Structured Prediction Problems in Natural Language Processing

Michael Collins

MIT

Acknowledgments: Xavier Carreras, Amir Globerson, Terry Koo

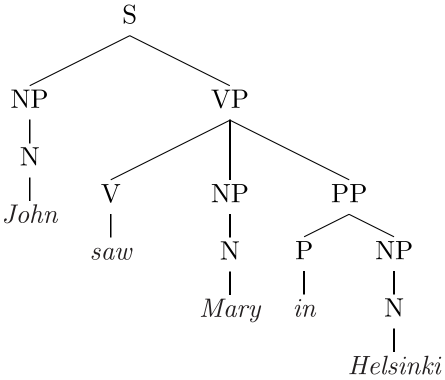
Structured Prediction Problems

- Supervised learning: learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ from examples $\{(x_i, y_i)\}_{i=1}^n$
- Binary classification: $\mathcal{Y} = \{-1, +1\}$
- Multi-class classification: $\mathcal{Y} = \{1, 2, \ldots, k\}$
- Structured prediction:
  - $\mathcal{Y}$ is a very large set
  - Each member of $\mathcal{Y}$ has internal structure

Examples of Structured Prediction Problems

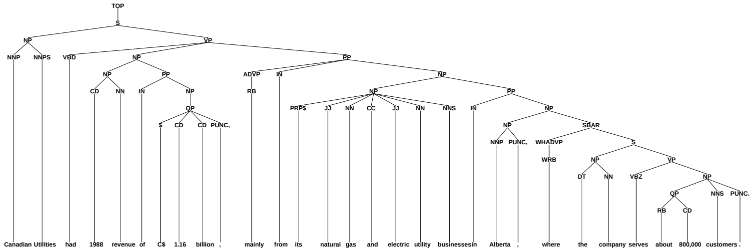
- Speech recognition: mapping acoustic inputs to sentences
- Computer vision: e.g., finding a segmentation of an image
- Computational biology: mapping a DNA sequence to an underlying segmentation
- Natural language parsing: mapping strings to parse trees
- Machine translation: mapping strings in one language to strings in another language

Syntactic Structures



- Natural language parsing: learning to map sentences to underlying parse trees

Syntactic Structures



Canadian Utilities had 1988 revenue of C$ 1.16 billion, mainly from its natural gas and electric utility businesses in Alberta, where the company serves about 800,000 customers.

Structured Prediction Models for Parsing

- Conditional random fields (CRFs), and other discriminative models, are a powerful alternative to HMMs
  - A key strength: flexible representations
- **Can we generalize CRF-style models to parsing?**
  Challenges:
  1. Choice of model structure/parameterization
  2. Inference
  3. Parameter estimation

Structured Prediction Models for Parsing

- Conditional random fields (CRFs), and other discriminative models, are a powerful alternative to HMMs
  - A key strength: flexible representations
- **Can we generalize CRF-style models to parsing?**
  Challenges:
  1. Choice of model structure/parameterization
  2. Inference
  3. Parameter estimation

Structured Prediction Models for Parsing

- Conditional random fields (CRFs), and other discriminative models, are a powerful alternative to HMMs
  - A key strength: flexible representations

- **Can we generalize CRF-style models to parsing?**
  Challenges:
  1. Choice of model structure/parameterization
  2. Inference
  3. Parameter estimation

Structured Prediction Models for Parsing

- Conditional random fields (CRFs), and other discriminative models, are a powerful alternative to HMMs
  - A key strength: flexible representations
- **Can we generalize CRF-style models to parsing?** Challenges:
  1. Choice of model structure/parameterization
  2. Inference
  3. Parameter estimation

Overview

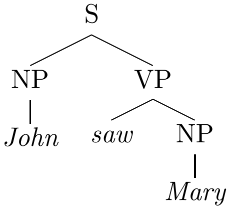
- Background
- Models
- An Application: Machine translation
- Inference
- Optimization/Learning

Context-Free Grammars (CFGs) for Language

- Basic elements in CFGs are rules. A simple CFG:

`S -> NP VP`	*A sentence is formed by a noun-phrase followed by a verb-phrase*
`NP -> John`	*A noun-phrase can be the string "John"*
`NP -> Mary`	*A noun-phrase can be the string "Mary"*
`VP -> slept`	*A verb-phrase can be the string "slept"*
`VP -> saw NP`	*A VP can be "saw" followed by a noun-phrase*

- A parse tree:



Motivation for Parsing: Grammatical Relations

- A sentence:

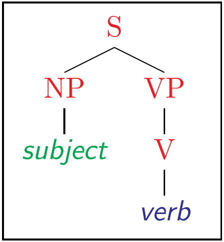
  John saw Mary in Helsinki

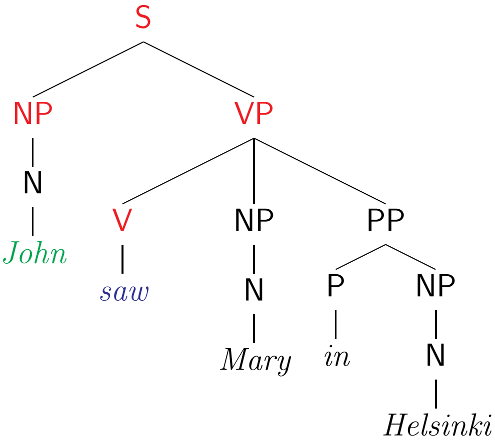
  Grammatical relations within the sentence:

  John is the subject of *saw*
  Mary is the object of *saw*
  in Helsinki is a (locative) modifier to *saw*

- Useful in many NLP applications: machine translation, information extraction, etc.

