Spectral Learning for Non-Deterministic Dependency Parsing

Franco M. Luque
Universidad Nacional de Córdoba
and CONICET
Córdoba X5000HUA, Argentina
francolq@famaf.unc.edu.ar

Ariadna Quattoni and **Borja Balle** and **Xavier Carreras**
Universitat Politècnica de Catalunya
Barcelona E-08034
{aquattoni,bballe,carreras}@lsi.upc.edu

Abstract

In this paper we study spectral learning methods for non-deterministic split head-automata grammars, a powerful hidden-state formalism for dependency parsing. We present a learning algorithm that, like other spectral methods, is efficient and non-susceptible to local minima. We show how this algorithm can be formulated as a technique for inducing hidden structure from distributions computed by forward-backward recursions. Furthermore, we also present an inside-outside algorithm for the parsing model that runs in cubic time, hence maintaining the standard parsing costs for context-free grammars.

1 Introduction

Dependency structures of natural language sentences exhibit a significant amount of non-local phenomena. Historically, there have been two main approaches to model non-locality: (1) increasing the order of the factors of a dependency model (e.g. with sibling and grandparent relations (Eisner, 2000; McDonald and Pereira, 2006; Carreras, 2007; Martins et al., 2009; Koo and Collins, 2010)), and (2) using hidden states to pass information across factors (Matsuzaki et al., 2005; Petrov et al., 2006; Musillo and Merlo, 2008).

Higher-order models have the advantage that they are relatively easy to train, because estimating the parameters of the model can be expressed as a convex optimization. However, they have two main drawbacks. (1) The number of parameters grows significantly with the size of the factors, leading to potential data-sparsity problems. A solution to address the data-sparsity problem is to explicitly tell the model what properties of higher-order factors need to be remembered. This can be achieved by means of feature engineering, but compressing such information into a state of bounded size will typically be labor intensive, and will not generalize across languages. (2) Increasing the size of the factors generally results in polynomial increases in the parsing cost.

In principle, hidden variable models could solve some of the problems of feature engineering in higher-order factorizations, since they could automatically induce the information in a derivation history that should be passed across factors. Potentially, they would require less feature engineering since they can learn from an annotated corpus an optimal way to compress derivations into hidden states. For example, one line of work has added hidden annotations to the non-terminals of a phrase-structure grammar (Matsuzaki et al., 2005; Petrov et al., 2006; Musillo and Merlo, 2008), resulting in compact grammars that obtain parsing accuracies comparable to lexicalized grammars. A second line of work has modeled hidden sequential structure, like in our case, but using PDFA (Infante-Lopez and de Rijke, 2004). Finally, a third line of work has induced hidden structure from the history of actions of a parser (Titov and Henderson, 2007).

However, the main drawback of the hidden variable approach to parsing is that, to the best of our knowledge, there has not been any convex formulation of the learning problem. As a result, training a hidden-variable model is both expensive and prone to local minima issues.

In this paper we present a learning algorithm for hidden-state split head-automata grammars (SHAG) (Eisner and Satta, 1999). In this for-

malism, head-modifier sequences are generated by a collection of finite-state automata. In our case, the underlying machines are probabilistic non-deterministic finite state automata (PNFA), which we parameterize using the *operator model* representation. This representation allows the use of simple spectral algorithms for estimating the model parameters from data (Hsu et al., 2009; Bailly, 2011; Balle et al., 2012). In all previous work, the algorithms used to induce hidden structure require running repeated inference on training data—e.g. Expectation-Maximization (Dempster et al., 1977), or split-merge algorithms. In contrast, spectral methods are simple and very efficient —parameter estimation is reduced to computing some data statistics, performing SVD, and inverting matrices.

The main contributions of this paper are:

- We present a spectral learning algorithm for inducing PNFA with applications to head-automata dependency grammars. Our formulation is based on thinking about the distribution generated by a PNFA in terms of the forward-backward recursions.
- Spectral learning algorithms in previous work only use statistics of prefixes of sequences. In contrast, our algorithm is able to learn from substring statistics.
- We derive an inside-outside algorithm for non-deterministic SHAG that runs in cubic time, keeping the costs of CFG parsing.
- In experiments we show that adding non-determinism improves the accuracy of several baselines. When we compare our algorithm to EM we observe a reduction of two orders of magnitude in training time.

The paper is organized as follows. Next section describes the necessary background on SHAG and operator models. Section 3 introduces Operator SHAG for parsing, and presents a spectral learning algorithm. Section 4 presents a parsing algorithm. Section 5 presents experiments and analysis of results, and section 6 concludes.

2 Preliminaries

2.1 Head-Automata Dependency Grammars

In this work we use split head-automata grammars (SHAG) (Eisner and Satta, 1999; Eisner, 2000), a context-free grammatical formalism whose derivations are projective dependency trees. We will use $x_{i:j} = x_i x_{i+1} \cdots x_j$ to denote a sequence of symbols x_t with $i \leq t \leq j$. A SHAG generates sentences $s_{0:N}$, where symbols $s_t \in \mathcal{X}$ with $1 \leq t \leq N$ are regular words and $s_0 = \star \notin \mathcal{X}$ is a special root symbol. Let $\bar{\mathcal{X}} = \mathcal{X} \cup \{\star\}$. A derivation y, i.e. a dependency tree, is a collection of *head-modifier* sequences $\langle h, d, x_{1:T} \rangle$, where $h \in \bar{\mathcal{X}}$ is a word, $d \in \{\text{LEFT}, \text{RIGHT}\}$ is a direction, and $x_{1:T}$ is a sequence of T words, where each $x_t \in \mathcal{X}$ is a *modifier* of h in direction d. We say that h is the *head* of each x_t. Modifier sequences $x_{1:T}$ are ordered head-outwards, i.e. among $x_{1:T}$, x_1 is the word closest to h in the derived sentence, and x_T is the furthest. A derivation y of a sentence $s_{0:N}$ consists of a LEFT and a RIGHT head-modifier sequence for each s_t. As special cases, the LEFT sequence of the root symbol is always empty, while the RIGHT one consists of a single word corresponding to the head of the sentence. We denote by $\mathcal{Y}$ the set of all valid derivations.

