Introduction to Natural Language Syntax and Parsing
Lecture 8: Parsing with CCG

Stephen Clark

October 23, 2015

Inducing a Grammar from CCGbank Since CCG is a lexicalised grammar, then the grammar can be induced from the treebank by effectively reading the lexicon off the leaves of the derivation trees, where the lexicon is a set of word-category pairs. So in the example on the slide, we would learn that *Marks* can have the category *NP*, for example. In addition to the lexicon, the grammar also consists of the manually-defined combinatory rules.

In practice, inducing a grammar is not quite as neat as the description suggests, primarily because of all the additional rules that have to be extracted from the derivations. These include the unary type-changing rules and punctuation rules we encountered in the previous lecture. There are also a number of additional rule instances in CCGbank which do not conform to the combinatory rules, since the conversion of the PTB to CCG was a semi-automatic process, which did not always produce "perfect" CCG derivations. Whether and how to include these additional rule instances is a design choice for the parser developer. If the goal is to score highly on a CCGbank evaluation, then it makes sense to include at least the most frequent of these additional rule instances in the grammar.

Chart Parsing with CCG The first stage in the CCG parsing pipeline is to assign lexical categories to the words in the input sentence. One way to achieve this is to simply assign all possible lexical categories to each word, as determined by the lexicon (with some strategy for dealing with unknown and rare words), and let the parser resolve the lexical category ambiguity. However, as we saw in the previous lecture, this would result in a huge number of categories being assigned to many frequent words.

Hence the usual practice is to use a sequence labelling algorithm (a tagger) to assign lexical categories to the words. The second bullet on the slide suggests using a standard maximum entropy tagger, which is the approach we took in the C&C parser. However, recent work by Mike Lewis and Wenduan Xu shows that a tagger based on a recurrent neural network is more accurate, and in particular is more robust to domain changes between the training and test data [3].

The chart-parsing algorithm itself is similar to the one we saw applied to dependency parsing. CCG is also a good fit with shift-reduce parsing architectures, and recent work by Yue Zhang and Wenduan Xu shows that it is possible to build a competitive shift-reduce parser for CCG [4].

CCG Supertagging Following Bangalore and Joshi's seminal work on tagging for lexicalised tree adjoining grammar, tagging when applied to lexicalised grammar formalisms is often referred to as *supertagging.* The examples on the slide are designed to demonstrate the difficulty of supertagging, compared to PTB POS tagging. The categories in blue are three different categories assigned to prepositions, which would all receive the *IN* PTB POS tag. Note that, for the two categories assigned to *with*, the supertagger is often making attachment decisions when choosing these categories. There is no choice in the second case, but in the first case *with* could attach to *the road* (in which case it would have the $(NP\backslash NP)/NP$ category).

A useful measure of the difficulty of a tagging task is to evaluate a simple baseline method which, given a word, assigns the tag most seen with that word in the training data. For the PTB POS tagging task, this baseline method has a per-word accuracy of around 90%; for CCG supertagging it's around 72%.

CCG Multitagging A maximum entropy tagger which assigns a single tag to each word has a per-word accuracy of around 92%, which may not sound so bad compared to the 72% baseline. However, bare in mind that for a dataset with sentences consisting of 20 to 25 words, this accuracy level equates to a couple of mistakes per sentence. Hence a better approach is to allow the tagger to assign multiple tags per word, using the probability distributions from the tagger as a measure of how confident the tagger is in its decisions. For words where there is little confidence, more tags can be assigned.

For a multitagger using the strategy described on the slide — which, as a measure of confidence, compares the probability of a tag for a word with the highest-probability tag for that word — the per-word accuracy is now over 97%. In addition, the lexical ambiguity level is still relatively low: only 1.4 categories per word.

CKY Algorithm A chart is a data structure consisting of cells (i, j), where i indicates the start of a constituent and j its span. Each cell contains sets of non-terminal symbols, in our case CCG categories. Charts enable efficient parsing by grouping together equivalent categories in the same cell. For a context-free grammar, non-terminals in the same cell with the same label are equivalent, and the parsing complexity is $O(n^3)$, where n is the length of the sentence. For a lexicalised context-free grammar, where the non-terminal label also has a linguistic head, the parsing complexity is $O(n^5)$ (as it was for the dependency parsing case).

