Statistical Parsing

Gerald Penn
CS224N
[based on slides by Christopher Manning, Jason Eisner and Noah Smith]

Example of uniform cost search vs. CKY parsing: CKY vs. order of agenda pops in chart

What can go wrong in parsing?

Speed: build promising items first.

Correctness: keep items on the agenda until you're sure you've seen their best parse.

Speeding up agenda-based parsers

- Two options for doing less work
  - The optimal way: A* parsing
    - Klein and Manning (2004)
  - The ugly but much more practical way: “best-first” parsing
    - Cardellino and Charniak (1998)
    - Charniak, Johnson, and Goldwater (2008)

A* Search

- Problem with uniform-cost:
  - Even utterly small edges have high score.
  - We end up processing every small edge

- Solution: A* Search
  - Small edges have to fit into a full pane.
  - The smaller the edge, the more the full pane will cost
    - Consider both the cost to build (β) and the cost to complete (α)

- We figure out β during parsing
  - We GUESS all α in advance (pre-processing)
  - Exactly calculating this quantity is as hard as parsing.
  - But we can do A* parsing if we can cheaply calculate underestimate of the true cost

Example of uniform cost search vs. CKY parsing: The grammar, lexicon, and sentence

• S → NP VP % 0.9
• S → VP % 0.1
• NP → V NP % 0.6
• NP → V % 0.4
• NP → N % 0.3
• N → people % 0.5
• N → fish % 0.1
• N → tanks % 0.1
• V → people % 0.1
• V → fish % 0.6
• V → tanks % 0.3
• people fish tanks
Using context for admissible outside estimates

- The more detailed the context used to estimate $\hat{c}$ is, the sharper our estimate is...

  Fix outside score: Score = 11.3
  Add left tag: Score = 13.5
  Add right tag: Score = 15.1

  Entire context gives the exact best parse.
  Score = 18.1

Categorical filters are a limit case of $A^*$ estimates

- Let projection $\pi$ collapse all prusal symbols to "X":

  $NP \rightarrow \pi CC NP CC NP$ ...

  $X \rightarrow \pi CC X CC X$

- When can $X \rightarrow \pi CC X CC X$ be completed?

  Whenever the right context includes two $CC$

  Gives an admissible lower bound for this projection that is very efficient to calculate.

A* Context Summary Sharpness

Adding local information changes the intercept, but not the slope!

Best-First Parsing

- In best-first parsing, we visit edges according to a
  figure-of-merit (FOM).

- A good FOM focuses work on "qualify" edges.

- The good leads to full parses quickly.

- The (potential) bad: leads to non-NF parses.

- The ugly: propagation
  * If we find a better way to build a
  parse tree, we need to rebuild everything above it

- In practice, works well

Beam Search

- State space search

  States are partial parses with an associated probability
  - Keep only the top scoring elements at each stage of the beam search
  - Find a way to ensure that all parses of a sentence have the same number $N$ steps
    * Or at least are roughly comparable
    * Leftmost top-down CFG derivations in true CNF
    * Shift-reduce derivations in true CNF
  - Partial parses that cover the same number of words

Beam Search

- Time-synchronous beam search

  Beam at time $i$

  Successors of beam elements

  Beam at time $i + 1$
Kinds of beam search

- Constant beam size $k$
- Constant beam width relative to best item
- Defined either additively or multiplicatively
- Sometimes combination of the above two
- Sometimes do fancier stuff like trying to keep the beam elements diverse
- Beam search can be made very fast
- No measure of how often you find model optimal answer
- But can track correct answer to see how often/far gold standard optimal answer remains in the beam

Beam search treebank parsers?

- Most people do bottom up parsing (CKY, shift-reduce parsing or a version of left-corner parsing)
- For treebank grammars, not much grammar constraint, so want to use data-driven constraints
- Adeel Raffel and Niki 1996 (invariant shift-reduce parser)
- Manning and Carpenter 1998 and Henderson 2000 left-corner parsers
- But top-down with rich conditioning is possible
- C.f. Brian Koob 2002
- Don’t actually want to store states as partial parses
- Store them as the last rule applied, with backpointers to the previous rules that built those configurations (and a probability)
- You get a linear time parser... but you may not find the best parses according to your model (things “fall off the beam”)

Search in modern lexicalized statistical parsers

- Klein and Manning (2003b) do optimal A* search
- Done in a restricted space of lexicalized PCFGs that “factors”, allowing very efficient A* search
- Collins (1999) exploits both the ideas of beams and agenda based parsing
  - He places a separate beam over each span, and then, roughly, does uniform cost search
- Charniak (2000) uses inadmissible heuristics to guide search
  - He uses very good (but inadmissible) heuristics – “best first search” – to find good parses quickly
  - Perhaps unsurprisingly this is the fastest of the 3.

