PCFGs are forever

from simple grammars to state-of-the-art phrase-based parsing

Joseph Le Roux,
with Antoine Rozenknop and Jennifer Foster

Université Paris 13

11/02/2014

Background

PCFGs

- Probabilistic Context-Free Grammars
- Well-known generative model to represent NL syntax
- Independence assumptions make them quite limited as such

Background

PCFGs

- Probabilistic Context-Free Grammars
- Well-known generative model to represent NL syntax
- Independence assumptions make them quite limited as such

PCFG-LAs

PCFGs with Latent Annotations [Matsuzaki et al., 2005]

- **the** model for phrase-based parsing nowadays
  - state of the art performance
  - available implementations (Berkeley, LORG, and Zhang's)

Objective

We will recall that there is no *PCFG-LA parsing* in practice

- *PCFG parsing*
- with a PCFG computed *online*
- with statistics gathered from a PCFG-LA and a sentence

Objective

We will recall that there is no *PCFG-LA parsing* in practice

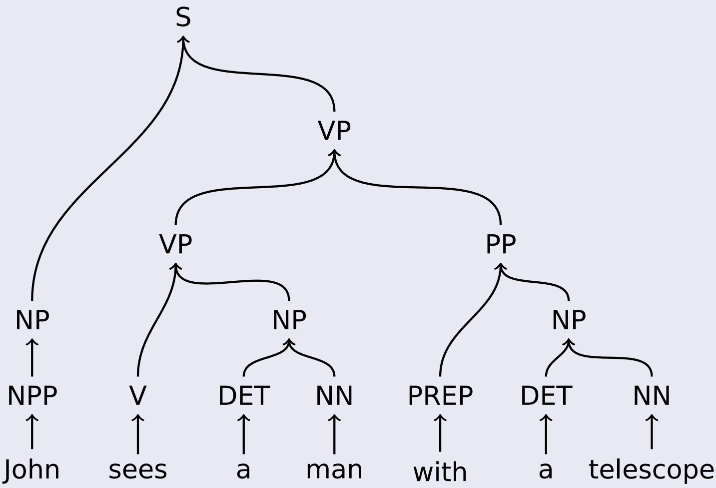
- *PCFG parsing*
- with a PCFG computed *online*
- with statistics gathered from a PCFG-LA and a sentence

We propose a novel algorithm to combine PCFGs

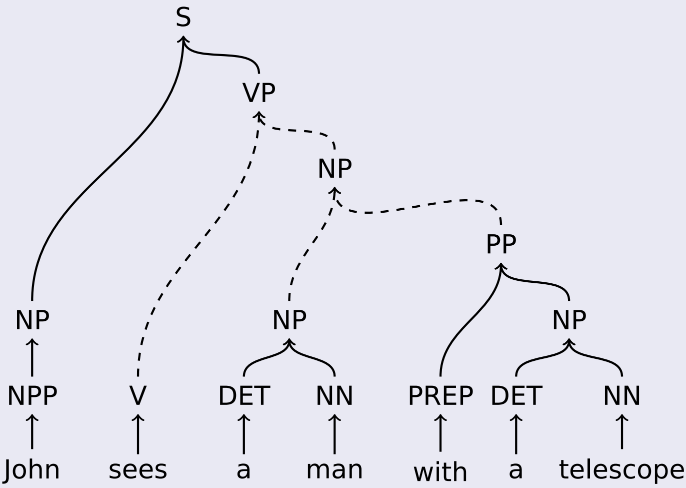
to address some of the deficiencies of the PCFG-LA learning process

- uses several grammars/parsers
  1. different sets of symbols
  2. different binarization schemes
- finds a parse that maximizes the sum of the scores of parsers
- relies on dual decomposition
  1. simplicity : reuse of vanilla CKY algorithm
  2. no additional heuristics : certificates of optimality
  3. not a "joint-system" : avoids search space explosion

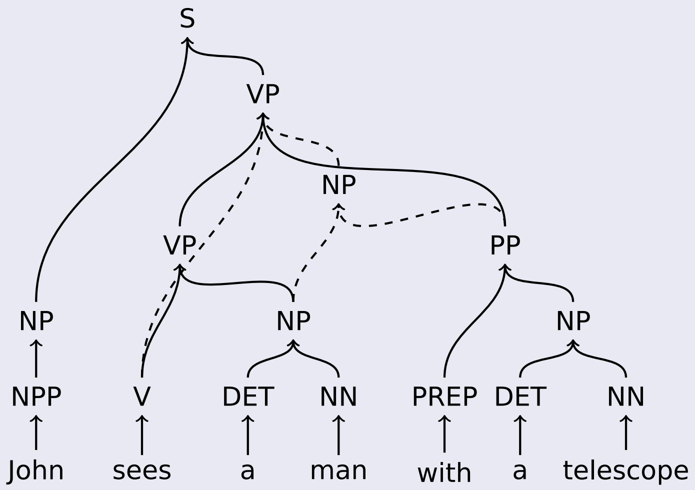
Find a parse (tree) for a given sentence (sequence)



Find a parse (tree) for a given sentence (sequence)



Find a parse (tree) for a given sentence (sequence)



Outline

Probabilistic Context-Free Grammars (PCFGs)