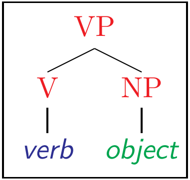
Syntactic Structures and Grammatical Relations

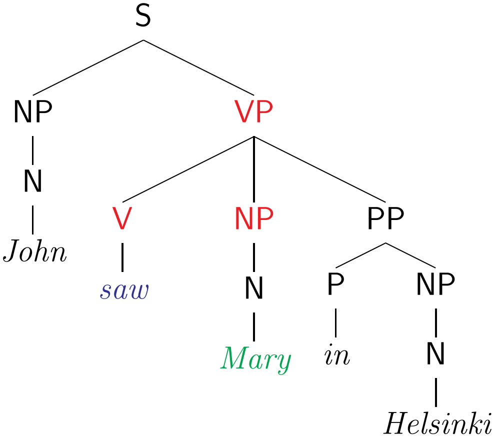




⇒ *John* is the subject of *saw*

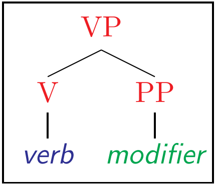
Syntactic Structures and Grammatical Relations

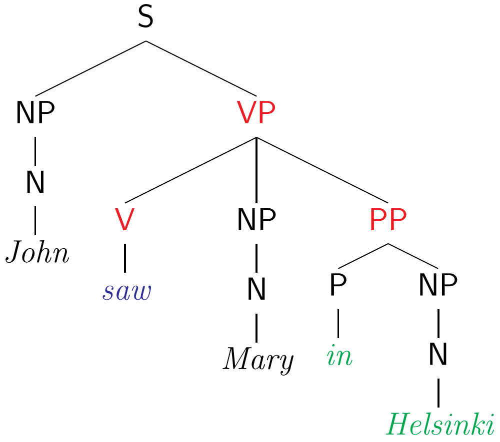




$\Rightarrow$ *Mary* is the object of *saw*

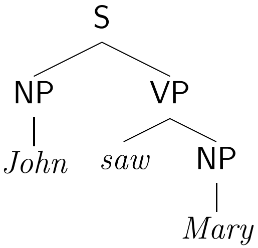
Syntactic Structures and Grammatical Relations





⇒ *In Helsinki* is a prepositional-phrase (PP) modifier to *saw*

Probabilistic Context-Free Grammars (PCFGs)



$$P(Tree) = P(\texttt{S -> NP VP} \mid \texttt{S}) \times P(\texttt{NP -> John} \mid \texttt{NP}) \times$$
$$P(\texttt{VP -> saw NP} \mid \texttt{VP}) \times P(\texttt{NP -> Mary} \mid \texttt{NP})$$

Overview

- Background
- Models
- An Application: Machine translation
- Inference
- Optimization/Learning

Models: Key Points

1. Probabilistic/weighted grammars in machine learning
2. Tree adjoining grammars as an alternative to context-free grammars
3. CRF-style models applied to learning weighted grammars

Conditional Random Fields

(Lafferty, McCallum, and Pereira, 2001)

- Goal: learn a function from $\mathbf{x}$ to $\mathbf{y}$ where
  - $\mathbf{x} = x_1 x_2 \ldots x_n$ is an input sequence (e.g., a sequence of words)
  - $\mathbf{y} = y_1 y_2 \ldots y_n$ is an output sequence (e.g., a sequence of underlying states)

The Building Blocks for CRFs: Feature Vectors

$\mathbf{y} =$	N	V	D	N	P	N
$\mathbf{x} =$	Mary	eats	the	cake	with	almonds

- $\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$ is a *feature vector* representing the transition $y_{i-1} \rightarrow y_i$ at position i in the sentence
- e.g., $i = 4$, $y_{i-1} =$ D, $y_i =$ N

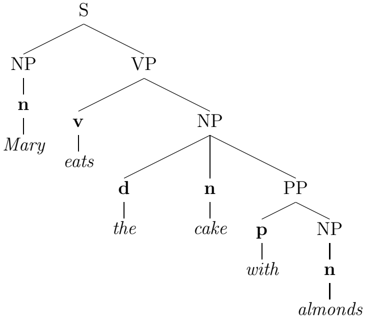
Conditional Random Fields

- Model form:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

  - $\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$ is a feature vector, $\mathbf{w}$ is a parameter vector
  - $\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$ is a measure of the plausibility/probability of state y_{i-1} being followed by state y_i at position i in the sentence $\mathbf{x}$
  - Can find $\mathbf{y}^*$ using the Viterbi algorithm

Generalizing CRFs to Parsing

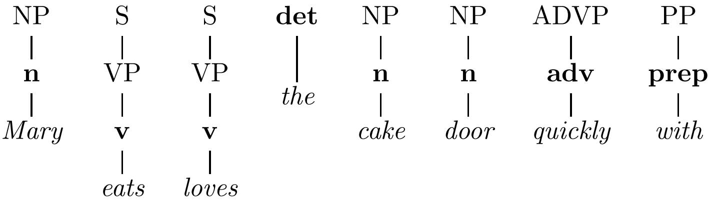


- One option: methods based on context-free grammars
- An alternative: *Tree Adjoining Grammars* (Joshi, 1985)

A TAG-Style Formalism

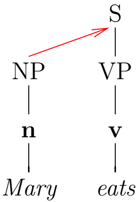
(Carreras, C, and Koo, 2008)

- In Tree Adjoining Grammar (TAG, Joshi, 1985) the grammar is defined by *a set of elementary trees*.
- Our elementary trees are Spines (See also Shen and Joshi, 2005):



A Combination Operation: *Sister Adjunction*

Sister adjunctions are used to combine spines to form trees.

