Advanced PCFG Parsing

Computational Linguistics

Alexander Koller

10 December 2019

Today

- Semiring parsing.
- Pruning techniques for chart parsing.

Semiring parsing

- We have seen a number of algorithms on CKY charts that all look basically the same.
  - decide word problem
  - compute best parse
  - compute inside probabilities
  - compute number of parse trees

- What exactly do they have in common?
  Can we use it to build better algorithms?

CKY for recognition

```
for each i from 1 to n:
  for each production rule A → wi:
    Ch(A, i, i+1) = true

for each width b from 2 to n:
  for each start position i from 1 to n-b+1:
    for each left width k from 1 to b-1:
      for each production rule A → B C:
        Ch(A, i, i+b)
          = Ch(A, i, i+b) ∨
            (Ch(B, i, i+k) ∧ Ch(C, i+k, i+b) ∧ true)

return Ch(S, 1, n+1)
```

Viterbi-CKY

```
for each i from 1 to n:
  for each production rule A → wᵢ:
    Ch(A, i, i+1) = P(A → wᵢ)

for each width b from 2 to n:
  for each start position i from 1 to n-b+1:
    for each left width k from 1 to b-1:
      for each production rule A → B C:
        Ch(A, i, i+b)
          = max(Ch(A, i, i+b),
                Ch(B, i, i+k) * Ch(C, i+k, i+b) * P(A → B C))

return Ch(S, 1, n+1)
```

Inside

```
for each i from 1 to n:
  for each production rule A → wi:
    Ch(A, i, i+1) = P(A → wi)

for each width b from 2 to n:
  for each start position i from 1 to n-b+1:
    for each left width k from 1 to b-1:
      for each production rule A → B C:
        Ch(A, i, i+b)
          = Ch(A, i, i+b) +
            (Ch(B, i, i+k) * Ch(C, i+k, i+b) * P(A → B C))

return Ch(S, 1, n+1)
```

Counting

```
for each i from 1 to n:
  for each production rule A → wi:
    Ch(A, i, i+1) = 1

for each width b from 2 to n:
  for each start position i from 1 to n-b+1:
    for each left width k from 1 to b-1:
      for each production rule A → B C:
        Ch(A, i, i+b)
          = Ch(A, i, i+b) +
            (Ch(B, i, i+k) * Ch(C, i+k, i+b) * 1)

return Ch(S, 1, n+1)
```

Semirings

- A *semiring* is a 5-tuple consisting of
  - a nonempty set V of values
  - an *addition* $\oplus : V \times V \to V$, associative and commutative
  - a *multiplication* $\otimes : V \times V \to V$, must be associative and distribute over $\oplus$
  - an *abstract zero* $0 \in V$ such that $0 \oplus v = v \oplus 0 = v$ and $0 \otimes v = v \otimes 0 = 0$, for all v
  - an *abstract one* $1 \in V$ such that $1 \otimes v = v \otimes 1 = v$, for all v

A semiring where $\oplus$ has inverse elements is called a *ring*
— really important in math, but not so much in this course.

Some important semirings

	values	addition	multiplication	zero	one
counting	$\mathbf{N}_0$	+	*	0	1
boolean	{true, false}	$\vee$	$\wedge$	false	true
Viterbi	[0, 1]	max	*	0	1
inside	$[0, \infty]$	+	*	0	1

Semiring parsing

- We are interested in calculating value V(w) for the string out of values R(r) for the individual rules:

$$V(w) = \bigoplus_{t \in \mathcal{T}(w)} V(t)$$
$$= \bigoplus_{t \in \mathcal{T}(w)} \bigotimes_{\text{rule } r \text{ in } t} R(r)$$

- For any semiring, we can do this CKY-style:

$$V(A, i, i+1) = R(A \to w_i)$$
$$V(A, i, k) = \bigoplus_{\substack{A \to B\ C \\ i<j<k}} V(B, i, j) \otimes V(C, j, k) \otimes R(A \to B\ C)$$

Generic CKY with semirings

assume evaluation function R: rules → V

```
for each i from 1 to n:
  for each production rule A → wᵢ:
    Ch(A, i, i+1) = R(A → wᵢ)

for each width b from 2 to n:
  for each start position i from 1 to n-b+1:
    for each left width k from 1 to b-1:
      for each production rule A → B C:
        Ch(A, i, i+b)
          = Ch(A, i, i+b) ⊕
            (Ch(B, i, i+k) ⊗ Ch(C, i+k, i+b) ⊗ R(A → B C))

return Ch(S, 1, n+1)
```

This generalizes all the variants we saw above.