$G = (\mathcal{N}, \mathcal{T}, \mathcal{R}, s, q)$

- $\mathcal{N}$ non-terminals (with axiom s) and $\mathcal{T}$ terminals
- $\mathcal{R}$ set of rewrite rules
  - $a \to \gamma$ with $a \in \mathcal{N}, \gamma \in \mathcal{N}^+$

$$
\begin{array}{cccc}
 & & a & \\
\gamma_1 & \gamma_2 & \cdots & \gamma_n
\end{array}
$$

  - $a \to w$ with $a \in \mathcal{N}, w \in \mathcal{T}$

$$
\begin{array}{c}
a \\
\uparrow \\
w
\end{array}
$$

- for each rule r, a parameter $q(r) \geq 0$

$$\forall a \in \mathcal{N} \qquad \sum_{\gamma} q(a \to \gamma) + \sum_{w} q(a \to w) = 1$$

Probabilistic Context-Free Grammars (PCFGs)

weight of a tree/derivation

$$Q(T) = \prod_{r \in T} q(r)^{c(r,T)}$$: probability of a derivation

$$s(T) = \sum_{r \in T} c(r,T) \cdot log(q(r))$$: score of a tree

Probabilistic Context-Free Grammars (PCFGs)

weight of a tree/derivation

$$Q(T) = \prod_{r \in T} q(r)^{c(r,T)} \quad : \text{probability of a derivation}$$

$$s(T) = \sum_{r \in T} c(r,T) \cdot log(q(r)) \quad : \text{score of a tree}$$

Parsing as a linear system

$$\begin{aligned} T^* &= \arg\max_T \; s(T) \\ &= \arg\max_T \sum_{r \in T} c(r,T) \cdot log(q(r)) \\ &= \arg\max_T \; w \cdot \mathbf{N}\{r \in T\} \end{aligned}$$

Probabilistic Context-Free Grammars (PCFGs)

weight of a tree/derivation

$$Q(T) = \prod_{r \in T} q(r)^{c(r,T)} \quad : \text{probability of a derivation}$$

$$s(T) = \sum_{r \in T} c(r,T) \cdot log(q(r)) \quad : \text{score of a tree}$$

Parsing as a linear system

$$\begin{aligned} T^* &= \arg\max_T \; s(T) \\ &= \arg\max_T \sum_{r \in T} c(r,T) \cdot log(q(r)) \\ &= \arg\max_T \; w \cdot \mathbf{N}\{r \in T\} \end{aligned}$$

Key property

parse tree = unique derivation

Efficient Parsing : CKY Algorithm (1967)

Bottom-up parsing algorithm

forest construction and best solution

Parsing as Deduction

$[a, i, j]$: partial parse-tree rooted in A between words i and j

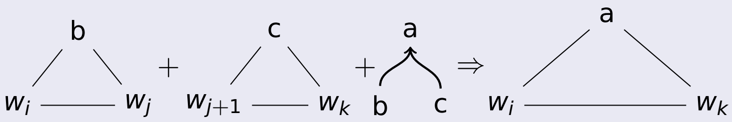
$|w| = n$: length of the sentence $w = w_1 \dots w_n$

- initialization:

$$\{[a, i, i] \mid a \rightarrow w_i \in \mathcal{R}\}$$

- deductive rules applied until stabilization, ie for binary rules :

$$\frac{[b, i, j] \quad [c, j+1, k]}{[a, i, k]} \quad \text{if } a \rightarrow bc \in \mathcal{R}$$



Bottom-up parsing algorithm

forest construction and best solution

Complexity

depends on the length of rules

- $O(|\mathcal{R}| \cdot |w|^{1+lg(\mathcal{R})})$
- with unary and binary rules $O(|\mathcal{R}| \cdot |w|^3)$

$\Rightarrow$ Grammar Binarization

Grammar binarization

Well-known : Exact Binarization (*Chomsky normal form*)

$a \rightarrow bcde$	$a \rightarrow ba_1$ $\quad a_2 \rightarrow de$ $a_1 \rightarrow ca_2$	$a \rightarrow a_1 e$ $\quad a_2 \rightarrow bc$ $a_1 \rightarrow da_2$
$a \rightarrow cde$	$a \rightarrow ca_3$ $a_3 \rightarrow de$	$a \rightarrow a_3 e$ $a_3 \rightarrow cd$

- Exact binarization $\rightarrow$ same trees (up to *debinarization*)
- Adds many new non-terminals/rules

Grammar binarization

Well-known : Exact Binarization (*Chomsky normal form*)

$a \to bcde$	$a \to ba_1$ $\quad a_2 \to de$ $a_1 \to ca_2$	$a \to a_1e$ $\quad a_2 \to bc$ $a_1 \to da_2$
$a \to cde$	$a \to ca_3$ $a_3 \to de$	$a \to a_3e$ $a_3 \to cd$

- Exact binarization $\to$ same trees (up to *debinarization*)
- Adds many new non-terminals/rules

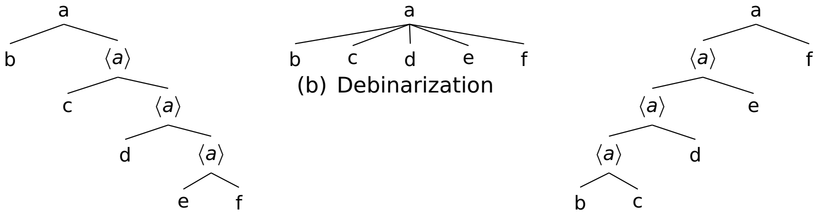
Less known *Markovized binarization*

$a \to bcde$	$a \to b\langle a\rangle$ $\quad \langle a\rangle \to de$ $\langle a\rangle \to c\langle a\rangle$
$a \to cde$	$a \to c\langle a\rangle$ $\langle a\rangle \to de$

- Inexact Binarization $\to$ overgeneration
- Compact $\to$ at most one new NT per original NT

Binarization

Rebuilding the original tree



(a) Left Bin.

