State-Splitting for Regular Tree Grammars

Toni Dietze

Institute of Theoretical Computer Science, Technische Universität Dresden

Abstract. We formalize the state-splitting algorithm of [2].

1 Introduction

The expectation-maximization algorithm (EM algorithm) [1] is a well-known procedure, which can be used, e.g., to estimate probabilities for the rules of a regular tree grammar (RTG) G on the basis of a corpus c of derived trees. The algorithm starts with an arbitrary probability assignment p_0 and iterates an expectation and a maximization step; in each iteration the likelihood of the corpus c under the current probabilistic RTG (PRTG) increases or stays the same. We denote the resulting PRTG by $\mathrm{EM}((G, p_0), c)$.

In [2], multiple calls to EM were interleaved with modifications of the underlying RTG (cf. SPLIT, MERGE in Fig. 1). This may improve the likelihood, because the state behavior of the current RTG may be adapted to characteristics of the trees in c. In [2], this approach was presented in an informal style, and no properties were proved. In our current work, we have formalized this approach and call it state-splitting algorithm.

2 Preliminaries

Let Σ be an alphabet. We denote the set of all trees over Σ by T_Σ. A *regular tree grammar (RTG)* is a tuple $(Q, \Sigma, q_\mathrm{s}, R)$ where Q and Σ are alphabets of *states* and *terminal symbols*, respectively, with $Q \cap \Sigma = \emptyset$, $q_\mathrm{s} \in Q$ is the *initial state*, and R is a finite set of *rules* of the form $q_0 \to \sigma(q_1, \ldots, q_n)$ where $\sigma \in \Sigma$, $n \geq 0$, and $q_i \in Q$. We denote the left-hand side of a rule $r \in R$ by $\mathrm{lhs}(r)$. A *probabilistic RTG (PRTG)* is a pair (G, p) where $G = (Q, \Sigma, q_\mathrm{s}, R)$ is an RTG and $p\colon R \to [0, 1]$ is a probability assignment which is *proper*, i.e., the probabilities of rules with the same left-hand side sum up to 1. A *corpus* is a mapping $c\colon \mathrm{T}_\Sigma \to \mathbb{R}_{\geq 0}$ such that $\{a \in A \mid c(A) > 0\}$ is finite. The *likelihood of c under* (G, p) is defined as $\mathrm{L}(c \mid (G, p)) = \prod_{t \in \mathrm{T}_\Sigma} p(t)^{c(t)}$.

3 The State-Splitting Algorithm

The state-splitting algorithm (see Fig. 1) uses the functions SPLIT and MERGE.

```
Input: alphabet $\Sigma$, corpus $c$ over $\mathrm{T}_\Sigma$,
PRTG $G_0$ over $\Sigma$ with initial state $q_\mathrm{s}$ and
$\mathrm{L}(c \mid G_0) > 0$, $\mu \in [0,1]$.
Output: sequence of PRTG.
 1: for $i \leftarrow 1, 2, \ldots$ do
 2:     $G' \leftarrow \mathrm{EM}(\textsc{split}(G_{i-1}), c)$
 3:     $G_i \leftarrow \mathrm{EM}(\textsc{merge}(G'), c)$
 4: function $\textsc{split}(G)$
 5:     $\pi \leftarrow$ $G$-splitter splitting every state
            $q$ in $G$ into $q^1$ and $q^2$ except $q_\mathrm{s}$
 6:     return a proper $\pi$-split of $G$
 7: function $\textsc{merge}(G')$
 8:     $\pi \leftarrow$ identity mapping
 9:     for all states $q$ s.t. $q^1$, $q^2$ in $G'$ do
10:         $\widehat{\pi} \leftarrow$ identity mapping
11:         $\widehat{\pi}(q^1) \leftarrow q$ and $\widehat{\pi}(q^2) \leftarrow q$
12:         $\lambda \leftarrow$ a good $\widehat{\pi}$-distributor
13:         if $\frac{\mathrm{L}(c \mid \mathrm{merge}_{\widehat{\pi}}^{\lambda}(G'))}{\mathrm{L}(c \mid G')} \geq \mu$ then
14:             $\pi(q^1) \leftarrow q$ and $\pi(q^2) \leftarrow q$
15:     $\lambda \leftarrow$ a good $\pi$-distributor
16:     return $\mathrm{merge}_{\pi}^{\lambda}(G')$
```

Fig. 1. The state-splitting algorithm.

Splitting Let $G = (Q, \Sigma, q_\mathrm{s}, R)$ be an RTG and let $q \in Q \setminus \{q_\mathrm{s}\}$. We can split q into new states, e.g., q^1 and q^2. Then we also have to split rules which use this state, e.g., we would split the rule $q_\mathrm{s} \to \sigma(q, q)$ into $q_\mathrm{s} \to \sigma(q^1, q^1)$, $q_\mathrm{s} \to \sigma(q^1, q^2)$, $q_\mathrm{s} \to \sigma(q^2, q^1)$, and $q_\mathrm{s} \to \sigma(q^2, q^2)$ (cf. Fig. 2), i.e., we create a rule for every possible combination of the split states.

