

# Approximation-aware Dependency Parsing by Belief Propagation

Matt Gormley

Mark Dredze

Jason Eisner

September 19, 2015

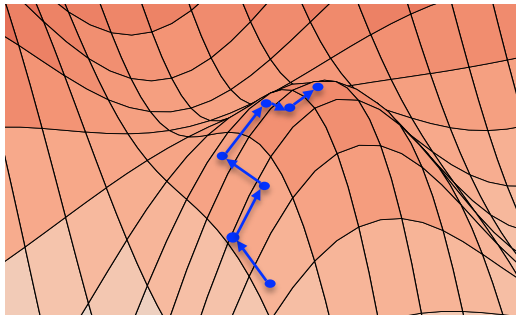
TACL at EMNLP

# Motivation #1:

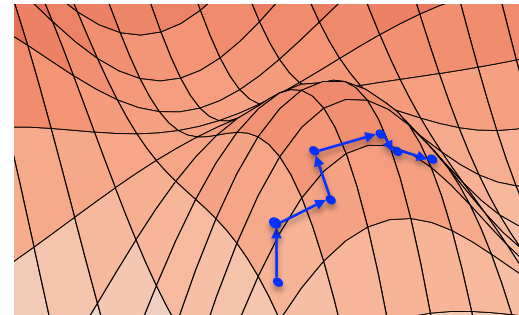
## Approximation-unaware Learning

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

with exact inference:



with approx. inference:

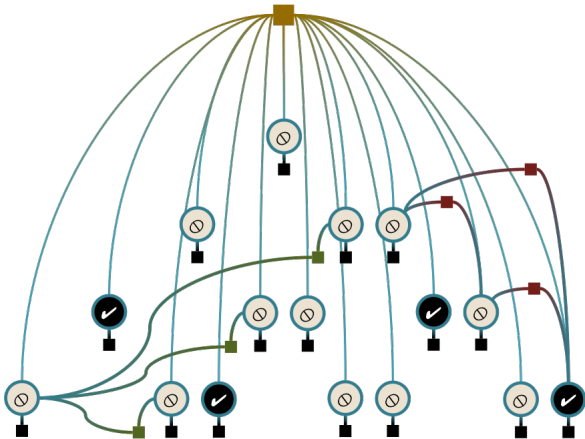


Can we take our  
approximations  
into account?

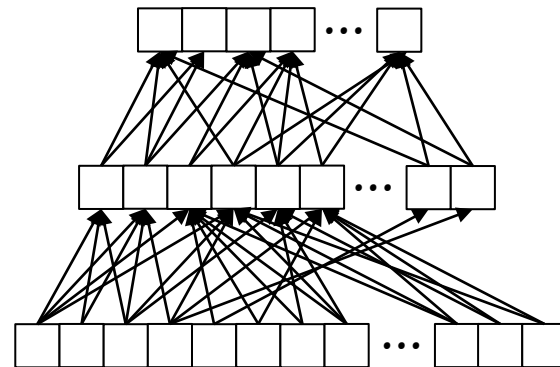
# Motivation #2:

## Hybrid Models

**Graphical models** let you encode domain knowledge



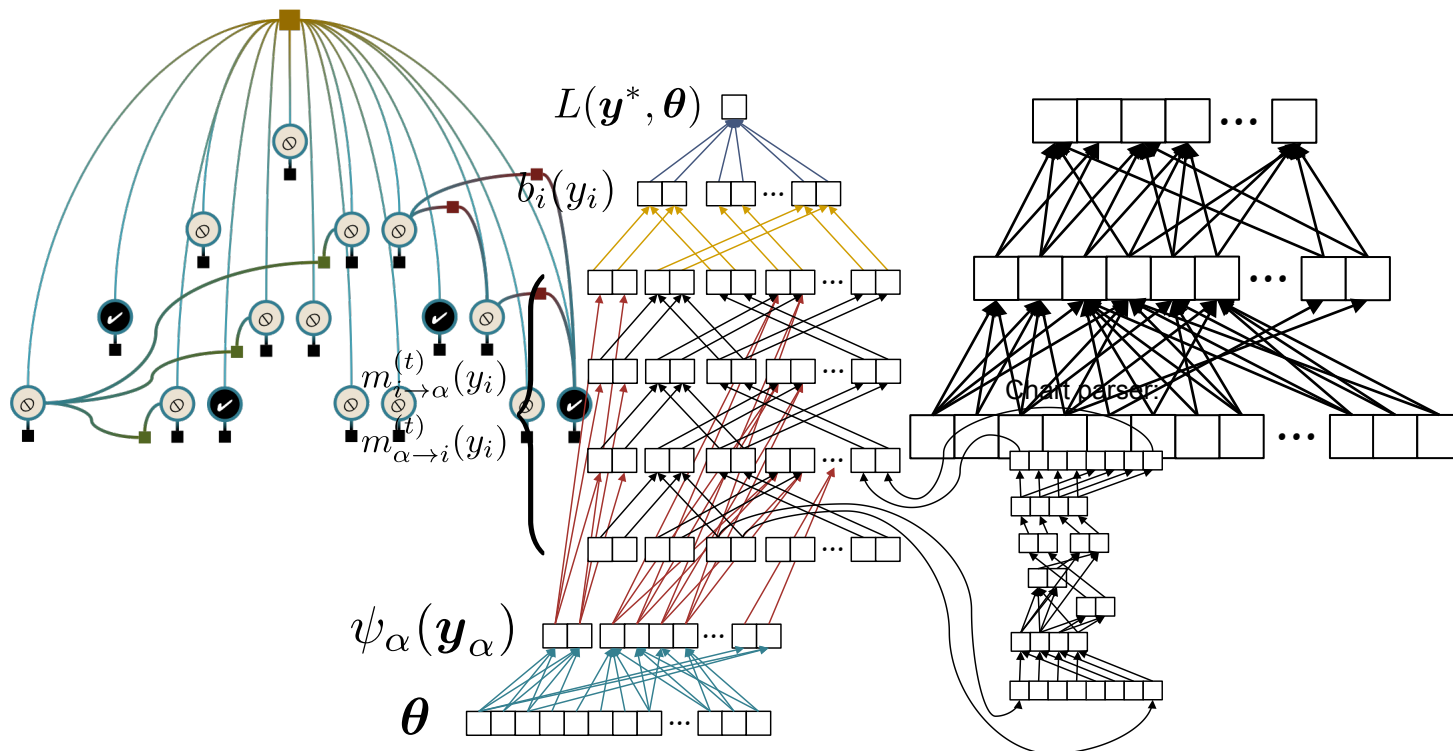
**Neural nets** are really good at fitting the data discriminatively to make good predictions



**Could we define a neural net  
that incorporates  
domain knowledge?**

# Our Solution

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





# Talk Summary

(Smith & Eisner, 2008)

Loopy BP + Dynamic Prog. = Structured BP



(Eaton & Ghahramani, 2009)  
(Stoyanov et al., 2011)

Loopy BP + Backprop.



= ERMA / Back-BP



Loopy BP + Dynamic Prog. + Backprop.



= This Talk



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



Graphical + Hypergraphs Models + 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

### LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition



Yoshua Bengio\*  
 bengioy@iro.umontreal.ca

Yann LeCun  
 yann@research.att.com



Craig Nohl  
 nohl@research.att.com

Chris Burges  
 burges@research.att.com



AT&T Bell Laboratories  
 Rm 4G332, 101 Crawfords Corner Road  
 Holmdel, NJ 07733

To appear in *Neural Computation*, Volume 7, Number 5, 1995

Loopy BP + Dynamic Prog. + Backprop.



= This Talk



Graphical Models + Hypergraphs + Neural Networks

= The models that interest me

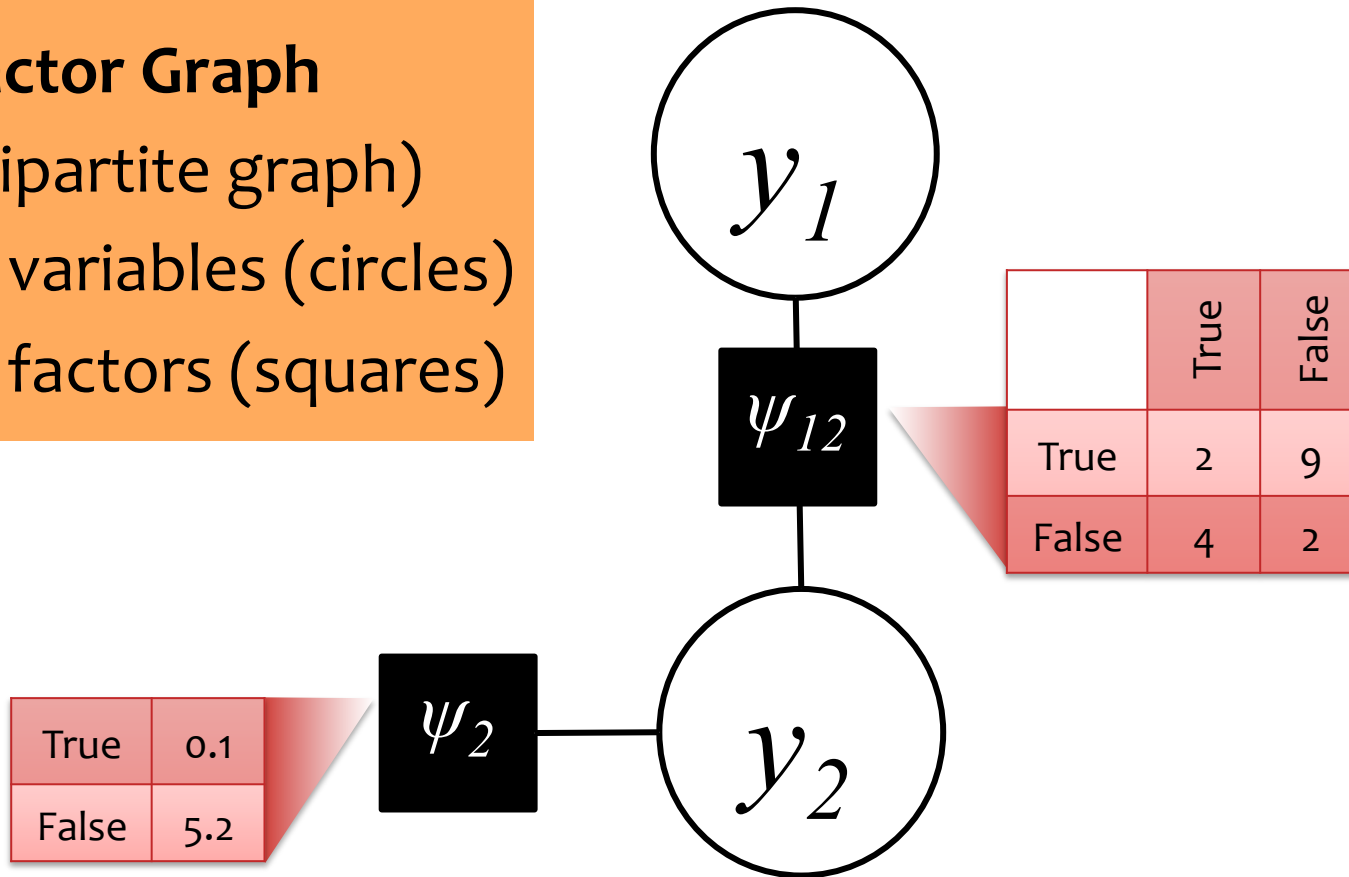
- So what's new since 1995?
- Two new emphases:
  1. Learning under approximate inference
  2. Structural constraints

