Inference of Inversion Transduction Grammars

Alexander Clark ALEXC@CS.RHUL.AC.UK
Department of Computer Science, Royal Holloway University of London, Egham TW20 0EX, United Kingdom

Abstract

We present the first polynomial algorithm for learning a class of inversion transduction grammars (ITGs) that implement context free transducers – functions from strings to strings. The class of transductions that we can learn properly includes all subsequential transductions. These algorithms are based on a generalisation of distributional learning; we prove correctness of our algorithm under an identification in the limit model.

1. Introduction

Many important problems can be cast as the problem of learning an unknown function from one domain $\mathcal{X}$ to another $\mathcal{Y}$ from a sequence of input-output pairs $(x_1, y_1), \ldots, (x_n, y_n)$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. Often $\mathcal{X}$ may have the structure of a vector space, and $\mathcal{Y}$ may be the real numbers, which gives the class of regression problems, or may just be $\{0, 1\}$ which leads to classification. The learning problem depends crucially on the algebraic structure of $\mathcal{X}$ and $\mathcal{Y}$. In this paper, we consider the case where the elements of both $\mathcal{X}$ and $\mathcal{Y}$ are strings — i.e. free monoids over a finite alphabet. These functions are then called transductions. Many Natural Language Processing problems can be cast in this form: for example POS-tagging can be viewed as a transduction which preserves length. Machine translation is a transduction where the length of the output string need not be the same as the length of the input string. Indeed natural language understanding is best viewed as a transduction from sentences to logical forms. Moreover, these transductions occur in many other areas of computer science, the most historically prominent example being that of compilation (Aho & Ullman, 1969), where they are called *syntax-directed translation schemes*.

Unfortunately until now, essentially the only algorithm with any formal guarantees for learning these functions or transductions was the OSTIA algorithm for learning subsequential functions (Oncina et al., 1993). This class of functions is clearly not powerful enough to do any but a very limited amount of re-ordering and moreover has poor out of domain performance. Thus the application of techniques based on learning algorithms with theoretical guarantees has been limited to language pairs that require very limited amounts of reordering such as Castilian to Catalan. Indeed, in many cases the models are inferred using algorithms not for the class of subsequential transducers but rather the much simpler class of locally-testable transductions, where the state is determined by the local context (Vilar et al., 1996).

In this paper, we present an algorithm for learning a class of context free transducers that are capable of performing radical reorderings of the input string, and where the class of languages that can be learned properly include the class of subsequential transducers. Note that the algorithms we present here are some way away from being directly applicable to learning MT models from raw text, and the theoretical learning results we present – an empirical evaluation is beyond the scope of this paper – are not as sharp as we would like.

Nonetheless, we consider that the study of these algorithms is essential for the next generation of syntax-based machine translation systems – this paper contains the first non-trivial algorithms for learning these models. Future progress will depend on a sound theoretical understanding of inference for these models. (Mylonakis & Sima'an, 2010) point out that "existing empirical work is largely based on successful heuristic techniques, and the learning of Hiero-like BITG/SCFG remains an unsolved problem.".

Current approaches rely either on raw heuristics or on a more principled stochastic search for a probabilistic model. For example, (Blunsom et al., 2009) use a Gibbs sampler to find suitable re-orderings.

We also note that in the inductive inference frame-

Appearing in *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

work, there are trivial enumerative algorithms that can learn all enumerable classes of computable functions, by taking the first function in an enumeration that is compatible with the data seen so far.

Until recently there have been only a limited number of algorithms with formal guarantees for learning context free grammars (CFGs), and essentially no algorithms for learning ITGs. Recently, techniques for learning CFGs and Multiple Context Free Grammars (MCFGs) based on distributional learning have been developed and a general theory proposed (Clark, 2010a) based on using objective models: models where nonterminal symbols have a well defined interpretation independent of other symbols in the grammar. This paper extends these approaches to the learnability of *synchronous* CFGs which are closely related to MCFGs. If we assume that the grammars define a function, then it is possible to have algorithms that learn merely from positive examples even under a very difficult learning environment where the sequence may be adversarially chosen. The reason for this is that if we have a function and we see an input-output pair (a, m), then we know not just that the pair (a, m) is a positive example, but also that all of the infinitely many other examples (a, n) where $m \neq n$ are *not* examples. This means that we do not have the problems of controlling over-generalisation that plague non-probabilistic learning models in the inductive inference framework.

2. Definitions

We assume that we are given two finite non-empty alphabets Σ and Δ. We define Σ^* (and similarly Δ^*) to be the set of all finite strings of symbols from Σ, and we use λ to define the empty string. We define concatenation of strings u and v by uv, and we will extend this to sets in the natural way: given $U, V \subseteq \Sigma^*$ we define $UV = \{uv | u \in U, v \in V\}$.

We will consider relations between Σ^* and Δ^*. We will use letters from the first half of the alphabet $a, b \ldots$ for elements of Σ^* and letters from the second half $m, n, \ldots$ for elements of Δ^*. We assume that these relations are (possibly partial) functions, which we will write both as $T : \Sigma^* \to \Delta^*$, and as $T \subset \Sigma^* \times \Delta^*$. We will write $\mathrm{Dom}(T) = \{a | (a, m) \in T\}$. If T is a partial function, i.e. if $\mathrm{Dom}(T) \neq \Sigma^*$, then we will extend it to a total function T' from $\Sigma^* \to \Delta^* \cup \{0\}$ where $0 \notin \Delta$ is a distinguished symbol.

