Training Tree Transducers

Jonathan Graehl
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
graehl@isi.edu

Kevin Knight
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
knight@isi.edu

Abstract

Many probabilistic models for natural language are now written in terms of hierarchical tree structure. Tree-based modeling still lacks many of the standard tools taken for granted in (finite-state) string-based modeling. The theory of tree transducer automata provides a possible framework to draw on, as it has been worked out in an extensive literature. We motivate the use of tree transducers for natural language and address the training problem for probabilistic tree-to-tree and tree-to-string transducers.

1 Introduction

Much of natural language work over the past decade has employed probabilistic finite-state transducers (**FST**s) operating on strings. This has occurred somewhat under the influence of speech recognition, where transducing acoustic sequences to word sequences is neatly captured by left-to-right stateful substitution. Many conceptual tools exist, such as Viterbi decoding (Viterbi, 1967) and forward-backward training (Baum and Eagon, 1967), as well as generic software toolkits. Moreover, a surprising variety of problems are attackable with **FST**s, from part-of-speech tagging to letter-to-sound conversion to name transliteration.

However, language problems like machine translation break this mold, because they involve massive reordering of symbols, and because the transformation processes seem sensitive to hierarchical tree structure. Recently, specific probabilistic tree-based models have been proposed not only for machine translation (Wu, 1997; Alshawi, Bangalore, and Douglas, 2000; Yamada and Knight, 2001; Gildea, 2003; Eisner, 2003), but also for summarization (Knight and Marcu, 2002), paraphrasing (Pang, Knight, and Marcu, 2003), natural language generation (Langkilde and Knight, 1998; Bangalore and Rambow, 2000; Corston-Oliver et al., 2002), and language modeling (Baker, 1979; Lari and Young, 1990; Collins, 1997; Chelba and Jelinek, 2000; Charniak, 2001; Klein and Manning, 2003). It is useful to understand generic algorithms that may support all these tasks and more.

(Rounds, 1970) and (Thatcher, 1970) independently introduced tree transducers as a generalization of **FST**s. Rounds was motivated by natural language. The Rounds tree transducer is very similar to a left-to-right **FST**, except that it works top-down, pursuing subtrees in parallel, with each subtree transformed depending only on its own passed-down state. This class of transducer is often nowadays called **R**, for "Root-to-frontier" (Gécseg and Steinby, 1984).

Rounds uses a mathematics-oriented example of an **R** transducer, which we summarize in Figure 1. At each point in the top-down traversal, the transducer chooses a production to apply, based *only* on the current state and the current root symbol. The traversal continues until there are no more state-annotated nodes. Nondeterministic transducers may have several productions with the same left-hand side, and therefore some free choices to make during transduction.

An **R** transducer compactly represents a potentially-infinite set of input/output tree pairs: exactly those pairs (T1, T2) for which some sequence of productions applied to T1 (starting in the initial state) results in T2. This is similar to an **FST**, which compactly represents a set of input/output string pairs, and in fact, **R** is a generalization of **FST**. If we think of strings written down vertically, as degenerate trees, we can convert any **FST** into an **R** transducer by automatically replacing **FST** transitions with **R** productions.

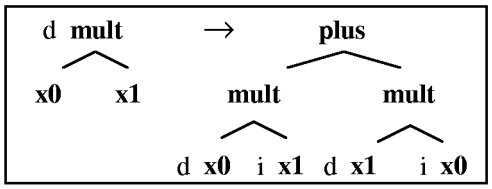
R does have some extra power beyond path following

This work was supported by DARPA contract F49620-00-1-0337 and ARDA contract MDA904-02-C-0450.

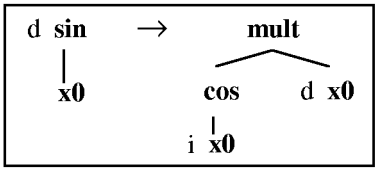
Transducer alphabet: {**0**, **1**, **a**, **y**, **sin**, **cos**, **plus**, **mult**}
Transducer states: {d for "derive", i for "identity"}
Transducer rules:

1. d **plus**(**x0**, **x1**) → **plus**(d **x0**, d **x1**)

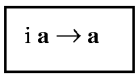
2. d **mult**(**x0**, **x1**) → **plus**(**mult**(d **x0**, i **x1**), **mult**(d **x1**, i **x0**))



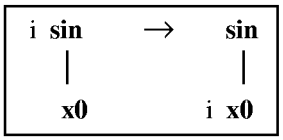
3. d **sin**(**x0**) → **mult**(**cos**(i **x0**), d **x0**)



4. i **a** → **a**



5. i **sin**(**x0**) → **sin**(i **x0**)



Transducer in action:

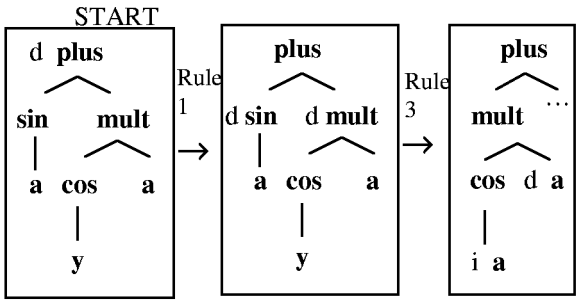


Figure 1: A sample **R** tree transducer that takes the derivative of its input.

and state-based record keeping. It can copy whole subtrees, and transform those subtrees differently. It can also delete subtrees without inspecting them (imagine by analogy an **FST** that quits and accepts right in the middle of an input string). Variants of **R** that disallow copying and deleting are called **RL** (for *linear*) and **RN** (for *nondeleting*), respectively.