Assume a derivation y contains $\langle h, \text{LEFT}, x_{1:T} \rangle$ and $\langle h, \text{RIGHT}, x'_{1:T'} \rangle$. Let $\mathcal{L}(y, h)$ be the derived sentence *headed* by h, which can be expressed as $\mathcal{L}(y, x_T) \cdots \mathcal{L}(y, x_1) \;\; h \;\; \mathcal{L}(y, x'_1) \cdots \mathcal{L}(y, x'_{T'})$.[1] The language generated by a SHAG are the strings $\mathcal{L}(y, \star)$ for any $y \in \mathcal{Y}$.

In this paper we use probabilistic versions of SHAG where probabilities of head-modifier sequences in a derivation are independent of each other:

$$\mathbb{P}(y) = \prod_{\langle h,d,x_{1:T} \rangle \in y} \mathbb{P}(x_{1:T}|h,d) \quad . \qquad (1)$$

In the literature, standard *arc-factored* models further assume that

$$\mathbb{P}(x_{1:T}|h,d) = \prod_{t=1}^{T+1} \mathbb{P}(x_t|h,d,\sigma_t) \quad , \qquad (2)$$

where x_{T+1} is always a special STOP word, and σ_t is the state of a deterministic automaton generating $x_{1:T+1}$. For example, setting $\sigma_1 = \text{FIRST}$ and $\sigma_{t>1} = \text{REST}$ corresponds to first-order models, while setting $\sigma_1 = \text{NULL}$ and $\sigma_{t>1} = x_{t-1}$ corresponds to sibling models (Eisner, 2000; McDonald et al., 2005; McDonald and Pereira, 2006).

[1] Throughout the paper we assume we can distinguish the words in a derivation, irrespective of whether two words at different positions correspond to the same symbol.

2.2 Operator Models

An operator model $\mathbf{A}$ with n states is a tuple $\langle \alpha_1, \alpha_\infty^\top, \{A_a\}_{a \in \mathcal{X}} \rangle$, where $A_a \in \mathbb{R}^{n \times n}$ is an *operator* matrix and $\alpha_1, \alpha_\infty \in \mathbb{R}^n$ are vectors. $\mathbf{A}$ computes a function $f : \mathcal{X}^* \to \mathbb{R}$ as follows:

$$f(x_{1:T}) = \alpha_\infty^\top \, A_{x_T} \, \cdots \, A_{x_1} \, \alpha_1 \quad . \qquad (3)$$

One intuitive way of understanding operator models is to consider the case where f computes a probability distribution over strings. Such a distribution can be described in two equivalent ways: by making some independence assumptions and providing the corresponding parameters, or by explaining the process used to compute f. This is akin to describing the distribution defined by an HMM in terms of a factorization and its corresponding transition and emission parameters, or using the inductive equations of the forward algorithm. The operator model representation takes the latter approach.

Operator models have had numerous applications. For example, they can be used as an alternative parameterization of the function computed by an HMM (Hsu et al., 2009). Consider an HMM with n hidden states and initial-state probabilities $\pi \in \mathbb{R}^n$, transition probabilities $T \in \mathbb{R}^{n \times n}$, and observation probabilities $O_a \in \mathbb{R}^{n \times n}$ for each $a \in \mathcal{X}$, with the following meaning:

- $\pi(i)$ is the probability of starting at state i,
- $T(i, j)$ is the probability of transitioning from state j to state i,
- O_a is a diagonal matrix, such that $O_a(i, i)$ is the probability of generating symbol a from state i.

Given an HMM, an equivalent operator model can be defined by setting $\alpha_1 = \pi$, $A_a = TO_a$ and $\alpha_\infty = \vec{1}$. To see this, let us show that the forward algorithm computes the expression in equation (3). Let σ_t denote the state of the HMM at time t. Consider a state-distribution vector $\alpha_t \in \mathbb{R}^n$, where $\alpha_t(i) = \mathbb{P}(x_{1:t-1}, \sigma_t = i)$. Initially $\alpha_1 = \pi$. At each step in the chain of products (3), $\alpha_{t+1} = A_{x_t} \, \alpha_t$ updates the state distribution from positions t to $t + 1$ by applying the appropriate operator, i.e. by emitting symbol x_t and transitioning to the new state distribution. The probability of $x_{1:T}$ is given by $\sum_i \alpha_{T+1}(i)$. Hence, $A_a(i, j)$ is the probability of generating symbol a and moving to state i given that we are at state j.

HMM are only one example of distributions that can be parameterized by operator models. In general, operator models can parameterize any PNFA, where the parameters of the model correspond to probabilities of emitting a symbol from a state *and* moving to the next state.

The advantage of working with operator models is that, under certain mild assumptions on the operator parameters, there exist algorithms that can estimate the operators from observable statistics of the input sequences. These algorithms are extremely efficient and are not susceptible to local minima issues. See (Hsu et al., 2009) for theoretical proofs of the learnability of HMM under the operator model representation.

In the following, we write $x = x_{i:j} \in \mathcal{X}^*$ to denote sequences of symbols, and use $A_{x_{i:j}}$ as a shorthand for $A_{x_j} \cdots A_{x_i}$. Also, for convenience we assume $\mathcal{X} = \{1, \ldots, l\}$, so that we can index vectors and matrices by symbols in $\mathcal{X}$.