Given pairs of strings $(a, m) \in \Sigma^* \times \Delta^*$, we define two concatenation operations: $(a, m) \oplus (b, n) = (ab, mn)$ and $(a, m) \ominus (b, n) = (ab, nm)$. We will also call a pair of strings (a, m) a *biword*. We will define a *bicontext* as an element of $\Sigma^* \times \Sigma^* \times \Delta^* \times \Delta^*$, which we will write as (a, b, m, n). We can combine a bicontext with a biword to get a biword, by wrapping which we write as: $(a, b, m, n) \odot (c, o) = (acb, mon)$. Our approach is based on modeling the relationship between bicontexts and biwords where $(a, b, m, n) \sim_T (c, o)$ iff $(acb, mon) \in T$. For a biword (a, m) we define $\mathrm{BiCon}(a, m) = \{(b, c, n, o) | \exists d, p \in \Sigma^* \times \Delta^* \, s.t. \, bdc = a, npo = m\}$ to be the set of all bicontexts that occur in a word. We define $\mathrm{BiSub}(a, m) = \{(d, p) | \exists (b, c, n, o) \, s.t. \, bdc = a, npo = m\}$. We extend these operations to sets as before.

We take $|u|$ to be the length of a word; and we define $|(a, m)| = |a| + |m|$, and $|(a, m, b, n)| = |a| + |m| + |b| + |n|$. The size of a set or sequence of words, biwords or bicontexts is defined to be the sum of the lengths of the elements of the set or sequence. Note that $|\,\mathrm{BiSub}(a.m)| \leq |\,\mathrm{BiCon}(a, m)| = |a+1||a+2||m+1||m+2|/4$.

The representations we will use will be Inversion Transduction Grammars (ITGs) (Wu, 1997), a limited class of grammars that have a 2-normal form and are therefore efficiently biparsable.

An ITG, in the normal form we use, consists of a set V of nonterminals (NTs), a distinguished nonterminal start symbol, which we denote by S, and a set of rules of the following three types, where A, B and C are in V, $a \in \Sigma^*$, $m \in \Delta^*$

$$A \to [BC], A \to \langle BC \rangle, A \to a/m \tag{1}$$

We will say that a nonterminal derives a biword if there is an appropriate sequence of derivation steps; we will write this derivation relation as $A \stackrel{*}{\Rightarrow}_G (a, m)$. Suppose that $B \stackrel{*}{\Rightarrow} (b, n)$ and $C \stackrel{*}{\Rightarrow} (c, o)$. Then if there is a rule $A \to [BC]$ then $A \stackrel{*}{\Rightarrow} (b, n) \oplus (c, o) = (bc, no)$ and if there is a rule $A \to \langle BC \rangle$ then $A \stackrel{*}{\Rightarrow} (b, n) \ominus (c, o) = (bc, on)$. Additionally if there is a rule $A \to a/m$ then $A \stackrel{*}{\Rightarrow} (a, m)$. The derivation relation is the smallest relation that satisfies these closure properties. For a grammar G, and an NT A we define $L(G, A) = \{(a, m) | A \stackrel{*}{\Rightarrow}_G (a, m)\}$. The transduction defined by G is defined as $T(G) = L(G, S)$.

We will now give a simple example that we will refer to as T_{PP}. Consider arithmetic expressions using the two binary operators $+$ and $\times$, and the set of digits $1, \ldots 9$. We can write these unambiguously in either prefix or postfix notation. For example $2 \times (3+4)$ would be written as $\times 2 + 34$ in prefix, or $234 + \times$ in postfix. We can consider the function from strings in prefix notation to those in postfix notation; since this is only a par-

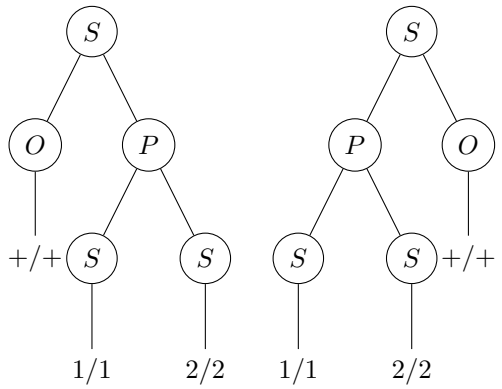

Figure 1. Derivation trees for input output pair $(+12, 12+)$.

