The Penn Treebank and Statistical Parsing

Taras Mychaskiw & Aaron Voelker

University of Waterloo

May 27th, 2015

Overview

1. Background
2. Penn Treebank
3. Probabilistic Context-Free Grammars
4. Maxiumum-Entropy Models
5. Semantic Parsing
6. Modern Statistical Parsing

Overview

Part of Speech Tags

- POS tagging is a process of marking each word in a corpus with a POS tag based on meaning, context, etc
- Initially, tagging was done by hand or by simple rules
- Over the last 20 years, more automated ways have been discovered, usually in the form of supervised learning
- Nine "categories" of tags (noun, verb...), corpora contained anywhere from 50-200 tags

Parse Tree

```
(S
  (NP
    (NNP  John)
  )
  (VP
    (VBZ  loves)
    (NP
      (NNP  Mary)
    )
  )
  (.  .)
)
```

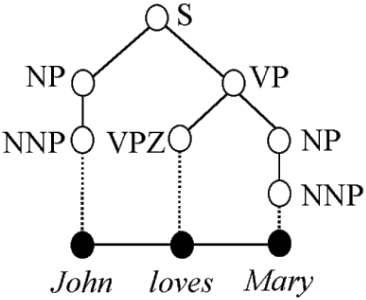


Figure 1: Example structure for *John loves Mary*.

Why Parse?

- Machine translation
- Information retrieval
  - Question-answering
  - Search
- Information extraction
  - Sentiment analysis
  - Text classification
  - Summarization

Brown Corpus

- Brown corpus was the first major POS tagged corpus available, developed by Kucera and Francis at Brown in the 60s
- Roughly 500 English works, 1 million words
- Used 87 different tags and allows compound tags
  - Such as *I'm* is PPSS+BEM for non-third person nominative pronoun and *am*
- Brown corpus is now dwarfed in size compared to modern corpora, which usually contain millions and millions of words

Context-Free Grammars

Context-Free Grammar

A context-free grammar (CFG) is a 4-tuple $G = (N, \Sigma, R, S)$ where:

N is a finite set of non-terminal symbols

Σ is the set of terminal symbols

R is the set of rules of the form $n \rightarrow \sigma$ where $n \in N$ and $\sigma \subset \Sigma \cup N$

$S \in N$ is the start symbol

Context-Free Grammars

CFG Example

$$
\begin{array}{rcl}
\text{S} & \rightarrow & \text{NP VP} \\
\text{NP} & \rightarrow & \text{NNP} \\
\text{NNP} & \rightarrow & \textit{John} \\
\vdots & &
\end{array}
$$

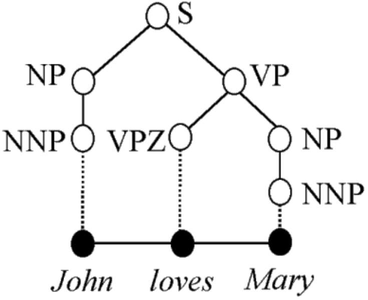


Figure 2: Example structure for *John loves Mary*.

Shift-Reduce Parsing

- Uses the "current" parse tree, the words of the sentence in a queue and partial parse trees in a stack
- Applies different state transitions until the queue is empty and the stack only contains the completed parse tree
- Possible transitions:
  - Shift: move a word from queue to the stack
  - Unary reduce: label on the top of the stack changes
  - Binary reduce: top 2 nodes on the stack are combined with a new label

Shift-Reduce Parsing

- Stanford parser uses a multiclass perceptron to determine the next transition
- POS tags are not assigned, but rather used as features
- Trained by iterating over the parse trees until "converged"
  - Start from base state and apply states until the actual tree can no longer be rebuilt
  - Once wrong, each transition's weights can be adjusted

CYK Algorithm

- Cocke–Younger–Kasami (CYK) algorithm is a parser for CFGs
- CYK is a dynamic programming algorithm with run time $\Theta(n^3|G|)$, where n is the sentence length
- Considers every possible consecutive subsequence of words $(i, \ldots, j)$ if the sequence can be generated by a rule r
- Does so for subsequences of length 1, 2, ...
- For length 2 and greater, also consider all possible partitions into two parts and check for rules that can lead to such a production

Classical Parsing

- Context-Free Grammars were used as symbolic parsing tools
- Did not scale well, many possible parses due to ambiguities
- Issues with this strategy:
  - Constrained grammars limit weird parses, but lead to many sentences having no parse
  - Less constrained grammars have a broad search space, simple sentences will end up with many possible parses
- Solution: need mechanism that finds the most likely parse out of all the possible parses, statistical parsing!