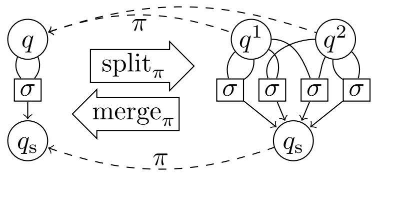

Fig. 2. Visualization of split/merge.

Formally, let Q' be an alphabet such that $q_\mathrm{s} \in Q'$, and let $\pi\colon Q' \to Q$ be a surjective mapping such that $\pi^{-1}(q_\mathrm{s}) = \{q_\mathrm{s}\}$. We call π a *G-splitter (w.r.t. Q')*. We use π to map a split state to its (unsplit) source. Hence, we can use π^{-1} to split a state. We will denote the splitting based on π by split_π and we will use this notion for the splitting of states, rules, sets of those, and RTG. Thus the *π-split of G* is the RTG $\mathrm{split}_\pi(G) = (\mathrm{split}_\pi(Q), \Sigma, q_\mathrm{s}, \mathrm{split}_\pi(R))$.

Now let us turn to PRTG. Let (G, p) and (G', p') be PRTG over the same terminal alphabet. We say (G', p') is a *proper π-split of* (G, p), if $G' = \mathrm{split}_\pi(G)$, and $p(r) = \sum_{r' \in \mathrm{split}_\pi(r)\colon \mathrm{lhs}(r') = q'} p'(r')$ for every rule r in G and for every $q' \in \mathrm{split}_\pi(\mathrm{lhs}(r))$. One can show that the probability of a tree in (G, p) is the same as in (G', p'), if (G', p') is a proper π-split of (G, p).

Merging The merge operation undoes splits. Formally, let $G' = (Q', \Sigma, q_\mathrm{s}, R')$ be an RTG, Q an alphabet such that $q_\mathrm{s} \in Q$, and π a surjective mapping such that $\pi^{-1}(q_\mathrm{s}) = \{q_\mathrm{s}\}$. We call π a *G'-merger (w.r.t. Q)*. We will denote the merging based on π by merge_π and we will use this notion for the merging of states, rules, sets of those, and RTG. Thus the merging results in the RTG $\mathrm{merge}_\pi(G') = (\mathrm{merge}_\pi(Q'), \Sigma, q_\mathrm{s}, \mathrm{merge}_\pi(R'))$.

Note that splitting and merging fit together nicely: Let G be an RTG, π be a G-splitter, and $G' = \mathrm{split}_\pi(G)$; then π is also a G'-merger and $\mathrm{merge}_\pi(G') = G$. The other direction is a bit more subtle: Let $G' = (Q', \Sigma, q_\mathrm{s}, R')$ be an RTG and π a G'-merger; then $Q' = \mathrm{split}_\pi(\mathrm{merge}_\pi(Q'))$, but $R' \subseteq \mathrm{split}_\pi(\mathrm{merge}_\pi(R'))$.

Now let us turn to PRTG again. Let (G', p') be a PRTG and π a G'-merger with $G' = (Q', \Sigma, q_\mathrm{s}, R')$ and $\mathrm{merge}_\pi(G') = (Q, \Sigma, q_\mathrm{s}, R)$. Roughly speaking, we need to merge p' now. Let $\lambda\colon Q' \to [0,1]$ be a mapping such that for every $q \in Q$ we have $\sum_{q' \in \mathrm{split}_\pi(q)} \lambda(q') = 1$. We call λ a *π-distributor.* We define the probability assignment $\mathrm{merge}_\pi^\lambda(p')$ to be the mapping $p\colon R \to [0,1]$ such that $p(r) = \sum_{r' \in R'\colon\ \mathrm{merge}_\pi(r')=r} \lambda(\mathrm{lhs}(r')) \cdot p'(r')$ for every $r \in R$. One can show that p is proper, which is guaranteed by the properties of λ. Yet, the λ provides enough flexibility for later tasks. We define $\mathrm{merge}_\pi^\lambda(G', p') = (\mathrm{merge}_\pi(G'), \mathrm{merge}_\pi^\lambda(p'))$.

In the MERGE function in Fig. 1, the algorithm decides which states to merge. For this purpose, it compares the likelihoods of the unmerged grammar and a grammar where only two states are merged. If this merge does not harm the likelihood too much, these two states will also be merged in the result. The parameter μ lets us configure the maximally accepted loss in the likelihood to find a trade-off between the loss of likelihood and the complexity of the grammar.

4 Further Research

We have to choose a "good" π-distributor λ for every merge. Our idea is, to do an EM iteration for the merged grammar, and since we do not have an initial probability assignment, we use the probability assignment of the split grammar to compute the counts needed for an EM iteration.

While merging, it is not clear how the likelihood changes in general. In [2], the ratio between the likelihoods (cf. Fig. 1, Line 13) was only approximated. Maybe the exact likelihood for the merged grammar can be calculated efficiently, if the likelihood of the unmerged grammar has already been calculated.

Also, in SPLIT we have to choose a proper π-split of G; it remains to investigate how this choice affects the outcome of the algorithm exactly. Additionally, good values for the parameter μ need to be found, which may be done empirically. Another idea is to get along without merging at all, if we split states only selectively, e.g., only the most frequent states in derivations of the trees in the corpus.

References

1. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.
2. S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 433–440, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.