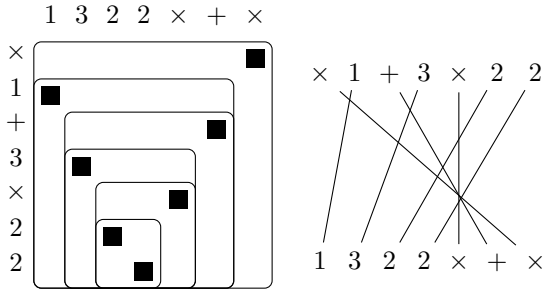

Figure 2. Example of alignment matrix for prefix order to postfix order transduction. The example is $(\times 1 + 3 \times 22, 1322 \times +\times)$. On the left an alignment matrix, and on the right an alignment diagram showing the crossing dependencies.

tial function we can extend this to a total function as discussed. This transduction will thus consist of pairs $(1, 1), (+12, 12+), (9, 9), (\times 2 + 34, 234 + \times) \ldots$ A simple ITG for this will have nonterminals S, O, P, with rules $S \to 1/1, \ldots S \to 9/9$, $O \to +/+, O \to \times/\times$, and $S \to \langle OP \rangle$, $P \to [SS]$. For each derivation we have a derivation tree which we can draw in two ways, ordered according to the input or according to the output. For example, we have that $S \stackrel{*}{\Rightarrow} (+12, 12+)$. Figure 1 shows the input and output trees for this derivation. Note how the rule $S \to \langle OP \rangle$ flips the tree in the output. Figure 2 gives an example alignment.

3. Principles

Our algorithm is based on the ideas of distributional learning presented in (Clark & Eyraud, 2007; Clark, 2010a). These were extended to the case of MCFGs by (Yoshinaka, 2011; 2010). Since synchronous CFGs are a special case of MCFGs, we can naturally take these techniques across, modifying them to take account of the fact that we are learning functions. This means that we can dispense with the need for Membership Queries and, as discussed in Section 1, rely purely on positive examples. We will also rely on a *dual* representation where the nonterminals are indexed by contexts rather than strings as in (Clark, 2010b).

For every non-terminal we define a *desired* set of biwords $I(N)$; we want $I(N)$ to be equal to $L(G, N)$. We will have two types of nonterminal in our grammar. We will have "primal" nonterminals N which are indexed by a biword (a, m); these are preterminals and each such nonterminal will just derive the biword itself, through a terminal rule of the form $N \to a/m$; each nonterminal will only appear on the left hand side of one rule and therefore $L(G, N) = \{(a, m)\}$. We will write the symbol corresponding to (a, m) as $\|\mathbf{a}, \mathbf{m}\|$. We define $I(\|\mathbf{a}, \mathbf{m}\|)$ to be the singleton set $\{(a, m)\}$.

The other type is "dual" and is indexed by a bicontext. For each nonterminal N we have a bicontext (a, b, m, n), where $a, b \in \Sigma^*$ and $m, n \in \Delta^*$. We will aim to have the set of biwords generated by this nonterminal correspond exactly to the set of biwords that can occur in that bicontext. For such a nonterminal N, which we will write as $\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\|$ we will define, given a transduction T:

$$I(N) = \{(c, o) \in \Sigma^* \times \Delta^* | (acb, mon) \in T\} \qquad (2)$$

We define the start symbol $S = \|\lambda, \lambda, \lambda, \lambda\|$. In this case $I(S) = T$. Note that the requirement that $L(G, N) = I(N)$. may seem unrealistic, but as we shall see below is satisfied by any subsequential transducer.

For our example T_{PP}, we have primal nonterminals like $\|\mathbf{1}, \mathbf{1}\|, \|\mathbf{2}, \mathbf{2}\|, \|+, +\|$, and dual nonterminals such as $\|\times, \lambda, \lambda, \times\|$.

Using this representation we can then say whether a rule is *valid* or not. Given a rule $A \to [BC]$ we say that it is valid iff $I(A) \supseteq I(B) \oplus I(C)$. Similarly, given a rule $A \to \langle BC \rangle$ we say that it is valid iff $I(A) \supseteq I(B) \ominus I(C)$. Terminal rules are always valid.

We first note that some branching rules will always be valid. Going back to our example we can see that the rule $\|\lambda, \lambda, \lambda, \lambda\| \to \langle \|\times, \times\| \| \times, \lambda, \lambda, \times\| \rangle$ is automatically valid.

Lemma 1. *The following four classes of rules are valid for any transduction.*

1. $\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to [\|\mathbf{a}, \mathbf{cb}, \mathbf{m}, \mathbf{on}\| \| \mathbf{c}, \mathbf{o}\|]$
2. $\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to [\|\mathbf{c}, \mathbf{o}\| \| \mathbf{ac}, \mathbf{b}, \mathbf{mo}, \mathbf{n}\|]$

3. $\|\mathbf{a},\mathbf{b},\mathbf{m},\mathbf{n}\| \rightarrow \langle\|\mathbf{a},\mathbf{cb},\mathbf{mo},\mathbf{n}\|\|\mathbf{c},\mathbf{o}\|\rangle$

4. $\|\mathbf{a},\mathbf{b},\mathbf{m},\mathbf{n}\| \rightarrow \langle\|\mathbf{c},\mathbf{o}\|\|\mathbf{ac},\mathbf{b},\mathbf{m},\mathbf{on}\|\rangle$

Proof. We will just prove two of the four cases. Case 1: Suppose $(d,p) \in I(\|\mathbf{a},\mathbf{cb},\mathbf{m},\mathbf{on}\|)$. This means that $(adcb, mpon) \in T$. Therefore $(dc, po) \in I(\|\mathbf{a},\mathbf{b},\mathbf{m},\mathbf{n}\|)$. Case 4: Suppose $(d,p) \in \|\mathbf{ac},\mathbf{b},\mathbf{m},\mathbf{on}\|$; then $(acdb, mpon) \in T$. $(c,o)\ominus(d,p) = (cd, po)$ which is therefore in $I(\|\mathbf{a},\mathbf{b},\mathbf{m},\mathbf{n}\|)$. □

A simple example of an invalid rule from our example T_{PP} is the rule $\|\lambda,\lambda,\lambda,\lambda\| \rightarrow [\|\lambda,\lambda,\lambda,\lambda\|\|\lambda,\lambda,\lambda,\lambda\|]$, since $(1,1) \oplus (1,1)$ is not in T.

