PBML

logo

The Prague Bulletin of Mathematical Linguistics

NUMBER ??? JULY 2014 1–4

Tree Transduction Tools for cdec

Austin Matthews[a], Paul Baltescu[b], Phil Blunsom[b], Alon Lavie[a], Chris Dyer[a]

[a] Carnegie Mellon University
[b] University of Oxford

Abstract

We describe a collection of open source tools for learning tree-to-string and tree-to-tree transducers and the extensions to the cdec decoder that enable translation with these. Our modular, easy-to-extned tools extract rules from trees or forests aligned to strings and trees subject to different structural constraints. A fast, multithreaded implementation of the Cohn and Blunsom (2009) model for extracting compact tree-to-string rules is also included. The implementation of the tree composition algorithm used by cdec is described, and translation quality and decoding time results are presented. Our results add to the body of evidence suggesting that tree transduers are a compelling option for translation, particularly when decoding speed and translation model size are important.

1. Bayesian Synchronous Tree to String Grammar Induction

In this section, we present an open source implementation of the synchronous tree-to-string grammar induction algorithm proposed by Cohn and Blunsom (2009) [1]. This algorithm relies on a Bayesian model which incorporates a prior preference for learning small, generalizable STSG rules. The model is designed to jointly learn translation rules and word alignments. This is important for capturing long distance reordering phenomena, which might otherwise be ignored if the rules are inferred using distance penalized alignments (e.g. as in the heuristic proposed by Galley et al. (2004)).

[1] Our code is publicly available here: `https://github.com/pauldb89/worm`

Corresponding author: `cdyer@cs.cmu.edu`
Cite as: Austin Matthews, Paul Baltescu, Phil Blunsom, Alon Lavie, Chris Dyer. Tree Transduction Tools for cdec. The Prague Bulletin of Mathematical Linguistics No. ???, 2014, pp. 1–4.
doi: HERE-WILL-BE-A-DOI-NUMBER

The model represents the tree-to-string grammar as a set of distributions $\{G_c\}$ over the productions of each non-terminal c. Each distribution G_c is assumed to be generated by a Dirichlet Process with a concentration parameter α_c and a base distribution $P_0(\cdot|c)$, i.e. $G_c \sim DP(\alpha_c, P_0(\cdot|c))$. The concentration parameter α_c controls the model's tendency towards reusing rules or creating new ones according to the base distribution and has a direct influence on the size of the resulting grammar. The base distribution is defined to assign probabilities to an infinite set of rules. The probabilities decrease exponentially as the sizes of the rules increase, biasing the model towards learning smaller rules.

Instead of representing the distributions G_c explicitly, we integrate over all the possible values of G_c. We obtain the following formula for estimating the probability of a rule r with root c, given a fixed set of derivations $\mathbf{r}$ for the training corpus:

$$p(r|\mathbf{r}, c; \alpha_c, P_0) = \frac{n_r + \alpha_c P_0(r \mid c)}{n_c + \alpha_c}, \tag{1}$$

where n_r is the number of times r occurs in $\mathbf{r}$ and n_c is the number of rules with root c in $\mathbf{r}$.

Cohn and Blunsom (2009) train their model using Gibbs sampling. To simplify the implementation, an alignment variable is defined for every internal node in the parsed corpus. An alignment variable specifies the interval of target words which are spanned by a source node. Alternatively, a node may not be aligned to any target words or may span a discontiguous group of words, in which case it is annotated with an empty interval. Non-empty alignment variables mark the substitution sites for the rules in a derivation of a parse tree. Overall, they are used to specify a set of sampled derivations $\mathbf{r}$ for the entire training data. Alignment spans are constrained to subsume the spans of their descendants and must be contained within the spans of their ancestors. In addition to this, sibling spans belonging to the frontier of the same rule must not overlap.

We implement Gibbs sampling with the help of two operators: `expand` and `swap`. The `expand` operator works by resampling a randomly selected alignment variable a, while keeping all the other alignment variables fixed. The set of possible outcomes consists of the empty interval and all the intervals assignable to a such that the previous conditions continue to hold. Each outcome is scored proportionally to the new rules it creates, using Equation 1, conditioned on all the rules in the training data that remain unaffected by the sampling operation. The `swap` operator randomly selects two frontier nodes labelled with non-terminals belonging to the same STSG rule and chooses to either swap their alignment variables or to leave them unchanged. The outcomes are weighted similarly to the previous case. The goal of the `swap` operator is to improve the sampler's ability to mix, especially in the context of improving word reordering, by providing a way to execute several low probability `expand` steps at once.

Our implementation of the grammar induction algorithm is written in `C++`. Compiling the code results in several binaries, including `sampler`, which implements our Gibbs sampler. The grammar induction tool expects the following as input:

- A file containing the parse trees for the source side of the parallel corpus, in the Penn Treebank format.
- A file containing the target side of the parallel corpus.
- A file containing the word alignments for the training data. The word alignments are needed in the heuristic (Galley et al., 2004) used to initialize the first set of derivations.
- Two files containing the translation tables $p(s|t)$ and $p(t|s)$. These probabilities are used in the base distribution to weight the rules according to the source and target words they contain.

The remaining input arguments (hyperparameters, rule restrictions, etc.) are initialized with sensible default values. Running the binary with the `--help` option will produce the complete list of arguments and a brief explanation for each. The tool produces several files as output, including one containing the set of rules together with their probabilities, computed based on the last set of sampled derivations. The documentation released with our code shows how to prepare the training data, run the tool and convert the output in the `cdec` format.

Our tool leverages the benefits of a multithreaded environment to speed up grammar induction. At every sampler iteration, each training sentence is dynamically allocated to one of the available threads. In our implementation, we use a hash-based implementation of a Chinese Restaurant Process (CRP) (Teh, 2010) to efficiently compute the rule probabilities given by Equation 1. The data structure is updated whenever one of the `expand` or `swap` operators is applied. To lock this data structure with every update would completely cancel the effect of parallelization, as all the basic operations performed by the sampler are dependent on the CRP. Instead, we distribute a copy of the CRP on every thread and synchronize the data structures at the end of each iteration. Although the CRPs diverge during an iteration through the training data, no negative effects are observed when inferring STSGs in multithreaded mode.

Bibliography

Cohn, Trevor and Phil Blunsom. A bayesian model of syntax-directed tree to string grammar induction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *HLT-NAACL*, 2004.

Teh, Yee Whye. Dirichlet process. In *Encyclopedia of Machine Learning*, pages 280–287. 2010.

Address for correspondence:
Chris Dyer

cdyer@cs.cmu.edu
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, United States