Classical Parsing

- Context-Free Grammars were used as symbolic parsing tools
- Did not scale well, many possible parses due to ambiguities
- Issues with this strategy:
  - Constrained grammars limit weird parses, but lead to many sentences having no parse
  - Less constrained grammars have a broad search space, simple sentences will end up with many possible parses
- Solution: need mechanism that finds the most likely parse out of all the possible parses, statistical parsing!
- But datas!

Overview

The Penn Treebank

- Several projects have extended the Brown corpus tagset
- These other projects include anywhere from 100 to 200 tags, the rationale being that more tags would lead to better classifications of words
- The Penn treebank consists of over 4.5 million words, but only 48 tags
- Their goal was to reduce redundancies by considering lexical and syntactic information
- Created by Marcus, Marcinkiewicz at U. Pennsylvania and Santorini at Northwestern
- Most recent release is 20 years old now and still requires a licensing fee and a cd-drive

Recoverability

- Brown corpus distinguishes five forms of verbs, VB (singular, past tense etc)
- The same paradigm is followed for *have* – although *have* is assigned it's own base tag, HV
- The Brown corpus also distinguishes 3 forms of *do* and 8 forms of *be*
- However, all these distinctions are lexically recoverable, so they are not included in the Penn treebank tagset

Consistency

- The Penn treebank also removed some inconsistent tags
- For example, *there* and *now* are always tagged as adverb, but *here* and *then* are tagged as adverb or as nominal adverb

Syntactic Function

- The Penn treebank encodes a word's syntactic function in the tag when possible
- For example, *one* is always labeled as a number by the Brown corpus, but it is labeled as a noun when appropriate in the Penn treebank
- Another example, the word *both* receives different tags depending on context, such as *the boys both* (postnominal), *both the boys* (prenominal) or *both of the boys* (noun phrase head)

Indeterminacy

- In some cases, annotators may simply not know how to tag a word
- To account for this, the treebank allows for disjunctions of tags
- Any combination of tags is allowed, however the vast majority of cases are restricted to a small set of two-tag options

POS Tagging

- The corpus is POS-tagged by an automated stage and a manual correction stage
- Initially used a stochastic algorithm called PARTS (Church, 1988), but now uses a “cascade of stochastic and rule-driven taggers”
- Manual correction stage uses output of automated stage
- Four annotators with graduate training in linguistics
- Experiment showed that accuracy and inter-annotator agreement rates were higher when correcting the automated output versus tagging from scratch

Bracketing

- Bracketing is a “skeletal syntactic structure” using an “impoverished flat context-free notation”

```
( (S
    (NP Battle-tested industrial managers here)
     always
    (VP buck
    up
    (NP nervous newcomers)
    ...))
```

- Basically just a relaxed parse tree for a CFG

Bracketing

- *Fidditch* is a deterministic parser developed at the University of Pennsylvania in 1983
- The parser produces a bunch of chunks that must be "glued" together manually
- Manual correction done on over half the corpus
- Special *null elements* included because they can be used to infer additional information like *predicate-argument structure*

Annotations

- Penn Treebank introduced the idea of an "annotated parse tree" to circumvent problems with ambiguity and/or consistency
- The X label is used whenever unsure of syntactic category
- Global ambiguity in determining the correct attachment point handled by "pseudo-attachment" notation

a boatload of warriors $\underbrace{\text{blown ashore 375 years ago}}_{\text{pseudo-attached}}$

Future of Penn Treebank

- In 1994, released paper on making the predicate-argument structure more explicit
- No other literature could be found
- Used as a training corpus for POS taggers and parsers
- Gold standard for evaluating new parsers
- Used especially within University of Pennsylvania
- Don't hear about it as much anymore, perhaps due to restricted availability, and many free alternatives [1]

[1] `http://en.wikipedia.org/wiki/Treebank#Syntactic_treebanks`

Overview

Motivating Example

Example from Ratnaparkhi (1999)

