Introduction to Natural Language Syntax and Parsing
Lecture 4: The Perceptron Parsing Model

Stephen Clark

October 14, 2015

Edge-Based Linear Model A minimal feature set for a dependency parsing model would simply look at the words in an edge, and score the edge based on those words; for example, how likely is an edge from *runs* to *lion*? However, such a feature set would be extremely sparse, in the sense that very few of the large number of possible word pairs would appear in the training data (which is typically of the order of 1M words). Hence, in practice, dependency parsing models are much richer, in particular making extensive use of part-of-speech (POS) tags, which are less sparse than words. Features are defined over the edges themselves, in terms of both words and POS tags (and combinations of the two); but also in terms of the POS tags in between dependent words, and outside of dependent words.

Features in the MST Parser The table on the slide shows the feature set from McDonald's MST parser [2]. The basic unigram features effectively ask questions such as: *how likely is the word* runs *to be the parent of an edge? how likely is the POS tag* NN *to be the child of an edge?* The bigram features effectively ask questions such as: *how likely is the word* runs *with POS tag* VBZ *to be the parent of the word* lion*?* The in-between and surrounding POS features are designed to capture patterns of POS tags seen frequently between, and either side of, dependent words. For example, the VBD_PRP$_NN feature effectively asks the question: how likely is a dependency edge where the POS tag sequence between the two dependent words (inclusive) is VBD_PRP$_NN?

Global Linear Model In the edge-based linear model, the score for a whole tree is defined as the sum of the scores for each edge; and the score for an edge is a dot product between a *local* feature vector and a weight vector. The derivation on the slide shows that, because of the properties of sums of products, the score for a tree can be written as a dot product between a *global* feature vector and a weight vector (the same weight vector used in the local dot products).

The function f_k is an indicator function which takes an edge as argument and has the value 1 or 0, depending on whether the edge displays a particular

pattern. Here we overload f_k by also have it take a whole tree τ as an argument, and return a non-negative integer, so that it now *counts* the number of times the corresponding pattern appears in the whole tree. If we encode all the values of the $f_k(\tau)$ counting functions as a global feature vector $\mathbf{F}(\tau)$, then the score for a tree τ can be written as the dot product between the weight vector and the global feature vector.

Generic Online Learning The pseudocode for a generic online learning algorithm shows the simplicity of the idea: start with a zero weights vector, use the current weight vector to decode each training instance in turn, updating the weight vector at each instance. For dependency parsing, the training instances are pairs of sentences and gold-standard dependency trees, taken from a dependency bank.

The final line outputs *averaged* weights [1], as a method to avoid overfitting. So $\mathbf{v}$ is just an accumulation of all the weight vectors encountered during the training process (N passes over T training instances), and $\mathbf{w}$ is the averaged output.

The Perceptron Update The perceptron update uses the highest-scoring dependency tree $\mathbf{z}_t$ given the current weight vector $\mathbf{w}_{t-1}$; $\mathbf{F}(\mathbf{x}_t, \mathbf{z}_t)$ is the global feature vector for tree $\mathbf{z}_t$ (and sentence $\mathbf{x}_t$). The update to the current weight vector is simple: add the global feature vector for the gold-standard tree, and take away the global feature vector for the tree returned by the decoder. Note that the update is *passive*, meaning that no update takes place if the decoder returns the gold-standard tree.

A useful intuition for the perceptron update can be given in terms of the *local* indicator features. The local features for a training instance can be divided into three sets: 1) those that are in the gold standard and returned by the decoder; 2) those that are in the gold standard and not returned by the decoder; and 3) those that are returned by the decoder but not in the gold standard. For the features in 1), their weights remain the same (no update); for the features in 2), a value of 1 is added to their weights; and for the features in 3), a value of 1 is taken from their weights. Intuitively the update is attempting to force the decoder to return correct features, and prevent it from returning the incorrect ones.

CoNLL Shared Task Data The table on the slide is taken from [4]. The point of the table is to demonstrate the range of languages for which dependency parsers can be built and evaluated (Arabic, Basque, Catalan, Chinese, Czech, English, Greek, Hungarian, Italian, and Turkish). Note that the amount of training data varies considerably across languages, from 51,000 tokens for Basque to 447,000 tokens for English. Notice also the variation in the level of projectivity, from 0% of sentences for Chinese to over 30% for Turkish.

Graph-based vs. Transition-based The table, from [3], compares the accuracy of the graph-based MST parser with the transition-based Malt parser (for the 2006 CoNLL shared task data). Interestingly the accuracies are similar across languages, although what the paper shows is that the different parsing architectures do lead to different parsing errors, paving the way for a fruitful ensemble (or combination).

The table also shows how the accuracies for some languages are much higher than for others. Whilst this may suggest that some languages are intrinsically harder to parse than others, this conclusion needs to be reached with care, since a number of other factors, such as the size of the corresponding dependency banks, need to be taken into account.

State-of-the-Art (2015) The New York Google parsing team published a paper in 2015 with the best reported results on English dependency parsing [5]. The parsing algorithm is shift-reduce, the training method is the perceptron, but what is interesting about the parser is that a neural network is used to automatically extract the features. One dissatisfying aspect of previous dependency parsers is the huge number of features required for top performance – as many as 30M in some cases! The neural network-based parsers use similar feature templates, extracting similar information, but because the information is distributed across dense feature vectors, rather than "one-hot" vectors, the effective number of features is greatly reduced.

Any empirical result from industrial labs such as Google has to be interpreted in the context of the vast resources available in such places, which can be used to great effect in, for example, tuning hyperparameters. However, there are currently many papers being published showing the benefits of the distributed representations in neural networks (NNs), and parsing accuracies are likely to continue to increase for a few years yet, using NN approaches.

Accuracy League Table (2015) The current best-performing parsers on English newspaper data are transition-based parsers, but the difference is relatively small. It's possible that similar accuracuies could be achieved with graph-based parsers using neural network models. The numbers may look impressively high — almost 94% for unlabelled parsing, and over 92% for labelled parsing — but bear in mind these are aggregate scores across all dependency types, including the easy ones such as determiner-noun edges. Accuracies for some individual dependency types, such as coordination and prepositional phrase attachment, are still much lower, and the overall parsing problem is still far from being solved.

One interesting feature of the latest transition-based parsing results is how the accuracies vary by beam size. For the neural network parsers, the fully greedy approach with a beam size of 1 is not far behind the scores with larger beam sizes, and the best reported results are for a beam size as low as 8. Recall that the beam size has a direct impact on the speed of the parser, which is an important consideration in the context of a web-search company.

Readings for Today's Lecture The McDonald technical report is still the core reading, although the online update described there is more complicated than the simple perceptron update given in the lecture. For a description of the perceptron, see [1], which shows how to apply perceptron models to the POS tagging problem (although the same techniques can easily be adapted to dependency parsing).

References

[1] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA, 2002.

[2] Ryan McDonald, Koby Crammer, and Fernando Pereira. Spanning tree methods for discriminative training of dependency parsers. Technical report, University of Pennsylvania, 2005.

[3] Ryan McDonald and Joakim Nivre. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1), 2011.

[4] J. Nivre, J. Hall, S. Kubler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. The CoNLL 2007 shared task on dependency parsing. In *Conference on Empirical Methods in Natural Language Processing and Natural Language Learning*, 2007.

[5] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the ACL*, Beijing, China, 2015.