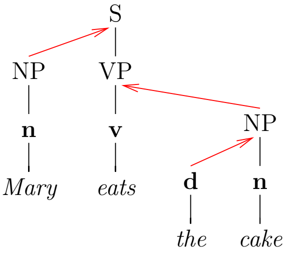


An adjunction operation attaches:

- A modifier spine
- To some position of a head spine

A Combination Operation: *Sister Adjunction*

Sister adjunctions are used to combine spines to form trees.

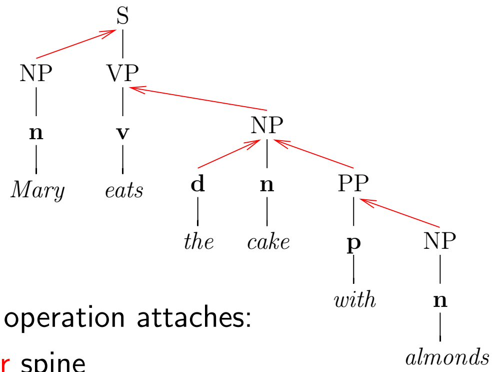


An adjunction operation attaches:

- A modifier spine
- To some position of a head spine

A Combination Operation: *Sister Adjunction*

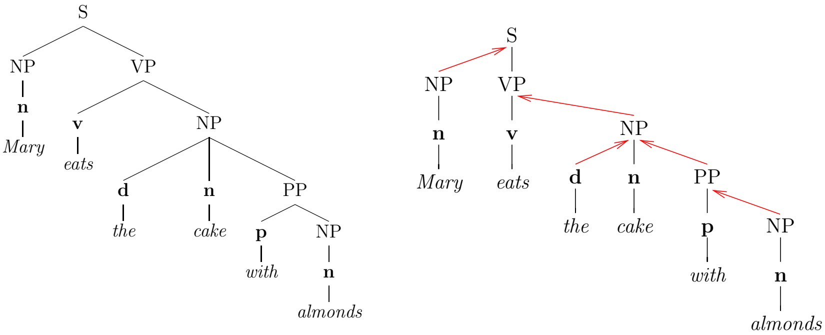
Sister adjunctions are used to combine spines to form trees.



An adjunction operation attaches:

- A modifier spine
- To some position of a head spine

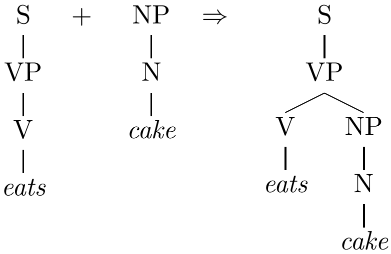
The Decomposition into Spines and Adjunctions



Advantages of TAG

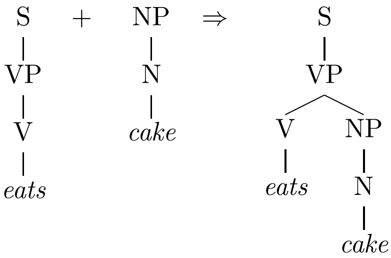
- Lexical entries naturally capture constraints associated with lexical items

- Probabilities/costs can be associated with combination operations:



The Contrast with Context-free Grammars

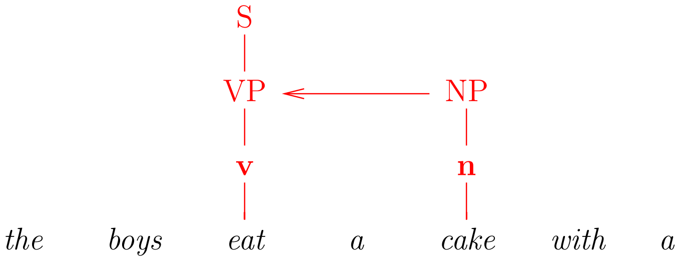
- In TAG, probabilities/costs are associated with combination operations:



- In CFGs, probabilities/costs are associated with rules:

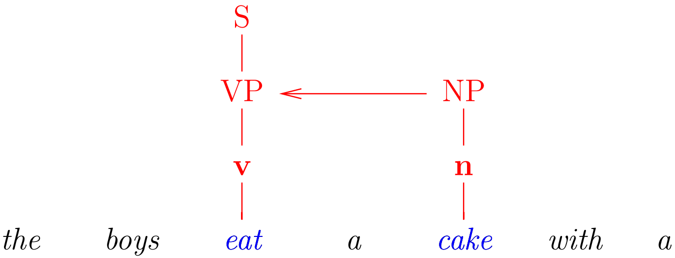
```
 S -> NP VP
VP -> saw NP
    ...
```

Features on Adjunctions



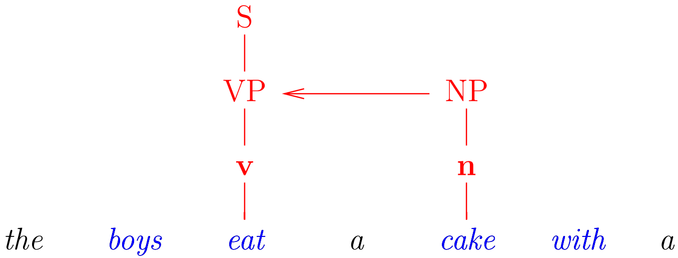
- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