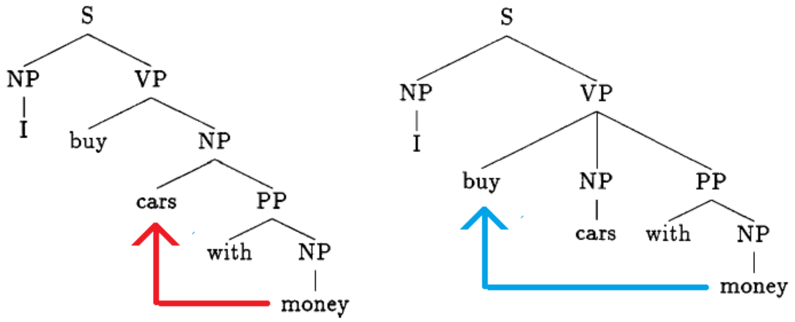


Figure 3: Unlikely parse (left); Likely parse (right)

- *Money* refers to *buy* (verb-phrase), not *cars* (noun-phrase)
- Both parses are legal, but we want the one that is more likely

Statistical Parsers

- Distinguishing the more likely parse requires semantic knowledge
- Or, for instance, the likelihood of someone buying a car *that contains* money versus the likelihood of someone buying a car *using* money
- Superior performance achieved using statistical parsers to learn a probabilistic context-free grammar from a treebank

Probabilistic Context-Free Grammars

- This section was adapted from course notes by Collins (2011)
- Key idea in probabilistic context-free grammars (PCFG) is to extend the definition of CFGs by assigning a probability $p(t)$ to each possible parse tree t that the grammar produces

$$p(t) \geq 0, \quad \sum_t p(t) = 1$$

- This seems an impossible task at a glance, the set of possible parse trees is large if not infinite
- Turns out there is a nice trick!

PCFG

- A PCFG consists of:
  - A CFG $G = (N, \Sigma, R, S)$
  - A parameter $q(\alpha \to \beta)$ for each rule $\alpha \to \beta \in R$
- $q(\alpha \to \beta)$ is the conditional probability of producing through the rule $\alpha \to \beta$ given the non-terminal being expanded is α
- Then, we can define $p(t)$ as

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \to \beta_i)$$

PCFG Training

- CFG parameters are taken from all the trees in the corpus
  - eg Σ is the set of all terminals found
- The maximum likelihood estimate of q is

$$q(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

where Count($\alpha \rightarrow \beta$) is the number of times the rule $\alpha \rightarrow \beta$ is seen and Count(α) is the number of times the non-terminal α is seen in the corpus

Parsing with PCFGs

- Given all possible parse trees T of a sentence, we can use the PCFG to find the most likely parse with

$$\arg\max_{t \in T} p(t)$$

- CYK algorithm can be extended for use with PCFGs
- Most likely parse is sometimes called the "Viterbi parse"
  - Viterbi algorithm finds most likely sequence of states in a hidden Markov model
  - The term became popular, and started being used simply to mean the "most likely parse"

Problems with PCFGs

- PCFGs exhibit two mean weaknesses:
  1. Lack of sensitivity to lexical information
  2. Lack of sensitivity to structure preferences

Lack of Sensitivity to Lexical Information

$p(t) =$
$q(S \rightarrow NP\ VP) \cdot q(NP \rightarrow NNP) \cdot$
$q(NPP \rightarrow John) \cdot q(VP \rightarrow VPZ\ NP) \cdot$
$q(VPZ \rightarrow loves) \cdot q(NP \rightarrow NNP) \cdot$
$q(NNP \rightarrow Mary)$

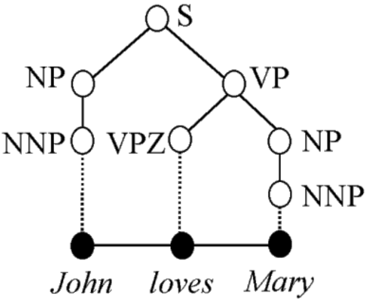


Figure 4: Example structure for *John loves Mary*.