(b) Debinarization

(c) Right Bin.

- *a* natural non-terminal
- $\langle a \rangle$ artificial non-terminal
- debinarization before outputting the trees

Estimation of parameters

Supervised Learning: learn the grammar from binarized trees (training set $\mathcal{E}$)

Maximize the log-likelihood of the training set

$$q^* \quad = \quad \arg\max_q log \prod_{t \in \mathcal{E}} Q(T) = \arg\max_q \sum_{t \in \mathcal{E}} s(t)$$

Estimation of parameters

Supervised Learning: learn the grammar from binarized trees (training set $\mathcal{E}$)

Maximize the log-likelihood of the training set

$$
\begin{aligned}
q^* &= \arg\max_q log \prod_{t \in \mathcal{E}} Q(T) = \arg\max_q \sum_{t \in \mathcal{E}} s(t) \\
&= \arg\max_q \sum_{t \in \mathcal{E}} \sum_{r \in T} c(r, T) \log(q(r)) = \arg\max_q \sum_{r \in \mathcal{E}} \mathrm{c}(r, \mathcal{E}) \log(q(r))
\end{aligned}
$$

Estimation of parameters

Supervised Learning: learn the grammar from binarized trees (training set $\mathcal{E}$)

Maximize the log-likelihood of the training set

$$\begin{aligned}
q^* &= \arg\max_q log \prod_{t \in \mathcal{E}} Q(T) = \arg\max_q \sum_{t \in \mathcal{E}} s(t) \\
&= \arg\max_q \sum_{t \in \mathcal{E}} \sum_{r \in T} c(r, T) \log(q(r)) = \arg\max_q \sum_{r \in \mathcal{E}} \mathrm{c}(r, \mathcal{E}) \log(q(r)) \\
&= \arg\max_q \sum_{a \in \mathcal{N}} \sum_{r \in \mathcal{E}, lhs(r) = a} \mathrm{c}(r) \log(q(r)) \\
& \text{with } \forall a \in \mathcal{N} \quad \sum_{\gamma} q(a \rightarrow \gamma) + \sum_{w} q(a \rightarrow w) = 1
\end{aligned}$$

Estimation of parameters

Supervised Learning: learn the grammar from binarized trees (training set $\mathcal{E}$)

Maximize the log-likelihood of the training set

$$
\begin{aligned}
q^* &= \arg\max_q log \prod_{t \in \mathcal{E}} Q(T) = \arg\max_q \sum_{t \in \mathcal{E}} s(t) \\
&= \arg\max_q \sum_{t \in \mathcal{E}} \sum_{r \in T} c(r, T) \log(q(r)) = \arg\max_q \sum_{r \in \mathcal{E}} \mathrm{c}(r, \mathcal{E}) \log(q(r)) \\
&= \arg\max_q \sum_{a \in \mathcal{N}} \sum_{r \in \mathcal{E}, lhs(r) = a} \mathrm{c}(r) \log(q(r)) \\
& \text{with } \forall a \in \mathcal{N} \quad \sum_{\gamma} q(a \rightarrow \gamma) + \sum_{w} q(a \rightarrow w) = 1
\end{aligned}
$$

Simple closed form

$$
q^*(a \rightarrow \gamma) = \frac{c(a \rightarrow \gamma)}{c(a)}
$$

Non-Terminals

The nodes in the corpus are tagged with:

1. a grammatical category (noun phrase, verb phrase,. . . and so on)
2. sometimes a function (subject, object. . .)
3. possibly some other info (trace of syntactic movement)

In this talk

- always keep the categories
- plus two approaches: with and without functions
  - functions may add interesting information → learning is more accurate
  - add data sparseness → learning can be less accurate
  - are not evaluated (in the main parse metrics)

Parseval F-score

Measure the score of tree in terms of constituents $[A, i, j]$

- $R = \frac{|\text{correct returned constituents}|}{|\text{reference constituents}|}$ $\quad P = \frac{|\text{correct returned constituents}|}{|\text{returned constituents}|}$
- $F = \frac{2PR}{P+R} \times 100$

Some results

Size of PCFGs / Parsing accuracy

Parseval F-score

Measure the score of tree in terms of constituents $[A, i, j]$

- $R = \frac{|\text{correct returned constituents}|}{|\text{reference constituents}|}$ $\quad P = \frac{|\text{correct returned constituents}|}{|\text{returned constituents}|}$
- $F = \frac{2PR}{P+R} \times 100$

Penn TreeBank evaluation

- train set $\approx 40k$ sentences, test set ≈ 2400 sentences
- all grammars right-binarized

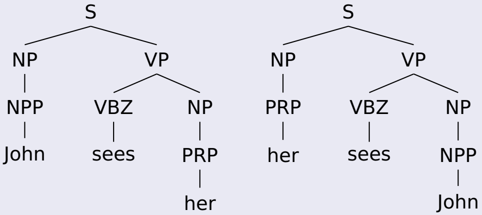
binarization / NT set	nb of NTs	nb of Rules	**F**	**Exact**
markov / no fun	98	4,076	65.27	6.75
markov / fun	459	9,963	67.55	7.37
exact / no fun	12,946	27,845	74.64	9.52
exact / fun	18,273	44,307	75.83	11.55

Outline

PCFG have issues modelling Natural Language :

- NL beyond context-free (won't be addressed directly)
- What is the correct/best set of non-terminals?