Features on Adjunctions



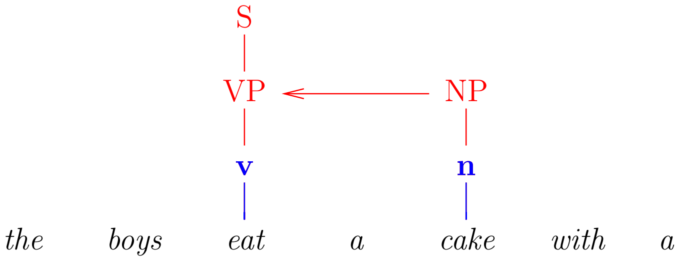
- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

Features on Adjunctions



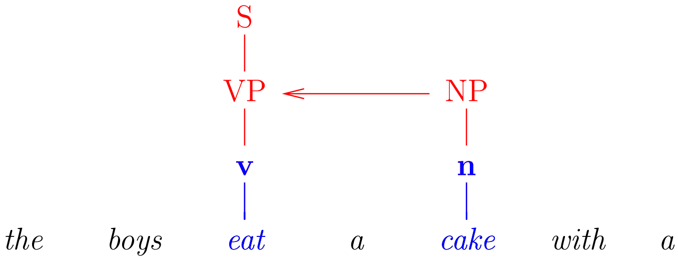
- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

Features on Adjunctions



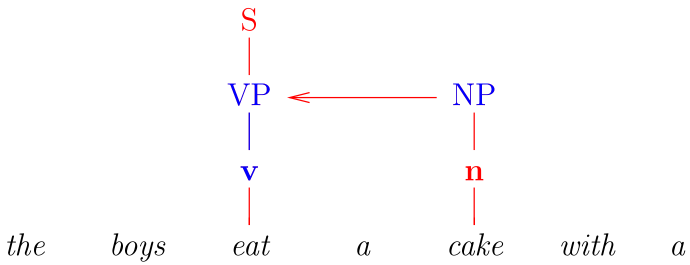
- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

Features on Adjunctions



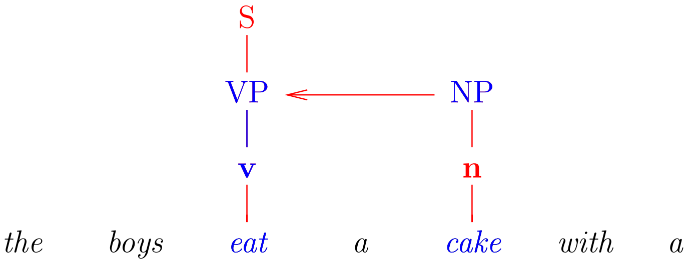
- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

Features on Adjunctions



- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

Features on Adjunctions



- Feature vectors $\mathbf{f}(\mathbf{x}, h, m, \sigma_h, \sigma_m, \text{POS})$ where
  - $\mathbf{x}$ is the sentence
  - $h = 3$ (index of head word), $m = 5$ (index of modifier word)
  - σ_h and σ_m are the head and modifier spines
  - POS is the position being adjoined into (e.g., VP)

A TAG-Based Model

- Goal: map an input sentence $\mathbf{x}$ to a parse tree $\mathbf{y}$
- Model form:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{r \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

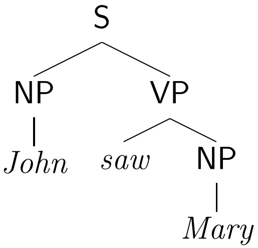
  where each r is a tuple $\langle h, m, \sigma_h, \sigma_m, \text{POS} \rangle$ representing a combination of two spines in $\mathbf{y}$
- Compare to the model form for CRFs:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

Experiments

- Inference: coarse-to-fine dynamic programming
- Training: averaged perceptron algorithm
- Data: Penn Wall Street Journal treebank
- Evaluation metric: precision, recall, and F1 score in recovering constituents in parse trees
- Comparison to PCFG-based models

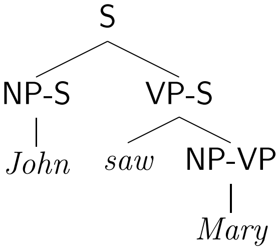
Probabilistic Context-Free Grammars (PCFGs)



$$P(Tree) = P(\texttt{S -> NP VP} \mid \texttt{S}) \times P(\texttt{NP -> John} \mid \texttt{NP}) \times$$
$$P(\texttt{VP -> saw NP} \mid \texttt{VP}) \times P(\texttt{NP -> Mary} \mid \texttt{NP})$$

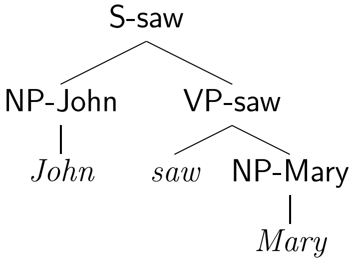
PCFGs with Parent Annotations

(Johnson, 1999)



$$
\begin{aligned}
P(Tree) \quad = \quad & P(\texttt{S -> NP-S VP-S} \mid \texttt{S}) \times \\
& P(\texttt{NP-S -> John} \mid \texttt{NP-S}) \times \\
& \ldots
\end{aligned}
$$