Coarse-to-fine parsing

- Uses grammar projections to guide search
  - VP|VBR, VP|VBG, VP|VBN, ... → VP
  - VP|VBR|IN, VP|VBR|VBN, VP|VBR|VBP, ... → VP
- You can parse much more quickly with a simple grammar because the grammar constant is way smaller
- You restrict the search of the expensive refined model to explore only spans and/or spans with compatible labels that the simple grammar liked
- Very successfully used in several recent parsers
  - Charniak and Johnson (2005)
  - Petrov and Klein (2007)

Coarse-to-fine parsing: A visualization of the span posterior probabilities from Petrov and Klein 2007

Dependency parsing
**Dependency Grammar/Parsing**

- A sentence is parsed by relating each word to other words in the sentence which depend on it.
- The idea of dependency structure goes back a long way.
- To Brach's grammar (5th century BC).
- Constituency is a new-fangled invention.
- 20th-century invention (B. Wells, 1941).
- Modern dependency work often linked to work of L. Tauskine (1959).
- Dominant approach in “East” Russia, Czech Rep., China, ....
- Basic approach of “First American” grammarians.
- Among the earliest kinds of parsers in NLP, even in the US.
- David Hoyle, one of the founders of compositional linguistics, built early (1968) dependency parser (Hey, 1982).

**Dependency structure**

- Words are linked from dependent to head (e.g., “pink”.
- Warning! Some people do the arrows one way; some the other way
  (Institut has them point from head to dependent...).
- Usually add a type “SG” (even $S$) so every word is a dependent of
  precisely 1 other node.

**Relation between CFG to dependency parse**

- A dependency grammar has a notion of a head.
- Officially, CFGs don’t.
- But modern linguistic theory and all modern statistical parsers
  (Charniak, Collins, Stanford, ...) do, via hand-written phrasal “head rules”:
  - The head of a noun phrase is a noun/num/number/adj/....
  - The head of a verb phrase is a verb.
  - The head rules can be used to extract a dependency
    parse from a CFG parse (follow the heads).
- A phrase structure tree can be got from a dependency tree, but dependents are flat (no VP!)

**Propagating head words**

- Small set of rules propagate heads.

**Extracted structure**

NB. Not all dependencies shown here.

- Dependencies often untyped, or if they are, usually by a very different scheme than the
  non-terminals of phrase structure. But Collins (1996) types them using phrasal
  categories.

**Quiz question!**

- Which of the following is a dependency (with arrow pointing from dependent to head) in the following sentence?
  Retail sales drop in April cools afternoon market trading
  a) afternoon --> drop
  b) cools --> drop
  c) drop --> cools
  d) afternoon --> cools
Dependency Conditioning Preferences

Sources of information:
- bilateral dependencies
- distance of dependencies
- valency of heads (number of dependents)

A word’s dependents (adjuncts, arguments) tend to fall near it in the string.

Probabilistic dependency grammar: generative model

1. Start with left wall $S$.
2. Generate root $e_0$.
3. Generate left children $e_i, w_i, \ldots, w_r$ from the FSA $h_w$.
4. Generate right children $e_i, w_i, \ldots, w_r$ from the FSA $h_r$.
5. Recurse on each $w_i$ for $i$ in $(4, \ldots, r)$, sampling a (steps 2-4).
6. Return $e_i, w_i, e_r, \ldots, e_0$.

Naive Recognition/Parsing

Dependency Grammar Cubic Recognition/Parsing (Esner & Satta, 1999)

- Triangles: span over words, where tall side of triangle is the head, other side is dependent, and no non-head words expecting more dependents.
- Trapezoids: span over words, where larger side is head, smaller side is dependent, and smaller side is still looking for dependents on its side of the trapezoid.

Dependency Grammar Cubic Recognition/Parsing (Esner & Satta, 1999)

- A triangle is $O(n^2)$ with upper left (or right) subtree.
- One trapezoid per dependency.

Cubic Recognition/Parsing (Esner & Satta, 1999)

- Gives $O(n^3)$ dependency grammar parsing.
Evaluation of Dependency Parsing:
Simply use (labeled) dependency accuracy

Accuracy = \frac{\text{number of correct dependencies}}{\text{total number of dependencies}}
= \frac{5 \times \text{DET}}{2 \times \text{sandwich}} = \frac{10}{4} = 40% 