Lack of Sensitivity to Lexical Information

$p(t) =$
$q(S \rightarrow NP\ VP) \cdot q(NP \rightarrow NNP) \cdot$
$q(NPP \rightarrow John) \cdot q(VP \rightarrow VPZ\ NP) \cdot$
$q(VPZ \rightarrow loves) \cdot q(NP \rightarrow NNP) \cdot$
$q(NNP \rightarrow Mary)$

- PCFG makes a strong independence assumption

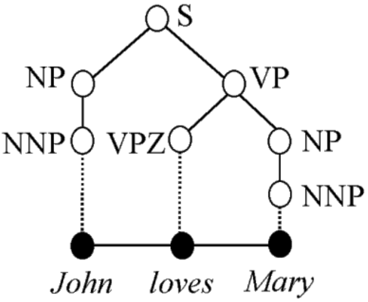
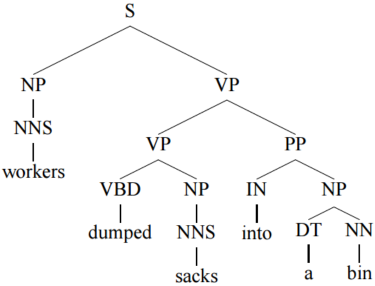
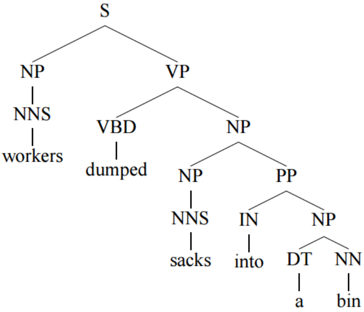


Figure 4: Example structure for *John loves Mary*.

Lack of Sensitivity to Lexical Information



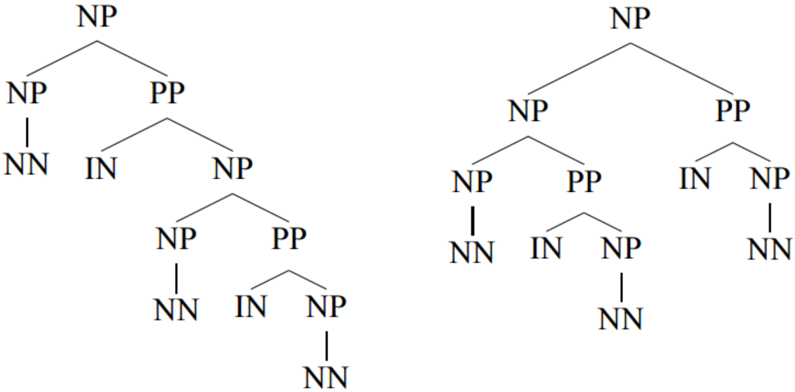


Lack of Sensitivity to Lexical Information

S	$\rightarrow$	NP VP		S	$\rightarrow$	NP VP
NP	$\rightarrow$	NNP		NP	$\rightarrow$	NNP
VP	$\rightarrow$	VBD NP		VP	$\rightarrow$	VBD NP
NP	$\rightarrow$	NNS		NP	$\rightarrow$	NNS
PP	$\rightarrow$	IN NP		PP	$\rightarrow$	IN NP
NP	$\rightarrow$	DT NN		NP	$\rightarrow$	DT NN
VP	$\rightarrow$	**VP PP**		**NP**	$\rightarrow$	**NP PP**

- Parse tree picked depends only on $q(\text{VP} \rightarrow \text{VP PP})$ and $q(\text{NP} \rightarrow \text{NP PP})$

Lack of Sensitivity to Structural Preferences



Lack of Sensitivity to Structural Preferences

- Searching the corpus, we find that the first structure is roughly twice as common as the second structure
- However, the PCFG assigns an identical probability to both trees since they have the same rules

Lexicalized PCFG

- Lexicalized PCFGs address the first of the two weaknesses
- Requires lexicalization of the corpus
- LPCFGs are essentially PCFGs were each non-terminal in each rule includes additional lexical information

$$\text{S} \rightarrow \text{NP VP}$$

becomes

$$\text{S(questioned)} \rightarrow \text{NP(lawyer) VP(questioned)}$$

- Otherwise, LPCFGs are the same as PCFGs
    - Larger set of non-terminals
    - Larger set of rules

Lexicalization of the Corpus

- For each context-free rule in the corpus, identify the “head” of of the rule
- Then, lexical information can be propagated bottom-up through the parse tree from the head to the parent

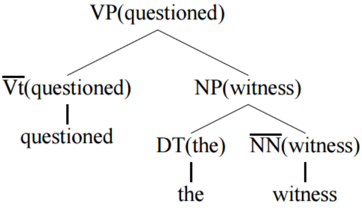


Figure 5: Lexicalized parse tree for *question the witness*.