# An Abstraction for Modeling

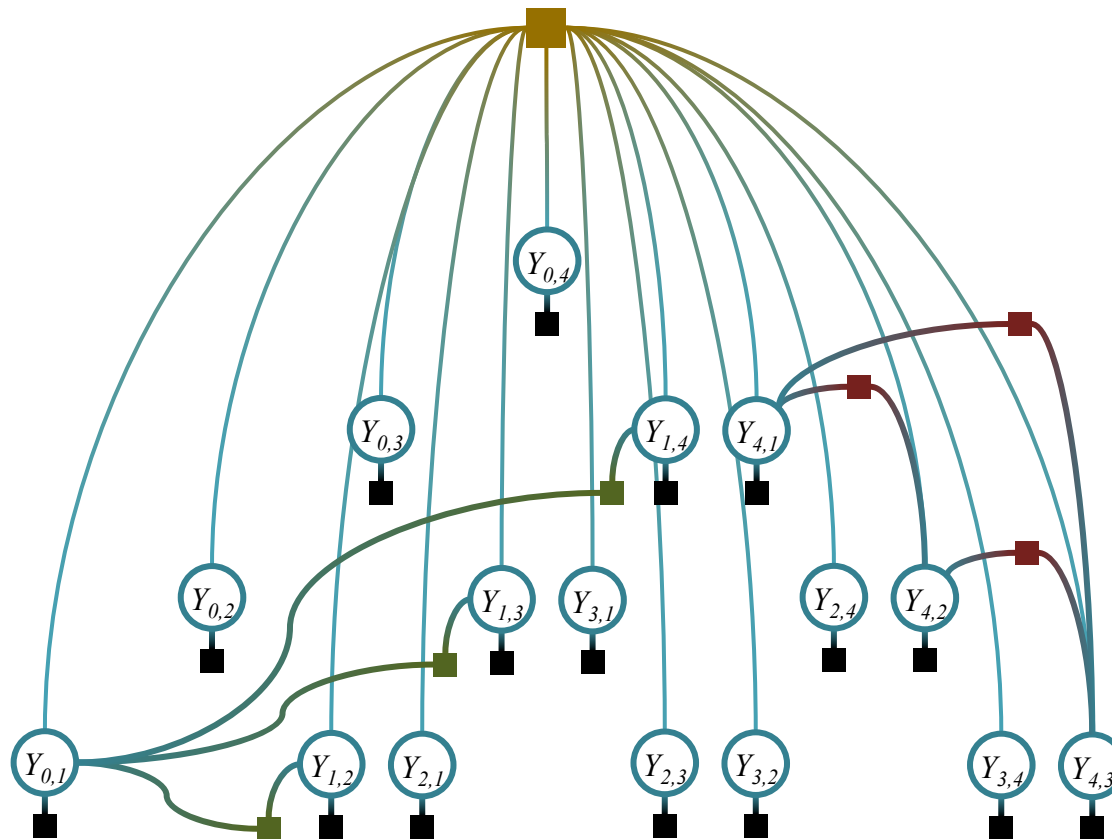
## Factor Graph

(bipartite graph)

- variables (circles)
- factors (squares)

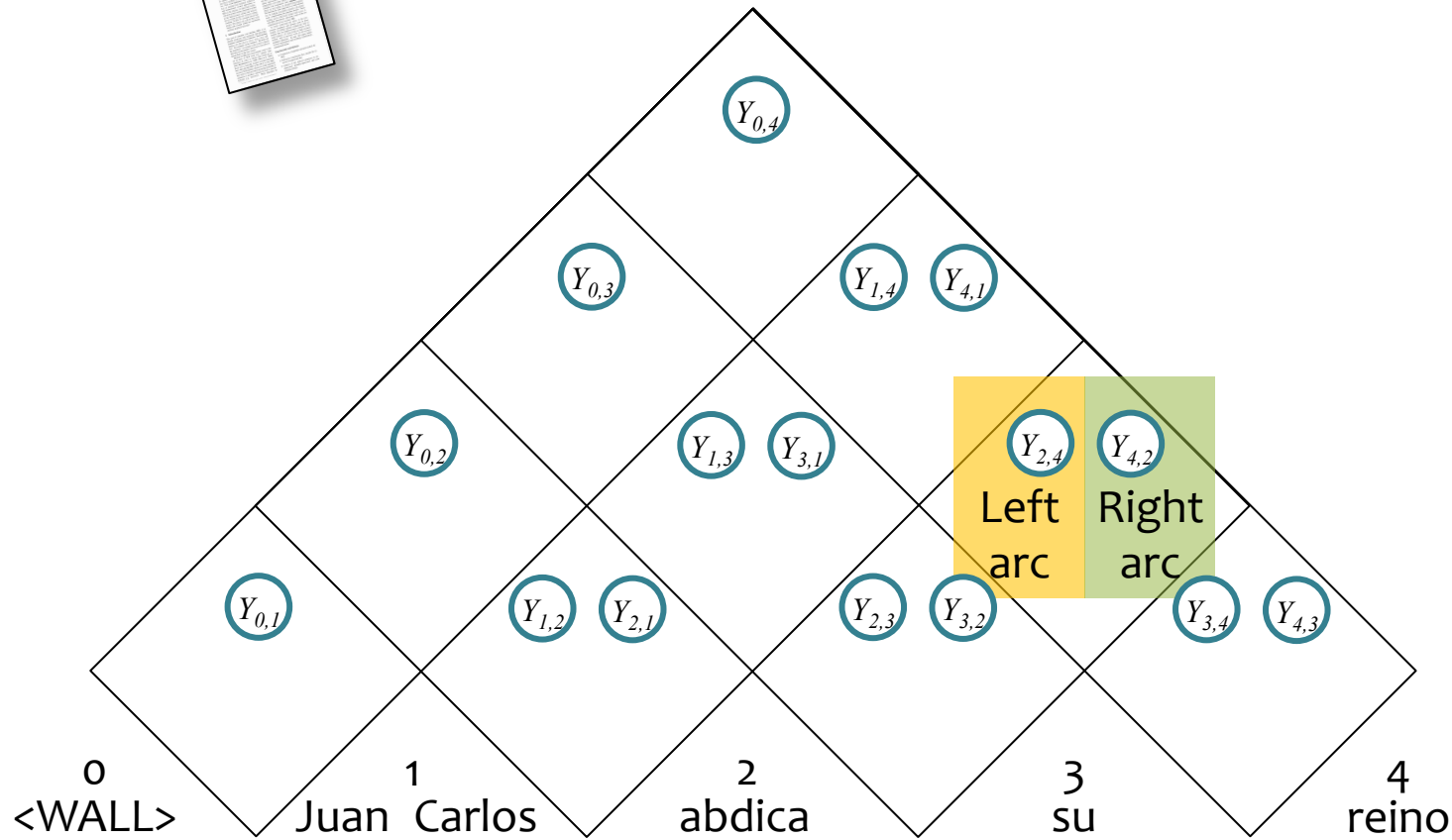


# Factor Graph for Dependency Parsing



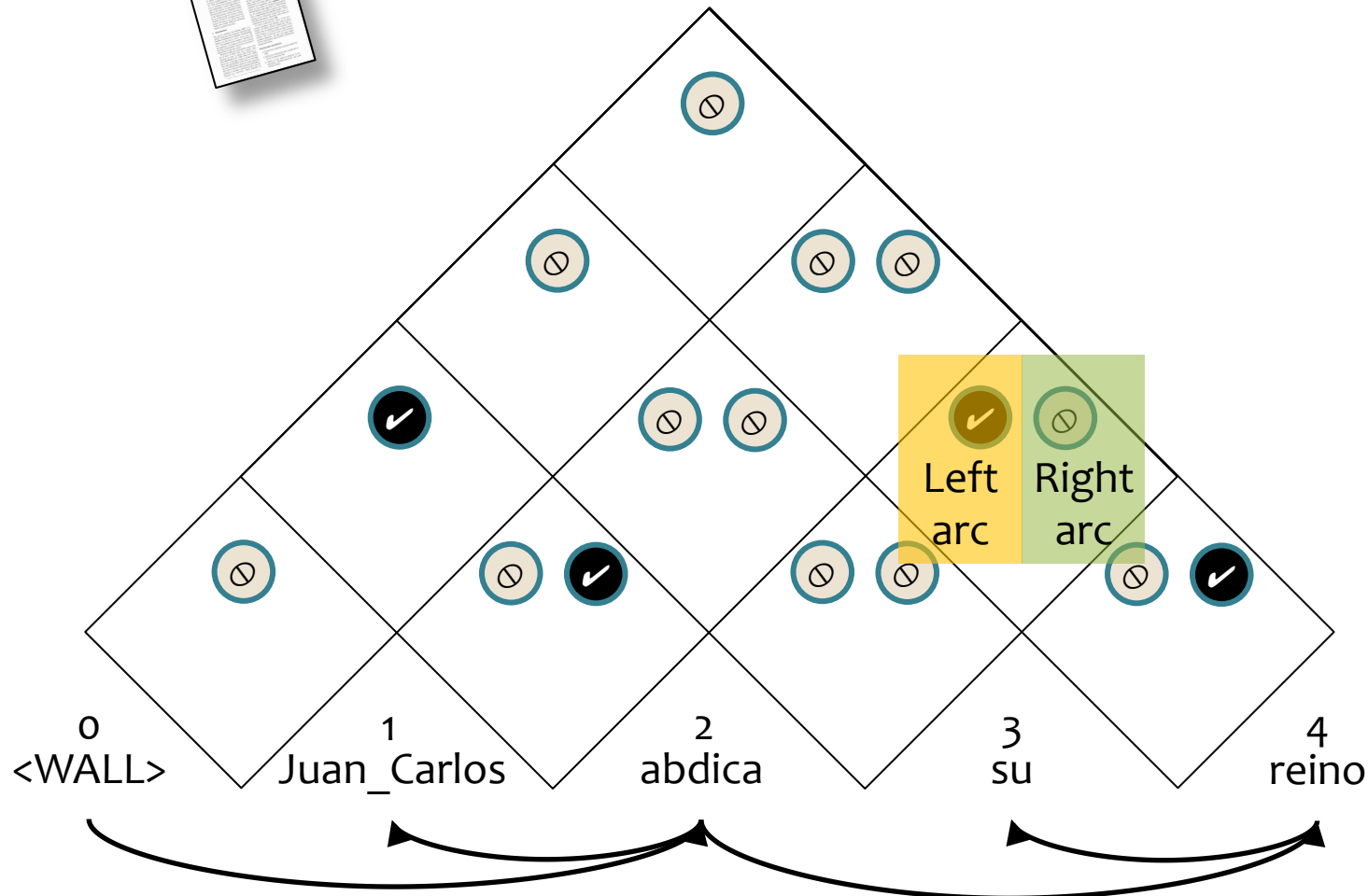
# Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)