One advantage of working with tree transducers is the large and useful body of literature about these automata; two excellent surveys are (Gécseg and Steinby, 1984) and (Comon et al., 1997). For example, **R** is not closed under composition (Rounds, 1970), and neither are **RL** or **F** (the "frontier-to-root" cousin of **R**), but the non-copying **FL** is closed under composition. Many of these composition results are first found in (Engelfriet, 1975).

R has surprising ability to change the structure of an input tree. For example, it may not be initially obvious how an **R** transducer can transform the English structure S(PRO, VP(V, NP)) into the Arabic equivalent S(V, PRO, NP), as it is difficult to move the subject PRO into position between the verb V and the direct object NP. First, **R** productions have no lookahead capability—the left-hand-side of the S production consists only of q S(x0, x1), although we want our English-to-Arabic transformation to apply only when it faces the entire structure q S(PRO, VP(V, NP)). However, we can simulate lookahead using states, as in these productions:

- q S(x0, x1) → S(qpro x0, qvp.v.np x1)
- qpro PRO → PRO
- qvp.v.np VP(x0, x1) → VP(qv x0, qnp x1)

By omitting rules like qpro NP → ..., we ensure that the entire production sequence will dead-end unless the first child of the input tree is in fact PRO. So finite lookahead is not a problem. The next problem is how to get the PRO to appear in between the V and NP, as in Arabic. This can be carried out using copying. We make two copies of the English VP, and assign them different states:

- q S(x0,x1) → S(qleft.vp.v x1, qpro x0, qright.vp.np x1)
- qpro PRO → PRO
- qleft.vp.v VP(x0, x1) → qv x0
- qright.vp.np VP(x0, x1) → qnp x1

While general properties of **R** are understood, there are many algorithmic questions. In this paper, we take on the problem of *training* probabilistic **R** transducers. For many language problems (machine translation, paraphrasing, text compression, etc.), it is possible to collect training data in the form of tree pairs and to distill linguistic knowledge automatically.

Our problem statement is: Given (1) a particular transducer with productions P, and (2) a finite training set of sample input/output tree pairs, we want to produce (3) a probability estimate for each production in P such that we maximize the probability of the output trees given the input trees.

As organized in the rest of this paper, we accomplish this by intersecting the given transducer with each input/output pair in turn. Each such intersection produces a set of weighted derivations that are packed into a regular tree grammar (Sections 3-5), which is equivalent to a tree substitution grammar. The inside and outside probabilities of this packed derivation structure are used to compute expected counts of the productions from the original, given transducer (Sections 6-7). Section 9 gives a sample transducer implementing a published machine translation model; some readers may wish to skip to this section directly.

2 Trees

T_Σ is the set of *(rooted, ordered, labeled, finite) trees over alphabet* Σ. An *alphabet* is just a finite set.

$T_\Sigma(X)$ are the *trees over alphabet* Σ*, indexed by* X—the subset of $T_{\Sigma \cup X}$ where only leaves may be labeled by X. ($T_\Sigma(\emptyset) = T_\Sigma$.) *Leaves* are nodes with no children.

The *nodes* of a tree *t* are identified one-to-one with its *paths*: $paths_t \subset paths \equiv \mathbb{N}^* \equiv \bigcup_{i=0}^{\infty} \mathbb{N}^i$ ($A^0 \equiv \{()\}$). The path to the root is the empty sequence $()$, and p_1 *extended by* p_2 is $p_1 \cdot p_2$, where $\cdot$ is concatenation.

For $p \in paths_t$, $rank_t(p)$ is the number of children, or *rank*, of the node at p in t, and $label_t(p) \in \Sigma \cup X$ is its *label*. The *ranked label* of a node is the pair $labelandrank_t(p) \equiv (label_t(p), rank_t(p))$. For $1 \leq i \leq rank_t(p)$, the i^{th} *child* of the node at p is located at *path* $p \cdot (i)$. The *subtree at path p of t* is $t \downarrow p$, defined by $paths_{t \downarrow p} \equiv \{q \mid p \cdot q \in paths_t\}$ and $labelandrank_{t \downarrow p}(q) \equiv labelandrank_t(p \cdot q)$.

The *paths to* X *in* t are $paths_t(X) \equiv \{p \in paths_t \mid label_t(p) \in X\}$. A *frontier* is a set of paths f that are *pairwise prefix-independent*:

$$\forall p_1, p_2 \in f, p \in paths : p_1 = p_2 \cdot p \implies p_1 = p_2$$

A *frontier of t* is a frontier $f \subseteq paths_t$.

For $t, s \in T_\Sigma(X), p \in paths_t$, $t[p \leftarrow s]$ is the *substitution of* s *for* p in t, where the subtree at path p is replaced by s. For a frontier f of t, the *mass substitution of* X *for the frontier* f *in* t is written $t[p \leftarrow X, \forall p \in f]$ and is equivalent to substituting the $X(p)$ for the p serially in any order.