Formal Definition

Lexicalized Probabilistic Context-Free Grammar

A lexicalized probabilistic context-free grammar (LPCFG) is a 6-tuple $G = (N, \Sigma, R, S, q, \gamma)$ where:

N is a finite set of non-terminal symbols

Σ is the set of terminal lexical items

R is the set of rules in one of the following forms:

$$X(h) \rightarrow_1 Y_1(h) Y_2(m), \quad X(h) \rightarrow_2 Y_1(m) Y_2(h), \quad X(h) \rightarrow h$$

where $X, Y_1, Y_2 \in N$ and $h, m \in \Sigma$

$S \in N$ is the start symbol

Formal Definition

Lexicalized Probabilistic Context-Free Grammar

For each $r \in R$, there is a $q(r) \geq 0$, and for any $X \in N, h \in \Sigma$

$$\sum_{LHS(r)=X(h)} q(r) = 1$$

For each $X \in N, h \in \Sigma$, there is a $\gamma(X, h) \geq 0$ and

$$\sum_{X,h} \gamma(X, h) = 1$$

Finally, the probability of a derivation of $r_1, \ldots, r_n$, $r_i \in R$, is

$$\gamma(LHS(r_1)) \times \prod_i q(r_i)$$

LPCFG Training

- The number of rules, and therefore parameters, is huge
- Make use of smoothing techniques to estimate each $q(r)$

$$q(\text{S(questioned)} \rightarrow_2 \text{NP(lawyer) VP(questioned)})$$
$$= P(R = \text{S} \rightarrow_2 \text{NP VP}, m = \text{lawyer} | X = S, h = \text{questioned})$$

- Use chain rule to decompose into two terms

$$= P(R = \text{S} \rightarrow_2 \text{NP VP} | X = S, h = \text{questioned})$$
$$\times P(m = \text{lawyer} | R = \text{S} \rightarrow_2 \text{NP VP}, X = S, h = \text{questioned})$$

LPCFG Training

- Find smoothed estimates for both terms
- Use MLE for $q(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | S, \mathrm{questioned})$ and $q(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | S)$ as derived from the corpus
- Then an estimate of the first term can be

$$a_1 \cdot q(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | S, \mathrm{questioned}) + b_1 \cdot q(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | S)$$

- A similar method can be used to estimate the second term
- Putting both together yields

$$(a_1 \cdot q(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | S, \mathrm{questioned}) + b_1 \cdot q(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | S)) \times$$
$$(a_2 \cdot q(\mathrm{lawyer} | \mathrm{S} \rightarrow_2 \mathrm{NP\ VP}, \mathrm{questioned}) + b_2 \cdot q(\mathrm{lawyer} | \mathrm{S} \rightarrow_2 \mathrm{NP\ VP}))$$

Parsing with LPCFGs

- Parsing strategy remains the same as for with PCFGs, an extended version of the CYK algorithm can be used to find the most likely parse tree
- Slower than PCFGs since grammar is much larger

Performance

- Collins (1996) compared lexicalized PCFGs to non-lexical PCFGs and showed that lexicalization in this fashion achieved significantly better parsing results; 85% compared to 75%
- Magerman (*Statistical Decision-Tree Models for Parsing*, 1995) showed similar results with a decision-tree learned parser
- Later, Collins (1997) improved his results with more sophisticated models to get 88% correct rates

Overview

History

- Ratnaparkhi (1999) provides the core model used by Charniak (2000)
- Interesting facts: Ratnaparkhi is also from the University of Pennsylvania, but Charniak is from Brown University

History

- Ratnaparkhi (1999) provides the core model used by Charniak (2000)
- Interesting facts: Ratnaparkhi is also from the University of Pennsylvania, but Charniak is from Brown University