3 Learning Operator SHAG

We will define a SHAG using a collection of operator models to compute probabilities. Assume that for each possible head h in the vocabulary $\bar{\mathcal{X}}$ and each direction $d \in \{\text{LEFT}, \text{RIGHT}\}$ we have an operator model that computes probabilities of modifier sequences as follows:

$$\mathbb{P}(x_{1:T} | h, d) = (\alpha_\infty^{h,d})^\top \, A_{x_T}^{h,d} \, \cdots \, A_{x_1}^{h,d} \, \alpha_1^{h,d} \quad .$$

Then, this collection of operator models defines an *operator SHAG* that assigns a probability to each $y \in \mathcal{Y}$ according to (1). To learn the model parameters, namely $\langle \alpha_1^{h,d}, \alpha_\infty^{h,d}, \{A_a^{h,d}\}_{a \in \mathcal{X}} \rangle$ for $h \in \bar{\mathcal{X}}$ and $d \in \{\text{LEFT}, \text{RIGHT}\}$, we use spectral learning methods based on the works of Hsu et al. (2009), Bailly (2011) and Balle et al. (2012).

The main challenge of learning an operator model is to infer a hidden-state space from observable quantities, i.e. quantities that can be computed from the distribution of sequences that we observe. As it turns out, we cannot recover the actual hidden-state space used by the operators we wish to learn. The key insight of the spectral learning method is that we can recover a hidden-state space that corresponds to a projection of the original hidden space. Such projected space is equivalent to the original one in the sense that we

can find operators in the projected space that parameterize the *same* probability distribution over sequences.

In the rest of this section we describe an algorithm for learning an operator model. We will assume a fixed head word and direction, and drop h and d from all terms. Hence, our goal is to learn the following distribution, parameterized by operators α_1, $\{A_a\}_{a\in\mathcal{X}}$, and α_∞:

$$\mathbb{P}(x_{1:T}) = \alpha_\infty^\top A_{x_T} \cdots A_{x_1} \alpha_1 \ . \quad (4)$$

Our algorithm shares many features with the previous spectral algorithms of Hsu et al. (2009) and Bailly (2011), though the derivation given here is based upon the general formulation of Balle et al. (2012). The main difference is that our algorithm is able to learn operator models from *substring* statistics, while algorithms in previous works were restricted to statistics on prefixes. In principle, our algorithm should extract much more information from a sample.

3.1 Preliminary Definitions

The spectral learning algorithm will use statistics estimated from samples of the target distribution. More specifically, consider the function that computes the expected number of occurrences of a substring x in a random string x' drawn from $\mathbb{P}$:

$$\begin{aligned} f(x) &= \mathbb{E}(x \sqsubseteq_\sharp x') \\ &= \sum_{x'\in\mathcal{X}^*} (x \sqsubseteq_\sharp x')\mathbb{P}(x') \\ &= \sum_{p,s\in\mathcal{X}^*} \mathbb{P}(pxs) \ , \end{aligned} \quad (5)$$

where $x \sqsubseteq_\sharp x'$ denotes the number of times x appears in x'. Here we assume that the true values of $f(x)$ for bigrams are known, though in practice the algorithm will work with empirical estimates of these.

The information about f known by the algorithm is organized in matrix form as follows. Let $P \in \mathbb{R}^{l\times l}$ be a matrix containing the value of $f(x)$ for all strings of length two, i.e. bigrams.[2]. That is, each entry in $P \in \mathbb{R}^{l\times l}$ contains the expected number of occurrences of a given bigram:

$$P(b,a) = \mathbb{E}(ab \sqsubseteq_\sharp x) \ . \quad (6)$$

[2]In fact, while we restrict ourselves to strings of length two, an analogous algorithm can be derived that considers longer strings to define P. See (Balle et al., 2012) for details.

Furthermore, for each $b \in \mathcal{X}$ let $P_b \in \mathbb{R}^{l\times l}$ denote the matrix whose entries are given by

$$P_b(c,a) = \mathbb{E}(abc \sqsubseteq_\sharp x) \ , \quad (7)$$

the expected number of occurrences of trigrams. Finally, we define vectors $p_1 \in \mathbb{R}^l$ and $p_\infty \in \mathbb{R}^l$ as follows: $p_1(a) = \sum_{s\in\mathcal{X}^*} \mathbb{P}(as)$, the probability that a string begins with a particular symbol; and $p_\infty(a) = \sum_{p\in\mathcal{X}^*} \mathbb{P}(pa)$, the probability that a string ends with a particular symbol.

Now we show a particularly useful way to express the quantities defined above in terms of the operators $\langle \alpha_1, \alpha_\infty^\top, \{A_a\}_{a\in\mathcal{X}}\rangle$ of $\mathbb{P}$. First, note that each entry of P can be written in this form:

$$\begin{aligned} P(b,a) &= \sum_{p,s\in\mathcal{X}^*} \mathbb{P}(pabs) \quad (8) \\ &= \sum_{p,s\in\mathcal{X}^*} \alpha_\infty^\top A_s A_b A_a A_p \alpha_1 \\ &= (\alpha_\infty^\top \sum_{s\in\mathcal{X}^*} A_s) A_b A_a (\sum_{p\in\mathcal{X}^*} A_p \alpha_1) \ . \end{aligned}$$

It is not hard to see that, since $\mathbb{P}$ is a probability distribution over $\mathcal{X}^*$, actually $\alpha_\infty^\top \sum_{s\in\mathcal{X}^*} A_s = \vec{1}^\top$. Furthermore, since $\sum_{p\in\mathcal{X}^*} A_p = \sum_{k\geq 0}(\sum_{a\in\mathcal{X}} A_a)^k = (I - \sum_{a\in\mathcal{X}} A_a)^{-1}$, we write $\tilde{\alpha}_1 = (I - \sum_{a\in\mathcal{X}} A_a)^{-1}\alpha_1$. From (8) it is natural to define a *forward* matrix $F \in \mathbb{R}^{n\times l}$ whose ath column contains the sum of all hidden-state vectors obtained after generating all prefixes ended in a:

$$F(:,a) = A_a \sum_{p\in\mathcal{X}^*} A_p \alpha_1 = A_a \tilde{\alpha}_1 \ . \quad (9)$$