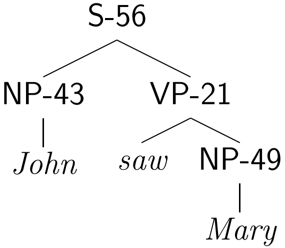
Lexicalized PCFGs



$$
\begin{aligned}
P(Tree) \quad = \quad & P(\texttt{S-saw -> NP-John VP-saw} \mid \texttt{S-saw}) \times \\
& P(\texttt{NP-John -> John} \mid \texttt{NP-John}) \times \\
& \ldots
\end{aligned}
$$

PCFGs with Latent Variables

(e.g., Petrov and Klein, 2007)



- Each non-terminals (e.g., `S`) is split into a number of new non-terminals (e.g., `S-1`, `S-2`, ..., `S-128`)
- Latent annotations learned using EM

A Comparison to PCFGs

PARSER	F_1 Error
Parent annotations (Johnson, 1999)	20.4%
Lexicalized PCFGs (Collins, 1999)	11.8%
Latent variables, EM (Petrov & Klein 2007)	9.9%
The TAG-based model	**8.9%**

A Comparison to PCFGs

PARSER	F_1 Error
Parent annotations (Johnson, 1999)	20.4%
Lexicalized PCFGs (Collins, 1999)	11.8%
Latent variables, EM (Petrov & Klein 2007)	9.9%
The TAG-based model	**8.9%**

A Comparison to PCFGs

PARSER	F_1 Error
Parent annotations (Johnson, 1999)	20.4%
Lexicalized PCFGs (Collins, 1999)	11.8%
Latent variables, EM (Petrov & Klein 2007)	9.9%
The TAG-based model	**8.9%**

Overview

- Background
- Models
- An Application: Machine translation
- Inference
- Optimization/Learning

An Application: Translation

In wenigen Tagen finden Parlamentswahlen in Slowenien statt

$\Downarrow$

In a few days, elections will take place in Slovenia

- Statistical machine translation: systems which learn from a corpus of example translations
- Possible approaches:
  - Learn a direct mapping from German to English
  - Learn a mapping where syntactic structures are used as latent/hidden structure

Phrase-based Translation (Och et al., 1999)

- In phrase-based systems, a major component is a lexicon of *phrase pairs*, learned from a corpus of example translations. E.g., (In wenigen $\Leftrightarrow$ In a few), (Tagen $\Leftrightarrow$ days)
- Translation involves:
  1. Segmenting the German into phrases, and choosing a translation for each phrase
  2. Choosing an ordering of the resulting English phrases

In wenigen Tagen finden Parlamentswahlen in Slowenien statt

Phrase-based Translation (Och et al., 1999)

- In phrase-based systems, a major component is a lexicon of *phrase pairs*, learned from a corpus of example translations. E.g., (In wenigen ⇔ In a few), (Tagen ⇔ days)

- Translation involves:
  1. Segmenting the German into phrases, and choosing a translation for each phrase
  2. Choosing an ordering of the resulting English phrases

[In wenigen]	[Tagen]	[finden]	[Parlamentswahlen]	[in Slowenien]	[statt]
[In a few]	[days]	[take]	[elections]	[in Slovenia]	[place]

Phrase-based Translation (Och et al., 1999)

- In phrase-based systems, a major component is a lexicon of *phrase pairs*, learned from a corpus of example translations. E.g., (In wenigen $\Leftrightarrow$ In a few), (Tagen $\Leftrightarrow$ days)

- Translation involves:
  1. Segmenting the German into phrases, and choosing a translation for each phrase
  2. Choosing an ordering of the resulting English phrases

[In wenigen]	[Tagen]	[finden]	[Parlamentswahlen]	[in Slowenien]	[statt]
[In a few]	[days]	[take]	[elections]	[in Slovenia]	[place]

$$\Downarrow$$

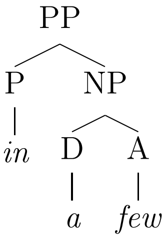
[In a few] [days] [elections] [take] [place] [in Slovenia]

Translation as Parsing

e.g., Marcu et al., (2006), Shen et al., (2008)

- Phrase entries are augmented to include target-language syntax, e.g.,

In wenigen ⇔

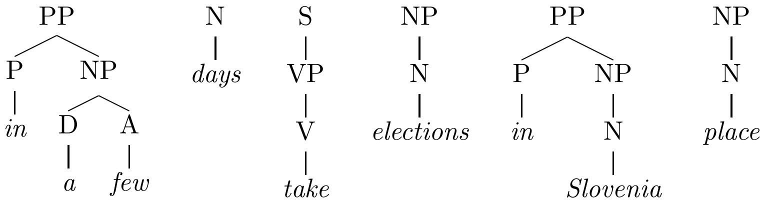


Translation as Parsing

Step 1 Choose a segmentation of the German input, and choosing a phrase entry for each German phrase

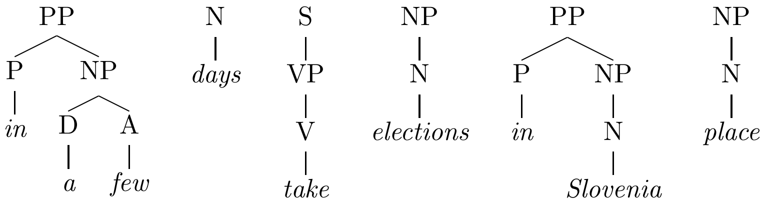
[In wenigen] [Tagen] [finden] [Parlamentswahlen] [in Slowenien] [statt]