"Charniak (1997) and Collins (1997) do not use general machine learning algorithms, but instead develop specialized statistical estimation techniques for their respective parsing tasks." Ratnaparkhi (1999)

- Idea of applying maximum entropy to NLP first proposed by Berger (1996) for machine translation
- Perhaps the first example of a general machine learning technique applied to parsing

Model Intuition

- Idea: *score* a parse by examining its derivation
- Derivations are built from the bottom up by adding *actions* to a list (similar to *shift-reduce* parsing)
- Every derivation corresponds to a unique parse tree (a complete parse T has exactly one derivation $d = \{a_1, \ldots, a_n\}$)
- Each action is scored using the maximum entropy approach
- Parsing is again just a standard search problem for the highest scoring parse

Maximum Entropy Framework

- Also referred to as a *log-linear* approach (a kind of *discriminative* model)
- The probability of taking an action a is conditioned on H, which includes some chosen *history* and *surrounding context* from the partial derivation

$$P(a|H) = \frac{1}{Z(H)} e^{\lambda_1 f_1(a,H) + \ldots + \lambda_m f_m(a,H)}$$

- $f_1, \ldots, f_m$ provide the scalar features of a, given H (fixed)
- λ_i are weights in $(-\infty, \infty)$ indicating the relative importance of feature i, depending on a and H (estimated)
- $Z(H)$ simply normalizes so that $\sum_a p(a|H) = 1$, for each H

Maximum Entropy Framework

- Advantages:
  - Model is easily changeable
  - Can incorporate arbitrarily diverse information
  - No independence assumption required for features
  - Smoothing comes for free
- Disadvantages:
  - Can be difficult to incorporate prior linguistic knowledge in a predictable manner
  - Feature selection is an "art"; evidence should have little noise (otherwise need smoothing)
- In other words, to design a parser, one only needs intuition about what evidence is "useful" to distinguish likely actions from unlikely actions in a derivation.

Alternative Representations

- Again, the model is $P(a|H) = \frac{1}{Z(H)} e^{\lambda_1 f_1(a,H) + \ldots + \lambda_m f_m(a,H)}$
- Taking the log of both sides,

$$\ln P(a|H) = \sum_{i=1}^{m} \lambda_i f_i(a, H) - \ln(Z(H))$$

gives a more familiar form (logistic regression).

Alternative Representations

- Again, the model is $P(a|H) = \frac{1}{Z(H)} e^{\lambda_1 f_1(a,H)+\ldots+\lambda_m f_m(a,H)}$
- Taking the log of both sides,

$$\ln P(a|H) = \sum_{i=1}^{m} \lambda_i f_i(a, H) - \ln(Z(H))$$

  gives a more familiar form (logistic regression).
- Let $g_0(a, H) = 1/Z(H)$, $g_i(a, H) = e^{\lambda_i f_i(a,H)}$, then

$$p(a|H) = g_0(a, H) \cdots g_m(a, H)$$

  is the form used by Charniak (2000) to connect each feature's contribution back to conditional probabilities.

Parameter Estimation

- *Generalized Iterative Scaling* algorithm (Darroch & Ratcliff, 1972)
- Let $\alpha_i = e^{\lambda_i}$, then

$$P(a|H) = \frac{1}{Z(H)} \prod_{i=1}^{m} \alpha_i^{f_i(a,H)}$$

- Introduce a "correction" feature $f_{m+1} = C - \sum_{i=1}^{m} f_i(a, H)$
- Iteratively update all α_i by computing the expectation of each f_i,

$$\sum_a P(a|H) f_i(a, H)$$

- Guaranteed to converge monotonically to the optimal solution

Maximum Likelihood Estimation

- Berger (1996) states that this algorithm gives the maximum likelihood estimate for the set of models under the given form

Maximum Likelihood Estimation

- Berger (1996) states that this algorithm gives the maximum likelihood estimate for the set of models under the given form

"The duality is appealing since as a maximum entropy model, it will not assume anything beyond the evidence, and as a maximum likelihood model, it will have a close fit to the observed data." Ratnaparkhi (1999)

Training and Searching

- Trained using the Penn Treebank
- Beam-search
- Achieves state of the art performance