Conversely, we also define a *backward* matrix $B \in \mathbb{R}^{l\times n}$ whose ath row contains the probability of generating a from any possible state:

$$B(a,:) = \alpha_\infty^\top \sum_{s\in\mathcal{X}^*} A_s A_a = \vec{1}^\top A_a \ . \quad (10)$$

By plugging the forward and backward matrices into (8) one obtains the factorization $P = BF$. With similar arguments it is easy to see that one also has $P_b = BA_bF$, $p_1 = B\ \alpha_1$, and $p_\infty^\top = \alpha_\infty^\top F$. Hence, if B and F were known, one could in principle invert these expressions in order to recover the operators of the model from empirical estimations computed from a sample. In the next section we show that in fact one does not need to know B and F to learn an operator model for $\mathbb{P}$, but rather that having a "good" factorization of P is enough.

3.2 Inducing a Hidden-State Space

We have shown that an operator model $\mathbf{A}$ computing $\mathbb{P}$ induces a factorization of the matrix P, namely $P = BF$. More generally, it turns out that when the rank of P equals the minimal number of states of an operator model that computes $\mathbb{P}$, then one can prove a duality relation between operators and factorizations of P. In particular, one can show that, for any rank factorization $P = QR$, the operators given by $\bar{\alpha}_1 = Q^+ p_1$, $\bar{\alpha}_\infty^\top = p_\infty^\top R^+$, and $\bar{A}_a = Q^+ P_a R^+$, yield an operator model for $\mathbb{P}$. A key fact in proving this result is that the function $\mathbb{P}$ is invariant to the basis chosen to represent operator matrices. See (Balle et al., 2012) for further details.

Thus, we can recover an operator model for $\mathbb{P}$ from any rank factorization of P, provided a rank assumption on P holds (which hereafter we assume to be the case). Since we only have access to an approximation of P, it seems reasonable to choose a factorization which is robust to estimation errors. A natural such choice is the thin SVD decomposition of P (i.e. using top n singular vectors), given by: $P = U(\Sigma V^\top) = U(U^\top P)$. Intuitively, we can think of U and $U^\top P$ as projected backward and forward matrices. Now that we have a factorization of P we can construct an operator model for $\mathbb{P}$ as follows: [3]

$$\bar{\alpha}_1 = U^\top p_1 \ , \tag{11}$$

$$\bar{\alpha}_\infty^\top = p_\infty^\top (U^\top P)^+ \ , \tag{12}$$

$$\bar{A}_a = U^\top P_a (U^\top P)^+ \ . \tag{13}$$

Algorithm 1 presents pseudo-code for an algorithm learning operators of a SHAG from training head-modifier sequences using this spectral method. Note that each operator model in the SHAG is learned separately. The running time of the algorithm is dominated by two computations. First, a pass over the training sequences to compute statistics over unigrams, bigrams and trigrams. Second, SVD and matrix operations for computing the operators, which run in time cubic in the number of symbols l. However, note that when dealing with sparse matrices many of these operations can be performed more efficiently.

Algorithm 1 Learn Operator SHAG

inputs:

- An alphabet $\mathcal{X}$
- A training set $\text{TRAIN} = \{\langle h^i, d^i, x^i_{1:T}\rangle\}_{i=1}^M$
- The number of hidden states n

for each $h \in \bar{\mathcal{X}}$ and $d \in \{\text{LEFT}, \text{RIGHT}\}$ **do**
  Compute an empirical estimate from TRAIN of statistics matrices $\widehat{p}_1$, $\widehat{p}_\infty$, $\widehat{P}$, and $\{\widehat{P}_a\}_{a\in\mathcal{X}}$
  Compute the SVD of $\widehat{P}$ and let $\widehat{U}$ be the matrix of top n left singular vectors of $\widehat{P}$
  Compute the observable operators for h and d:
    $\widehat{\alpha}_1^{h,d} = \widehat{U}^\top \widehat{p}_1$
    $(\widehat{\alpha}_\infty^{h,d})^\top = \widehat{p}_\infty^\top (\widehat{U}^\top \widehat{P})^+$
    $\widehat{A}_a^{h,d} = \widehat{U}^\top \widehat{P}_a (\widehat{U}^\top \widehat{P})^+$ for each $a \in \mathcal{X}$
end for
return Operators $\langle \widehat{\alpha}_1^{h,d}, \widehat{\alpha}_\infty^{h,d}, \widehat{A}_a^{h,d}\rangle$ for each $h \in \bar{\mathcal{X}}$, $d \in \{\text{LEFT}, \text{RIGHT}\}$, $a \in \mathcal{X}$

4 Parsing Algorithms

Given a sentence $s_{0:N}$ we would like to find its most likely derivation, $\hat{y} = \text{argmax}_{y\in\mathcal{Y}(s_{0:N})} \mathbb{P}(y)$. This problem, known as MAP inference, is known to be intractable for hidden-state structure prediction models, as it involves finding the most likely tree structure while summing out over hidden states. We use a common approximation to MAP based on first computing posterior marginals of tree edges (i.e. dependencies) and then maximizing over the tree structure (see (Park and Darwiche, 2004) for complexity of general MAP inference and approximations). For parsing, this strategy is sometimes known as MBR decoding; previous work has shown that empirically it gives good performance (Goodman, 1996; Clark and Curran, 2004; Titov and Henderson, 2006; Petrov and Klein, 2007). In our case, we use the non-deterministic SHAG to compute posterior marginals of dependencies. We first explain the general strategy of MBR decoding, and then present an algorithm to compute marginals.