Trees may be written as strings over $\Sigma \cup \{(,)\}$ in the usual way. For example, the tree $t = \text{S}(\text{NP}, \text{VP}(\text{V}, \text{NP}))$ has $labelandrank_t((2)) = (\text{VP}, 2)$ and $labelandrank_t((2,1)) = (\text{V}, 0)$. For $t \in T_\Sigma, \sigma \in \Sigma$, $\sigma(t)$ is the tree whose root has label σ and whose single child is t.

The *yield of* X *in* t is $yield_t(X)$, the string formed by reading out the leaves labeled with X in left-to-right order. The usual case (the *yield of t*) is $yield_t \equiv yield_t(\Sigma)$.

Σ = {S, NP, VP, PP, PREP, DET, N, V, run, the, of, sons, daughters}
N = {qnp, qpp, qdet, qn, qprep}
S = q
P = {q $\rightarrow^{1.0}$ S(qnp, VP(V(run))),
qnp $\rightarrow^{0.6}$ NP(qdet, qn),
qnp $\rightarrow^{0.4}$ NP(qnp, qpp),
qpp $\rightarrow^{1.0}$ PP(qprep, qnp),
qdet $\rightarrow^{1.0}$ DET(the),
qprep $\rightarrow^{1.0}$ PREP(of),
qn $\rightarrow^{0.5}$ N(sons),
qn $\rightarrow^{0.5}$ N(daughters)}

Figure 2: A sample weighted regular tree grammar (**wRTG**)

3 Regular Tree Grammars

In this section, we describe the *regular tree grammar*, a common way of compactly representing a potentially infinite set of trees (similar to the role played by the finite-state acceptor **FSA** for strings). We describe the version (equivalent to **TSG** (Schabes, 1990)) where the generated trees are given weights, as are strings in a **WFSA**.

A *weighted regular tree grammar* (**wRTG**) G is a quadruple (Σ, N, S, P), where Σ is the alphabet, N is the finite set of *nonterminals*, $S \in N$ is the *start (or initial) nonterminal*, and $P \subseteq N \times T_\Sigma(N) \times \mathbb{R}^+$ is the finite set of *weighted productions* ($\mathbb{R}^+ \equiv \{r \in \mathbb{R} \mid r > 0\}$). A production (lhs, rhs, w) is written $lhs \rightarrow^w rhs$. Productions whose rhs contains no nonterminals ($rhs \in T_\Sigma$) are called *terminal productions*, and rules of the form $A \rightarrow^w B$, for $A, B \in N$ are called ϵ*-productions*, or *epsilon productions*, and can be used in lieu of multiple initial nonterminals.

Figure 2 shows a sample **wRTG**. This grammar accepts an infinite number of trees. The tree S(NP(DT(the), N(sons)), VP(V(run))) comes out with probability 0.3.

We define the binary relation $\Rightarrow_G$ (*single-step derives in G*) on $T_\Sigma(N) \times (paths \times P)^*$, pairs of trees and *derivation histories*, which are logs of (location, production used):

$$\Rightarrow_G \equiv \Big\{((a,h),(b,h \cdot (p,(l,r,w))) \mid (l,r,w) \in P \wedge p \in paths_a(\{l\}) \wedge b = a[p \leftarrow r]\Big\}$$

where $(a,h) \Rightarrow_G (b, h \cdot (p,(l,r,w)))$ iff tree b may be derived from tree a by using the rule $l \rightarrow^w r$ to replace the nonterminal leaf l at path p with r. For a derivation history $h = ((p_1,(l_1,r_1,w_1)),\ldots,(p_n,(l_1,r_1,w_1)))$, the *weight of* h is $w(h) \equiv \prod_{i=1}^{n} w_i$, and call h *leftmost* if $L(h) \equiv \forall 1 \leq i < n : p_{i+1} \not<_{lex} p_i$.[1]

[1] $() <_{lex} (a)$, $(a_1) <_{lex} (a_2)$ iff $a_1 < a_2$, $(a_1) \cdot b_1 <_{lex} (a_2) \cdot b_2$ iff $a_1 < a_2 \vee (a_1 = a_2 \wedge b_1 <_{lex} b_2)$

The reflexive, transitive closure of $\Rightarrow_G$ is written $\Rightarrow_G^*$ (*derives in* G), and the restriction of $\Rightarrow_G^*$ to leftmost derivation histories is $\Rightarrow_G^{L*}$ (*leftmost derives in* G).

The *weight of* a *becoming* b *in* G is $w_G(a,b) \equiv \sum_{h:(a,())\Rightarrow_G^{L*}(b,h)} w(h)$, the sum of weights of all unique (leftmost) derivations transforming a to b, and the *weight of* t *in* G is $W_G(t) = w_G(S,t)$. The *weighted regular tree language produced by* G is $L_G \equiv \{(t,w) \in T_\Sigma \times \mathbb{R}^+ \mid W_G(t) = w\}$.

For every weighted context-free grammar, there is an equivalent **wRTG** that produces its weighted derivation trees with yields being the string produced, and the yields of regular tree grammars are context free string languages (Gécseg and Steinby, 1984).