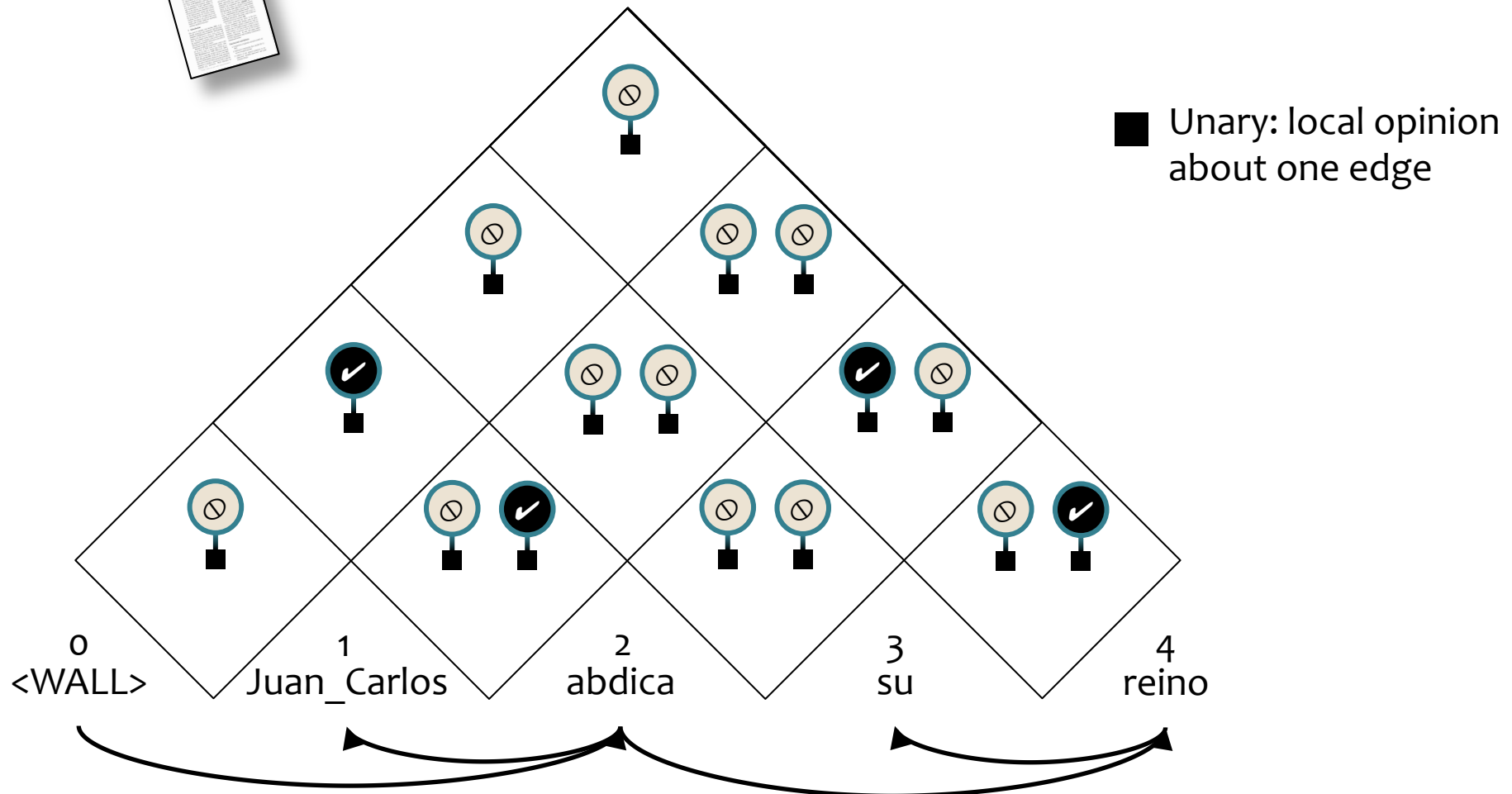
# Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)



# Factor Graph for Dependency Parsing

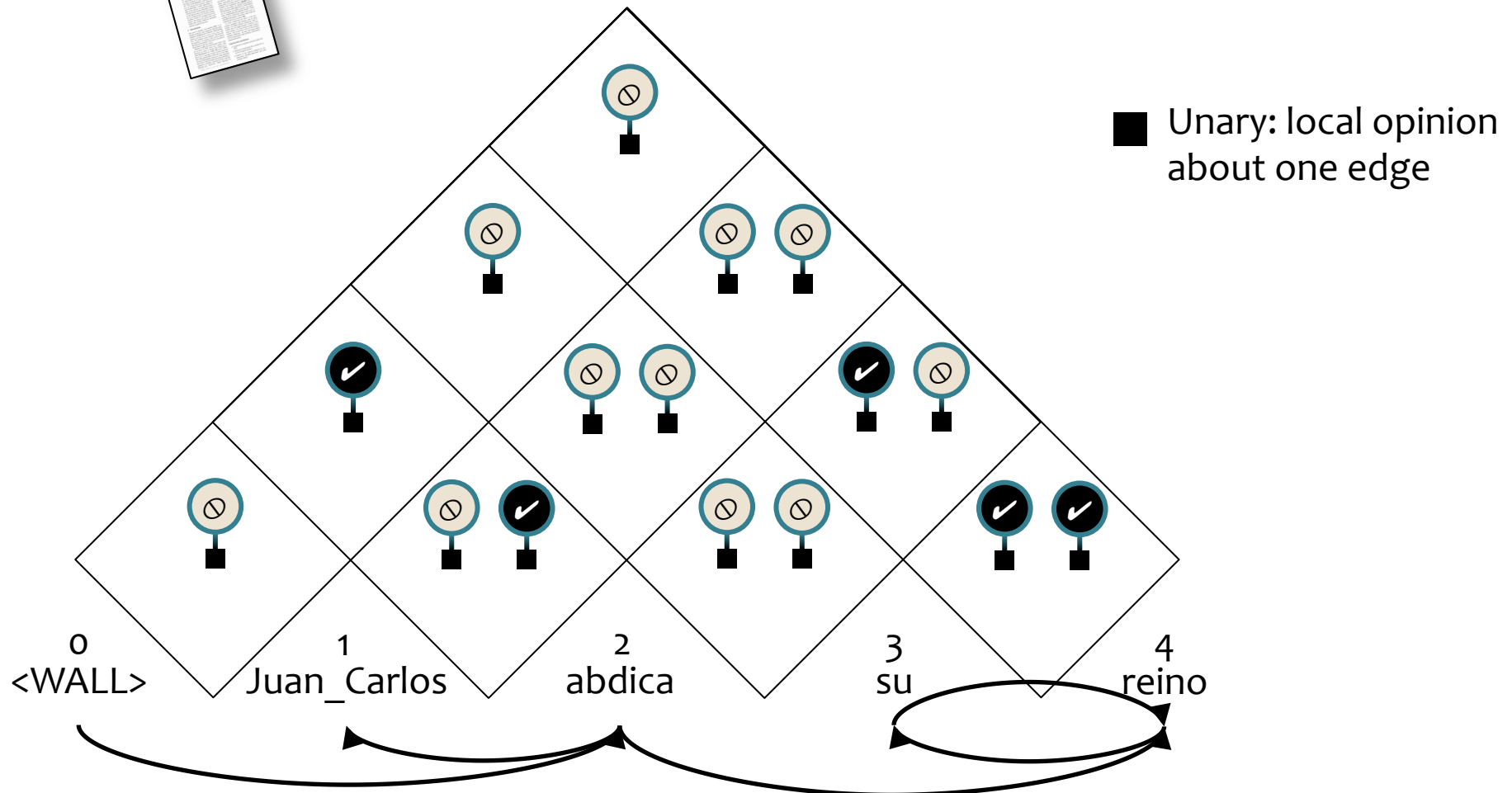
(Smith & Eisner, 2008)





# Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

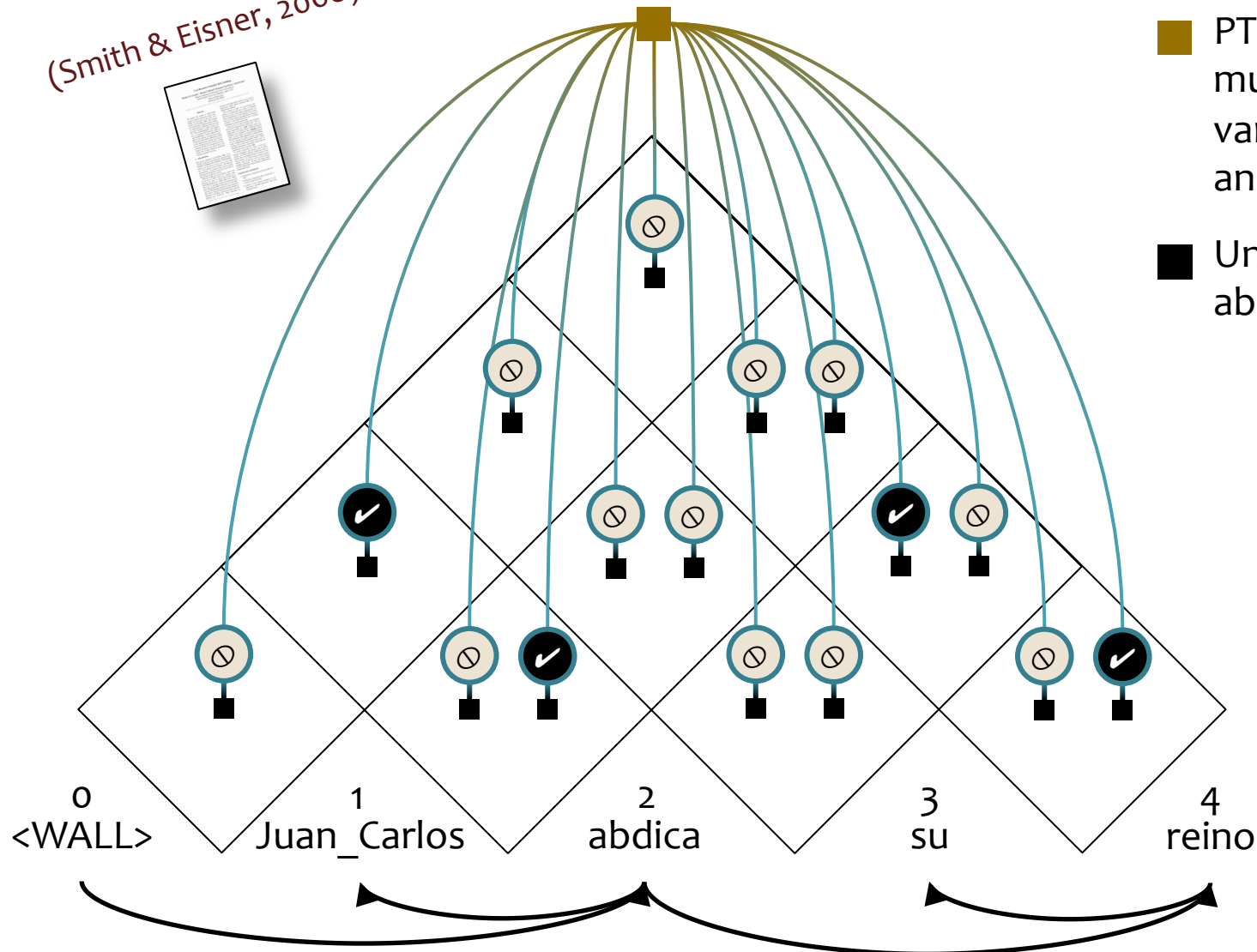


# Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)



