \alpha}(y_i) \]

Messages at time \( t=1 \)

Messages at time \( t=0 \)

Factors

Model parameters
Feed-forward Topology

\[ L(y^*, \theta) \]

\[ b_i(y_i) \]

\[ m^{(t)}_{i \rightarrow \alpha}(y_i) \]

\[ m^{(t)}_{\alpha \rightarrow i}(y_i) \]

\[ \psi_{\alpha}(y_{\alpha}) \]

\[ \theta \]
Messages from PTree factor rely on a variant of inside-outside

\[ m_{i \rightarrow \alpha}(y_i) = \frac{1}{\kappa_{i \rightarrow \alpha}} \sum_{y_i \sim y_i} \psi_{\alpha}(y_{\alpha}) \prod_{j \in N(\alpha) \setminus i} m_{j \rightarrow \alpha}(y_i) \]

Factors

Model parameters

Arrows in This Diagram:
A different semantics given by the algorithm

Messages at time \( t=1 \)

Messages at time \( t=0 \)
Messages from PTree factor rely on a variant of inside-outside

\[ m_{i \rightarrow \alpha}^{(t)}(y_i) \]

\[ m_{\alpha \rightarrow i}^{(t)}(y_i) \]

\[ \psi_{\alpha}(y_{\alpha}) \]

\[ \theta \]
Approximation-aware Learning

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)

Key idea: Open up the black box!
Experimental Setup

Goal: Compare two training approaches

1. Standard approach (CLL)
2. New approach (Backprop)

Data: English PTB

– Converted to dependencies using Yamada & Matsumoto (2003) head rules
– Standard train (02-21), dev (22), test (23) split
– TurboTagger predicted POS tags

Metric: Unlabeled Attachment Score
(higher is better)
Results

Speed-Accuracy Tradeoff

New training approach yields models which are:

1. Faster for a given level of accuracy
2. More accurate for a given level of speed
Results

Increasingly Cyclic Models

• As we add more factors to the model, our model becomes loopier.

• Yet, our training by Backprop consistently improves as models get richer.
See our TACL paper for...

1) Results on 19 languages from CoNLL 2006 / 2007

2) Results with alternate training objectives

3) Empirical comparison of exact and approximate inference

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>DEV UAS</th>
<th>TEST UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>91.99</td>
<td>91.62</td>
</tr>
<tr>
<td>French</td>
<td>91.37</td>
<td>91.25</td>
</tr>
<tr>
<td>German</td>
<td>91.91</td>
<td>91.88</td>
</tr>
<tr>
<td>Italian</td>
<td>91.91</td>
<td>91.66</td>
</tr>
<tr>
<td>Spanish</td>
<td>91.83</td>
<td>91.63</td>
</tr>
</tbody>
</table>

```
\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|c|c|c|c|}
\hline
\textbf{Train} & \textbf{Inference} & \textbf{Dev UAS} & \textbf{Test UAS} \\
\hline
CLL & Exact & 91.99 & 91.62 \\
CLL & BP 4 iters & 91.37 & 91.25 \\
L_2 & Exact & 91.91 & 91.66 \\
L_2 & BP 4 iters & 91.83 & 91.63 \\
\hline
\end{tabular}
\end{table}
```
Comparison of Two Approaches

1. **CLL with approximate inference**
   - A totally ridiculous thing to do!
   - But it’s been done for years because it often works well
   - (Also named “surrogate likelihood training” by Wainright (2006))
Comparison of Two Approaches

Key idea: Open up the black box!

2. Approximation-aware Learning for NLP

- In hindsight, treating the approximations as part of the model is the obvious thing to do (Domke, 2010; Domke, 2011; Stoyanov et al., 2011; Ross et al., 2011; Stoyanov & Eisner, 2012; Hershey et al., 2014)

- Our contribution: Approximation-aware learning with **structured factors**

- But there's some challenges to get it right (numerical stability, efficiency, backprop through structured factors, annealing a decoder’s argmin)

- Sum-Product Networks are similar in spirit (Poon & Domingos, 2011; Gen & Domingos, 2012)
Takeaways

• New learning approach for Structured BP maintains high accuracy with fewer iterations of BP, even with cycles

• Need a neural network? Treat your unrolled approximate inference algorithm as a deep network
Questions?

Pacaya - Open source framework for hybrid graphical models, hypergraphs, and neural networks

Features:
- Structured BP
- Coming Soon: Approximation-aware training

Language: Java

URL: https://github.com/mgormley/pacaya