The case for CCG is complicated somewhat, since there are variants of CCG, namely those with the generalised composition rules, where the CKY algorithm is not even polynomial: it's exponential. Vijay-Shankar and Weir developed a polynomial-time parsing algorithm for CCG, albeit with a relatively large exponent, in the early 90s, and this problem has recently been revived with the work of Kuhlmann and Satta [2]. The C&C parser deals with this issue by restricting the use of the combinatory rules, in particular by only allowing categories to combine if they have been seen to combine in the training data, effectively making the grammar context-free.

C&C also appends linguistic heads to the category labels, both for use in the statistical parsing model, and to enable the recovery of predicate-argument dependencies, as described in Clark and Curran (2007). Hence the grammar used in the C&C parser is effectively a lexicalised context-free grammar extracted from CCGbank. What this means for the CKY algorithm is that, for two categories to be equivalent, they must be in the same cell, have the same category label, and have the same linguistic head.

The algorithm itself traverses the cells in the chart, bottom up, filling in the cells with all possible combinations of contiguous categories from those that have already been built.

Chart Parsing The example on the slide, which uses a CFG, demonstrates how charts naturally lead to a *packed* structure. For the two VPs spanning words 2 to 7, only one needs to be considered when performing further parsing higher in the chart (in particular when filling the "corner" cell). This packing occurs in all cells, and explains how an exponential number of parse trees can be represented efficiently in a polynomial-sized structure.

Furthermore, as long as the features of the statistical parsing model are restricted to be local to the rule instantiations — i.e. they are not allowed to "look outside" of a single rule instance — then dynamic programming in the form of the Viterbi algorithm applied to trees can be used to find the highest-scoring tree. A PCFG naturally has this property, since the probabilities making up the probability of a tree are by definition restricted to single rule applications.

A useful intuition for Viterbi is the following: suppose that the VP on the left in the example has a higher score than the one on the right. Can we safely discard the one on the right when doing further parsing? As long as the features of the statistical parsing model are sufficiently local as described above, then there is no way that the VP on the right can "overtake" the one on the left in any subsequent parsing; the VP on the right cannot be part of the highest-scoring parse, and can therefore be discarded.

Linear Parsing Model This is the same linear parsing model that we've already seen for dependency parsing. The difference is that, for CCG, we're defining a model over derivations, d (for sentence S). The model is a global model, so the feature functions are integer-valued, counting the number of times that certain patterns are observed in the derivation tree. The features are fairly standard for a model of this type, picking out the category at the root of the derivation; the category-word pairs at the leaves; and the category triples (or pairs) defining a rule instantiation. Each one of the category-only features will have an additional version featuring the lexical head, and another version with the lexical head replaced by its POS tag (since the lexical features will be sparse). Finally, there is a set of features defined in terms of the word-word dependencies created by the derivation.

Training Data from CCGbank CCGbank has around 40,000 newspaper sentences with gold-standard derivation trees, together with a set of word-word dependencies for each derivation, from which the features will be extracted.

Feature Representation Mathematically each feature is an integer-valued function on a derivation. The number of features can be very large, especially because of the word-word features, leading to feature sets in the order of millions. Remarkably, simple regularisation techniques mean that the resulting models are not overfitted to the training data, despite the large number of features and relatively small number of training instances. For the preceptron, averaging all the weight vectors created during training, and using the average weight vector as the final model, is an effective regulariser.

Linear Parsing Model In Clark and Curran (2007) we defined a number of probabilistic parsing models for CCG. Here we'll use a (non-probabilistic) linear model, trained with the structured perceptron. The beauty of the perceptron is its simplicity, and competitiveness against more complicated alternatives.

Perceptron Training We've already seen perceptron training for dependency parsing. It works the same way here: take each training instance one at a time; find the highest-scoring derivation using the current set of weights (decoding); and then update the weights using that derivation and the gold-standard.

The example updates on the slides demonstrate this process nicely. First we start with a zero set of weights, and decode the first sentence in the training data, using chart-parsing with Viterbi. This will produce three types of feature: the ones in blue are the features which appear in the derivation returned by the parser and the gold standard; the ones in red are the ones returned by the parser, but not in the gold standard; and finally the ones in green are in the gold standard but not returned by the parser. Intuitively we'd like the parser to keep returning the blue features; stop returning the red features; and begin returning the green features. Hence we leave the weights of the blue features alone; the weights of the red features get decreased by one; and the weights of the green features increased by one. Then we move on to the next sentence and repeat. And we pass through the whole data set a number of times (around 10 passes will suffice for convergence).