What is sometimes called a *forest* in natural language generation (Langkilde, 2000; Nederhof and Satta, 2002) is a *finite* **wRTG** without loops, i.e., $\forall n \in N(n,()) \Rightarrow_G^* (t,h) \implies paths_t(\{n\}) = \emptyset$. Regular tree languages are strictly contained in tree sets of tree adjoining grammars (Joshi and Schabes, 1997).

4 Extended-LHS Tree Transducers (xR)

Section 1 informally described the root-to-frontier transducer class **R**. We saw that **R** allows, by use of states, finite lookahead and arbitrary rearrangement of nonsibling input subtrees removed by a finite distance. However, it is often easier to write rules that explicitly represent such lookahead and movement, relieving the burden on the user to produce the requisite intermediary rules and states. We define **xR**, a convenience-oriented generalization of weighted **R**. Because of its good fit to natural language problems, **xR** is already briefly touched on, though not defined, in (Rounds, 1970).

A *weighted extended-lhs root-to-frontier tree transducer* X is a quintuple $(\Sigma, \Delta, Q, Q_i, R)$ where Σ is the input alphabet, and Δ is the output alphabet, Q is a finite set of states, $Q_i \in Q$ is the *initial (or start, or root) state*, and $R \subseteq Q \times \boldsymbol{XRPAT}_\Sigma \times T_\Delta(Q \times paths) \times \mathbb{R}^+$ is a finite set of *weighted transformation rules*, written $(q, pattern) \rightarrow^w rhs$, meaning that an input subtree matching $pattern$ while in state q is transformed into rhs, with $Q \times paths$ leaves replaced by their (recursive) transformations. The $Q \times paths$ leaves of a rhs are called $nonterminals$ (there may also be $terminal$ leaves labeled by the output tree alphabet Δ).

$\boldsymbol{XRPAT}_\Sigma$ is the set of *finite tree patterns*: predicate functions $f : T_\Sigma \rightarrow \{0,1\}$ that depend only on the label and rank of a finite number of fixed paths their input. **xR** is the set of all such transducers. **R**, the set of conventional top-down transducers, is a subset of **xR** where the rules are restricted to use finite tree patterns that depend only on the root: $\boldsymbol{RPAT}_\Sigma \equiv \{p_{\sigma,r}(t)\}$ where $p_{\sigma,r}(t) \equiv (label_t(()) = \sigma \wedge rank_t(()) = r)$.

Rules whose rhs are a pure T_Δ with no states/paths for further expansion are called *terminal rules*. Rules of the form $(q, pat) \rightarrow^w (q', ())$ are *ϵ-rules*, or *epsilon rules*, which substitute state q' for state q without producing output, and stay at the current input subtree. Multiple initial states are not needed: we can use a single start state Q_i, and instead of each initial state q with starting weight w add the rule $(Q_i, \boldsymbol{TRUE}) \rightarrow^w (q, ())$ (where $\boldsymbol{TRUE}(t) \equiv 1, \forall t$).

We define the binary relation $\Rightarrow_X$ for **xR** tranducer X on $T_{\Sigma\cup\Delta\cup Q} \times (paths \times R)^*$, pairs of partially transformed (*working*) trees and derivation histories:

$$\Rightarrow_X \equiv \left\{ ((a,h),(b,h\cdot(i,(q,pat,r,w)))) \;\middle|\; \begin{array}{r} (q,pat,r,w) \in R \wedge i \in paths_a \wedge \\ q = label_a(i) \wedge pat(a \downarrow (i\cdot(1))) = 1 \wedge \\ b = a\left[i \leftarrow r\left[\begin{array}{c} p \leftarrow q'(a \downarrow (i\cdot(1)\cdot i')), \\ \forall p : label_r(p) = (q', i') \end{array}\right]\right] \end{array} \right\}$$

That is, b is derived from a by application of a rule $(q, pat) \rightarrow^w r$ to an unprocessed input subtree $a \downarrow i$ which is in state q, replacing it by output given by r, with its nonterminals replaced by the instruction to transform descendant input subtrees at relative path i' in state q'. The *sources* of a rule $r = (q, l, rhs, w) \in R$ are the input-path parts of the rhs nonterminals:

$$sources(rhs) \equiv \{i' \mid \begin{array}{l} \exists p \in paths_{rhs}(Q \times paths), \\ q' \in Q : label_{rhs}(p) = (q', i')\} \end{array}$$

If the sources of a rule refer to input paths that do not exist in the input, then the rule cannot apply (because $a \downarrow (i \cdot (1) \cdot i')$ would not exist). In the traditional statement of **R**, $sources(rhs)$ is always $\{(1), \ldots, (n)\}$, writing x_i instead of (i), but in **xR**, we identify mapped input subtrees by arbitrary (finite) paths.

An input tree is transformed by starting at the root in the initial state, and recursively applying output-generating rules to a frontier of (copies of) input subtrees (each marked with their own state), until (in a *complete derivation*, finishing at the leaves with terminal rules) no states remain.