We now give a lemma that justifies our use of the term *valid* above.

Lemma 2. *For any relation T, if the* ITG *G consists only of valid rules, then $L(G,N) \subseteq I(N)$, i.e. if $N \overset{*}{\Rightarrow}_G (a,m)$ it holds that $(a,m) \in I(N)$, and specifically $T(G) \subseteq T$.*

Proof. Proof is by induction on the length of the derivation. Note that if N is primal this is immediate. Otherwise suppose true for all derivations of length $\leq k \geq 1$. Let $N \overset{*}{\Rightarrow}_G (a,m)$ be a derivation of length $k+1$. Suppose it starts with a rule $N \rightarrow [PQ]$ and derivations of length $\leq K$ of the form $P \overset{*}{\Rightarrow}_G (b,n)$, $Q \overset{*}{\Rightarrow}_G (c,o)$ where $(a,m) = (b,n)\oplus(c,o)$. By inductive hypothesis $(b,n) \in I(P), (c,o) \in I(Q)$; since the rule $N \rightarrow [PQ]$ is valid, we have that $I(P) \oplus I(Q) \subseteq I(N)$ and therefore $(a,m) \in I(N)$. An identical argument holds for rules of the form $N \rightarrow \langle PQ\rangle$ and so the lemma holds. □

We now discuss how we can eliminate invalid rules. Suppose we have an invalid rule of the form $A \rightarrow [BC]$. This means that there must be some $(b,n) \in I(B)$ and $(c,o) \in I(C)$ such that (bc, no) is not in $I(A)$. Suppose $A = \|\mathbf{a},\mathbf{d},\mathbf{m},\mathbf{p}\|$. This means that $(abcd, mnop)$ is not in T; and therefore since T is a total function, there will be some $(abcd, q)$ where q is not equal to $mnop$. Thus when we observe the pair $(abcd, q)$ we will know that the rule $A \rightarrow [BC]$ is invalid, and we can discard it. The crucial point here is the local validity of the rules: since each nonterminal has a well defined set of biwords associated with it we can test the validity independently.

Given a rule of the form $\|\mathbf{a},\mathbf{b},\mathbf{m},\mathbf{n}\| \rightarrow [\|\mathbf{c},\mathbf{d},\mathbf{o},\mathbf{p}\|\|\mathbf{e},\mathbf{f},\mathbf{q},\mathbf{r}\|]$ we say that a triple of biwords contradicts this rule if they are of the form $(cgd, osp), (ehf, qtr), (aghb, w)$ where $w \neq mstn$. If a transduction contains such a triple then clearly the rule is invalid for that transduction, since $(g,s) \in I(c,d,o,p)$, $(h,t) \in I(e,f,q,r)$ yet $(g,s) \oplus (h,t)$ is not in $I(a,b,m,n)$. Similarly, given a rule of the form $\|\mathbf{a},\mathbf{b},\mathbf{m},\mathbf{n}\| \rightarrow \langle\|\mathbf{c},\mathbf{d},\mathbf{o},\mathbf{p}\|\|\mathbf{e},\mathbf{f},\mathbf{q},\mathbf{r}\|\rangle$ it is contradicted by a triple of biwords if they are of the form $(cgd, osp), (ehf, qtr), (aghb, w)$ where $w \neq mtsn$. Crucially, it might be the case that $aghb \notin \text{Dom}(T)$, in which case $w = 0$. It is at this point that it becomes important for the function to be total or extended to be total.

Given a finite set of biwords D we say that a dual branching rule is invalid with respect to D, if there is a triple of biwords in D that contradicts the rule. Otherwise we say it is valid w.r.t. D.

We can now define an algorithm for constructing an ITG given a finite set of NTs some of which are primal and some of which are dual, and a finite set of input-output pairs.

Algorithm 1 describes the procedure for making an ITG in pseudocode. Note that the algorithm runs in time polynomial in the total size of N_P, N_D and D. There are a number of obvious optimisations we could make that we neglect here for clarity. The algorithm works by taking all possible rules, and removing those that are invalid. We could also add rules of the form $\|\mathbf{ab},\mathbf{mn}\| \rightarrow [\|\mathbf{a},\mathbf{m}\|\|\mathbf{b},\mathbf{n}\|]$ and so on, which would be automatically valid, but these would be redundant.

4. Algorithm

Given this construction we can define a simple algorithm. Given a finite sample of data K, for each pair of strings (a,m) in the data we consider every possible split such that $bcde = a$ and $nopq = m$. We then hypothesise that there might be two rules, both of which have a nonterminal that derives the string biword (cd, op): one rule will have a nonterminal that derives (c,o) and another that derives (d,p); the second will have nonterminals that derive (c,p) and (d,o) and compose them backwards (i.e. with a $\langle\rangle$ rule).