[3] To see that equations (11-13) define a model for $\mathbb{P}$, one must first see that the matrix $M = F(\Sigma V^\top)^+$ is invertible with inverse $M^{-1} = U^\top B$. Using this and recalling that $p_1 = B\alpha_1$, $P_a = BA_aF$, $p_\infty^\top = \alpha_\infty^\top F$, one obtains that:

$$\begin{aligned}
\bar{\alpha}_1 &= U^\top B\alpha_1 = M^{-1}\alpha_1 \ , \\
\bar{\alpha}_\infty^\top &= \alpha_\infty^\top F(U^\top BF)^+ = \alpha_\infty^\top M \ , \\
\bar{A}_a &= U^\top BA_aF(U^\top BF)^+ = M^{-1}A_aM \ .
\end{aligned}$$

Finally:

$$\begin{aligned}
\mathbb{P}(x_{1:T}) &= \alpha_\infty^\top A_{x_T} \cdots A_{x_1} \alpha_1 \\
&= \alpha_\infty^\top MM^{-1}A_{x_T}M \cdots M^{-1}A_{x_1}MM^{-1}\alpha_1 \\
&= \bar{\alpha}_\infty^\top \bar{A}_{x_T} \cdots \bar{A}_{x_1}\bar{\alpha}_1
\end{aligned}$$

Let (s_i, s_j) denote a dependency between head word i and modifier word j. The posterior or *marginal probability* of a dependency (s_i, s_j) given a sentence $s_{0:N}$ is defined as

$$\mu_{i,j} = \mathbb{P}((s_i, s_j) \mid s_{0:N}) = \sum_{y \in \mathcal{Y}(s_{0:N}) \,:\, (s_i, s_j) \in y} \mathbb{P}(y) \ .$$

To compute marginals, the sum over derivations can be decomposed into a product of inside and outside quantities (Baker, 1979). Below we describe an inside-outside algorithm for our grammars. Given a sentence $s_{0:N}$ and marginal scores $\mu_{i,j}$, we compute the parse tree for $s_{0:N}$ as

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}(s_{0:N})} \sum_{(s_i, s_j) \in y} \log \mu_{i,j} \qquad (14)$$

using the standard projective parsing algorithm for arc-factored models (Eisner, 2000). Overall we use a two-pass parsing process, first to compute marginals and then to compute the best tree.

4.1 An Inside-Outside Algorithm

In this section we sketch an algorithm to compute marginal probabilities of dependencies. Our algorithm is an adaptation of the parsing algorithm for SHAG by Eisner and Satta (1999) to the case of non-deterministic head-automata, and has a runtime cost of $O(n^2 N^3)$, where n is the number of states of the model, and N is the length of the input sentence. Hence the algorithm maintains the standard cubic cost on the sentence length, while the quadratic cost on n is inherent to the computations defined by our model in Eq. (3). The main insight behind our extension is that, because the computations of our model involve state-distribution vectors, we need to extend the standard inside/outside quantities to be in the form of such state-distribution quantities.[4]

Throughout this section we assume a fixed sentence $s_{0:N}$. Let $\mathcal{Y}(x_{i:j})$ be the set of derivations that yield a subsequence $x_{i:j}$. For a derivation y, we use $\mathrm{root}(y)$ to indicate the root word of it, and use $(x_i, x_j) \in y$ to refer a dependency in y from head x_i to modifier x_j. Following Eisner and Satta (1999), we use decoding structures related to complete half-constituents (or "triangles", denoted C) and incomplete half-constituents (or "trapezoids", denoted I), each decorated with a direction (denoted L and R). We assume familiarity with their algorithm.

We define $\theta^{\mathrm{I,R}}_{i,j} \in \mathbb{R}^n$ as the inside score-vector of a right trapezoid dominated by dependency (s_i, s_j),

$$\theta^{\mathrm{I,R}}_{i,j} = \sum_{\substack{y \in \mathcal{Y}(s_{i:j}) \,:\, (s_i, s_j) \in y \ , \\ y = \{\langle s_i, \mathrm{R}, x_{1:t} \rangle\} \cup y' \ , \ x_t = s_j}} \mathbb{P}(y') \alpha^{s_i, \mathrm{R}}(x_{1:t}) \ . \qquad (15)$$

The term $\mathbb{P}(y')$ is the probability of head-modifier sequences in the range $s_{i:j}$ that do not involve s_i. The term $\alpha^{s_i, \mathrm{R}}(x_{1:t})$ is a *forward* state-distribution vector —the qth coordinate of the vector is the probability that s_i generates right modifiers $x_{1:t}$ and remains at state q. Similarly, we define $\phi^{\mathrm{I,R}}_{i,j} \in \mathbb{R}^n$ as the outside score-vector of a right trapezoid, as

$$\phi^{\mathrm{I,R}}_{i,j} = \sum_{\substack{y \in \mathcal{Y}(s_{0:i} s_{j:n}) \,:\, \mathrm{root}(y) = s_0, \\ y = \{\langle s_i, \mathrm{R}, x_{t:T} \rangle\} \cup y' \ , \ x_t = s_j}} \mathbb{P}(y') \beta^{s_i, \mathrm{R}}(x_{t+1:T}) \ , \qquad (16)$$

where $\beta^{s_i, \mathrm{R}}(x_{t+1:T}) \in \mathbb{R}^n$ is a *backward* state-distribution vector —the qth coordinate is the probability of being at state q of the right automaton of s_i and generating $x_{t+1:T}$. Analogous inside-outside expressions can be defined for the rest of structures (left/right triangles and trapezoids). With these quantities, we can compute marginals as

$$\mu_{i,j} = \begin{cases} (\phi^{\mathrm{I,R}}_{i,j})^\top \, \theta^{\mathrm{I,R}}_{i,j} \, Z^{-1} & \text{if } i < j \ , \\ (\phi^{\mathrm{I,L}}_{i,j})^\top \, \theta^{\mathrm{I,L}}_{i,j} \, Z^{-1} & \text{if } j < i \ , \end{cases} \qquad (17)$$

where $Z = \sum_{y \in \mathcal{Y}(s_{0:N})} \mathbb{P}(y) = (\alpha^{\star, \mathrm{R}}_\infty)^\top \, \theta^{\mathrm{C,R}}_{0,N}$.