With a coarse NT set, PCFGs cannot give different weights to:



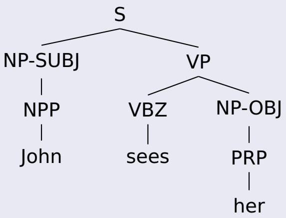
(d) Correct (e) Incorrect

Limitations of PCFGs

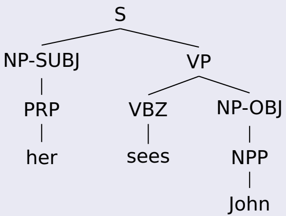
PCFG have issues modelling NL :

- NL beyond contex-free (won't be addressed directly)
- What is the correct/best set of non-terminals?

With a richer NT set, PCFGs can give different weights to:



(f) Correct



(g) Incorrect

Set of NT symbols

Late 90s / Early 00s: Quest for the perfect set of NTs

- simple re-annotation [Johnson, 1999]
  - *parent annotation*
  - cannot be extended very far
  - $F = 79.6$
- very complex re-annotation[Klein and Manning, 2003]
  - almost complete rewrite of the treebank
  - requires understanding of the language, the treebank, the parser. . .
  - some refinements are detrimental, too precise/sparse
  - $F = 85.7$

PCFG-LA

Idea: use refined symbols with coarse symbols

A grammar where each NT is of the form $a[x]$:

- a is non-terminal symbol, as appearing in the treebank (coarse-grain)
- x is a specialization drawn from a small set of possible refinements of a

Determiners

- in the treebank : DET for *the, a, this*
- we want DET[def] for *the*, DET[undef] for *a*, DET[dem] for *this*

Learn such a grammar from a regular (coarse-grain) treebank

PCFG-LA

Definition [Matsuzaki et al., 2005]

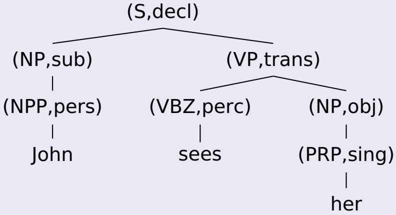
$G = (\mathcal{N}, \mathcal{H}, \mathcal{T}, \mathcal{R}, s, p)$

- $\mathcal{N}$ non-terminals (with axiom s) and $\mathcal{T}$ terminals
- $\mathcal{H}$ hidden refinement of categories
- $\mathcal{R}$ set of rewriting rules
  - $a[x] \to b[y]c[z]$ with $a, b, c \in \mathcal{N}$ and $x, y, z \in \mathcal{H}$
  - $a[x] \to w$ with $a \in \mathcal{N}, x \in \mathcal{H}, w \in \mathcal{T}$
- parameters p with each rule s.t. $\forall a[x] \in \mathcal{N} \times \mathcal{T}$

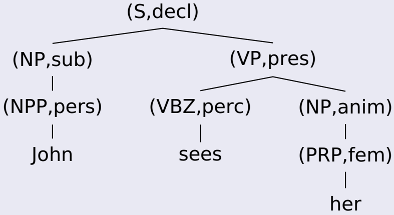
$$\sum_{\gamma} p(a[x] \to \gamma) + \sum_{w} p(a[x] \to w) = 1$$

Two kinds of trees

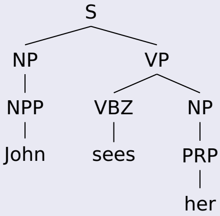
Derivations vs. Parses



(h) Derivation tree 1



(i) Derivation tree 2



(j) Parse tree

Two kinds of trees

Probability of a derivation tree $T_{\mathcal{H}}$

$$P(T_{\mathcal{H}}) = \prod_{r \in T_{\mathcal{H}}} p(r)$$

Probability of a parse tree T

$$P(T) = \sum_{T_{\mathcal{H}} \in \rho^{-1}(T)} \prod_{r \in T_{\mathcal{H}}} p(r)$$

- where ρ is a projection from annotated trees to skeletals.
- ρ keeps only the first component of NTs

Learning PCFG-LA

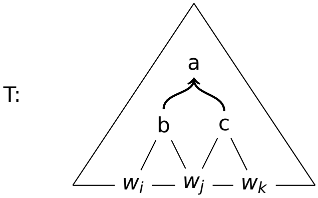
Objective

- parameters that maximize log-likelihood of a training set of parse trees
- but this time we want to learn refined categories!
- one parse tree: exponentially many derivations trees
- we restrict to $\mathcal{H} = \{1..n\}$ with fixed n
- we suppose axiom can have only one refinement.