As the algorithm proceeds we receive a stream of input-output pairs. At each step, we add the new pair to the pool of examples that we use to eliminate invalid rules. If the new pair is not generated by the transduction then we will add new NTs. The precise algorithm for adding new rules depends on the learning model. In this paper we make no assumptions about the sequence of examples, and thus the sequence may be chosen by an adversary. We therefore have to pick all possible rules, which while formally efficient (in P), gives an impractical explosion of possible rules. Under a probabilistic model, more refined algorithms could

be chosen that would still be correct.

Algorithm 1 Make ITG

Require: N_D is finite set of bicontexts including $(\lambda, \lambda, \lambda, \lambda)$, N_P is a finite set of biwords and D is finite set of input output pairs;
  $S := (\lambda, \lambda, \lambda, \lambda)$;
  $P := \emptyset$;
  for (a, m) in N_P **do**
    $P := P \cup \{\|\mathbf{a}, \mathbf{m}\| \to a/m\}$;
  end for
  for $(a, b, m, n), (c, d, o, p), (e, f, q, r) \in N_D$ **do**
    $P := P \cup \{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to [\|\mathbf{c}, \mathbf{d}, \mathbf{o}, \mathbf{p}\|\|\mathbf{e}, \mathbf{f}, \mathbf{q}, \mathbf{r}\|]$
    $\cup\{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to \langle\|\mathbf{c}, \mathbf{d}, \mathbf{o}, \mathbf{p}\|\|\mathbf{e}, \mathbf{f}, \mathbf{q}, \mathbf{r}\|\rangle\}$;
  end for
  for $(a, b, m, n) \in N_D$, $(c, o) \in N_P$ **do**
    if $(acb, mon) \in D$ **then**
      $P := P \cup \{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to c/o\}$;
    end if
  end for
  for $(a, b, m, n), (c, d, o, p) \in N_D$, $(g, s) \in N_P$ **do**
    $P := P \cup \{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to [\|\mathbf{c}, \mathbf{d}, \mathbf{o}, \mathbf{p}\|\|\mathbf{g}, \mathbf{s}\|]\}$;
    $P := P \cup \{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to \langle\|\mathbf{c}, \mathbf{d}, \mathbf{o}, \mathbf{p}\|\|\mathbf{g}, \mathbf{s}\|\rangle\}$;
    $P := P \cup \{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to [\|\mathbf{g}, \mathbf{s}\|\|\mathbf{c}, \mathbf{d}, \mathbf{o}, \mathbf{p}\|]\}$;
    $P := P \cup \{\|\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}\| \to \langle\|\mathbf{g}, \mathbf{s}\|\|\mathbf{c}, \mathbf{d}, \mathbf{o}, \mathbf{p}\|\rangle\}$;
  end for
  for each branching rule in P **do**
    if there is a triple of strings in D that show it is invalid **then**
      remove rule from P ;
    end if
  end for
  return $G = (N_P \cup N_D, S, P)$

The pseudocode is presented in Algorithm 2.

Algorithm 2 Learn ITG

  $D := \emptyset, Pos := \emptyset$;
  $S := (\lambda, \lambda, \lambda, \lambda)$, $N := \{S\}$;
  for each input-output pair (a_i, m_i) **do**
    $D := D \cup \{(a_i, m_i)\}$;
    if $m_i \neq 0$ **then**
      $Pos := Pos \cup \{(a_i, m_i)\}$;
      if it is not the case that $S \stackrel{*}{\Rightarrow}_{\hat{G}} (a_i, m_i)$ **then**
        $N_D :=$ BiCon(Pos), $N_P :=$ BiSub(Pos);
      end if
    end if
    let $\hat{G} = MakeITG(N_D, N_P, D)$;
    output $\hat{G}$;
  end for

We can consider our running example in the light of this algorithm. Suppose we first observe the biword $(+12, 12+)$. This has 49 biword substrings: we will add primal nonterminals and associated terminal rules for each of these: such as the useful rules $\|+, +\| \to +/+$, and the useless $\|+\mathbf{1}, \lambda\| \to +1/\lambda$. We also have 100 bicontexts including the useful $\|+, \lambda, \lambda, +\|$ and $\|\lambda, \lambda, \lambda, \lambda\|$, as well as many useless ones. Using this we will construct many rules: we will have some useful valid rules such as $\|\lambda, \lambda, \lambda, \lambda\| \to \langle\|+, +\|\|+, \lambda, \lambda, +\|\rangle$ and many other invalid rules. Consider the pair $(+11, 11+)$; from this we will hypothesise many rules including, from the segmentation $(+, 1) \oplus (1, 1) = (+1, 11)$, the rule $\|\lambda, \mathbf{1}, \lambda, +\| \to [\|\lambda, \mathbf{11}, \lambda, \mathbf{1}+\|\|+, \mathbf{1}, \mathbf{1}, +\|]$. This is invalid; though there are some elements for which it is correct. $(+ + 3, 31+)$ is in $I(\lambda, 11, \lambda, 1+)$ since $(+ + 311, 31 + 1+)$ is correct; $(\times 13, 3 \times 1)$ is in $I(+, 1, 1, +)$ since $(+ \times 131, 13 \times 1+)$ is correct, but $(+ + 3, 31+) \oplus (\times 13, 3 \times 1) = (+ + 3 \times 13, 31 + 3 \times 1)$ is not in $I(\lambda, 1, \lambda, +)$ since $(+ + 3 \times 131, 31 + 3 \times 1+)$ is incorrect. The correct output for $+ + 3 \times 131$ is $313 \times +1+$. Therefore, once we see the triple of inputs: $(+ + 311, 31 + 1+)$, $(+ \times 131, 13 \times 1+)$ and $(+ + 3 \times 131, 313 \times +1+)$ we will know that this rule is invalid. There are also valid rules that do not conform to our expectation of what the correct rule should be, in the sense that they may not be length preserving.