Finally, we sketch the equations for computing inside scores in $O(N^3)$ time. The outside equations can be derived analogously (see (Paskin, 2001)). For $0 \leq i < j \leq N$:

$$\theta^{\mathrm{C,R}}_{i,i} = \alpha^{s_i, \mathrm{R}}_1 \qquad (18)$$

$$\theta^{\mathrm{C,R}}_{i,j} = \sum_{k=i+1}^{j} \theta^{\mathrm{I,R}}_{i,k} \left((\alpha^{s_k, \mathrm{R}}_\infty)^\top \, \theta^{\mathrm{C,R}}_{k,j} \right) \qquad (19)$$

$$\theta^{\mathrm{I,R}}_{i,j} = \sum_{k=i}^{j} A^{s_i, \mathrm{R}}_{s_j} \theta^{\mathrm{C,R}}_{i,k} \left((\alpha^{s_j, \mathrm{L}}_\infty)^\top \, \theta^{\mathrm{C,L}}_{k+1,j} \right) \qquad (20)$$

[4] Technically, when working with the projected operators the state-distribution vectors will not be distributions in the formal sense. However, they correspond to a projection of a state distribution, for some projection that we do not recover from data (namely M^{-1} in footnote 3). This projection has no effect on the computations because it cancels out.

5 Experiments

The goal of our experiments is to show that incorporating hidden states in a SHAG using operator models can consistently improve parsing accuracy. A second goal is to compare the spectral learning algorithm to EM, a standard learning method that also induces hidden states.

The first set of experiments involve fully unlexicalized models, i.e. parsing part-of-speech tag sequences. While this setting falls behind the state-of-the-art, it is nonetheless valid to analyze empirically the effect of incorporating hidden states via operator models, which results in large improvements. In a second set of experiments, we combine the unlexicalized hidden-state models with simple lexicalized models. Finally, we present some analysis of the automaton learned by the spectral algorithm to see the information that is captured in the hidden state space.

5.1 Fully Unlexicalized Grammars

We trained fully unlexicalized dependency grammars from dependency treebanks, that is, $\mathcal{X}$ are PoS tags and we parse PoS tag sequences. In all cases, our modifier sequences include special START and STOP symbols at the boundaries. [5] [6] We compare the following SHAG models:

- DET: a baseline deterministic grammar with a single state.
- DET+F: a deterministic grammar with two states, one emitting the first modifier of a sequence, and another emitting the rest (see (Eisner and Smith, 2010) for a similar deterministic baseline).
- SPECTRAL: a non-deterministic grammar with n hidden states trained with the spectral algorithm. n is a parameter of the model.
- EM: a non-deterministic grammar with n states trained with EM. Here, we estimate operators $\langle \widehat{\alpha}_1, \widehat{\alpha}_\infty, \widehat{A}_a^{h,d} \rangle$ using forward-backward for the E step. To initialize, we mimicked an HMM initialization: (1) we set $\widehat{\alpha}_1$ and $\widehat{\alpha}_\infty$ randomly; (2) we created a random transition matrix $T \in \mathbb{R}^{n \times n}$; (3) we created a diagonal matrix $O_a^{h,d} \in \mathbb{R}^{n \times n}$, where $O_a^{h,d}(i,i)$ is the probability of generating symbol a from h and d (estimated from training); (4) we set $\widehat{A}_a^{h,d} = TO_a^{h,d}$.

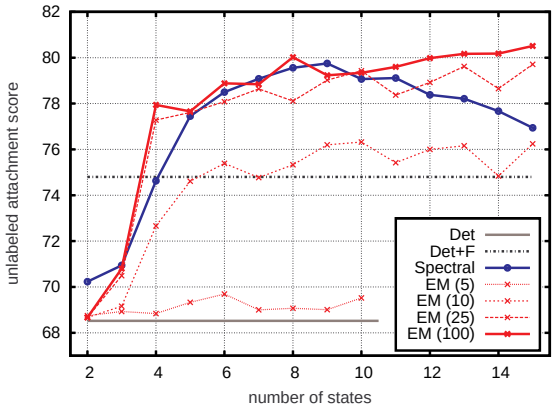

Figure 1: Accuracy curve on English development set for fully unlexicalized models.

We trained SHAG models using the standard WSJ sections of the English Penn Treebank (Marcus et al., 1994). Figure 1 shows the Unlabeled Attachment Score (UAS) curve on the development set, in terms of the number of hidden states for the spectral and EM models. We can see that DET+F largely outperforms DET[7], while the hidden-state models obtain much larger improvements. For the EM model, we show the accuracy curve after 5, 10, 25 and 100 iterations.[8]

In terms of peak accuracies, EM gives a slightly better result than the spectral method (80.51% for EM with 15 states versus 79.75% for the spectral method with 9 states). However, the spectral algorithm is much faster to train. With our Matlab implementation, it took about 30 seconds, while *each iteration* of EM took from 2 to 3 minutes, depending on the number of states. To give a concrete example, to reach an accuracy close to 80%, there is a factor of 150 between the training times of the spectral method and EM (where we compare the peak performance of the spectral method versus EM at 25 iterations with 13 states).

[5] Even though the operators α_1 and α_∞ of a PNFA account for start and stop probabilities, in preliminary experiments we found that having explicit START and STOP symbols results in more accurate models.

[6] Note that, for parsing, the operators for the START and STOP symbols can be packed into α_1 and α_∞ respectively. One just defines $\alpha'_1 = A_{\text{START}}\ \alpha_1$ and $\alpha'^{\top}_\infty = \alpha^{\top}_\infty\ A_{\text{STOP}}$.