DP vs. Beam Search There is a simple modification to the chart-parsing algorithm which can result in higher accuracies, without compromising speed. The idea is to retain only the K highest-scoring items in each cell (where K

is typically between 8 and 64). Here an item is a sub-derivation spanning the relevant part of the sentence. Hence there is no longer any packing, and the chart is simply being used to provide an order of combination of the constituents, and a means by which to compare items when applying the beam.

Since we're using heuristic beam search, the search is no longer optimal and not guaranteed to return the highest-scoring derivation. So why use it? The key point is that we are no longer confined to defining features which only cover a single rule instance. Since there is no longer the optimal sub-problem property required for dynamic programming, features can be defined over any part of the derivation. This increase in flexibility is enough to offset the lack of optimality, and increased accuracies can be obtained. The training process is still based on the perceptron, although a modified version — the so-called max-violation perceptron — has to be employed to accommodate the beam search.

A recent technical report describes some experiments using beam search, and can be obtained from my RESOURCES webpages, along with a new, Java version of the C&C parser.

Parser Evaluation Most of the CCG parsing evaluations use the CCG predicate-argument dependencies from CCGbank as the gold standard, adopting the standard split of sections 2-21 for grammar extraction and model training, 00 for development and 23 for testing. One problem with this approach is that it only allows the comparison of CCG parsers. What if we'd like to know how the C&C parser compares with the Stanford or Berkeley parsers?

A proposal was made in the late 90s (in fact originating from Cambridge and Sussex) to use generic grammatical relation dependencies as the gold standard, and compare all parsers against that. The idea was that a parser would only need some representation based on linguistic heads, and extracting the GRs would be straightforward. In fact it turned out to be a little more complicated than that [1], and the problem of how to effectively perform cross-formalism parser comparison is still unsolved. However, the idea of evaluating against GRs is still an attractive one.

Head-based GRs The examples on the slide use the Briscoe and Carroll (Cambridge-Sussex) GRs scheme. Other dependency schemes include the Stanford dependencies — similar to Briscoe and Carroll but more fine-grained — and the ongoing universal dependencies framework for dependency parsing.

Mapping CCG Dependencies to GRs It turned out that mapping from the native CCG dependencies to GRs was a non-trivial task, requiring a fairly complex set of rules with a number of exceptions [1]. An interesting research question which, as far as I know, has not been investigated, is to machine learn a mapping between the two representations.

Test Suite: DepBank DepBank contains 700 newspaper sentences manually annotated with GRs. Standard precision, recall and F-score measures can be used for evaluation.

Parsing Accuracy The overall accuracy number is probably a little below the state-of-the-art[1], although with the difficulties of mapping between formalisms it's difficult to know what the state-of-the-art number is. Beware also any comparisons of these numbers with, say, the accuracy numbers obtained in dependency parser evaluations. We showed convincingly in [1] that any such comparisons are not meaningful.

Perhaps the most interesting feature of the results is the accuracy breakdown per grammatical relation. Some GRs are easy to get: determiners and auiliaries, whereas others, such as coordination, are still very difficult. The classic disambiguation problems we saw in the first lecture are still there and unsolved, despite decades of research on the problem. Recent parsing models based on neural networks have produced another incremental improvement, but still not solved the problem.

Readings for Today

- Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. Stephen Clark and James R. Curran. *Computational Linguistics*, 33(4), 493-552.

References

[1] Stephen Clark and James R. Curran. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Meeting of the ACL*, pages 248–255, Prague, Czech Republic, 2007.

[2] Marco Kuhlmann and Giorgio Satta. A new parsing algorithm for combinatory categorial grammar. *Transactions of the Association for Computational Linguistics*, 2, 2014.

[3] Wenduan Xu, Michael Auli, and Stephen Clark. CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the ACL*, Beijing, China, 2015.

[4] Yue Zhang and Stephen Clark. Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the ACL*, Portland, OR, 2011.

[1]The new Java C&C parser will have higher accuracies, but hasn't yet been evaluated on GRs.