Parsing Schemata

- Parsing algorithm derives claims about the string. Record such claims in *parse items*.
- At each step, apply a *parsing rule* to infer new parse items from earlier ones.
- If there is a way to derive a *goal item* from the *start item(s)* for a given input string, then claim that this string is in the language.

Examples for schemas

	CKY	shift-reduce
items	(A, i, k)	(s,w')
claims	$A \Rightarrow^* w_i \ldots w_{k-1}$	$s\, w' \Rightarrow^* w$
rules	$\dfrac{A \rightarrow B\ C \quad (B, i, j) \quad (C, j, k)}{(A, i, k)}$	$\dfrac{(s,\ a \cdot w')}{(s \cdot a,\ w')}$ (shift) $\dfrac{(s \cdot s',\ w') \quad A \rightarrow s' \text{ in } P}{(s \cdot A,\ w')}$ (reduce)
start items	(A, i, i+1) if $A \rightarrow w_i$	(ε, w)
goal items	(S,1,n+1)	(S, ε)

Implementing schemas

- Can generally implement parser for given schema in the following way:
  - maintain an *agenda*: queue of items that we have discovered, but not yet attempted to combine with other items
  - maintain a *chart* of all seen items for the sentence

```
initialize chart and agenda with all start items

while agenda not empty:
  item = dequeue(agenda)
  for each combination c of item with other item in the chart:
    if c not in chart:
      add c to chart
      enqueue c in agenda

if chart contains a goal item, claim w ∈ L(G)
```

rules of parsing schema used here

essential to do this efficiently

Example

agenda:

(PP, 5, 8) (V, 2, 3) (Det, 3, 4) (N, 4, 5) (N, 4, 8) (NP, 3, 5)
(NP, 3, 8) (VP, 2, 5) (VP, 2, 8)

chart:

	2…	3…	4…	5…
…8	VP	NP	N	PP
…5	VP	NP	N	
…4		Det		
…3	V			

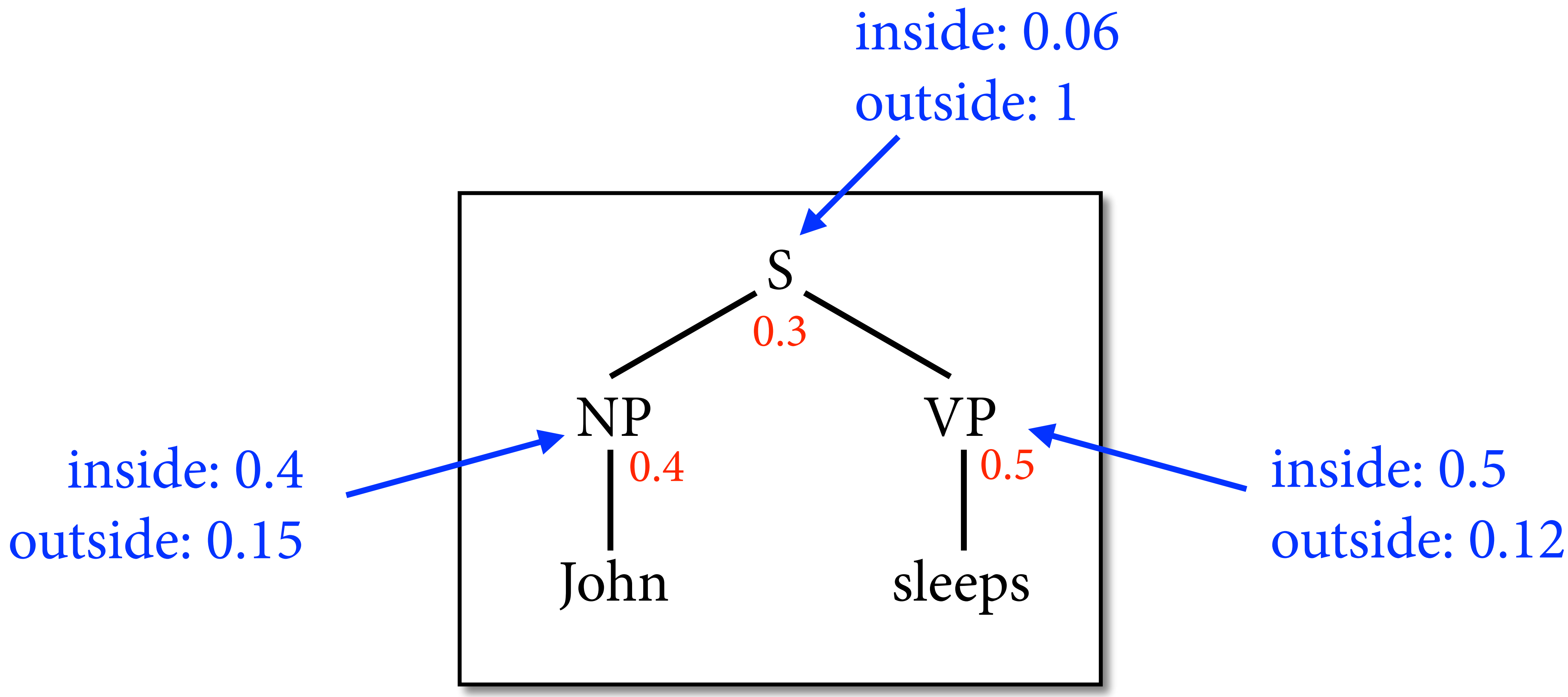
VP → V NP
NP → Det N
N → N PP

$$\frac{A \rightarrow B\ C \quad (B, i, j) \quad (C, j, k)}{(A, i, k)}$$

Pruning techniques

- If grammar is big and sentence is not short, computing the full chart is expensive.
  - runtime of CKY is $O(|G| * n^3)$
  - for treebank grammars, almost every substring can be derived from some nonterminal
- Most chart entries not used to build best parse tree.
- *Pruning:* avoid computing the full chart
  - *beam search:* limit number of entries per chart cell
  - *best-first search:* manipulate order in which items are taken from the agenda

Inside and outside probs



- For each individual parse tree, the product of inside and outside probabilities is same at every node.
- If we could calculate (inside * outside) for each chart item, then we could focus search on just the items that are needed for best parse.

Figures of Merit

- Challenge in bottom-up parsing:
  - We can easily compute (Viterbi) inside of each item. (Viterbi inside = max P(t); inside = Σ P(t).)
  - We cannot easily compute (Viterbi) outside, because we haven't combined item with other words yet.
- Idea: estimate (inside * outside) with a *figure of merit* (FOM) of the parse item.
  - FOM = Viterbi inside prob: underestimates quality of long substrings
  - FOM = (Viterbi inside)$^{1 / |\text{substring}|}$: works okay in practice, but still ignores outside probs

Beam search

- In CKY parsing, easiest way of using FOMs is *beam search:*
  - fix a number *k* of nonterminals that can be stored in each chart cell
  - only retain the *k* nonterminals with the best FOM
  - variant: only retain the nonterminals whose FOM is at least $\theta * f$, where *f* is FOM of best nonterminal in same cell
- Beam search very standard technique in parsing and machine translation (including decoding of neural network outputs).

Best-first parsing

- Idea: Agenda contains parse items (A, i, k); order them in descending order of their FOMs.