[7] For parsing with deterministic SHAG we employ MBR inference, even though Viterbi inference can be performed exactly. In experiments on development data DET improved from 62.65% using Viterbi to 68.52% using MBR, and DET+F improved from 72.72% to 74.80%.

[8] We ran EM 10 times under different initial conditions and selected the run that gave the best absolute accuracy after 100 iterations. We did not observe significant differences between the runs.

	DET	DET+F	SPECTRAL	EM
WSJ	69.45%	75.91%	80.44%	81.68%

Table 1: Unlabeled Attachment Score of fully unlexicalized models on the WSJ test set.

Table 1 shows results on WSJ test data, selecting the models that obtain peak performances in development. We observe the same behavior: hidden-states largely improve over deterministic baselines, and EM obtains a slight improvement over the spectral algorithm. Comparing to previous work on parsing WSJ PoS sequences, Eisner and Smith (2010) obtained an accuracy of 75.6% using a deterministic SHAG that uses information about dependency lengths. However, they used Viterbi inference, which we found to perform worse than MBR inference (see footnote 7).

5.2 Experiments with Lexicalized Grammars

We now turn to combining lexicalized deterministic grammars with the unlexicalized grammars obtained in the previous experiment using the spectral algorithm. The goal behind this experiment is to show that the information captured in hidden states is complimentary to head-modifier lexical preferences.

In this case $\mathcal{X}$ consists of lexical items, and we assume access to the PoS tag of each lexical item. We will denote as t_a and w_a the PoS tag and word of a symbol $a \in \bar{\mathcal{X}}$. We will estimate conditional distributions $\mathbb{P}(a \mid h, d, \sigma)$, where $a \in \mathcal{X}$ is a modifier, $h \in \bar{\mathcal{X}}$ is a head, d is a direction, and σ is a deterministic state. Following Collins (1999), we use three configurations of deterministic states:

- LEX: a single state.
- LEX+F: two distinct states for first modifier and rest of modifiers.
- LEX+FCP: four distinct states, encoding: first modifier, previous modifier was a coordination, previous modifier was punctuation, and previous modifier was some other word.

To estimate $\mathbb{P}$ we use a back-off strategy:

$$\mathbb{P}(a|h,d,\sigma) = \mathbb{P}_A(t_a|h,d,\sigma)\mathbb{P}_B(w_a|t_a,h,d,\delta)$$

To estimate $\mathbb{P}_A$ we use two back-off levels, the fine level conditions on $\{w_h, d, \sigma\}$ and the coarse level conditions on $\{t_h, d, \sigma\}$. For $\mathbb{P}_B$ we use three levels, which from fine to coarse are $\{t_a, w_h, d, \sigma\}$, $\{t_a, t_h, d, \sigma\}$ and $\{t_a\}$. We follow Collins (1999) to estimate $\mathbb{P}_A$ and $\mathbb{P}_B$ from a treebank using a back-off strategy.

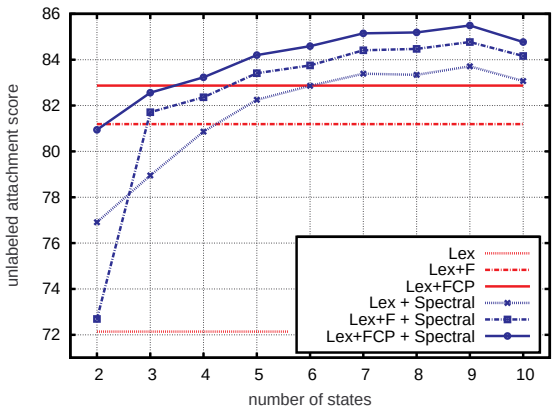

Figure 2: Accuracy curve on English development set for lexicalized models.

We use a simple approach to combine lexical models with the unlexical hidden-state models we obtained in the previous experiment. Namely, we use a log-linear model that computes scores for head-modifier sequences as

$$\begin{aligned} \mathrm{s}(\langle h, d, x_{1:T}\rangle) = \; & \log \mathbb{P}_{\mathrm{sp}}(x_{1:T}|h,d) \\ & + \log \mathbb{P}_{\mathrm{det}}(x_{1:T}|h,d) \quad , \end{aligned} \tag{21}$$

where $\mathbb{P}_{\mathrm{sp}}$ and $\mathbb{P}_{\mathrm{det}}$ are respectively spectral and deterministic probabilistic models. We tested combinations of each deterministic model with the spectral unlexicalized model using different number of states. Figure 2 shows the accuracies of single deterministic models, together with combinations using different number of states. In all cases, the combinations largely improve over the purely deterministic lexical counterparts, suggesting that the information encoded in hidden states is complementary to lexical preferences.

5.3 Results Analysis

We conclude the experiments by analyzing the state space learned by the spectral algorithm. Consider the space $\mathbb{R}^n$ where the forward-state vectors lie. Generating a modifier sequence corresponds to a path through the n-dimensional state space. We clustered sets of forward-state vectors in order to create a DFA that we can use to visualize the phenomena captured by the state space.

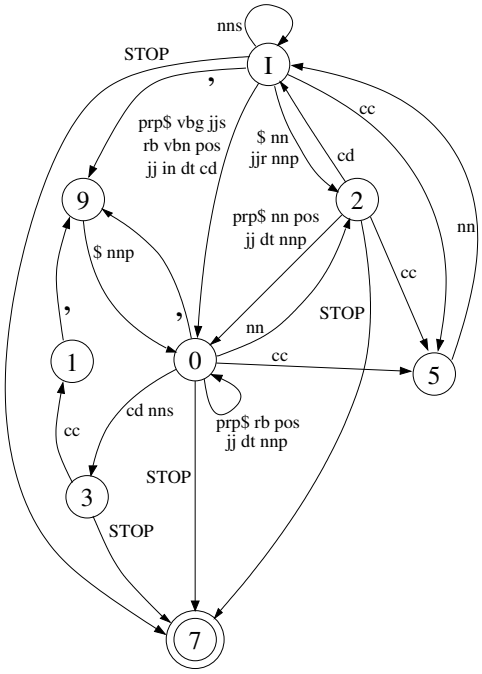

Figure 3: DFA approximation for the generation of NN left modifier sequences.