5. Learnable class

We wish to show that this algorithm will converge for a certain class of transductions. We will start by defining a class $\mathcal{T}_{fk}$ of transductions; and we will prove some basic properties of this class including that it is large enough to include some standard classes of transductions.

For any transduction T, if we have a finite set of nonterminals, and a collection of only valid rules then we will define a transduction $\hat{T}$ which will be a subset of T. The question is for which transductions is there a finite set of NTs such that the valid rules define exactly T. There is no precise syntactic way of characterising this class, since, as with CFGs, many of the decision problems associated with ITGs are undecidable.

We will say that a set of primal and dual NTs is a *kernel* for a transduction T, if the set of all valid rules, as defined in Algorithm 2, defines the transduction T. $\mathcal{T}_{fk}$ is defined as the set of all functions with a finite kernel.

On its own this is slightly circular – we will now discuss some of the properties of $\mathcal{T}_{fk}$. First note that $\mathcal{T}_{fk}$includes all subsequential transducers: this is the largest class of rational transductions that is widely

used (Mohri, 1997). Recall that a transduction is a subsequential transducer if there is a complete finite automaton, all of whose states are accepting, where the transitions are labelled with an output string from Δ^* and additionally each state has a "final" output string which it outputs when the input ends at that state. We will write $\sigma(q)$ for this extra string output from state q. Given a subsequential transducer A, with states Q, and a set of transitions $E \subset Q \times \Sigma \times \Delta^* \times Q$, we can easily define a suitable set of nonterminals. For each state q, we let a_q be a string such that $\delta(q_0, a_q) = q$, and we let m_q be the corresponding output. We add for each state a nonterminal $\|\mathbf{a_q}, \lambda, \mathbf{m_q}, \lambda\|$. For each transition labelled with input a and output m, we add a preterminal $\|\mathbf{a}, \mathbf{m}\|$, and for each final string n we add a preterminal $\|\lambda, \mathbf{n}\|$.

We can see then that given a transition (p, a, m, q) where $a \in \Sigma, m \in \Delta^*$, we will have a valid rule of the form

$$\|\mathbf{a_p}, \lambda, \mathbf{m_p}, \lambda\| \rightarrow [\|\mathbf{a}, \mathbf{m}\|\|\mathbf{a_q}, \lambda, \mathbf{m_q}, \lambda\|]$$

To model the final output function σ, we also add a terminal rule for each transition (p, a, m, q) of the form $\|\mathbf{a_p}, \lambda, \mathbf{m_p}, \lambda\| \rightarrow a/w\sigma(q)$.

We also need the special case for the start symbol q_0 where $a_{q_0} = \lambda$ and $m_{q_0} = \lambda$. $\|\lambda, \lambda, \lambda, \lambda\| \rightarrow \lambda/\sigma(q_0)$. This is responsible for dealing with the case where the input string has zero length, in which case the output is $\sigma(q_0)$.

Given these NTs we can see that all of these rules will be valid. As in other approaches of this kind, it is enough to show that there is a set of valid rules; all invalid rules will ultimately be removed, and thus irrelevant or useless nonterminals will not be part of the hypothesis, if we trim unreachable or unusable NTs.

We also note that $\mathcal{T}_{fk}$includes some non-subsequential rational transductions, as well as many non-rational transductions. To give a trivial example, the ITG with a single nonterminal S, indexed by $(\lambda, \lambda, \lambda, \lambda)$ and with rules: $S \rightarrow \langle SS \rangle$, and with $S \rightarrow \lambda/\lambda$ $S \rightarrow a/a$ for all $a \in \Sigma$, will defined a function which maps each string to its reverse.

We are also interested in transductions where the domain of the function is not necessarily Σ^*; in this case we have a difficulty that does not present itself in the case of a rational transduction. The domain of an ITG will be a context free language, but the complement of a context-free language, in contrast to the case of a regular language, is not necessarily context-free. In this case we will model the transduction using an ITG, and when the ITG does not define an output for a given input, we return the symbol 0.

