From Finite-State to Inversion Transductions: Toward Unsupervised Bilingual Grammar Induction

Markus SAERS   Karteek ADDANKI   Dekai WU
Human Language Technology Center
Hong Kong University of Science and Technology, Hong Kong
`masaers@cs.ust.hk, vskaddanki@cs.ust.hk, dekai@cs.ust.hk`

ABSTRACT
We report a wide range of comparative experiments establishing for the first time contrastive foundations for a completely unsupervised approach to bilingual grammar induction that is cognitively oriented toward early category formation and phrasal chunking in the bootstrapping process up the expressiveness hierarchy from finite-state to linear to inversion transduction grammars. We show a consistent improvement in terms of cross-entropy throughout the bootstrapping process, as well as promising decoding experiments using the learned grammars. Rather than relying on external resources such as parses, POS tags or dictionaries, our method is fully unsupervised (in the way this term is typically understood in the machine translation community). This means that the bootstrapping can only rely on information gathered during the previous step, which necessitates some strategy for expanding the expressiveness of the grammars. We present principled approaches for moving from finite-state to linear transduction grammars as well as from linear to inversion transduction grammars. It is our belief that early, integrated category formation and phrasal chunking in this unsupervised bootstrapping process is better aligned to child language acquisition. Finally, we also report exploratory decoding results using some of the learned grammars. This is the first step towards an end-to-end grammar-based statistical machine translation system.

KEYWORDS: Grammar & Formalisms, Empirical machine translation, Multilinguality and Bilingual grammar induction.

Proceedings of COLING 2012: Technical Papers, pages 2325–2340,
COLING 2012, Mumbai, December 2012.

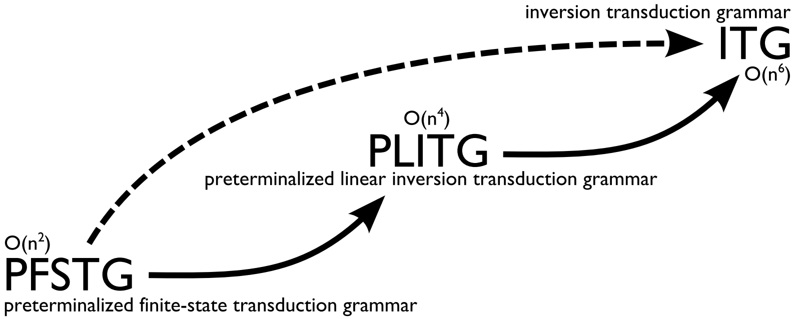
inversion transduction grammar
ITG
O(n6)
O(n4)
PLITG
preterminalized linear inversion transduction grammar
O(n2)
PFSTG
preterminalized finite-state transduction grammar

Figure1: Bootstrapping paths through the grammar hierarchy along with biparsing time complexities of grammar formalisms.

1 Introduction

We report a wide range of comparative experiments establishing for the first time contrastive foundations for a completely unsupervised approach to bilingual grammar induction that is cognitively oriented toward early category formation and phrasal chunking in the bootstrapping process up the expressiveness hierarchy from finite-state to linear to inversion transduction grammars.

In the context of bilingual grammar induction, "unsupervised" means that a transduction grammar (or synchronous grammar) is learned without using any external resources such as parses, POS tags, or dictionaries. In contrast, many, if not most, tree-based statistical MT approaches rely on monolingually parsed, chunked, and/or tagged parallel corpora (e.g., Galley et al. (2006)), from which a transduction grammar is extracted. Such approaches must compensate for monolingual analyses that often are not designed optimally for expressing the relationship between, say, English and Chinese. Exceptions include the hierarchical phrase-based SMT method of learning ITGs (Chiang, 2005), which does not rely on external resources.

However, unlike Chiang (2005) where huge numbers of (linguistically questionable) phrase translations are essentially memorized, our present work aims at inducing syntactic categories at an early stage in the learning, as occurs in child language acquisition. While only time will tell whether our more cognitively motivated approach to the induction of structural and lexical relationships between two languages will lead to better machine translation models, it is our belief that ultimately, learning the correct categories as early as humans do will provide a more accurate generalization bias for subsequent learning of more complex transduction rules.

In this paper, we report a large body of foundational experiments that, for the first time, illuminate how EM-based bilingual grammar induction behaves as we move up the complexity hierarchy of transduction grammars. We investigate various bootstrapping approaches to induction of finite-state transduction grammars (FSTGs), linear transduction grammars (LTGs), and inversion transduction grammars (ITGs). We then turn to various combinations of phrasal chunking to induce segmental transduction grammars, which can represent various classes of phrase-based translations. We then integrate category induction methods in various combinations.

This research thread represents a long term effort to investigate how a more principled, generalization-oriented model of bilingual grammar induction could gradually replace various parts of the long chain of heuristics used in the more memorization-oriented approaches. Although we do also look at preliminary BLEU and NIST scores from MT decoding using the induced transduction grammars, we believe it is premature to place excessive emphasis on such scores because too many other factors (such as the language model, to name one) are entangled. At this early stage of establishing the foundations for understanding category formation in bottom-up bilingual grammar induction, it is more important to compare more *directly* how well the relationships between the two languages are learned by the very many possible various combinations of bootstrapping between transduction expressivity levels, phrasal chunking, and category formation. Therefore, we focus more on the cross entropy of the induced bilingual grammars—which is exactly the bilingual form of the standard way to evaluate how well monolingual grammar induction captures monolingual data.