Let $\Rightarrow_X^*$, $\Rightarrow_X^{L*}$, and $w_X(a,b)$ follow from $\Rightarrow_X$ exactly as in Section 3. Then the *weight of* (i,o) *in* X is $W_X(i,o) \equiv w_X(Q_i(i), o)$. The *weighted tree transduction given by* X is $X_X \equiv \{(i,o,w) \in T_\Sigma \times T_\Delta \times \mathbb{R}^+ | W_X(i,o) = w\}$.

5 Parsing a Tree Transduction

Derivation trees for a transducer $X = (\Sigma, \Delta, Q, Q_i, R)$ are trees labeled by rules (R) that dictate the choice of rules in a complete X-derivation. Figure 3 shows derivation trees for a particular transducer. In order to generate

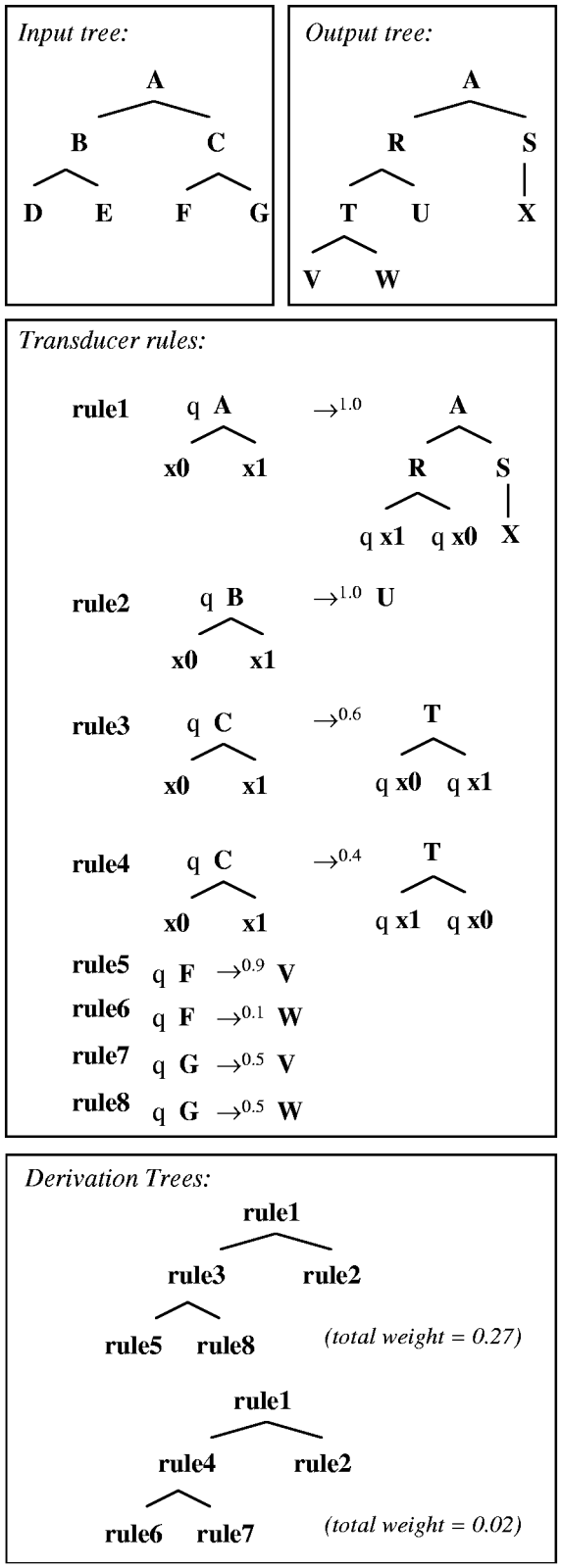


Figure 3: Derivation trees for an **R** tree transducer.

derivation trees for X automatically, we build a modified transducer X'. This new transducer produces derivation trees on its output instead of normal output trees. X' is (Σ, R, Q, Q_i, R'), with

$$R' \equiv \{(q, pattern, rule(yield_{rhs}(Q \times paths)), w) \mid rule = (q, pattern, rhs, w) \in R\}$$

That is, the original rhs of rules are flattened into a tree of depth 1, with the root labeled by the original rule, and all the non-expanding Δ-labeled nodes of the rhs removed, so that the remaining children are the nonterminal yield in left to right order. Derivation trees deterministically produce a single weighted output tree.

The derived transducer X' nicely produces derivation trees for a given input, but in explaining an observed (input/output) pair, we must restrict the possibilities further. Because the transformations of an input subtree depend only on that subtree and its state, we can (Algorithm 1) build a compact **wRTG** that produces exactly the weighted derivation trees corresponding to X-transductions $(I, ()) \Rightarrow_X^* (O, h)$ (with weight equal to $w_X(h)$).

6 Inside-Outside for wRTG

Given a **wRTG** $G = (\Sigma, N, S, P)$, we can compute the sums of weights of trees derived using each production by adapting the well-known inside-outside algorithm for weighted context-free (string) grammars (Lari and Young, 1990).

The *inside weights using G* are given by $\beta_G : T_\Sigma \rightarrow (\mathbb{R} - \mathbb{R}^-)$, giving the sum of weights of all tree-producing derivatons from trees with nonterminal leaves:

$$\beta_G(t) \equiv \begin{cases} \sum_{(t,r,w) \in P} w \cdot \beta_G(r) & \text{if } t \in N \\ \prod_{p \in paths_t(N)} \beta_G(label_t(p)) & \text{otherwise} \end{cases}$$

By definition, $\beta_G(S)$ gives the sum of the weights of all trees generated by G. For the **wRTG** generated by ***DERIV***(X, I, O), this is exactly $W_X(I, O)$.