$\Downarrow$



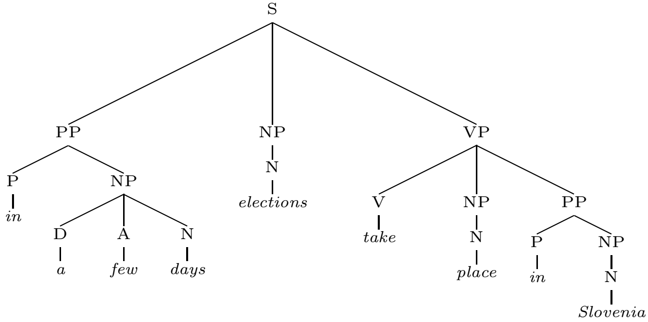
Translation as Parsing

Step 2 Assemble the English parse tree fragments to form a complete tree (some reordering allowed)



Translation as Parsing

Step 2 Assemble the English parse tree fragments to form a complete tree (some reordering allowed)



Properties of Translation as Parsing

- The translation process can be implemented using modified parsing algorithms
- Potential Advantages:
  - Building an English parse tree gives a direct model of grammaticality/fluency
  - Reordering operations can be based on the parse-tree structure

Overview

- Background
- Models
- An Application: Machine translation
- Inference
- Optimization/Learning

Inference

- Goal: map an input sentence $\mathbf{x}$ to a parse tree $\mathbf{y}$
- Model form:

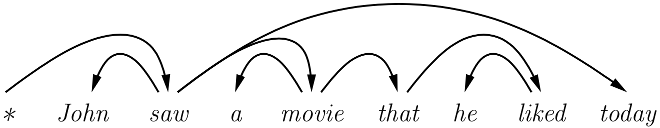
$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{r \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

  where each r is a tuple $\langle h, m, \sigma_h, \sigma_m, \text{POS} \rangle$ representing a combination of two spines in $\mathbf{y}$
- How to compute $\mathbf{y}^*$?

Inference: Key Points

- Dynamic programming algorithms can be applied to the TAG grammars
- Exact inference is still very expensive
- A solution: *coarse-to-fine* dynamic programming (e.g., (Charniak, 1997; Charniak and Johnson, 2005))
  - Use a first-pass, simple, computationally-cheap model to restrict the search space of the full model

Dependency Structures



- Directed arcs represent *dependencies* between a *head word* and a *modifier word*.
- Dependency parsing models of McDonald et al. (2005, 2006):

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{r \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

where each r is a tuple $\langle h, m \rangle$ representing a dependency from modifier m to head h

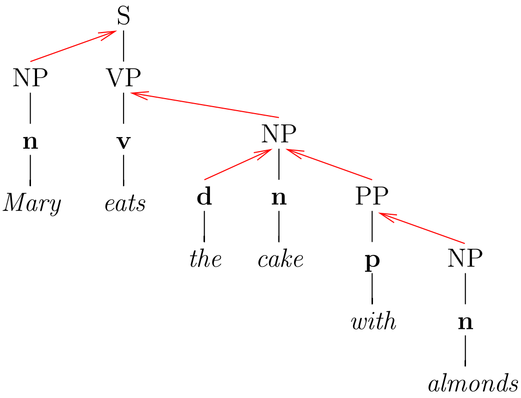
Efficient Parsing Algorithms (Eisner 1997, 2000)

- Dependency parsing models of McDonald et al. (2005, 2006):

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{r \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

  where each r is a tuple $\langle h, m \rangle$ representing a dependency from modifier m to head h
- Most probable/lowest cost dependency structure can be found in $O(n^3)$ time where n is the length of the sentence
- Similar to probabilistic context-free grammars, where parsing time is $O(n^3 G)$, with G being a grammar constant

TAG Parses and Dependency Structures



- A dependency structure augmented with *spines*, and *attachment positions*

Applying Eisner's Algorithms to our Formalism

- The TAG model form:

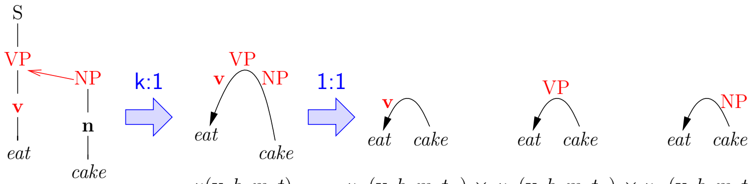
$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{r \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

  where each r is a tuple $\langle h, m, \sigma_h, \sigma_m, \text{POS} \rangle$ representing a combination of two spines in $\mathbf{y}$
- Most probable/lowest cost dependency structure can be found in $O(Gn^3)$ time where n is the length of the sentence, G is a grammar constant
- The constant G is polynomial in the number of possible spines for any word, and the maximum *height* of any spine

Coarse-to-fine Dynamic Programming

- Parsing time is at least $O(n^3G)$
(for some of our models it is $O(n^4G)$)
- Grammar constant G is prohibitive
(can easily have $G > 1000$ or $G > 10000$)
- Coarse-to-fine solution: build a simple dependency model with a much lower grammar constant G (e.g., $G \approx 60$), and *use this to prune the search space of the full model*

Three Simple Dependencies In Every Adjunction



$$\mu(\mathbf{x}, h, m, t) \quad = \mu_{\mathrm{H}}(\mathbf{x}, h, m, t_{\mathrm{H}}) \times \mu_{\mathrm{P}}(\mathbf{x}, h, m, t_{\mathrm{P}}) \times \mu_{\mathrm{M}}(\mathbf{x}, h, m, t_{\mathrm{M}})$$

- Coarse-to-fine approach: we only allow the full TAG model to consider dependencies that have high probability under a (simple) dependency model
- The simple model estimates dependency probabilities in $O(n^3G)$ time, where $G \approx 60$ is the number of non-terminals (i.e., *VP*, *NP*, *S*, etc.)