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

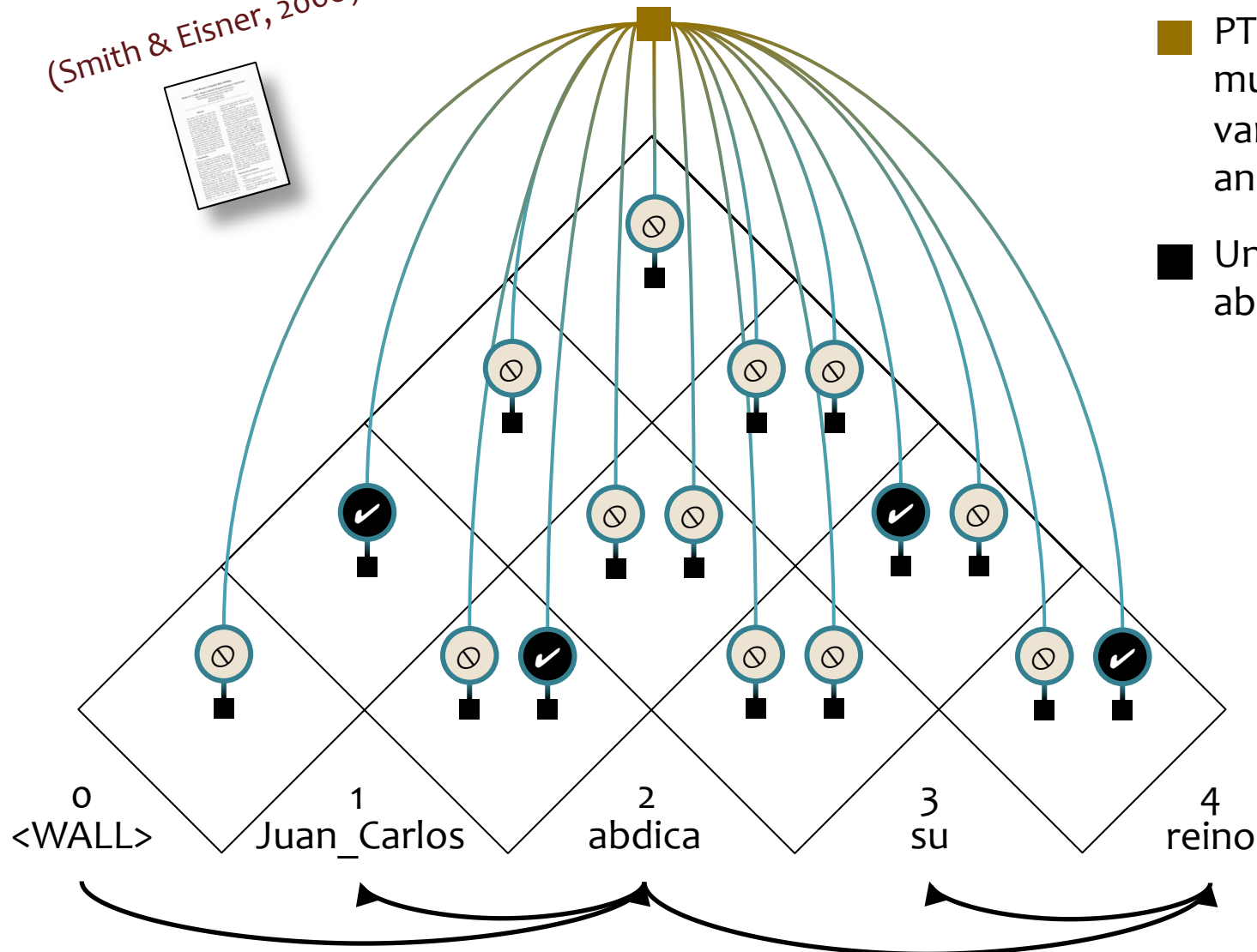


# Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)



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

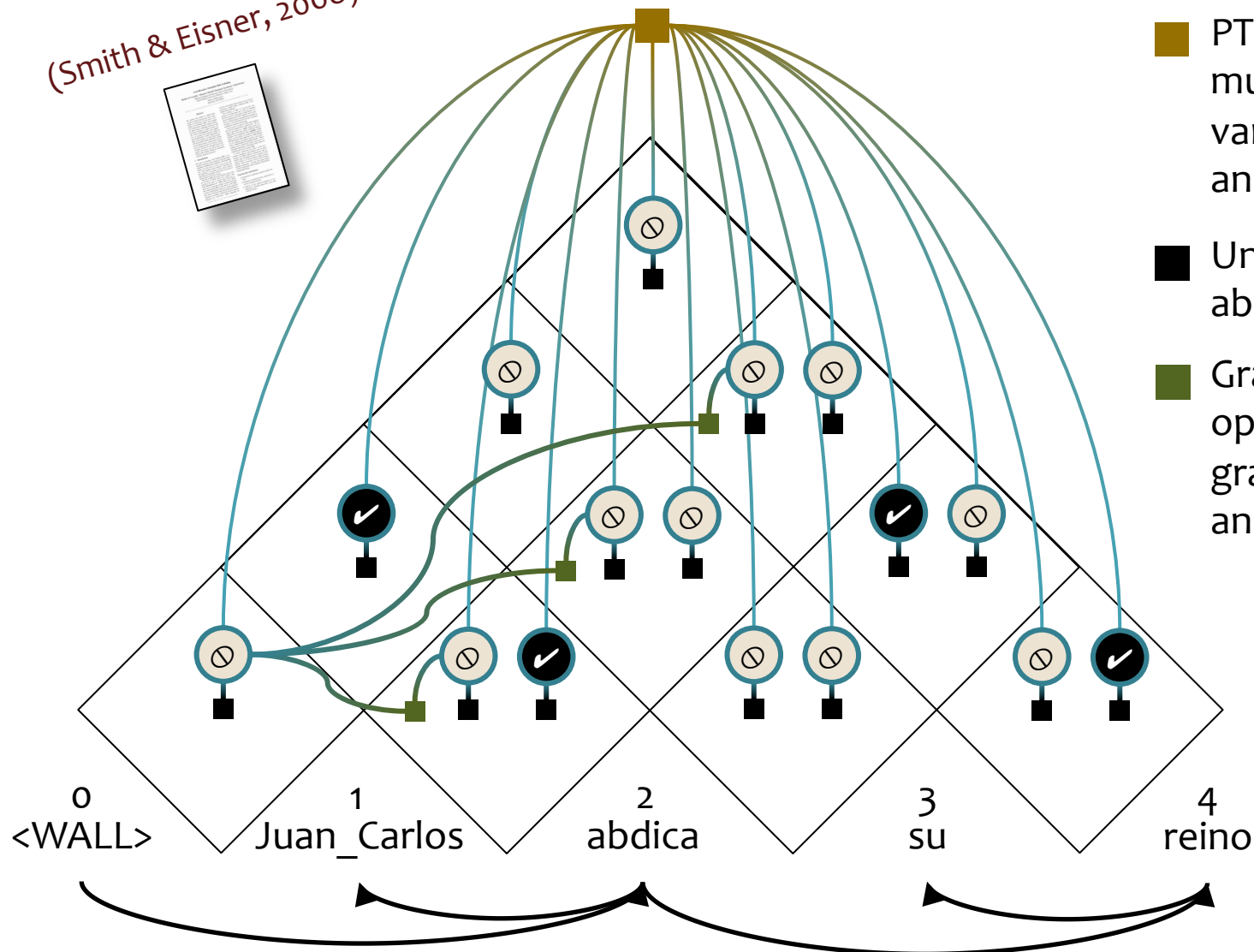


# Factor Graph for Dependency Parsing

(Smith & Eisner, 2008)

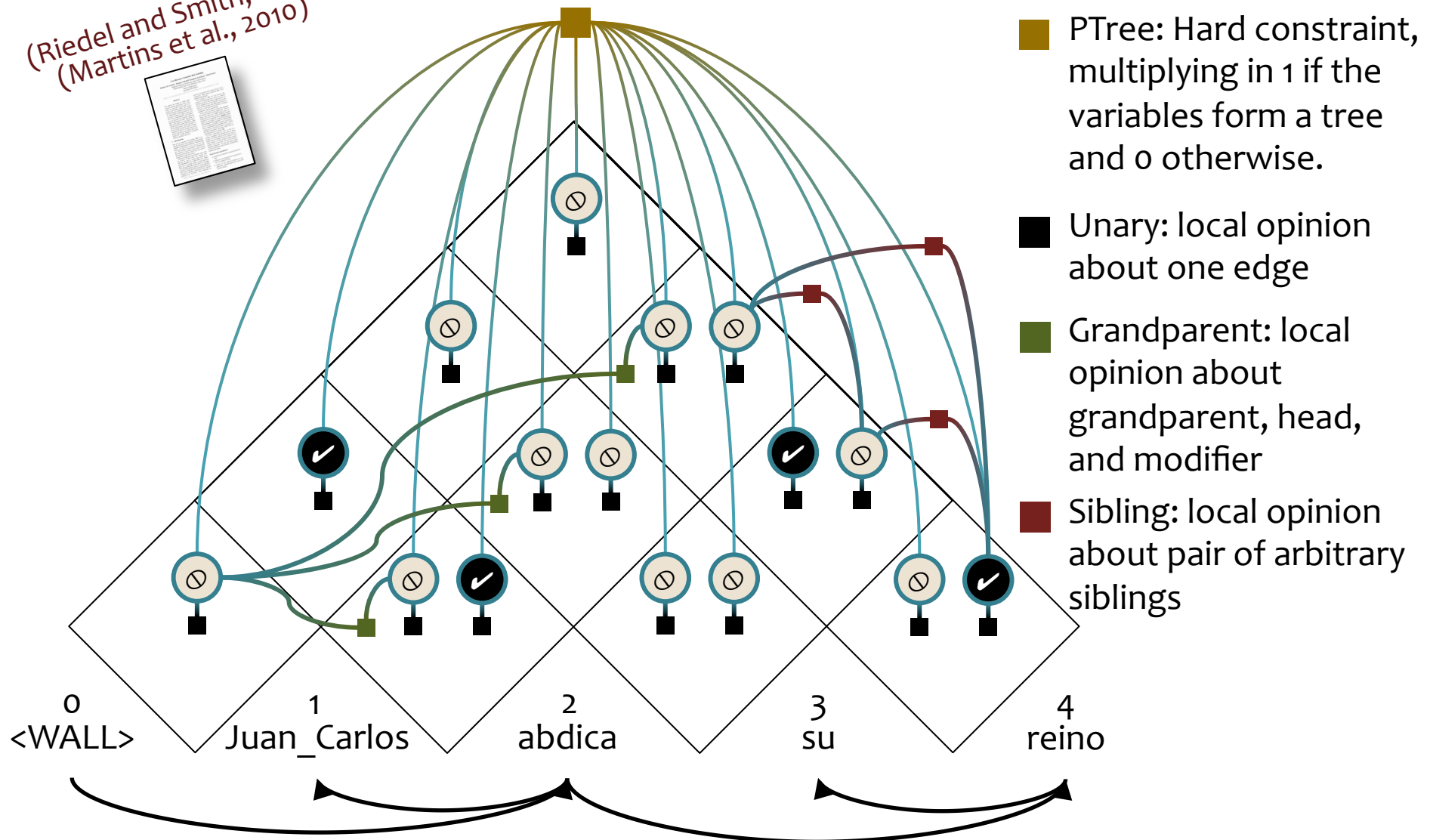


- PTree: Hard constraint, multiplying in 1 if the variables form a tree and 0 otherwise.
- Unary: local opinion about one edge
- Grandparent: local opinion about grandparent, head, and modifier



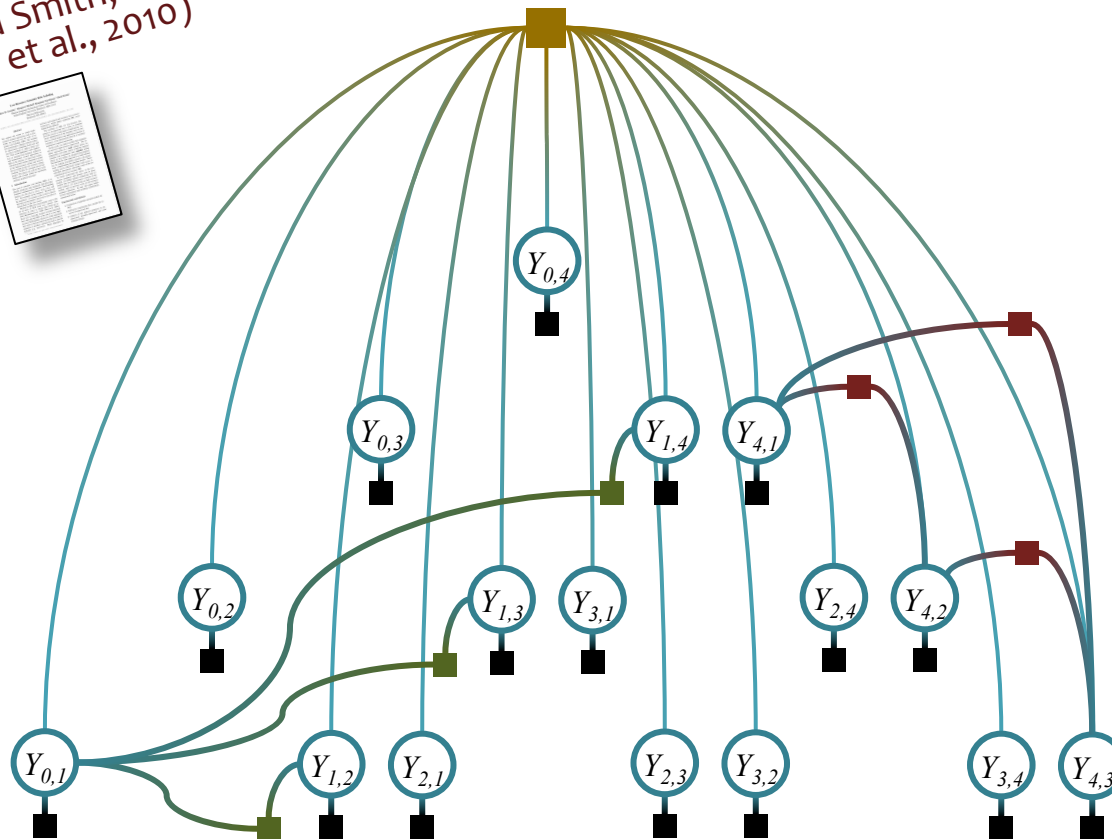
# Factor Graph for Dependency Parsing

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



# Factor Graph for Dependency Parsing

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



Now we can  
work at this  
level of  
abstraction.