Outside weights α_G for a nonterminal are the sums of weights of trees generated by the **wRTG** that have derivations containing it, but excluding its inside weights (that is, the weights summed do not include the weights of rules used to expand an instance of it).

$\alpha_G(n \in N) \equiv 1$ *if* $n = S$, *else:*

$$\overbrace{\sum_{p,(n',r,w) \in P : label_r(p) = n} w \cdot \alpha_G(n')}^{\text{uses of n in productions}} \cdot \underbrace{\prod_{p' \in paths_r(N) - \{p\}} \beta_G(label_r(p'))}_{\text{sibling nonterminals}}$$

Algorithm 1: ***DERIV***

Input: xR transducer $X = (\Sigma, \Delta, Q, Q_i, R)$ and observed tree pair $I \in T_\Sigma$, $O \in T_\Delta$.

Output: derivation **wRTG** $G = (R, N \subseteq Q \times paths_I \times paths_O, S, P)$ generating all weighted derivation trees for X that produce O from I. Returns $false$ instead if there are no such trees.

begin
  $S \leftarrow (Q_i, (), ())$, $N \leftarrow \emptyset$, $P \leftarrow \emptyset$
  if PRODUCE$_{I,O}(S)$ **then**
    return (R, N, S, P)
  else
    return $false$
end

memoized PRODUCE$_{I,O}(q, i, o)$ ***returns boolean*** $\equiv$
begin
  $anyrule? \leftarrow false$
  for $r = (q, pattern, rhs, w) \in R : pattern(I \downarrow i) = 1 \wedge$ **MATCH**$_{O,\Delta}(rhs, o)$ **do**
    $(o_1, \ldots, o_n) \leftarrow paths_{rhs}(Q \times paths)$ sorted by $o_1 <_{lex} \ldots <_{lex} o_n$
    //$n = 0$ `if there are none`
    $labelandrank_{derivrhs}(()) \leftarrow (r, n)$
    for $j \leftarrow 1$ ***to*** n **do**
      $(q', i') \leftarrow label_{rhs}(o_j)$
      $c \leftarrow (q', i \cdot i', o \cdot o_i)$
      if $\neg$**PRODUCE**$_{I,O}(c)$ **then next** r
      $labelandrank_{derivrhs}((j)) \leftarrow (c, 0)$
    $anyrule? \leftarrow true$
    $P \leftarrow P \cup \{((q, i, o), derivrhs, w)\}$
  if $anyrule?$ **then** $N \leftarrow N \cup \{(q, i, o)\}$
  return $anyrule?$
end

MATCH$_{t,\Sigma}(t', p) \equiv \forall p' \in path(t') : label(t', p') \in \Sigma \implies labelandrank_{t'}(p') = labelandrank_t(p \cdot p')$

The possible derivations for a given ***PRODUCE***$_{I,O}(q, i, o)$ are constant and need not be computed more than once, so the function is memoized. We have in the worst case to visit all $|Q| \cdot |I| \cdot |O|$ (q, i, o) pairs and have all $|R|$ transducer rules match at each of them. If enumerating rules matching transducer input-patterns and output-subtrees has cost L (constant given a transducer), then ***DERIV*** has time complexity $O(L \cdot |Q| \cdot |I| \cdot |O| \cdot |R|)$.

Finally, given inside and outside weights, the sum of weights of trees using a particular production is $\gamma_G((n, r, w) \in P) \equiv \alpha_G(n) \cdot w \cdot \beta_G(r)$.

Computing α_G and β_G for nonrecursive **wRTG** is a straightforward translation of the above recursive definitions (using memoization to compute each result only once) and is $O(|G|)$ in time and space.

7 EM Training

Estimation-Maximization training (Dempster, Laird, and Rubin, 1977) works on the principle that the corpus likelihood can be maximized subject to some normalization constraint on the parameters by repeatedly (1) *estimating* the expectation of decisions taken for all possible ways of generating the training corpus given the current parameters, accumulating parameter counts, and (2) *maximizing* by assigning the counts to the parameters and renormalizing. Each iteration is guaranteed to increase the likelihood until a local maximum is reached.

Algorithm 2 implements EM **xR** training, repeatedly computing inside-outside weights (using fixed transducer derivation **wRTG**s for each input/output tree pair) to efficiently sum each parameter contribution to likelihood over all derivations. Each EM iteration takes time linear in the size of the transducer and linear in the size of the derivation tree grammars for the training examples. The size of the derivation trees is at worst $O(|Q| \cdot |I| \cdot |O| \cdot |R|)$. For a corpus of K examples with average input/output size M, an iteration takes (at worst) $O(|Q| \cdot |R| \cdot K \cdot M^2)$ time—quadratic, like the forward-backward algorithm.

8 Tree-to-String Transducers (xRS)

We now turn to tree-to-string transducers (**xRS**). In the automata literature, these were first called *generalized syntax-directed translations* (Aho and Ullman, 1971) and used to specify compilers. Tree-to-string transducers have also been applied to machine translation (Yamada and Knight, 2001; Eisner, 2003).