Effect of the Beam (Validation Data)

	1st stage		2nd stage		
α	active	cov.	orac.	speed	F_1 error
10^{-4}	0.07	97.7	97.0	5:15	8.9
10^{-5}	0.16	98.5	97.9	11:45	8.4
10^{-6}	0.34	99.0	98.5	21:50	8.0

We can discard 99.6% of the possible adjunctions
and retain 98.5% of the correct syntactic constituents

Overview

- Background
- Models
- An Application: Machine translation
- Inference
- Optimization/Learning

Conditional Random Fields

- Model form:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

  - $\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$ is a feature vector, $\mathbf{w}$ is a parameter vector
  - $\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$ is a measure of the plausibility/probability of state y_{i-1} being followed by state y_i at position i in the sentence $\mathbf{x}$
  - Can find $\mathbf{y}^*$ using the Viterbi algorithm
- **Next question: algorithms for training the parameter vector $\mathbf{w}$**

Efficiency is a Key Problem

- Parsing and other NLP problems are often large-scale, with > 1000 or > 10000 training examples. Discriminative approaches for structured problems typically require repeated inference over the training examples.
- "Online" algorithms (e.g., stochastic gradient descent, the perceptron) are much more efficient than batch gradient methods (e.g., conjugate gradient, L-BFGS).
- Two "online" algorithms I'll describe:
  - The (averaged) perceptron
  - Exponentiated-gradient algorithms for CRFs

Parameter Estimation: the Structured Perceptron

(C, 2002)

- Set $\mathbf{w} = \mathbf{0}$
- For $t = 1 \ldots T$
  - For each training example $(\mathbf{x}, \mathbf{y})$
    1. Compute $\mathbf{z} = \arg\max_{\mathbf{z}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, z_{i-1}, z_i)$
    2. If $\mathbf{z} \neq \mathbf{y}$

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i) - \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, z_{i-1}, z_i)$$

- Return $\mathbf{w}$

Parameter Estimation: Averaging

(Freund and Schapire, 1998)

- Set $\mathbf{w} = \mathbf{0}$, $\mathbf{w_a} = \mathbf{0}$
- For $t = 1 \ldots T$
  - For each training example $(\mathbf{x}, \mathbf{y})$
    1. Compute $\mathbf{z} = \arg\max_{\mathbf{z}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, z_{i-1}, z_i)$
    2. If $\mathbf{z} \neq \mathbf{y}$

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i) - \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, z_{i-1}, z_i)$$

    3. $\mathbf{w_a} = \mathbf{w_a} + \mathbf{w}$
- Return $\mathbf{w_a}/NT$, where N is the number of training examples

Properties of the Perceptron

- If the data is separable, it will converge to parameter values with 0 errors
- Number of errors before convergence is related to a definition of *margin*. Can also relate margin to generalization properties
- In practice:
  1. Averaging improves performance **a lot**
  2. Typically reaches a good solution after only a few (say 5) iterations over the training set
  3. Often performs nearly as well as CRFs, or max-margin Markov networks
  4. Returns relatively *sparse* solutions, as each update only involves two state sequences ($\mathbf{y}$ and $\mathbf{z}$), and T is small

Averaged Perceptron Convergence

Iteration	Accuracy
1	90.79
2	91.20
3	91.32
4	91.47
5	91.58
6	91.78
7	91.76
8	91.82
9	91.88
10	91.91
11	91.92
12	91.96
13	91.97

- Results on validation set for treebank parsing

Regularized Log-Likelihood for Training CRFs

- Define

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

- Define

$$P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \frac{\exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\}}{Z(\mathbf{x}; \mathbf{w})}$$

- Given training examples $\{\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\}_{k=1}^{N}$, minimize

$$L(\mathbf{w}) = -\sum_{k} \log P(\mathbf{y}^{(k)} \mid \mathbf{x}^{(k)}; \mathbf{w}) + \frac{1}{2}||\mathbf{w}||^2$$

The Dual

- Dual variables: $\alpha_{k,y}$ where k ranges over all training examples, and y ranges over all state sequences for the k'th training example
- Define $\mathbf{w}(\boldsymbol{\alpha}) = \sum_k \mathbf{f}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) - \sum_k \sum_y \alpha_{k,y} \mathbf{f}(\mathbf{x}^{(k)}, \mathbf{y})$
- Dual objective: minimize

$$Q(\boldsymbol{\alpha}) = \sum_k \sum_y \alpha_{k,y} \log \alpha_{k,y} + \frac{1}{2} ||\mathbf{w}(\boldsymbol{\alpha})||^2$$

  under the constraints $0 \leq \alpha_{k,y} \leq 1, \;\; \sum_y \alpha_{k,y} = 1$

- Duality: $\mathbf{w}^* = \mathbf{w}(\boldsymbol{\alpha}^*)$

Dual Coordinate Descent

- Basic idea: pick one training example at a time, and update the dual variables on that one training example
- Has "online" flavour, in that the algorithm updates parameters after single training examples
- A nice property: can easily measure impact on the dual objective of any updates, and thereby choose learning rate