McDonald et al. (2005 ACL):
Online Large-Margin Training of Dependency Parsers

- Builds a discriminative dependency parser
- Can condition on rich features in that context
- Best-known recent dependency parser
- Lots of recent dependency parsing activity connected with CoNLL 2006/2007 shared task
- Doesn’t/can’t report constituent LPSJ, but evaluating dependencies correct:
  - Accuracy is similar to but a fraction below dependencies extracted from Collins.
  - 90.9% vs. 91.4% ... combining them gives 92.2% (all lengths)
  - Stanford parser on length up to 40:
    - Pure generative dependency model: 03.0%
    - Language-based parser: 03.0%

Extracting grammatical relations from statistical constituency parsers

McDonald et al. (2005 ACL):
Score of a parse is the sum of the scores of its dependencies
Each dependency is a linear function of features times weights
Feature weights are learned by MBA, an online large-margin algorithm
- But you could think of it as using a perceptron or nearest classifier
Features cover:
- Head and dependent word and POS separately
- Head and dependent word and POS bigram features
- Words between head and dependent
- Length and direction of dependency

Collapsing to facilitate semantic analysis
Bell, based in LA, makes and distributes electronic and computer products.

Collapsing to facilitate semantic analysis
Bell, based in LA, makes and distributes electronic and computer products.
Collapsing to facilitate semantic analysis

Dependency paths to identify IE relations like protein interaction

Discriminative Parsing

Classification problem
- Given a training set of i.i.d. samples \(T = ((X_1, Y), ..., (X_n, Y))\) of input and class variables from an unknown distribution \(P(X,Y)\), estimate a function that predicts the class from the input variables.
- The observed \(X\)'s are the sentences.
- The class \(Y\) of a sentence is its parse tree.
- The model has a large (infinite) space of classes, but we can still assign them probabilities.
- The way we can do this is by breaking whole parse trees into component parts.

1. Distribution-free methods
2. Probabilistic model methods

Motivating discriminative estimation

100
6
2

A training corpus of 108 (imperative) sentences.

Motivating discriminative estimation (2)

- In discriminative models, it is easy to incorporate different kinds of features.
  - Often just about anything that seems linguistically interesting.
- In generative models, it's often difficult, and the model suffers because of false independence assumptions.
- This ability to add informative features is the real power of discriminative models for NLP.
  - Can still do it for parsing, though it's trickier.
Discriminative Parsers

- Discriminative Dependency Parsing
  - Not as computationally hard (tiny grammar constant)
  - Explored considerably recently, e.g., by McDonald et al. 2005
  - Make parser action decisions discriminatively
    - E.g. with a shift-reduce parser
  - Dynamic-programmed Phrase Structure Parsing
    - Resource intensive! Most work on sentences of length <15
    - The need to be able to dynamic program limits the feature types you can use
  - Post-Processing: Parse reranking
    - Just work with output of k-best generative parser

Discriminative models

- Shift/reduce parser Ratinov (1998)
  - Learns a distribution \( P(T|S) \) of parse trees given sentences using the sequence of actions of a shift-reduce parser
    - \( P(T|S) = \prod P(A_i|A_{i-1}, \ldots, S) \)
  - Uses a maximum entropy model to learn conditional distribution of parse action given history
  - Suffers from independence assumptions that actions are independent of future observations as with CKN/MEMM
  - Higher parameter estimation cost to learn local maximum entropy models
  - Lower but still good accuracy: 80% - 87% labeled precision/recall

Discriminative dynamic-programmed parsers

- Taskar et al. (2004 EMNLP) show how to do joint
  discriminative SVM-style (“max margin”) parsing
  building a phrase structure tree also conditioned on
  words in \( O(n) \) time
  - In practice, totally impractically slow. Results were never demonstrated on sentences longer than 35 words
  - Turian et al. (2006 NIPS) do a decision-tree based discriminative parser
  - Finkel, et al. (2006 ACL) feature-based discriminative parser is just about practical.
    - Does dynamic programming discriminative parsing of long sentences (train and test on up to 48 word sentences)
    - \( \text{BILU F1} \)

Discriminative Models – Distribution Free
Re-ranking (Collins 2000)

- Represent sentence-parse tree pairs by a feature vector
  \( F(X, Y) \)
- Learn a linear ranking model with parameters
  using the boosting loss
  - 13% error reduction
  - Still very close in accuracy to generative model (Charniak 2000)

Charniak and Johnson (2005 ACL):
Coarse-to-fine k-best parsing and Maxent discriminative reranking

- Builds a maxent discriminative reranker over parses produced by (a slightly bugfixed and improved version of) Charniak (2000).
- Gets 50 best parses from Charniak (2000) parser
  - Doing this exploits the “coarse-to-fine” idea to heuristically find good candidates
- Maxent model for reranking uses heads, etc. as
  generative model, but also nice linguistic features:
    - Conjunction parallelism
    - Right branching preference
    - Heaviness (length) of constituents factored in
  - Gets 91.4\% LP/LR F1 (on all sentences! – to 80 words)