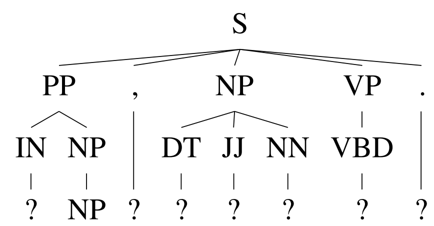
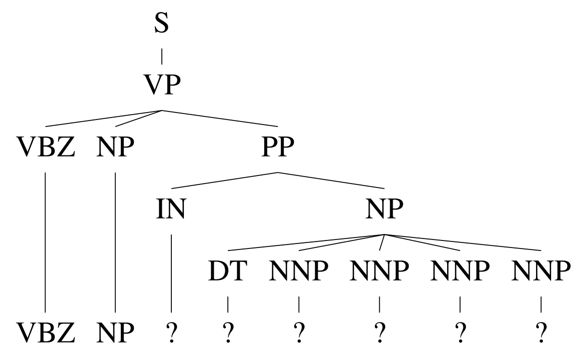
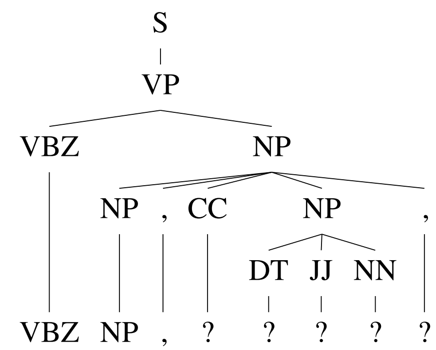
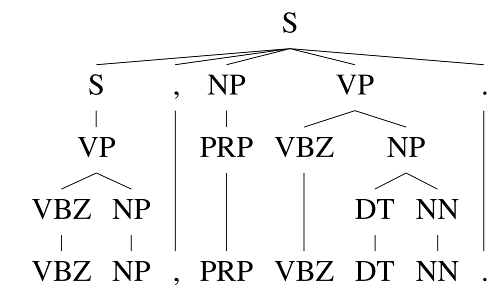
- If FOM were perfect, then first discovered goal item represents the best parse, and many unexplored items still on agenda ⇒ faster parser.

- If FOM is not perfect, parser can make *search errors:* first discovered goal item is not optimal.
  - can still be much faster than exhaustive parsing
  - accuracy depends on quality of FOM

A* parsing

- A* search: general method for heuristic search in AI
  - FOM h = (distance f from start) + (estimated distance g to goal)
  - g must *underestimate* distance, i.e. never be larger than true distance
  - guarantees that first path to goal we find is optimal
- Apply this to parsing (Klein & Manning 03):
  - f = - log inside
  - g = estimate of - log outside

Outside estimates

Estimate	SX	SXL	SXLR	TRUE
Summary	(1,6,NP)	(1,6,NP,VBZ)	(1,6,NP,VBZ,”,”)	(entire context)
Best Tree				
Score	-11.3	-13.9	-15.1	-18.1
	(a)	(b)	(c)	(d)

- Represent each parse item with a *summary,* which abstracts over the concrete sentence we are parsing.
- Compute outside estimates for each possible summary from grammar, before we start parsing actual sentences.

A* parsing: Results

Estimate	Savings	w/ Filter	Storage	Precomp
NULL	11.2	58.3	0K	none
S	40.5	77.8	2.5K	1 min
SX	80.3	95.3	5M	1 min
SXL	83.5	96.1	250M	30 min
S_1XLR	93.5	96.5	500M	480 min
SXR	93.8	96.9	250M	30 min
SXMLR	94.3	97.1	500M	60 min
B	94.6	97.3	1G	540 min

Coarse-to-fine parsing

- Idea: make coarser-grained grammar by combining "similar" nonterminals into one (Charniak et al. 06).
    - combine S, VP, S-bar, etc. into "S_"
    - combine S_ and N_ into "HP" (head phrase); etc.

- Compute complete parse chart with coarse-grained grammar; calculate exact inside and outside.

- Prune out entries with low inside * outside.
  Refine the others, then repeat until we have chart of original grammar.

CTF parsing: Results

Level	Constits Produced $*10^6$	Constits Pruned $*10^6$	% Pruned
0	8.82	7.55	86.5
1	9.18	6.51	70.8
2	11.2	9.48	84.4
3	11,8	0	0.0
total	40.4	–	–
3-only	392.0	0	0

Figure 5: Total constituents pruned at all levels for WSJ section 23, sentences of length ≤ 100

Level	Time for Level	Running Total
0	1598	1598
1	2570	4168
2	4303	8471
3	1527	9998
3-only	114654	–

Figure 6: Running times in seconds on WSJ section 23, with and without pruning

… at no loss in f-score with their grammar.

Summary

- PCFG parsing one of the most successful fields of NLP research.
- Current parsers are fast and quite accurate.
  - in practice, most people use Berkeley or Stanford parser for good speed-accuracy-convenience tradeoff
- Techniques from PCFG parsing carry over to many other problems in computational linguistics.