An Exponentiated Gradient (EG) Algorithm

(C, Globerson, Koo, Carreras, Bartlett, 2008)

- Dual objective: $Q(\boldsymbol{\alpha}) = \sum_k \sum_y \alpha_{k,y} \log \alpha_{k,y} + \frac{1}{2}||\mathbf{w}(\boldsymbol{\alpha})||^2$
- Initialization: choose initial $\alpha_{k,y}$ values (must be non-zero)
- Choose learning rate $\eta > 0$
- For $t = 1 \ldots T$

  1. Choose a training example k uniformly at random
  2. Update dual variables on k'th example:

$$\alpha_{k,y} \leftarrow \frac{\alpha_{k,y} \exp\{-\eta \nabla_{k,y}\}}{\sum_y \alpha_{k,y} \exp\{-\eta \nabla_{k,y}\}}$$

  where $\nabla_{k,y} = \frac{\partial}{\partial \alpha_{k,y}} Q(\boldsymbol{\alpha})$

Properties of the EG Algorithm

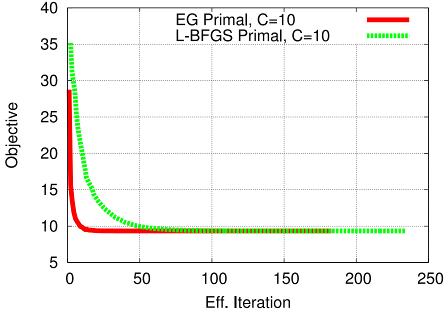
(C, Globerson, Koo, Carreras, and Bartlett, 2008)

- To get within ϵ of the optimal dual value, need $O(\log 1/\epsilon)$ updates. Online updates have faster convergence than batch methods, both in theory and practice.
- In structured problems, the algorithm can be implemented compactly/efficiently, using representation

$$\alpha_{k,y} = \frac{\exp\{\sum_{i=1}^{n} \theta_{k,i,y_{i-1},y_i}\}}{Z}$$

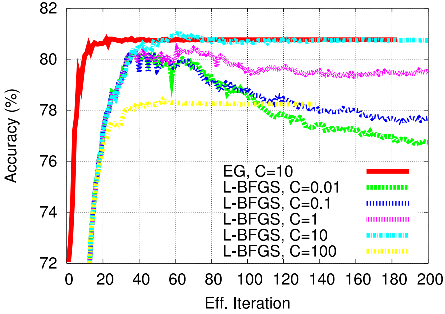
  where $\theta_{k,i,y_{i-1},y_i} \in \mathbb{R}$ are alternative dual variables.
- Main cost in the algorithm is then the *forward-backward* algorithm.

Comparison to L-BFGS on a Parsing Problem (Objective Value)



- Graph shows results for regularizer constant $C = 10$; results for other regularizer constants are similar

Comparison to L-BFGS on a Parsing Problem (Accuracy)

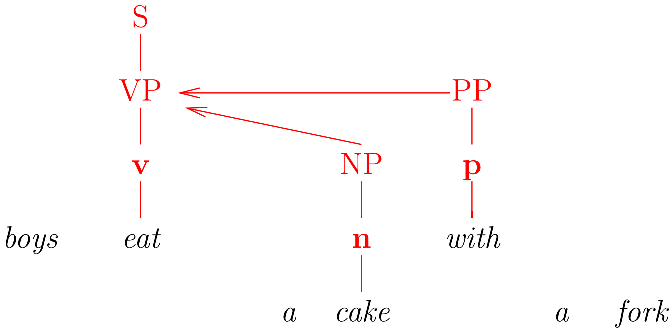


- Graph shows results for EG with regularizer constant $C = 10$, and L-BFGS for a range of regularizer constants

Conclusions

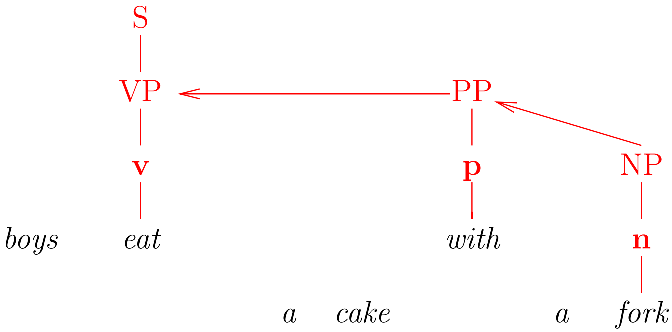
- Models:
  - Weighted grammars (like graphical models) offer useful generalizations of HMMs
  - Lexicalized grammars (e.g., TAGs, dependency grammars) lead to alternative parameterizations to PCFGs
- Inference: coarse-to-fine dynamic programming can be very effective
- Parameter estimation:
  - Averaged perceptron, EG algorithms are efficient, and are widely applicable to structured prediction problems

Second-Order Features with Siblings



- Can add sensitivity to sibling dependencies, and still retain $O(n^3)$ time algorithms

Second-Order Features with Grandchildren



- Can add sensitivity to grandparent dependencies, with $O(n^4)$ time algorithms

Regular Adjunctions

We also consider a regular adjunction operation.

It adds one level to the syntactic constituent it attaches to.