Use the Expectation-Maximization algorithm

Learning PCFG-LA

Sentence w and its parse tree T



Let us define the probability of deriving:

- the substring from a refined symbol

$$P_{IN}^{i,k}(a[x]) = P(w_i \ldots w_k | a[x])$$

- the complete context for $a[x]$ from i to k from the axiom

$$P_{OUT}^{i,k}(a[x]) = P(w_1 \ldots w_{i-1} a[x] w_{k+1} \ldots w_n)$$

Learning PCFG-LA

These quantities can be efficiently computed :

Inside:

$$
\begin{aligned}
P_{IN}^{i,i}(a[x]) &= p(a[x] \rightarrow w_i) \\
P_{IN}^{i,k}(a[x]) &= \sum_{y,z} p(a[x]) \rightarrow (b[y])(c[z])) \cdot P_{IN}^{i,j}(b[y]) \cdot P_{IN}^{j+1,k}(c[z])
\end{aligned}
$$

Outside:

$$
\begin{aligned}
P_{OUT}^{1,n}(s[0]) &= 1 \\
P_{OUT}^{i,j}(b[y]) &= \sum_{x,z} p(a[x] \rightarrow b[y]c[z]) \cdot P_{OUT}^{i,k}(a[x]) \cdot P_{IN}^{j+1,k}(c[z]) \\
P_{OUT}^{j+1,k}(c[z]) &= \sum_{x,y} p(a[x] \rightarrow b[y]c[z]) \cdot P_{OUT}^{i,k}(a[x]) \cdot P_{IN}^{i,j}(b[y])
\end{aligned}
$$

Learning PCFG-LAs

We can calculate $P(T[r], i, j, k)$, the probability of all derivations containing $r = a[x] \rightarrow b[y]c[z]$ in T at position (i, j, k)

$$p(a[x] \rightarrow b[y]c[z]) \cdot P_{in}^{i,k}(b[y]) \cdot P_{in}^{k+1,j}(c[z]) \cdot P_{out}^{i,j}(a[x])$$

Expectation Maximization

EM : Maximum Likelihood estimation with omitted data

- counts $\rightarrow$ fractional counts (based on expectation)
- likelihood improves between iterations [Smith, 2011]
- can be seen as coordinate block ascent
- can get stuck to a local maximum

2 steps repeated until stabilization

1. get expected counts for $r = a[x] \rightarrow b[y]c[z]$:

$$EC[r] = \sum_{t \in \mathcal{E}} \frac{1}{P(t)} \sum_{r \text{ spans } (i,j,k) \in t} P(t[r], i, j, k)$$

2. compute new probabilities (results of a maximization)

$$p(a[x] \rightarrow b[y]c[z]) = \frac{EC[a[x] \rightarrow b[y]c[z]]}{EC[a[x]]}$$

Parsing with PCFG-LA

Find the best tree with the given sentence as its yield:

$$T^* = \arg\max_T \sum_{T_{\mathcal{H}} \in \rho^{-1}(T)} \prod_{r \in T_{\mathcal{H}}} p(r) \qquad (1)$$

Intractability

Reduction to the problem of finding the best 'unfolding' (tree) from an *ambiguous* (tree-local) Weighted Tree Automaton [Maletti and Satta, 2009]

Can we find an efficient *good* approximate?

Approximate Parsing with PCFG-LA (1)

PCFG-LA as a big PCFG

Compute the best derivation and *clean* it

$$T^* = \rho(\arg\max_{T_{\mathcal{H}}} \prod_{r \in T_{\mathcal{H}}} p(r)) \qquad (2)$$

- We can reuse CKY parsing as is
- This assumes that the distribution is *dominated* by a best derivation. This is rarely the case $\rightarrow$ suboptimal accuracy

A few remarks

- Although exact parsing is intractable, computing inside/outside probabilities in the parse forest is tractable
- Learning PCFG from a corpus (even as small as a single parse forest) is easy

$\rightarrow$ **Variational Inference!**

- We want to learn a PCFG that recognizes only the current sentence
- With statistics gathered from the PCFG-LA shared forest

Rules of the PCFG we want to learn

- $a^{i,k} \rightarrow b^{i,j}\ c^{j+1,k}$ or $a^{i,i} \rightarrow w_i$
- Symbols have a fixed position
- denoted: $(a \rightarrow b\ c, i, j, k)$ or $(a \rightarrow w, i)$

Rules of the PCFG we want to learn

- $a^{i,k} \rightarrow b^{i,j}\ c^{j+1,k}$ or $a^{i,i} \rightarrow w_i$
- Symbols have a fixed position
- denoted: $(a \rightarrow b\ c, i, j, k)$ or $(a \rightarrow w, i)$

We want the distributions to be as close as possible

For a given sentence w

$$\arg\min_Q KL(P||Q) = \arg\min_Q \sum_{T \in \mathcal{F}(w)} P(T) \log \frac{P(T)}{Q(T)}$$

where

- P is the probability of the PCFG-LA model (sum of products)
- Q is the probability of the PCFG model (product)

We want the distributions to be as close as possible

For a given sentence

$$
\begin{aligned}
Q^* &= \arg\min_Q \sum_{T \in \mathcal{F}} P(T) \log \frac{P(T)}{Q(T)} = \arg\min_Q \sum_{T \in \mathcal{F}} K_T - P(T) \log Q(T) \\
&= \arg\max_Q \sum_{T \in \mathcal{F}} P(T) \log Q(T)
\end{aligned}
$$

We want the distributions to be as close as possible

For a given sentence

$$
\begin{aligned}
Q^* &= \arg\min_Q \sum_{T\in\mathcal{F}} P(T)\log\frac{P(T)}{Q(T)} = \arg\min_Q \sum_{T\in\mathcal{F}} K_T - P(T)\log Q(T) \\
&= \arg\max_Q \sum_{T\in\mathcal{F}} P(T)\log Q(T) \\
&= \arg\max_Q \sum_{T\in\mathcal{F}} P(T)\sum_{R\in T} c(R,T)\log q(R)
\end{aligned}
$$