$$p_{\theta}(\mathbf{y}) = \frac{1}{Z} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}_{\alpha})$$

# ***Why dependency parsing?***

- 1. Simplest example for Structured BP**
- 2. Exhibits both polytime and NP-hard problems**

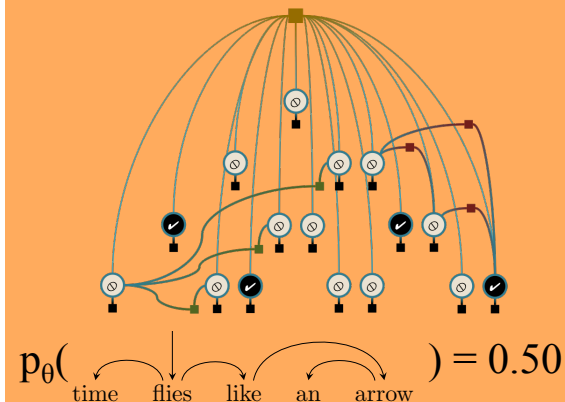
# The Impact of Approximations

## Linguistics

time flies like an arrow



## Model



## Inference



**NP-hard**

(Inference is usually called as a subroutine in learning)



## Learning





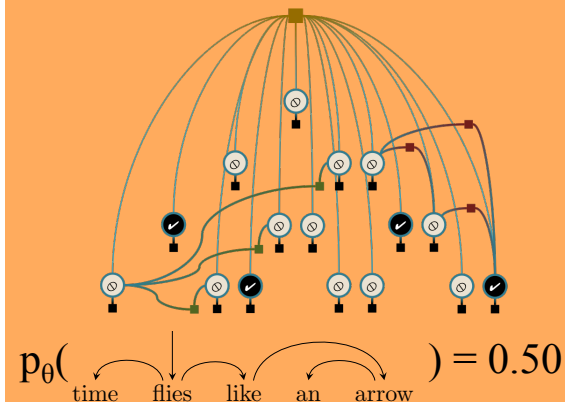
# The Impact of Approximations

## Linguistics

time flies like an arrow

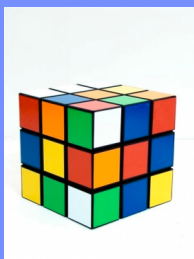


## Model



## Learning

## Inference



**Poly-time approximation!**



(Inference is usually called as a subroutine in learning)



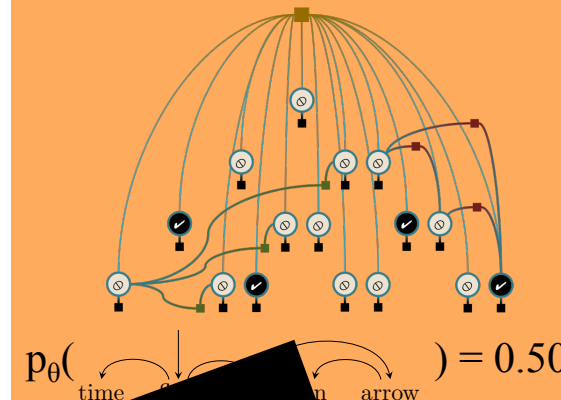
# The Impact of Approximations

## Linguistics

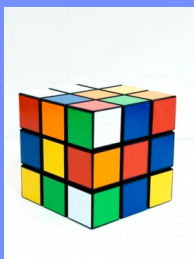
time flies like an arrow



## Model



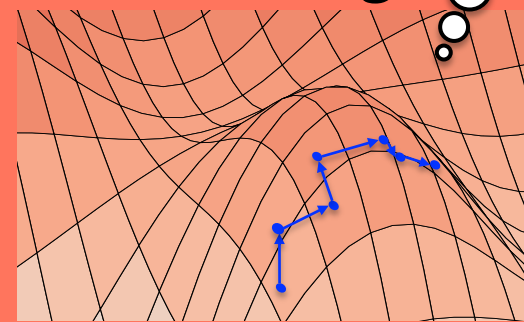
## Inference



Poly-time  
approximation!

Does learning  
know inference is  
approximate?

## Learning



(Inference is usually  
called as a subroutine  
in learning)



# Conditional Log-likelihood Training

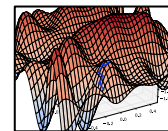
1. Choose **model**

$$p_{\theta}(\mathbf{y}) = \frac{1}{Z} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}_{\alpha})$$

2. Choose **objective**:

Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\mathbf{y} \in \mathcal{D}} \log p_{\theta}(\mathbf{y})$$



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

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

# Conditional Log-likelihood Training

1. Choose **model**

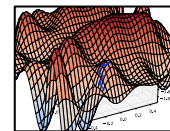
(3. comes from log-linear factors)

$$p_{\theta}(\mathbf{y}) = \frac{1}{Z} \prod_{\alpha} \exp(\theta \cdot \mathbf{f}_{\alpha}(\mathbf{y}_{\alpha}))$$

2. Choose **objective**:

Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\mathbf{y} \in \mathcal{D}} \log p_{\theta}(\mathbf{y})$$



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

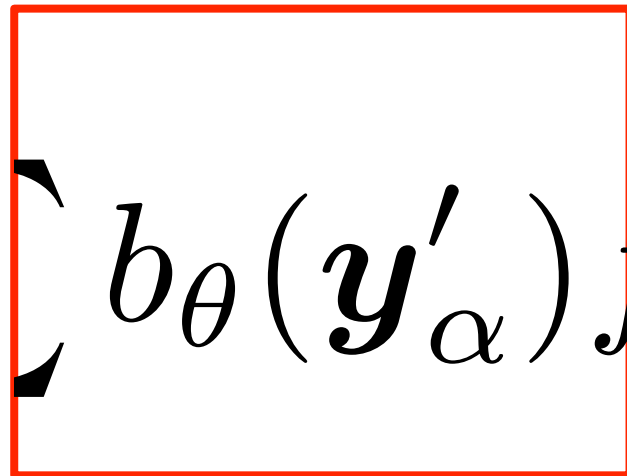
$$\frac{dL(\theta)}{d\theta_j} = \sum_{\mathbf{y} \in \mathcal{D}} \left( \sum_{\alpha} \left[ f_{\alpha,j}(\mathbf{y}_{\alpha}) - \sum_{\mathbf{y}'} p_{\theta}(\mathbf{y}'_{\alpha}) f_{\alpha,j}(\mathbf{y}'_{\alpha}) \right] \right)$$

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

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

# What's wrong with CLL?

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


$$b_{\theta}(y'_{\alpha})$$

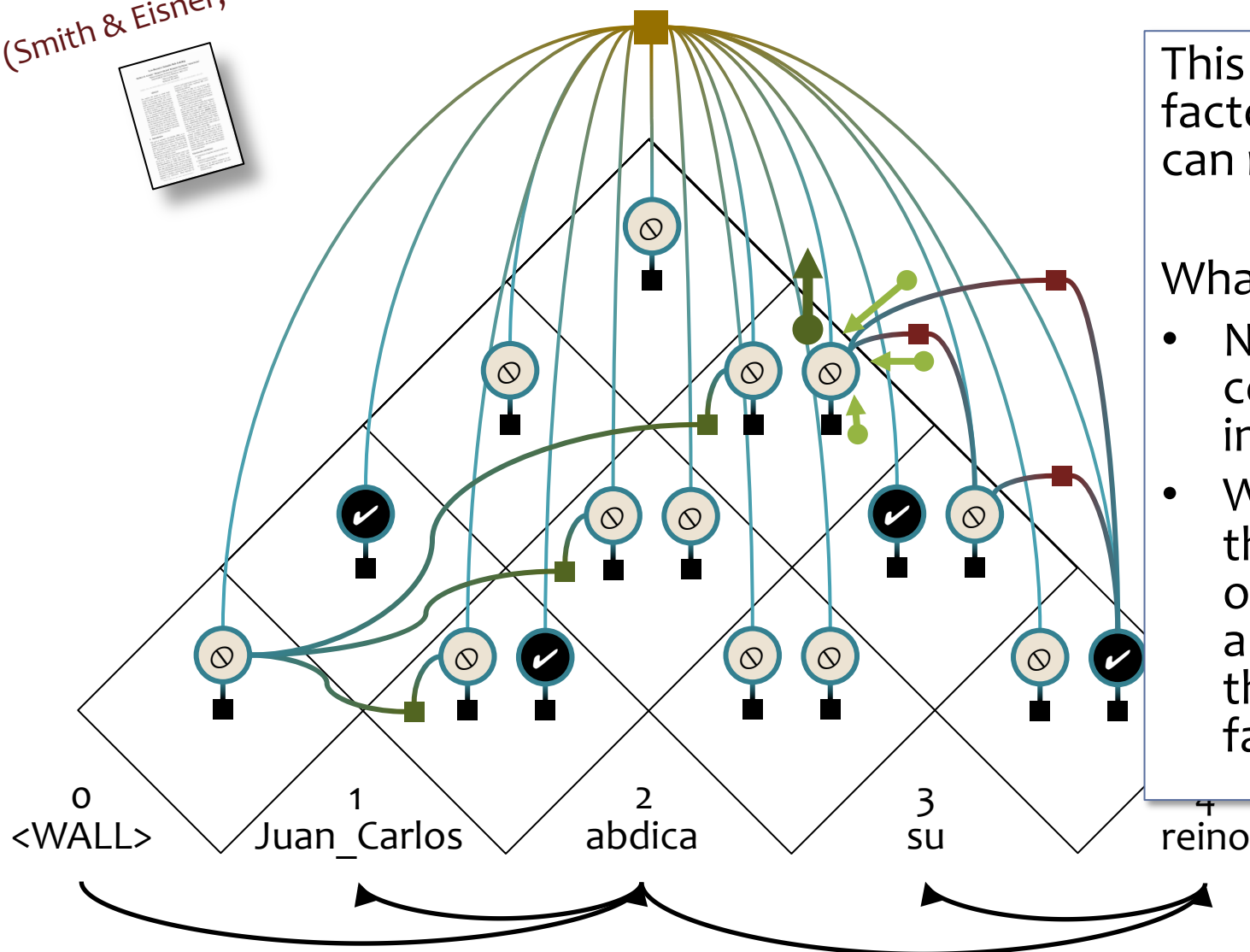
By **Structured Belief Propagation** of course!