The simple example T_{PP} of a prefix to postfix transduction is also in $\mathcal{T}_{fk}$, since we can define an ITG with the following NTs: the primal NTs are $\|+, +\|$, $\|\times, \times\|$, and the dual NTs are $\|+, \lambda, \lambda, +\|$, $\|\lambda, \lambda, \lambda, \lambda\|$. We will have valid rules of the form

$$\|\lambda, \lambda, \lambda, \lambda\| \rightarrow \langle \|+, +\|\|+, \lambda, \lambda, +\| \rangle \quad (3)$$

$$\|\lambda, \lambda, \lambda, \lambda\| \rightarrow \langle \|\times, \times\|\|+, \lambda, \lambda, +\| \rangle \quad (4)$$

$$\|+, \lambda, \lambda, +\| \rightarrow [\|\lambda, \lambda, \lambda, \lambda\|\|\lambda, \lambda, \lambda, \lambda\|] \quad (5)$$

as well as terminal rules of the form $\|\lambda, \lambda, \lambda, \lambda\| \rightarrow 1/1$ and so on for the other digits, and $\|\times, \times\| \rightarrow \times/\times$, $\|+, +\| \rightarrow +/+$.

6. Correctness

We will now prove the main result of our paper; the correctness of the algorithm for $\mathcal{T}_{fk}$.

Our learning model is identification in the limit. Given T we have T' which is the completion of T using the extra symbol 0. A presentation of a transduction T is a sequence $(a_1, w_1), \ldots$ such that $T' = \{(a_i, w_i) | i \in \mathbb{N}\}$. An algorithm will produce a grammar G_n from each finite sequence $(a_1, w_1) \ldots (a_n, w_n)$. We say that an algorithm identifies in the limit a transduction T, if for every presentation of T, there is some natural number N such that for all $n \geq N$, $G_n = G_N$ and $T(G_N) = T$. We say that an algorithm identifies in the limit a class of transductions $\mathcal{T}$ if for every $T \in \mathcal{T}$, the algorithm identifies in the limit T. It is important to remember that the transduction output by the algorithm at various points need not be a function. In what follows we will assume a fixed transduction T in $\mathcal{T}_{fk}$and a given presentation of T. First of all note that when the ITG only has valid rules, and defines the correct transduction T, then the algorithm will have converged and after that point it will never change.

Lemma 3. *If there is some n such that N_P, N_D is a kernel at step n, then there is some $N > n$ when the algorithm will converge to the correct transduction.*

Proof. Since we have a kernel, we know that the transduction will include the true transduction. So N_P, N_D will never increase after that point. Consider every possible invalid rule from these NTs; for each such rule there will be a triple of examples, which rule it out. Each of these examples is in T' and so will be present at some point; once all of these examples have appeared in the presentation, the algorithm will give exactly the right answer and will not change further. □

Theorem 1. *Algorithm 2 identifies in the limit $\mathcal{T}_{fk}$.*

Proof. Suppose T is a transduction in $\mathcal{T}_{fk}$ and $(a_1, m_1), (a_2, m_2) \ldots$ is a presentation of T. We will use $P_n = \{(a_i, m_i) | m_i \neq 0, i \leq n\}$ to refer to the set of positive examples seen in the first n examples. Let N_D be a set of bicontexts and N_P be a set of biwords such that $N_D \cup N_P$ is a kernel for T. Let n be the smallest number such that $\text{BiCon}(P_n)$ contains N_D and $\text{BiSub}(P_n)$ contains N_P. Consider the ITG produced at this step: G_n.

There are various cases which we need to consider. Case 1: the ITG has only valid rules and is correct in which case we have converged. Case 2: the ITG has only valid rules but does not define a sufficiently large transduction. In this case, there must be some point at which an element of $T \setminus T(G_n)$ occurs in the presentation; at this point N_D, N_P will be a kernel and thus by the preceding lemma it will converge. Case 3: the ITG has some invalid rules. By an argument similar to the previous lemma, either at some point we will remove all invalid rules, which will take us to Case 1 or Case 2, or we will increase N_D, N_P and therefore again by the preceding lemma we will converge; there are no other cases, and so we have established the lemma. □

Finally we would like to say something about the efficiency of the algorithm, It is easy to see that the algorithm runs in polynomial update time – if ℓ is the maximum length of the biwords seen so far, and n is the number of examples, then there is a polynomial $p(\ell, n)$ which bounds the running time of the algorithm to point n.

Ideally, we would like to be able to bound the number of examples, or rather the total size of examples needed to learn using the idea of a characteristic set. We would like to have a bound that says: given an ITG with n NTs which defines a transduction $T \in \mathcal{T}_{fk}$ there is a set X of biwords of total size $p(n)$ such that when the algorithm has seen any set that includes X it will have converged to the right answer. This will be very hard to do: while it is possible to do this for some classes of FSTs, ITGs have some additional problems, primarily that there is no canonical form and that the length of the shortest biwords generated by an ITG can, just as with CFGs, be exponential in n. Moreover, it could be the case that there is a much smaller ITG where the NTs do not correspond to individual contexts.

However we can say something. For a transduction T in $\mathcal{T}_{fk}$, let n be the size of the smallest ITG definable using primal and dual NTs as above, where the size is defined as the sum of the lengths of all biwords and bicontexts. Then there is a set X of polynomial size in n that will contain a kernel. Once we have a kernel, then there is another set Y of polynomial size in the size of the data seen so far after which it will converge. This gives a sort of "double polynomial" bound which is the best we can do at the moment.

Ideally we would like a pure PAC result for a large class of transducers – it is easy to see that this would fall foul of the well-known negative results for PAC-learning of finite automata. Distribution-free learning is too onerous a requirement.