Charniak's Model

- Uses a top-down generative parser to narrow down the search space to a set of candidate parses
- Then uses the methods from Ratnaparkhi (1999)
- However, instead of feature selecting only the common features, includes all features and smooths with deleted interpolation
- Also uses a less flexible formulation with a specific decomposition of probabilities, equivalent (?) to using 6 particular features and $Z(H) = 1$ (see eq. 7 from Charniak, 2000)

Overview

Semantic Parsing

- Often a traditional parse tree is not enough for an application
- Parse trees only reflect a language's grammar, which gives few clues as to the meaning of a sentence
- For instance, in question-answering tasks, it is much easier to work with *logical forms*
- The task of parsing to a logical form is called "semantic parsing"

Logical Forms

- A *logical form* can be thought of as a parse tree for a special grammar where the rules in the derivation encode logical relationships[2]
- Most often this is written using lambda calculus,

Example from Kwiatkowski et. al (2010)

Sentence: Which states border Texas?
Meaning: $\lambda\ x.state(x) \wedge next\_to(x, Texas)$

- In 2007, Collins went on to coauthor a paper titled *Online Learning of Relaxed CCG Grammars for Parsing to Logical Form.*

[2] Formalized as something called a *combinatory categorial grammar* (CCG)

Dependency-Based Compositional Semantics

- A hot topic in the field of semantic parsing is *Dependency-based Compositional Semantics* (DCS), first introduced by Liang et al (2011)
- Fun fact: Michael Jordan was a coauthor!
- Inspired by *Discourse Representation Theory* (DRT)
- They define a new semantic representation called DCS, which is a tree corresponding to a *latent* logical form, that is much less stringent than lambda calculus

Dependency-Based Compositional Semantics

- Interestingly, they use a discriminative log-linear feature model and beam-search, much like the Ratnaparkhi (1999) model discussed earlier
- Model trained using an EM-like algorithm and evaluated on a question-answer corpus
- This approach outperforms all existing semantic parsers, without even learning from annotated logical forms
- Success owed to using a flexible representation in a latent factor space

Overview

Charniak Extension

- Charniak and Johnson (*Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking*, 2005) build a discriminative ranker of parses produced by the maximum entropy parser
- Get top 50 parses, exploiting a "course-to-fine" heuristic they defined
- Improvements over the vanilla maximum entropy model, achieving about 91% on sentences up to 80 words

Stanford's Factored Parser

- *Fast Exact Inference with a Factored Model for Natural Language Parsing* (Klein and Manning, 2003)
- Version that is combined with a PCFG parser is available online for play: `http://nlp.stanford.edu:8080/parser/`
- Source code also available for download, including some more recent models

Neural Networks

- Compositional Vector Grammars (CVG)
  - *Parsing With Compositional Vector Grammars* (Socher et al., 2013)
  - Andrew Ng and Stanford
  - Combines PCFG with a recurrent neural network to learn a "syntactico-semantic" representation
  - Faster and more accurate than Stanford's factored parser
- Dependency Parser
  - *A Fast and Accurate Dependency Parser using Neural Networks* (Chen and Manning, 2014)
  - Dependency parsing establishes relationships between head words and the words that modify those heads
  - Their parser is transition based, where decisions are made by a neural network

Conclusions

- CFGs served as a decent foundation for parsing, but were fundamentally flawed
- Development of PCFGs "was one of the biggest breakthroughs in natural language processing in the 1990s"
- Modern parsers improved by use of discriminative techniques (maximum entropy)
- Stanford's newest neural network models are state of the art with 92% accuracy, fast, and freely available

Bioinformatics

- PCFGs are used to predict RNA structure
- Trained using a database of RNA structures in a similar fashion to using a treebank
- A maximum probability parse corresponds to a maximum probability RNA structure
- Can model long range interactions, pairwise structure and nested structures
- Pseudoknots (essentially loops in the structure) cannot be modeled however

Bioinformatics

- Generally a grammar is chosen by modeling RNA sequence structure[3]

$$\begin{aligned} S &\rightarrow LS \mid L \\ L &\rightarrow s \mid dFd \\ F &\rightarrow dFd \mid LS \end{aligned}$$

- Transition probabilities learned from training data
- Used to rank most likely structures
- *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids* (Durbin, 1998)

[3] `http://en.wikipedia.org/wiki/Stochastic_context-free_grammar#Building_a_PCFG_model`