# 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

(Smith & Eisner, 2008)



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

# Algorithmic Differentiation

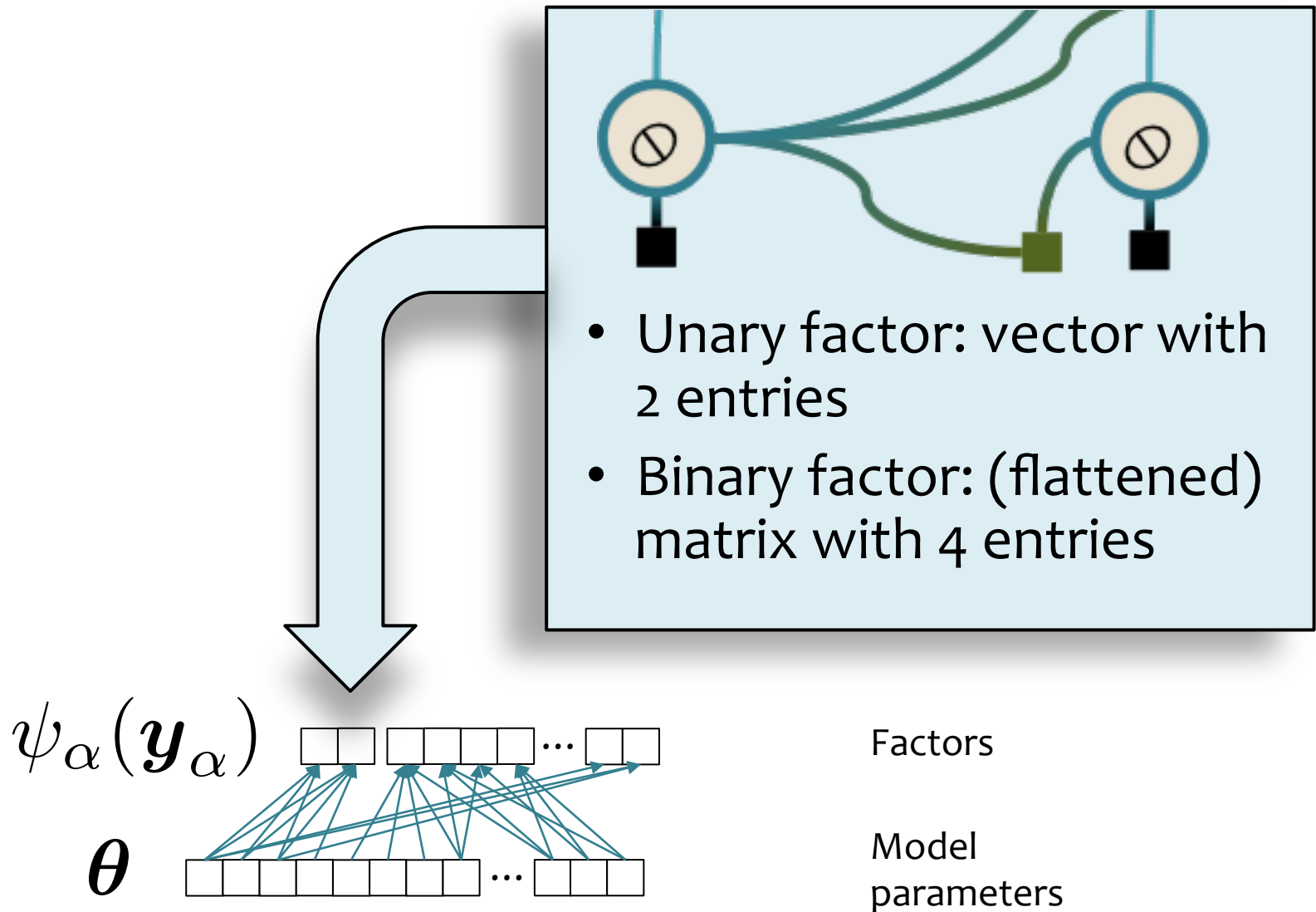
- Backprop works on more than just neural networks
- You can apply the chain rule to any arbitrary differentiable algorithm

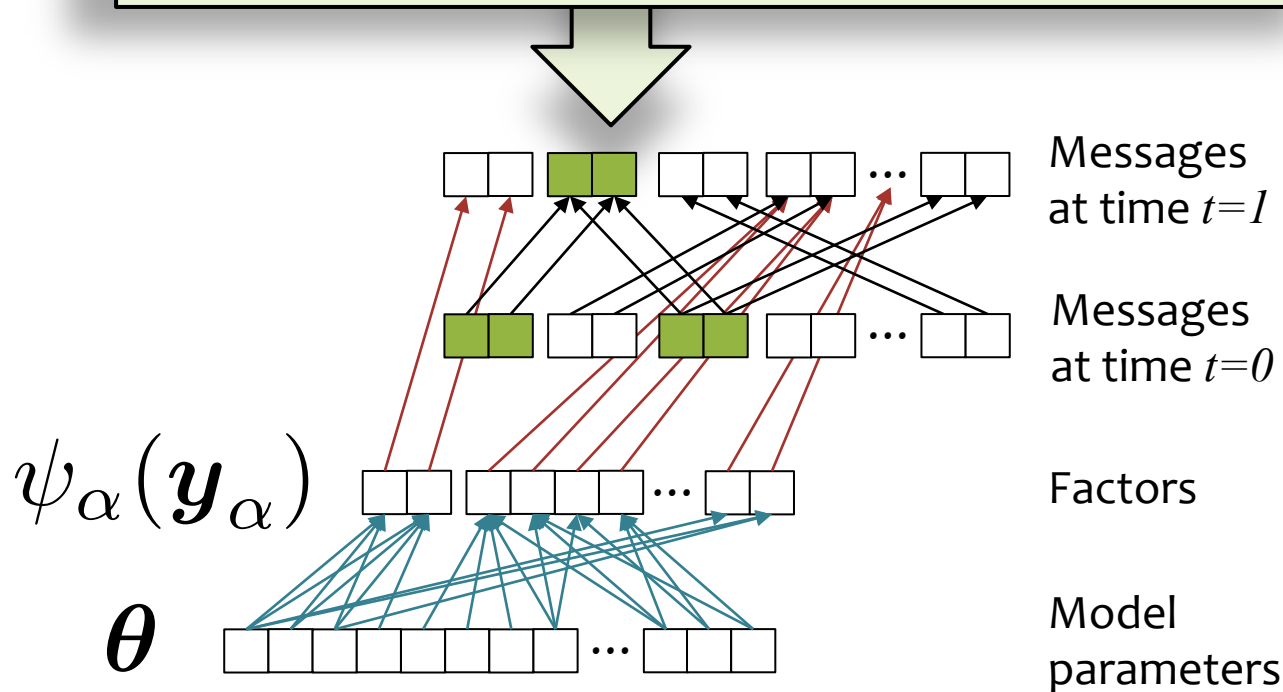
**That's the key (old) idea behind this talk.**

- Alternatively: could estimate a gradient by finite-difference approximations – but algorithmic differentiation is much more efficient!



# Feed-forward Topology of Inference, Decoding and Loss

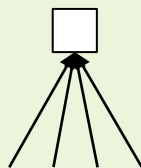




## Arrows in Neural Net:

Linear combination, then a sigmoid

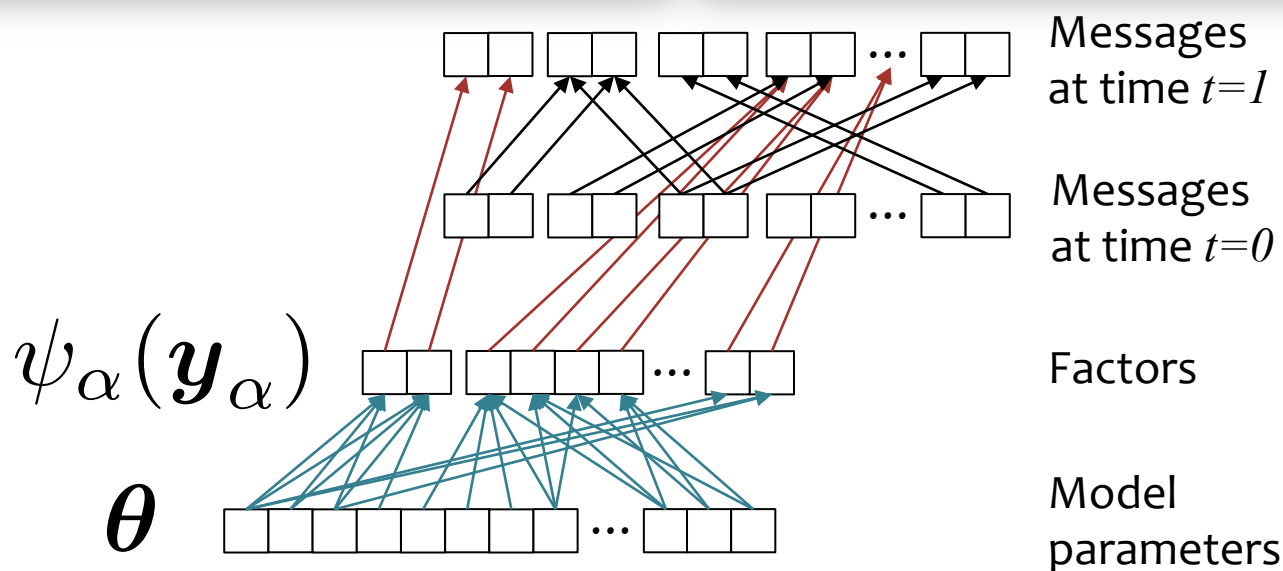
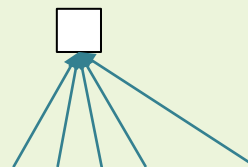
$$a_i = \sigma \left( \sum_j \theta_j b_j \right)$$



## Arrows in This Diagram:

A different semantics given by the algorithm

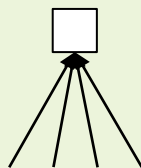
$$\psi_{\alpha}(\mathbf{y}_{\alpha}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\alpha}(\mathbf{y}_{\alpha}, \mathbf{x}))$$



## Arrows in Neural Net:

Linear combination, then a sigmoid

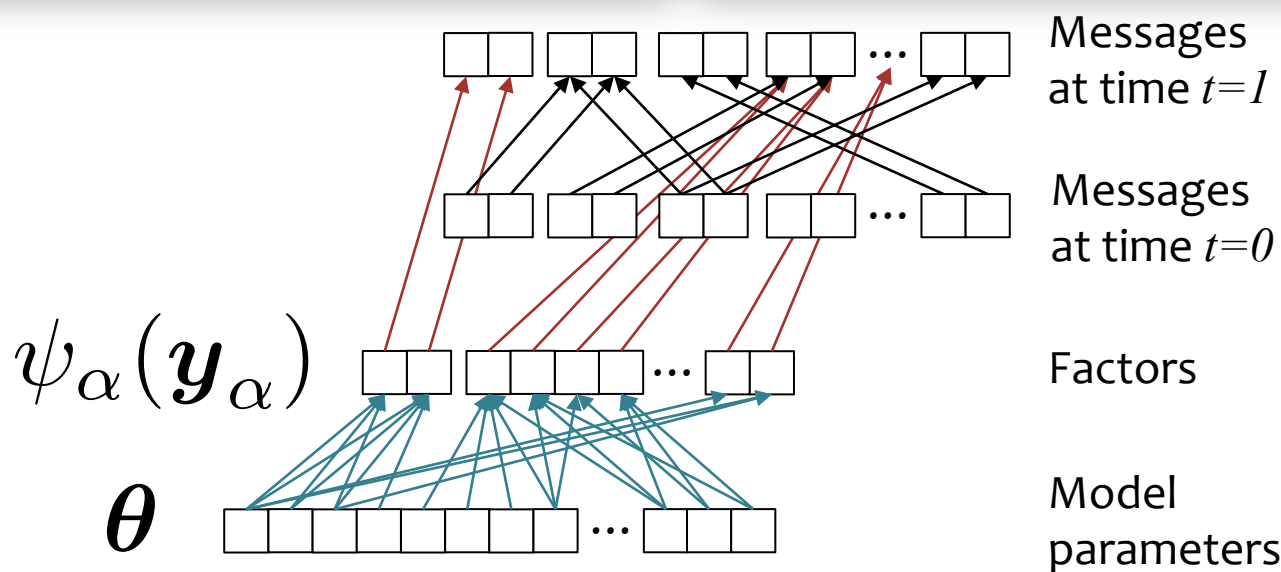
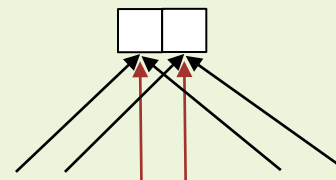
$$a_i = \sigma \left( \sum_j \theta_j b_j \right)$$