We give an explicit tree-to-string transducer example in the next section. Formally, a *weighted extended-lhs root-to-frontier tree-to-string transducer* X is a quintuple $(\Sigma, \Delta, Q, Q_i, R)$ where Σ is the input alphabet, and Δ is the output alphabet, Q is a finite set of states, $Q_i \in Q$ is the *initial (or start, or root) state*, and $R \subseteq Q \times XRPAT_\Sigma \times (\Delta \cup (Q \times paths))^\star \times \mathbb{R}^+$ are a finite set of *weighted transformation rules*, written $(q, pattern) \rightarrow^w rhs$. A rule says that to transform (with weight w) an input subtree matching $pattern$ while in state q, replace it by the string of rhs with its nonterminal ($Q \times paths$) letters replaced by their (recursive) transformation.

xRS is the same as **xR**, except that the rhs are strings containing some nonterminals instead of trees containing nonterminal leaves (so the intermediate derivation objects

Algorithm 2: ***TRAIN***

Input: xR transducer $X = (\Sigma, \Delta, Q, Q_d, R)$, observed weighted tree pairs $T \in T_\Sigma \times T_\Delta \times \mathbb{R}^+$, normalization function $Z(\{count_r \mid r \in R\}, r' \in R)$, minimum relative log-likelihood change for convergence $\epsilon \in \mathbb{R}^+$, maximum number of iterations $maxit \in \mathbb{N}$, and prior counts (for a so-called *Dirichlet prior*) $\{prior_r \mid r \in R\}$ for smoothing each rule.

Output: New rule weights $W \equiv \{w_r \mid r \in R\}$.

begin
  for $(i, o, w) \in T$ **do**
    $d_{i,o} \leftarrow$
    DERIV(X, i, o)//`Alg. 1`
    if $d_{i,o} = false$ **then**
      $T \leftarrow T - \{(i, o, w)\}$
      warn(more rules are needed to explain (i,o))
    compute inside/outside weights for $d_{i,o}$ and remove all useless nonterminals n whose $\beta_{d_{i,o}}(n) = 0$ or $\alpha_{d_{i,o}}(n) = 0$
  $itno \leftarrow 0$, $lastL \leftarrow -\infty$, $\delta \leftarrow \epsilon$
  for $r = (q, pat, rhs, w) \in R$ **do** $w_r \leftarrow w$
  while $\delta \geq \epsilon \wedge itno < maxit$ **do**
    for $r \in R$ **do** $count_r \leftarrow prior_r$
    $L \leftarrow 0$
    for $(i, o, w_{example}) \in T$
    //`Estimate`
    do
      let $D \equiv d_{i,o} \equiv (R, N, S, P)$
      compute α_D, β_D using latest $W \equiv \{w_r \mid r \in R\}$
      //`see Section 6`
      for $prod = (n, rhs, w) \in P$ **do**
        $\gamma_D(prod) \leftarrow \alpha_D(n) \cdot w \cdot \beta_D(rhs)$
        let $rule \equiv label_{rhs}(())$
        $count_{rule} \leftarrow count_{rule} + w_{example} \cdot \frac{\gamma_D(prod)}{\beta_D(S)}$
      $L \leftarrow L + \log \beta_D(S) \cdot w_{example}$
    for $r = (q, pattern, rhs, w) \in R$
    //`Maximize`
    do
      $w_r \leftarrow \dfrac{count_r}{Z(\{count_r | r \in R\}, r)}$
      //`e.g.` $Z((q, a, b, c)) \equiv \displaystyle\sum_{r=(q,d,e,f) \in R} count_r$
    $\delta \leftarrow \dfrac{L - lastL}{|L|}$
    $lastL \leftarrow L$, $itno \leftarrow itno + 1$
end

are strings containing state-marked input subtrees). We have developed an **xRS** training procedure similar to the **xR** procedure, with extra computational expense to consider how different productions might map to different spans of the output string. Space limitations prohibit a detailed description; we refer the reader to a longer version of this paper (submitted). We note that this algorithm subsumes normal inside-outside training of PCFG on strings (Lari and Young, 1990), since we can always fix the input tree to some constant for all training examples.

9 Example

It is possible to cast many current probabilistic natural language models as **R**-type tree transducers. In this section, we implement the translation model of (Yamada and Knight, 2001). Their generative model provides a formula for P(Japanese string | English tree), in terms of individual parameters, and their appendix gives special EM re-estimation formulae for maximizing the product of these conditional probabilities across the whole tree/string corpus.