To build a DFA, we computed the forward vectors corresponding to frequent prefixes of modifier sequences of the development set. Then, we clustered these vectors using a Group Average Agglomerative algorithm using the cosine similarity measure (Manning et al., 2008). This similarity measure is appropriate because it compares the angle between vectors, and is not affected by their magnitude (the magnitude of forward vectors decreases with the number of modifiers generated). Each cluster i defines a state in the DFA, and we say that a sequence $x_{1:t}$ is in state i if its corresponding forward vector at time t is in cluster i. Then, transitions in the DFA are defined using a procedure that looks at how sequences traverse the states. If a sequence $x_{1:t}$ is at state i at time $t-1$, and goes to state j at time t, then we define a transition from state i to state j with label x_t. This procedure may require merging states to give a consistent DFA, because different sequences may define different transitions for the same states and modifiers. After doing a merge, new merges may be required, so the procedure must be repeated until a DFA is obtained.

For this analysis, we took the spectral model with 9 states, and built DFA from the non-deterministic automata corresponding to heads and directions where we saw largest improvements in accuracy with respect to the baselines.

A DFA for the automaton $(\texttt{NN}, \textsc{left})$ is shown in Figure 3. The vectors were originally divided in ten clusters, but the DFA construction required two state mergings, leading to a eight state automaton. The state named I is the initial state. Clearly, we can see that there are special states for punctuation (state 9) and coordination (states 1 and 5). States 0 and 2 are harder to interpret. To understand them better, we computed an estimation of the probabilities of the transitions, by counting the number of times each of them is used. We found that our estimation of generating `STOP` from state 0 is 0.67, and from state 2 it is 0.15. Interestingly, state 2 can transition to state 0 generating `prp$`, `POS` or `DT`, that are usual endings of modifier sequences for nouns (recall that modifiers are generated head-outwards, so for a left automaton the final modifier is the left-most modifier in the sentence).

6 Conclusion

Our main contribution is a basic tool for inducing sequential hidden structure in dependency grammars. Most of the recent work in dependency parsing has explored explicit feature engineering. In part, this may be attributed to the high cost of using tools such as EM to induce representations. Our experiments have shown that adding hidden-structure improves parsing accuracy, and that our spectral algorithm is highly scalable.

Our methods may be used to enrich the representational power of more sophisticated dependency models. For example, future work should consider enhancing lexicalized dependency grammars with hidden states that summarize lexical dependencies. Another line for future research should extend the learning algorithm to be able to capture vertical hidden relations in the dependency tree, in addition to sequential relations.

Acknowledgements We are grateful to Gabriele Musillo and the anonymous reviewers for providing us with helpful comments. This work was supported by a Google Research Award and by the European Commission (PASCAL2 NoE FP7-216886, XLike STREP FP7-288342). Borja Balle was supported by an FPU fellowship (AP2008-02064) of the Spanish Ministry of Education. The Spanish Ministry of Science and Innovation supported Ariadna Quattoni (JCI-2009-04240) and Xavier Carreras (RYC-2008-02223 and "KNOW2" TIN2009-14715-C04-04).

References

Raphael Bailly. 2011. Quadratic weighted automata: Spectral algorithm and likelihood maximization. *JMLR Workshop and Conference Proceedings – ACML*.

James K. Baker. 1979. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. 2012. Local loss optimization in operator models: A new insight into spectral learning. Technical Report LSI-12-5-R, Departament de Llenguatges i Sistemes Informàtics (LSI), Universitat Politècnica de Catalunya (UPC).

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic, June. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2004. Parsing the wsj using ccg and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 103–110, Barcelona, Spain, July.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society, Series B*, 39(1):1–38.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464, University of Maryland, June.

Jason Eisner and Noah A. Smith. 2010. Favor short dependencies: Parsing with soft and hard constraints on dependency length. In Harry Bunt, Paola Merlo, and Joakim Nivre, editors, *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, chapter 8, pages 121–150. Springer.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers, October.

Joshua Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 177–183, Santa Cruz, California, USA, June. Association for Computational Linguistics.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. 2009. A spectral algorithm for learning hidden markov models. In *COLT 2009 - The 22nd Conference on Learning Theory*.

Gabriel Infante-Lopez and Maarten de Rijke. 2004. Alternative approaches for generating bodies of grammar rules. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 454–461, Barcelona, Spain, July.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July. Association for Computational Linguistics.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, first edition, July.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19.

Andre Martins, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec, Singapore, August. Association for Computational Linguistics.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 75–82, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Gabriele Antonio Musillo and Paola Merlo. 2008. Unlexicalised hidden variable models of split dependency grammars. In *Proceedings of ACL-08: HLT, Short Papers*, pages 213–216, Columbus, Ohio, June. Association for Computational Linguistics.

James D. Park and Adnan Darwiche. 2004. Complexity results and approximation strategies for map

explanations. *Journal of Artificial Intelligence Research*, 21:101–133.

Mark Paskin. 2001. Cubic-time parsing and learning algorithms for grammatical bigram models. Technical Report UCB/CSD-01-1148, University of California, Berkeley.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July. Association for Computational Linguistics.

Ivan Titov and James Henderson. 2006. Loss minimization in parse reranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 560–567, Sydney, Australia, July. Association for Computational Linguistics.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155, Prague, Czech Republic, June. Association for Computational Linguistics.