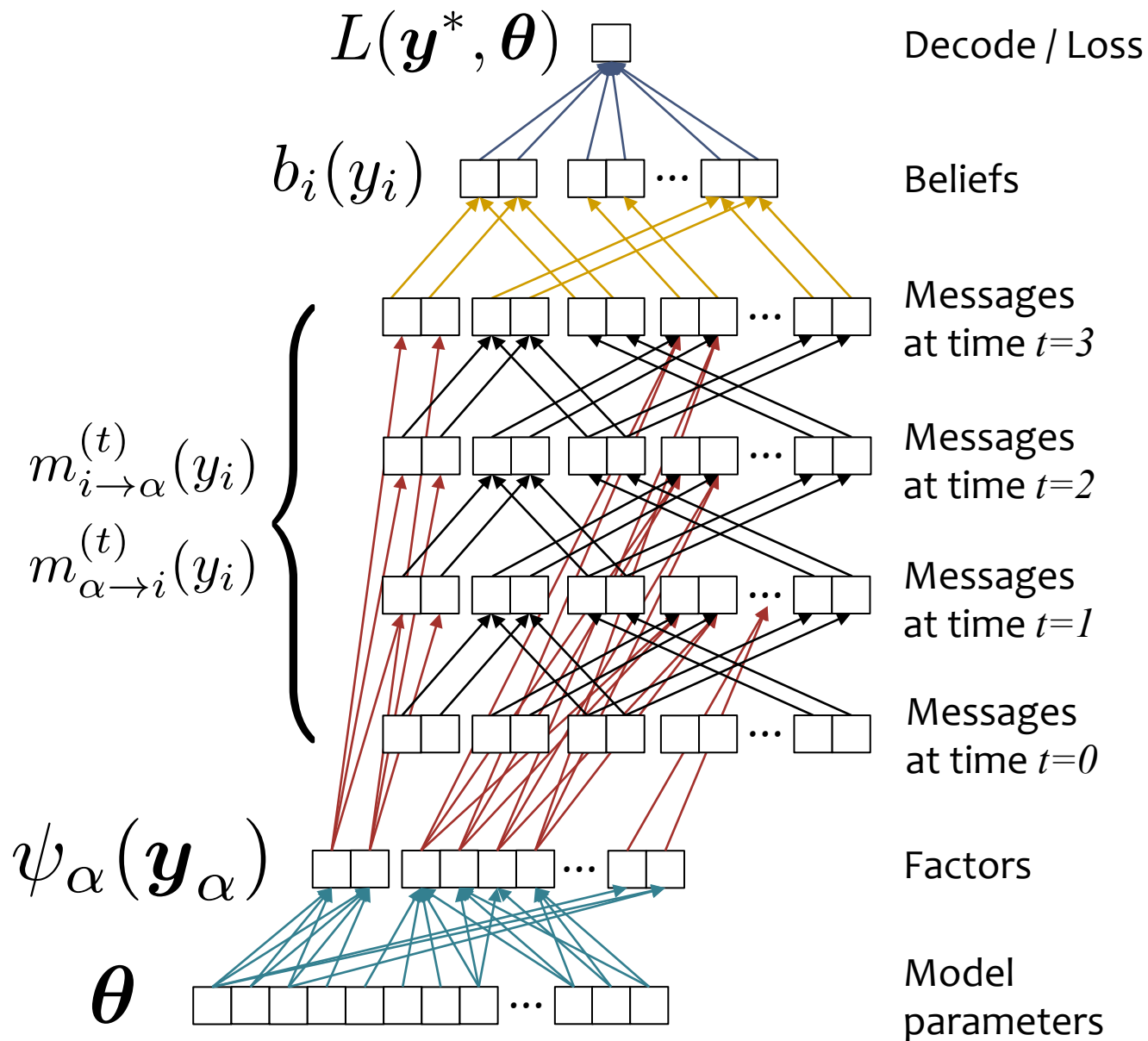
## Arrows in This Diagram:

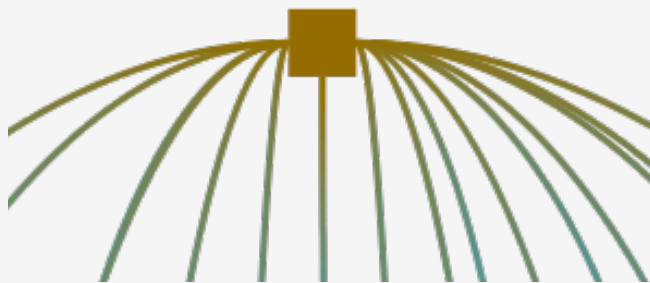
A different semantics given by the algorithm

$$m_{\alpha \rightarrow i}(y_i) = \frac{1}{\kappa_{\alpha \rightarrow i}} \sum_{\mathbf{y}_{\alpha} \sim y_i} \psi_{\alpha}(\mathbf{y}_{\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} m_{j \rightarrow \alpha}(y_i)$$



# Feed-forward Topology





Messages from PTree factor rely on a variant of **inside-outside**

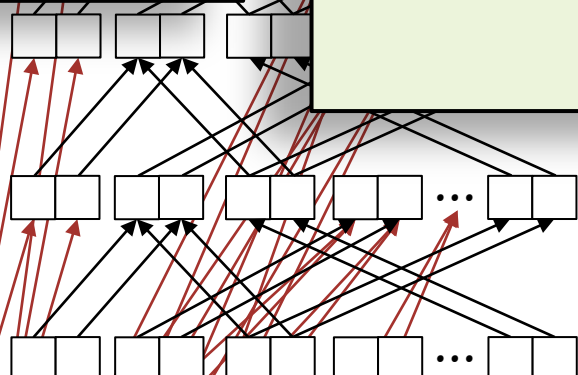
## Arrows in This Diagram:

A different semantics given by the algorithm

$$m_{\alpha \rightarrow i}(y_i) = \frac{1}{\kappa_{\alpha \rightarrow i}} \sum_{\mathbf{y}_{\alpha} \sim y_i} \psi_{\alpha}(\mathbf{y}_{\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} m_{j \rightarrow \alpha}(y_j)$$

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

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

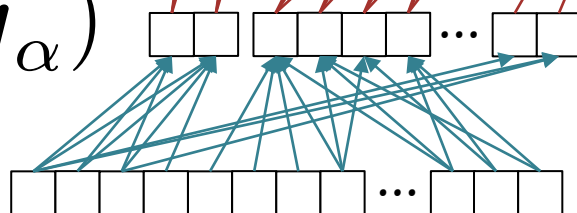


Messages at time  $t=l$

Messages at time  $t=0$

$\psi_{\alpha}(\mathbf{y}_{\alpha})$

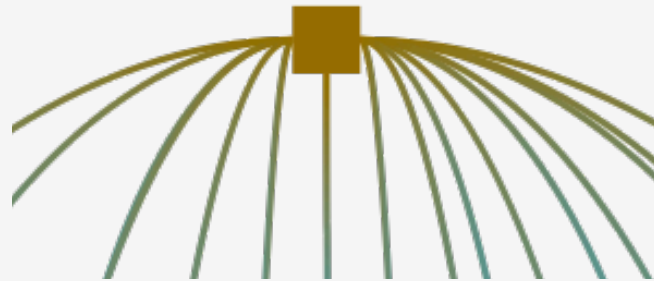
$\theta$



Factors

Model parameters

# Forward Topology



Messages from PTree  
factor rely on a variant  
of **inside-outside**

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

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

$$\psi_{\alpha}(\mathbf{y}_{\alpha})$$

$$\theta$$

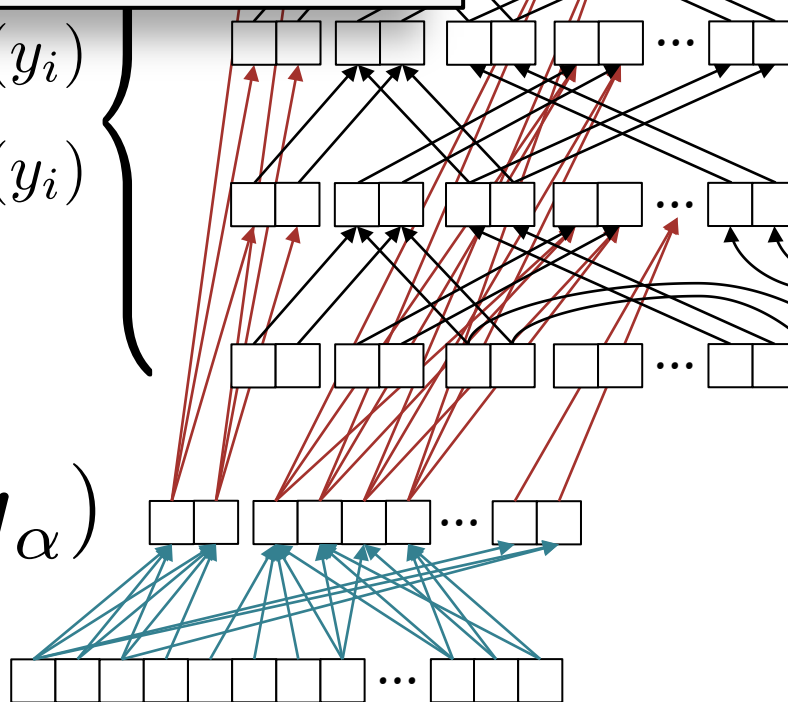
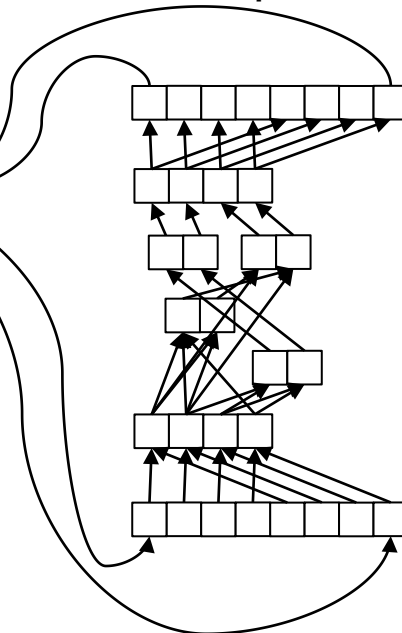


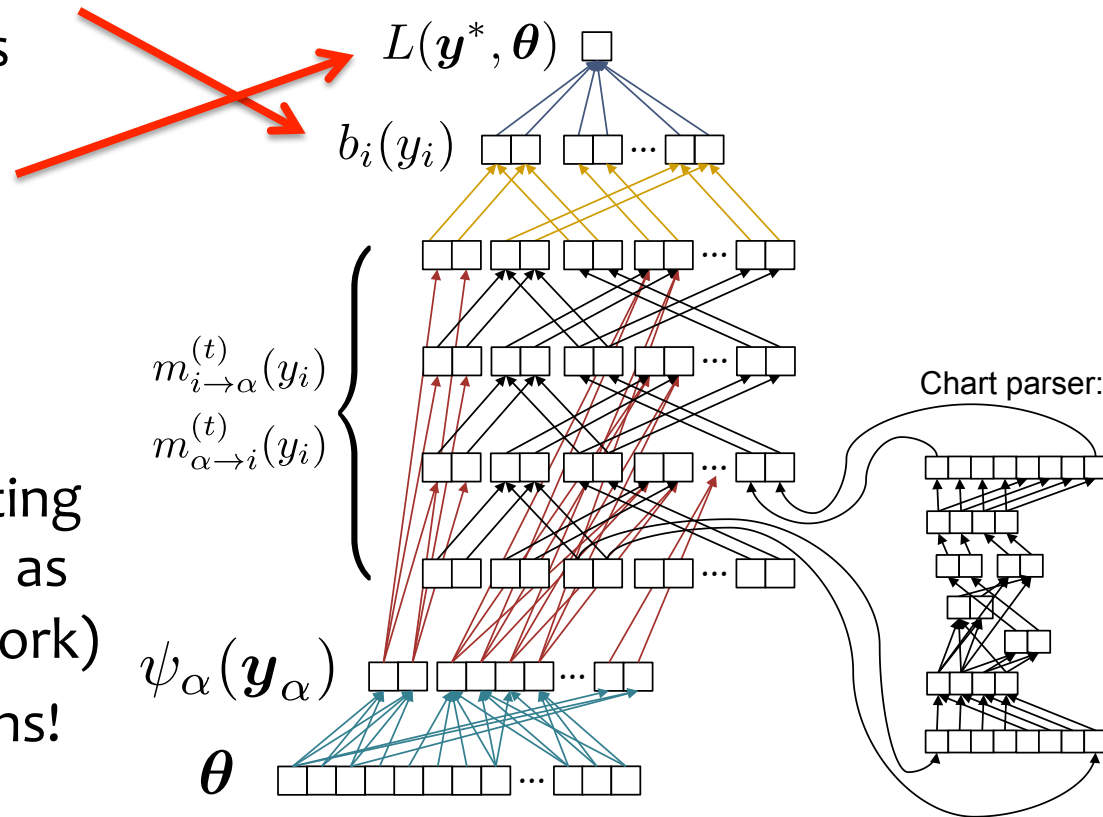
Chart parser:



# Approximation-aware Learning

**Key idea: Open up the black box!**

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)





# 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)

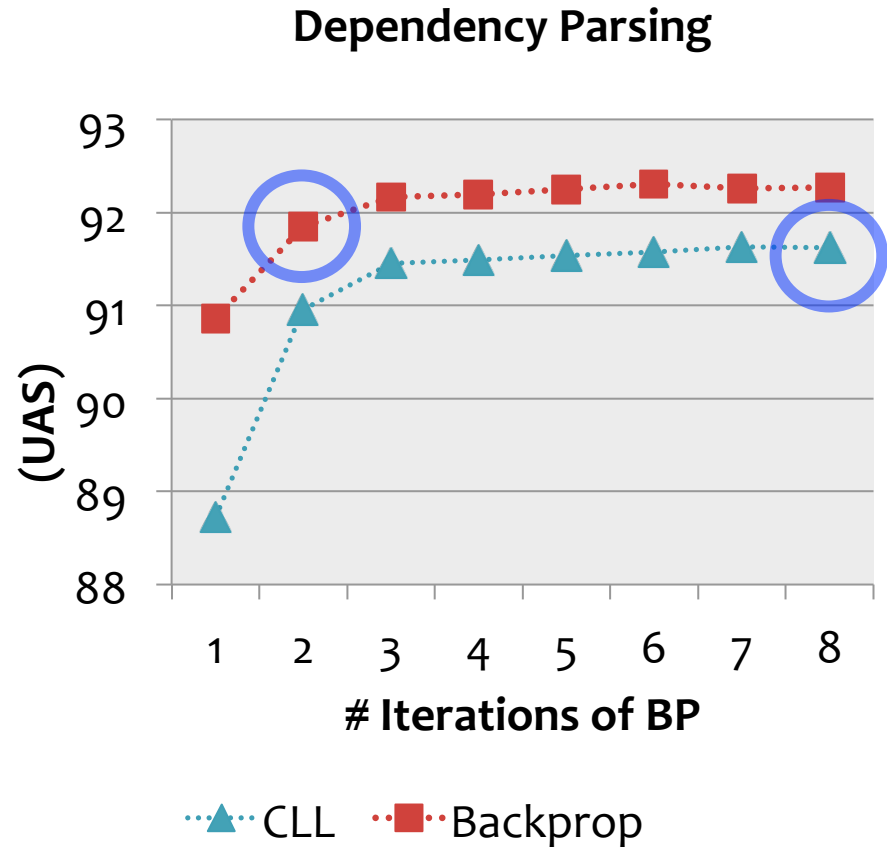
# Results

## 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

More accurate  
Unlabeled Attachment Score

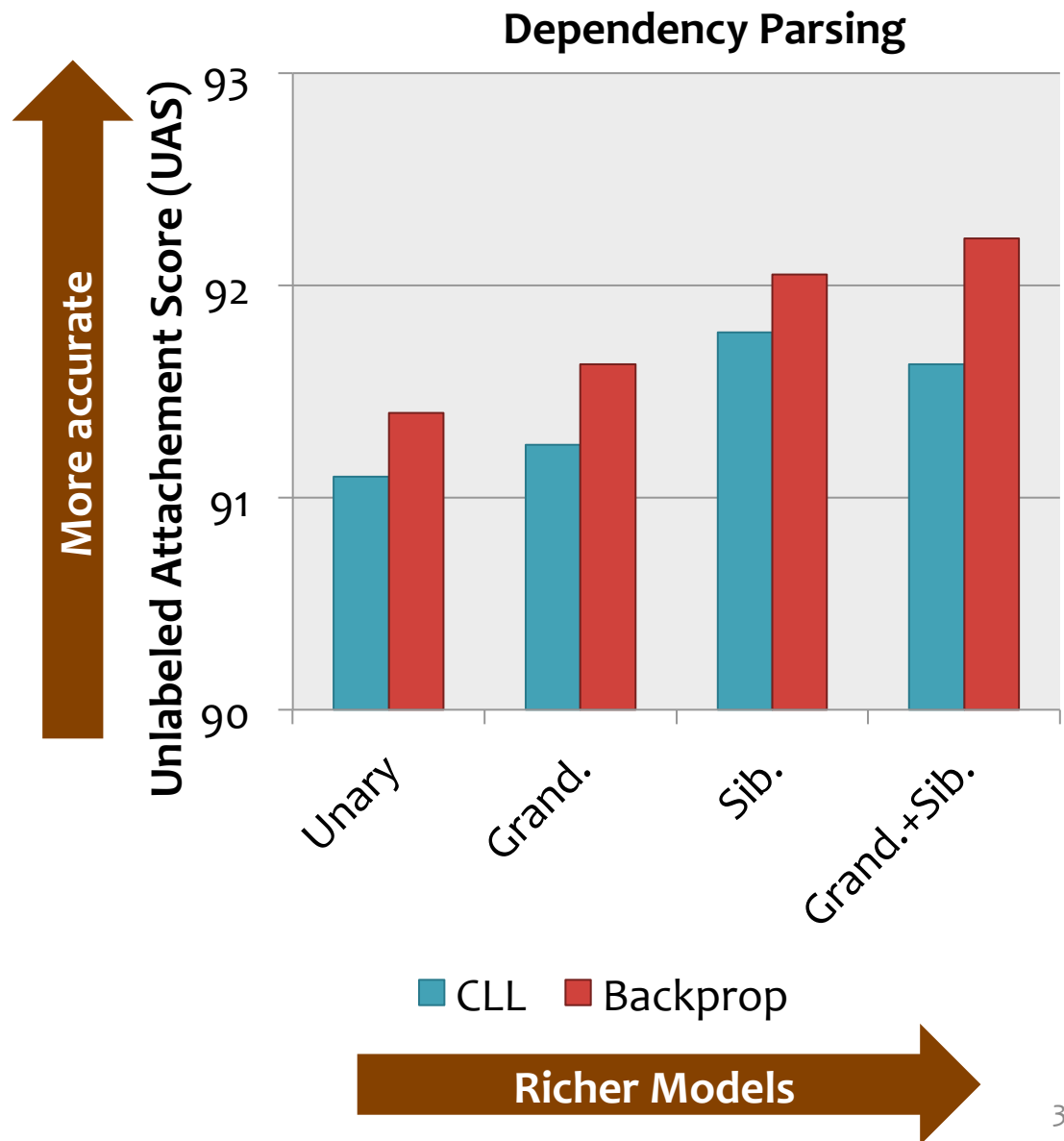


Faster

# 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

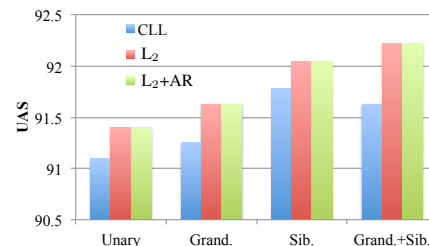
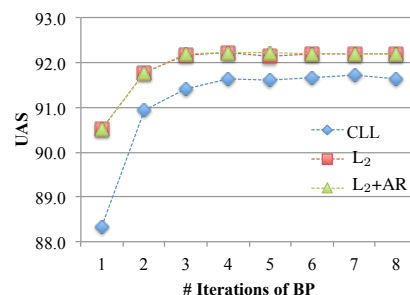


# See our TACL paper for...

1) Results on 19 languages from CoNLL 2006 / 2007

LANGUAGE	1ST-ORDER		2ND-ORDER (WITH GIVEN NUM. BP ITERATIONS)							
	CLL	L <sub>2</sub> - CL	CLL	1	2	4	8	CLL	L <sub>2</sub> - CL	L <sub>2</sub> - CL
AR	77.63	-0.26	73.39	+2.21	+77.05	-0.17	+77.20	+0.02	+77.16	-0.07
BG	90.38	-0.76	89.18	-0.45	+90.44	+0.04	+90.73	+0.25	+90.63	-0.19
CA	90.47	+0.30	88.90	+0.17	+90.79	+0.38	+91.21	+0.78	+91.49	+0.66
CS	84.69	-0.07	79.92	+3.78	+82.08	+2.27	+83.02	+2.94	+81.60	+4.42
DA	87.15	-0.12	86.31	-1.07	+87.41	+0.03	+87.65	-0.11	+87.68	-0.10
DE	88.55	+0.81	88.06	0.00	+89.27	+0.46	+89.85	-0.05	+89.87	-0.07
EL	82.43	-0.54	80.02	+0.29	+81.97	+0.09	+82.49	-0.16	+82.66	-0.04
EN	88.31	+0.32	85.53	+1.44	+87.67	+1.82	+88.63	+1.14	+88.85	+0.96
ES	83.49	-0.09	79.08	-0.37	+80.73	+0.14	+81.75	-0.66	+81.52	+0.02
EU	73.69	+0.11	71.45	+0.85	+74.16	+0.24	+74.92	-0.32	+74.94	-0.38
HU	78.79	-0.52	76.46	+1.24	+79.10	+0.03	+79.07	+0.60	+79.28	+0.31
IT	84.75	+0.32	84.14	+0.04	+85.15	+0.01	+85.66	-0.51	+85.81	-0.59
JA	93.54	+0.19	93.01	+0.44	+93.71	-0.10	+93.75	-0.26	+93.47	+0.07
NL	76.96	+0.53	74.23	+2.08	+77.12	+0.53	+78.03	-0.27	+77.83	-0.09
PT	86.31	+0.38	85.68	-0.01	+87.01	+0.29	+87.34	+0.08	+87.30	+0.17
SL	79.89	+0.30	78.42	+1.50	+79.56	+1.02	+80.91	+0.03	+80.80	+0.34
SV	87.22	+0.60	86.14	-0.02	+87.68	+0.74	+88.01	+0.41	+87.87	+0.37
TR	78.53	-0.30	77.43	-0.64	+78.51	-1.04	+78.80	-1.06	+78.91	-1.13
ZH	84.93	-0.39	82.62	+1.43	+84.27	+0.95	+84.79	+0.68	+84.77	+1.14
Avg.	83.98	+0.04	82.10	+0.68	+83.88	+0.41	+84.41	+0.19	+84.34	+0.31

2) Results with alternate training objectives



3) Empirical comparison of exact and approximate inference

TRAIN	INFERENCE	DEV UAS	TEST UAS
CLL	Exact	91.99	91.62
CLL	BP 4 iters	91.37	91.25
L <sub>2</sub>	Exact	91.91	91.66
L <sub>2</sub>	BP 4 iters	91.83	91.63

# 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>