We now build a trainable **xRS** tree-to-string transducer that embodies the same P(Japanese string | English tree). First, we need start productions like these, where q is the start state:

- q x:S → q.TOP.S x
- q x:VP → q.TOP.VP x

These set up states like q.TOP.S, which means "translate this tree, whose root is S." Then every q.parent.child pair gets its own set of three insert-function-word productions, e.g.:

- q.TOP.S x → i x, r x
- q.TOP.S x → r x, i x
- q.TOP.S x → r x
- q.NP.NN x → i x, r x
- q.NP.NN x → r x, i x
- q.NP.NN x → r x

State i means "produce a Japanese function word out of thin air." We include an i production for every Japanese word in the vocabulary, e.g.:

- i x → de
- i x → kuruma
- i x → wa

State r means "re-order my children and then recurse." For internal nodes, we include a production for every parent/child-sequence and every permutation thereof, e.g.:

- r NP(x0:CD, x1:NN) → q.NP.CD x0, q.NP.NN x1
- r NP(x0:CD, x1:NN) → q.NP.NN x1, q.NP.CD x0

The rhs sends the child subtrees back to state q for recursive processing. However, for English leaf nodes, we instead transition to a different state t, so as to prohibit any subsequent Japanese function word insertion:

- r NN(x0:car) → t x0

- r CC(x0:and) → t x0

State t means "translate this word," and we have a production for every pair of co-occurring English and Japanese words:

- t car → kuruma
- t car → wa
- t car → *e*

This follows (Yamada and Knight, 2001) in also allowing English words to disappear, or translate to epsilon.

Every production in the **xRS** transducer has an associated weight and corresponds to exactly one of the model parameters.

There are several benefits to this **xRS** formulation. First, it clarifies the model, in the same way that (Knight and Al-Onaizan, 1998; Kumar and Byrne, 2003) elucidate other machine translation models in easily-grasped **FST** terms. Second, the model can be trained with generic, off-the-shelf tools—versus the alternative of working out model-specific re-estimation formulae and implementing custom training software. Third, we can easily extend the model in interesting ways. For example, we can add productions for multi-level and lexical re-ordering:

- r NP(x0:NP, PP(IN(of), x1:NP)) → q x1, no, q x0

We can add productions for phrasal translations:

- r NP(JJ(big), NN(cars)) → ooki, kuruma

This can now include crucial non-constituent phrasal translations:

- r S(NP(PRO(there),VP(VB(are), x0:NP) → q x0, ga, arimasu

We can also eliminate many epsilon word-translation rules in favor of more syntactically-controlled ones, e.g.:

- r NP(DT(the),x0:NN) → q x0

We can make many such changes without modifying the training procedure, as long as we stick to tree automata.

10 Related Work

Tree substitution grammars or **TSG** (Schabes, 1990) are equivalent to regular tree grammars. **xR** transducers are similar to (weighted) Synchronous **TSG**, except that **xR** can copy input trees (and transform the copies differently), but does not model deleted input subtrees. (Eisner, 2003) discusses training for Synchronous TSG. Our training algorithm is a generalization of forward-backward EM training for finite-state (string) transducers, which is in turn a generalization of the original forward-backward algorithm for Hidden Markov Models.

11 Acknowledgments

Thanks to Daniel Gildea and Kenji Yamada for comments on a draft of this paper, and to David McAllester for helping us connect into previous work in automata theory.

References

Aho, A. V. and J. D. Ullman. 1971. Translations of a context-free grammar. *Information and Control*, 19:439–475.

Alshawi, Hiyan, Srinivas Bangalore, and Shona Douglas. 2000. Learning dependency translation models as collections of finite state head transducers. *Computational Linguistics*, 26(1):45–60.

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

Bangalore, Srinivas and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proc. COLING*.

Baum, L. E. and J. A. Eagon. 1967. An inequality with application to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematicians Society*, 73:360–363.

Charniak, Eugene. 2001. Immediate-head parsing for language models. In *Proc. ACL*.

Chelba, C. and F. Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14(4):283–332.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. ACL*.

Comon, H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 1997. Tree automata techniques and applications. Available on `www.grappa.univ-lille3.fr/tata`. release October, 1st 2002.

Corston-Oliver, Simon, Michael Gamon, Eric K. Ringger, and Robert Moore. 2002. An overview of Amalgam, a machine-learned generation module. In *Proc. IWNLG*.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Eisner, Jason. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proc. ACL (companion volume)*.

Engelfriet, J. 1975. Bottom-up and top-down tree transformations—a comparison. *Math. Systems Theory*, 9(3):198–231.

Gécseg, F. and M. Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Gildea, Daniel. 2003. Loosely tree-based alignment for machine translation. In *Proc. ACL*.

Joshi, A. and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages (Vol. 3)*. Springer, NY.

Klein, Dan and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. ACL*.

Knight, K. and Y. Al-Onaizan. 1998. Translation with finite-state devices. In *Proc. AMTA*.

Knight, K. and D. Marcu. 2002. Summarization beyond sentence extraction—a probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1).

Kumar, S. and W. Byrne. 2003. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of HLT-NAACL*.

Langkilde, I. 2000. Forest-based statistical sentence generation. In *Proc. NAACL*.

Langkilde, I. and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proc. ACL*.

Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4.

Nederhof, Mark-Jan and Giorgio Satta. 2002. Parsing non-recursive CFGs. In *Proc. ACL*.

Pang, Bo, Kevin Knight, and Daniel Marcu. 2003. Syntax-based alignment of multiple translations extracting paraphrases and generating new sentences. In *Proc. HLT/NAACL*.

Rounds, William C. 1970. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287.

Schabes, Yves. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Thatcher, J. W. 1970. Generalized^2 sequential machine maps. *J. Comput. System Sci.*, 4:339–367.

Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Information Theory*, IT-13.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404.

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proc. ACL*.