We want the distributions to be as close as possible

For a given sentence

$$
\begin{aligned}
Q^* &= \arg\min_Q \sum_{T\in\mathcal{F}} P(T)\log\frac{P(T)}{Q(T)} = \arg\min_Q \sum_{T\in\mathcal{F}} K_T - P(T)\log Q(T) \\
&= \arg\max_Q \sum_{T\in\mathcal{F}} P(T)\log Q(T) \\
&= \arg\max_Q \sum_{T\in\mathcal{F}} P(T)\sum_{R\in T} c(R,T)\log q(R) \\
&= \arg\max_Q \sum_{R\in\mathcal{F}} \Big(\sum_{R=\rho^{-1}(r)} P(\mathcal{F}[r], span(R))\Big)\log q(R)
\end{aligned}
$$

Variational Inference

The minimization under constraints has a closed form:

$$
\begin{aligned}
score(a \rightarrow b\ c, i, j, k) &= \sum_{x,y,z \in \mathcal{H}} P^{i,k}_{\text{out}}(a[x]) \cdot p(\ a[x] \rightarrow b[y]\ c[z]\ ) \cdot P^{i,j}_{\text{in}}(b[y]) \cdot P^{j,k}_{\text{in}}(c[z]) \\
norm(a \rightarrow b\ c, i, j, k) &= \sum_{x \in \mathcal{H}} P^{i,k}_{\text{in}}(a[x]) \cdot P^{i,k}_{\text{out}}(a[x]) \\
score(a \rightarrow w, i) &= \sum_{x \in \mathcal{H}} P^{i,i}_{\text{out}}(a[x]) \cdot p(a[x] \rightarrow w) \\
norm(a \rightarrow w, i) &= \sum_{x \in \mathcal{H}} P^{i,i}_{\text{in}}(a[x]) \cdot P^{i,i}_{\text{out}}(a[x]) \\
q(r_s) &= \left[\frac{score(r_s)}{norm(r_s)} \text{ (V. Inference)} \right]
\end{aligned}
$$

Variational Inference

The minimization under constraints has a closed form:

$$
\begin{aligned}
score(a \to b\ c, i, j, k) &= \sum_{x,y,z \in \mathcal{H}} P_{\text{out}}^{i,k}(a[x]) \cdot p(\ a[x] \to b[y]\ c[z]\ ) \cdot P_{\text{in}}^{i,j}(b[y]) \cdot P_{\text{in}}^{j,k}(c[z]) \\
norm(a \to b\ c, i, j, k) &= \sum_{x \in \mathcal{H}} P_{\text{in}}^{i,k}(a[x]) \cdot P_{\text{out}}^{i,k}(a[x]) \\
score(a \to w, i) &= \sum_{x \in \mathcal{H}} P_{\text{out}}^{i,i}(a[x]) \cdot p(a[x] \to w) \\
norm(a \to w, i) &= \sum_{x \in \mathcal{H}} P_{\text{in}}^{i,i}(a[x]) \cdot P_{\text{out}}^{i,i}(a[x]) \\
q(r_s) &= \left[\frac{score(r_s)}{norm(r_s)} \text{ (V. Inference)} \right] \text{ or } \left[\frac{score(r_s)}{P_{\text{in}}^{0,n}(S[0])} \text{ (Petrov's MR)} \right]
\end{aligned}
$$

Some Results: Approximate PCFG-LA parsing

Penn TreeBank evaluation PTB Sec 23

Markov Right bin / no function

$\|\mathcal{H}\|$	**F**	**Exact**
1	65.27	6.75
2	75.76	10.76
4	84.09	21.52
8	87.19	28.52
16	89.06	33.32
32	90.03	35.82
64	90.30	36.02

Some Results: Approximate PCFG-LA parsing

Penn TreeBank evaluation PTB Sec 23

Markov Right bin / no function

$\vert\mathcal{H}\vert$	**F**	**Exact**
1	65.27	6.75
2	75.76	10.76
4	84.09	21.52
8	87.19	28.52
16	89.06	33.32
32	90.03	35.82
64	90.30	36.02

With $|\mathcal{H}| = 64$

approx/ binarization / NT set	**F**	**Exact**
PCFG/ markov right / no fun	88.81	33.54
PCFG / markov right / fun	88.70	33.50
V.I. / markov right / no fun	90.30	36.02
V.I. / markov right / fun	89.85	36.22
V.I. / markov left / no fun	90.38	36.01
V.I. / markov left / fun	89.56	34.13

Product of PCFG-LAs [Petrov, 2010]

EM grammar accuracy depends on initial settings (up to 10% ER)

Petrov's idea

- train several grammars that only differ in their initial settings
- and combine their scores

$$T^* = \arg\max_T \prod_{i=1}^{n} Q_{G_i}(T)$$

Scoring with n grammars:

$$\begin{aligned} T^* &= \arg\max_T \sum_{i=1}^{n} \sum_{r \in T} \log q_{G_i}(r) \\ &= \arg\max_T \sum_{r \in T} \sum_{i=1}^{n} \log q_{G_i}(r) \end{aligned}$$