7. Discussion

To the best of our knowledge these algorithms are the first with any non-trivial theoretical guarantees, that can learn any non-rational transductions. One exception is the work of (Yoshinaka, 2011) on learning MCFGs which implicitly defines a learning algorithm for ITGs, albeit for a very limited class and without considering a functional dependency. There have been a number of heuristic algorithms for learning ITGs; typically these approaches learn ITGs with *one* nonterminal. That is a rather different task. A simpler task is that of learning tree to tree transductions from input-output pairs of fully labelled trees. In the simple cases this reduces to algorithms for various types of tree transducers, typically top-down deterministic tree transducers (Eisner, 2003; Lemay et al., 2010).

We note that if we study the learnability of partial functions in the case where we do not extend it to a total function, then it is enough for the domain to be *substitutable* in the sense defined by (Clark & Eyraud, 2007): a language is substitutable if $abc, adc, ebf \in L$ implies $edf \in L$. In this case, the closure properties of the language will be sufficient to allow incorrect rules to be eliminated using only examples from within the transduction; as a special case note that Σ^* is substitutable.

The algorithm presented here is a "dual" algorithm in the sense of (Clark, 2010a). It is worth considering the corresponding primal algorithm, where instead of defining the non-trivial NTs in terms of bicontexts we consider them in terms of biwords. A natural approach to this would be to consider *congruential* ITGs in the sense of (Yoshinaka & Clark, 2010) result for congruential multiple context free grammars. We consider for any biword (u, v) its distribution in T to be $C_T(u, v) = \{(a, b, c, d) | (aub, cvd) \in T\}$, and define a congruence relation $(u, v) \equiv_T (u', v')$ iff $C_T(u, v) = C_T(u', v')$. We would then have primal NTs indexed by biwords, such that the set of biwords derived from the NT indexed by (u, v) would consist of all biwords

congruent to (u, v). However there are problems; consider for example, the identity transduction $T(w) = w$ over a two letter alphabet $\Sigma = \Delta = \{a, b\}$. In this case, $(a, a) \not\equiv_T (b, b)$; in particular $(a, \lambda, \lambda, a) \in C_T(a, a)$ but is not in $C_T(b, b)$ since $(ab, ba) \notin T$. Thus these congruence classes may be too small to be useful. However it might be possible to overcome this difficulty by considering a boundary symbol as discussed in (Yoshinaka, 2010).

Conceptually the existence of these algorithms could be important for debates about the nature of linguistic representation, since they demonstrate that it is not necessary to learn a fixed constituent structure in order to model a transduction that is sensitive to hierarchical structure in the input.

In their current form, though correct, the quantity of data they require and the computational requirements render them impractical for large scale application. We conjecture that the use of lattice based approaches as advocated in (Clark, 2010b) could be a productive direction; the assumption that each nonterminal is defined by a single context is perhaps too restrictive. The use of lattice based approaches would allow an exponentially large number of NTs indexed by all subsets of a finite set of bicontexts. It is also natural to consider a query-learning model of learning functions, where the learner can pick an input string a and get the corresponding output $T(a)$, or to consider a probabilistic model where we assume that the conditional entropy of the output given the input is bounded in some way. Both of these approaches would allow learning of transductions that are not functional which would be important for machine translation.

Acknowledgments

We would like to thank the reviewers for their very detailed and helpful reviews; and Ryo Yoshinaka for discussions.

References

Aho, A.V. and Ullman, J.D. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56, 1969.

Blunsom, P., Cohn, T., Dyer, C., and Osborne, M. A Gibbs sampler for phrasal synchronous grammar induction. In *Proceedings of the 47th Annual Meeting of the ACL*, pp. 782–790, 2009.

Clark, Alexander. Towards general algorithms for grammatical inference. In *Proceedings of ALT*, Canberra, Australia, October 2010a. Invited Paper.

Clark, Alexander. Efficient, correct, unsupervised learning of context-sensitive languages. In *Proceedings of CoNLL*, pp. 28–37, Uppsala, Sweden, 2010b.

Clark, Alexander and Eyraud, Rémi. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745, August 2007.

Eisner, J. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the ACL*, pp. 205–208, 2003.

Lemay, A., Maneth, S., and Niehren, J. A learning algorithm for top-down XML transformations. In *Proceedings of the twenty-ninth ACM symposium on Principles of database systems*, pp. 285–296, 2010.

Mohri, Mehryar. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

Mylonakis, M. and Sima'an, K. Learning Probabilistic Synchronous CFGs for Phrase-based Translation. In *Proceedings of CoNLL-2010*, pp. 117, 2010.

Oncina, J., García, P., and Vidal, E. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458, 1993.

Vilar, J.M., Jiménez, V.M., Amengual, J.C., Castellanos, A., Llorens, D., and Vidal, E. Text and speech translation by means of subsequential transducers. *Natural Language Engineering*, 2(4):351–354, 1996.

Wu, Dekai. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403, September 1997.

Yoshinaka, Ryo. Polynomial-time identification of multiple context-free languages from positive data and membership queries. In *Proceedings of the International Colloquium on Grammatical Inference*, 2010.

Yoshinaka, Ryo. Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. *Theoretical Computer Science*, 412(19):1821 – 1831, 2011.

Yoshinaka, Ryo and Clark, Alexander. Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In *Proceedings of the 15th Conference on Formal Grammar*, Copenhagen, Denmark, 2010.