The product can be treated as one grammar: CKY still applies

Some results

Products of 16 grammars

Evaluation on PTB test

Binarization/NT set	**F**	**EX**
Markov Right Bin / No Fun	91.76	40.73
Markov Left Bin / No Fun	91.57	39.07
Markov Right Bin / Fun	91.73	41.47
Markov Left Bin / Fun	91.45	40.11

Some results

Products of 16 grammars

Evaluation on PTB test

Binarization/NT set	F	EX
Markov Right Bin / No Fun	91.76	40.73
Markov Left Bin / No Fun	91.57	39.07
Markov Right Bin / Fun	91.73	41.47
Markov Left Bin / Fun	91.45	40.11

- Binarizations $\rightarrow$ different errors
- Expert system with different binarization schemes

Outline

Combined Parsing System

- Different grammars (binarizations, symbols)

Combined Parsing System

- Different grammars (binarizations, symbols)
- Different grammars $\rightarrow$ different parsers p

$$T_p^* = \arg\max_T s_p(T)$$

Combined Parsing System

- Different grammars (binarizations, symbols)
- Different grammars $\rightarrow$ different parsers p

$$T_p^* = \arg\max_T s_p(T)$$

- May not give the same solution in general

$$T_{p_i}^* \neq T_{p_j}^*$$

Combined Parsing System

- Different grammars (binarizations, symbols)
- Different grammars $\rightarrow$ different parsers p

$$T_p^* = \arg\max_T s_p(T)$$

- May not give the same solution in general

$$T_{p_i}^* \neq T_{p_j}^*$$

- Combination as a vote of experts

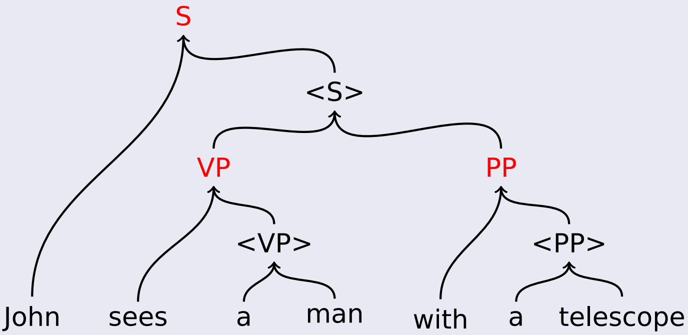
$$T^* = \arg\max_T \sum_{p=1}^{n} s_p(\mathcal{C}_p(T))$$

Issues: How to combine really different trees?

Combine the different binarizations

Parse agreement

- Each parser is an expert with it own binarization
- They must agree on the debinarized tree

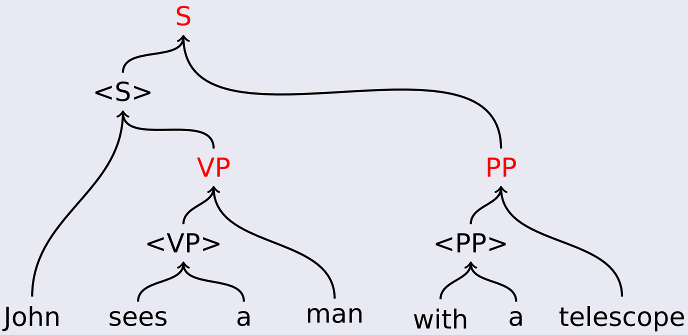


Expert 1 : left binarization

Combine the different binarizations

Parse agreement

- Each parser is an expert with it own binarization
- They must agree on the debinarized tree



Expert 2 : right binarization

Combine the different binarizations

Parse agreement

- Each parser is an expert with it own binarization
- They must agree on the debinarized tree

[S,1,7]

[VP,1,4]   [PP,5,7]

John sees a man with a telescope

Agreement

The new parsing problem

Combining different parsers

expert system:

$$T^* = \arg\max_T \sum_{p=1}^{n} s_p(T)$$

Our problem becomes

Find the best solution such that parsers agree on Natural NTs

$$(P): \quad T^* = \arg\max_{(T_1 \ldots T_n) \in \mathcal{C}} \sum_{p=1}^{n} s_p(T_p)$$
$$s.t. \quad z(T_i) = z(T_j), \qquad \forall (i,j) \in [\![1,n]\!]^2$$

$z(T_p)$: boolean vector indexed by Natural NTs

- $z(T_p)[A,i,j] = 1$ if A is in T_p and spans from i to j
- $z(T_p)[A,i,j] = 0$ otherwise

Dynamic Programming Solution

We can write a CKY variant that solve the new parsing problem

Intractable

- amounts to debinarize in the different parsers *on the fly*
- the length of debinarized rules is the bottleneck (CKY)
- because of Markovization, debinarized rules can be arbitrarily long (up to the length of the sentence)

The joint approach is a dead-end

Dual Decomposition Solution

Rewrite with a witness vector u

$$
\begin{aligned}
&(P) : \textit{Find} \qquad && o_P = \max_{(T_1 \dots T_n) \in \mathcal{C}} \sum_{i=1}^{n} s_i(T_i) && \\
&\qquad s.t. && z(T_i) = u && \exists u \in \mathbb{R}^d, \forall i \in [\![1, n]\!]
\end{aligned}
$$

Sub-problems tractable, coupling is not

Idea:

1. Transform coupling constraints into numerical penalties (Lagrangian)
2. Integrate this penalties into the objective

Relaxation

For each parser p_k, there is a real vector Λ_k indexed by $[A, i, j]$.

Relaxation

$$(RP): \quad o_{RP} = \max_{u, T_{1\ldots n}} \min_{\Lambda} \sum_i s_i(T_i) + \sum_i (z(T_i) - u) \cdot \Lambda_i$$

- If the coupling constraints are satisfied $\rightarrow$ same as before
- otherwise, we can set the values in Λ_i to $\pm\infty$ and get a minimum of $-\infty$

We get the same maximal solution iff the contraints are satisfied.

Dualization

Dualization 1: we permute max and min

$$(LP1): \quad o_{LP1} = \quad \min_{\Lambda} \max_{u, T_{1\ldots n}} \sum_i s_i(T_i) + \sum_i z(T_i) \cdot \Lambda_i - u \cdot \sum_i \Lambda_i$$

- To obtain finite solutions: $\sum_i \Lambda_i = \mathbf{0}$

Dualization

Dualization 1: we permute max and min

$$(LP1): \quad o_{LP1} = \quad \min_{\Lambda} \max_{u, T_{1\ldots n}} \sum_i s_i(T_i) + \sum_i z(T_i) \cdot \Lambda_i - u \cdot \sum_i \Lambda_i$$

- To obtain finite solutions: $\sum_i \Lambda_i = \mathbf{0}$

Dualization(2) with constraints

$$(LP): \quad o_{LP} = \quad \min_{\Lambda} \sum_{i=1}^{n} \max_{T_i \in \mathcal{F}_i} \left(s_i(T_i) + z(T_i) \cdot \Lambda_i \right)$$

$$s.t. \quad \sum_i \Lambda_i = \mathbf{0}$$

- If we search only on $\sum_i \Lambda_i = \mathbf{0}$, the subproblems are separated.
- $\rightarrow$ projected subgradient method

Minimization algorithm

Projected sub-gradient method

- Initialize Λ_i to 0
- Until parsers agree on natural NTs:
  - We solve each sub-problem:
    - $\max_{T \in \mathcal{F}_i} (s_i(T) + z(T) \cdot \Lambda_i)$
    - the penalties Λ_i are integrated in the CKY algorithm
    - (best path with node penalties)
  - Update Λ_i
    - with constraints $\sum_i \Lambda_i = \mathbf{0}$
    - proportionally to the difference between the solution $z(T_i)$ of parser i and the average solution.

Algorithm

Require: n parsers $\{p_i\}_{1 \leq i \leq n}$
  for all i, **do**
    $\Lambda_i^{(0)} = \mathbf{0}$
  end for
  for $t = 0 \to \tau$ **do**
    for all parsers p_i **do**
    $T_i^{(t)} \leftarrow \arg\max_{T \in \mathcal{F}_i} \left(s_i(T) + z(T) \cdot \Lambda_i^{(t)} \right)$
    end for
    for all parsers p_i **do**
      $\Delta_i^{(t)} \leftarrow \alpha_t \left(z\left(T_i^{(t)}\right) - \frac{\sum_{1 \leq j \leq n} z\left(T_j^{(t)}\right)}{n} \right)$
      $\Lambda_i^{(t+1)} \leftarrow \Lambda_i^{(t)} + \Delta_i^{(t)}$
    end for
    if $\Delta_i^{(t)} = 0$ for all i **then**
      Break
    end if
  end for
  return $(T_1^{(\tau)}, \cdots, T_n^{(\tau)})$

Remarks

This method applies to any PCFG

- Not just learned from PCFG-LA
- Need to define some agreement constraints over grammars

Certificate of optimality

If we find a solution, it is optimal

Subproblems are independent

The computation of the best solutions can be parallelized

Iterative algorithm

Additive complexity (vs. Multiplicative for joint systems)

Final results

Evaluation on PTB-WSJ23

System	F	EX
Markov Right Bin / No Fun	91.76	40.73
Markov Left Bin / No Fun	91.57	39.07
Markov Right Bin / Fun	91.73	41.47
Markov Left Bin / Fun	91.45	40.11
DD No Fun	92.09	41.51
DD Fun	92.26	42.09
DD Markov Right Bin	92.16	42.05
DD Markov Left Bin	91.89	40.65
DD4	**92.44**	42.38

- threshold: 1 000 iterations (95% converging)
- 85 iterations on average (39 on converging instances)

Outline

Conclusion & Future Work

PCFGs are a nice formalism!

- simple to understand
- efficient algorithms (polynomial...)
- no feature engineering
- generative model
  - probabilities we can use in many ways
  - inference is simple: we can learn PCFGs from parse forests

Conclusion & Future Work

PCFGs are a nice formalism!

- simple to understand
- efficient algorithms (polynomial...)
- no feature engineering
- generative model
  - probabilities we can use in many ways
  - inference is simple: we can learn PCFGs from parse forests

Combinations of PCFGs: state-of-the art systems

- on English (PTB sec23) [EMNLP 2013]
- unpublished on French (SPRML 2013 data)

Conclusion & Future Work

PCFGs are a nice formalism!

- simple to understand
- efficient algorithms (polynomial...)
- no feature engineering
- generative model
  - probabilities we can use in many ways
  - inference is simple: we can learn PCFGs from parse forests

Combinations of PCFGs: state-of-the art systems

- on English (PTB sec23) [EMNLP 2013]
- unpublished on French (SPRML 2013 data)

Future Work

- use treebank information on long distance dependencies
- combine parsing with other tasks (MWE tokenization)