Note that, as observed in Wu (1997), this way of analyzing bilingual grammar induction can also be viewed as the problem of *bilingual language modeling*—modeling two languages simultaneously. We attack this problem of bilingual language modeling by extending a surprisingly effective finite-state baseline in two different ways: adding reordering capabilities and adding context information in the form of categories. The ultimate aim of this research direction is to have a grammar-based end-to-end statistical machine translation system, which would enable the usage of the same model in training and decoding instead of relying on a pipeline of different mismatched models trained independently. The bilingual language models we learn in this paper are grammar-based, and they thus represent a step towards this goal. Our new model is still extremely simple, even with the two proposed extensions. Reordering is added by first promoting the finite-state model to a linear model, which allows reading from the beginning or end of the two sentences independently and then by promoting the linear model to a fully nesting model. Contextual information is added in the form of a handful of categories, which are learned exclusively from the raw data.

We start by allowing any input token to translate into any output token, we also allow the input or output token to be empty. The lexical rules can contain input and output strings of length zero or one, but at least one of them must be nonempty. This initial search space is reigned in by training, which eliminates some of the (very unlikely) rules, but it is mainly expanded by allowing observed sequences to be taken up as lexicalized sequences (chunking), and by allowing category diversification (splitting). This initial stage of search space expansion is carried out in a very weak grammar formalism, which forces it to focus on learning lexical rules, since the weak structure of the grammar makes it rely heavily on surface form. Once the lexical chunks have been found and tentatively categorized, we will move the grammar into a more expressive formalism, and further train it. The idea is that we will have found enough good rules in the expansion phase that training with the more expressive formalism is mainly about establishing the structure and getting rid of useless rules. This approach also has the benefit that a lot of the “heavy lifting” can be done in the weaker grammar formalism, which is also less expensive to compute.

We will consistently use *preterminalized finite-state transduction grammars* (PFSTGs) as the weak grammar formalism, and *preterminalized linear inversion transduction grammars* (PLITGs) and *inversion transduction grammars* (ITGs) as the more expressive grammar formalisms.

The PFSTG's weakness comes from the fact that they have no structural way to reorder the input into the output. Instead they have to rely on singletons – lexical rules that delete from the input or insert into the output. These rules allow the PFSTG to disregard parts of a sentence pair, but still account for the parts that it can account for. It also means that PFSTGs need to keep a lot of singleton rules around that a more expressive grammar could do without.

ITGs are capable of considerable structural reordering, they are in fact the most expressive transduction grammar that can be used to parse a parallel corpus in polynomial time. They are, in other words, the most expressive grammar that can possibly be afforded.

PLITGs are wedged between PFSTGs and ITGs in terms of expressiveness. They do allow for some structured reordering. Like PFSTGs, the parse trees are chains rather than trees, but unlike PFSTGs, the links in these chains do not have to be physically adjacent. Allowing the first input token to be lexically associated with the last output token is an example of the kind of reordering that PLITGs allow, which make them more expressive than PFSTGs. The fact that these reorderings are limited to lexical units only is what makes them less expressive than ITGs (Saers et al., 2011) .

The rest of the paper is structured so that we begin by describing the initial finite-state transduction grammar that we will start our bootstrapping sequence (Section 2). We then describe how the lexical chunking (Section 3) and category splitting (Section 4) is carried out before moving on to expressivity expansion: first from finite-state to linear transduction grammars (Section 5), and then from linear to inversion transduction grammar (Section 6). We then move on to an illustrative example of what the bootstrapping process actually entails (Section 7), along with some tentative decoding results, before offering some concluding remarks (Section 8).

2 Initial grammar

As a grammar-based translation model baseline, we will use the simplest possible transduction grammar: the finite-state transduction grammar. A finite-state transduction grammar is the grammar form of a finite-state transducer. The transformation into a grammar is trivial and gives us FSTGs.

2.1 Definition 1

A FSTG over languages L_0 and L_1 is a tuple $G = \langle N, \Sigma, \Delta, S, R \rangle$, where N is a finite nonempty set of nonterminal symbols, Σ is a finite nonempty set of L_0 tokens, Δ is a finite nonempty set of L_1 tokens, $S \in N$ is the designated start symbol and R is a finite nonempty set of finite-state transduction rules on the forms:

$$A \rightarrow e/f\,B, \quad A \rightarrow \epsilon/\epsilon$$

where $A, B \in N$ *and* $e/f \in (\Sigma^* \times \Delta^*) - \{\epsilon/\epsilon\}$.

To harmonize this kind of grammar with future extensions to it, we will preterminalized it. A *preterminalized* FSTG has a special class of nonterminals called *preterminals*, which have a monopoly on rewriting to biterminals. This does not change the expressivity of the grammar, nor its asymptotic time complexity. To see why, simply imagine the degenerate case where every biterminal has exactly one unique preterminal symbol associated with it. In this paper, the only FSTGs we will use are *preterminalized finite-state transduction grammars*, or PFSTGs.

2.2 Definition 2

A PFSTG over languages L_0 and L_1 is a tuple $G = \langle N, P, \Sigma, \Delta, S, R\rangle$, where N is a finite nonempty set of nonterminal symbols, P is a finite nonempty set of preterminal symbols, disjoint from N, Σ is a finite nonempty set of L_0 tokens, Δ is a finite nonempty set of L_1 tokens, $S \in N$ is the designated start symbol and R is a finite nonempty set of preterminalized finite-state transduction rules on the forms:

$$A \rightarrow QB, \quad A \rightarrow \epsilon/\epsilon, \quad Q \rightarrow e/f$$

where $A, B \in N$, $Q \in P$ and $e/f \in (\Sigma^* \times \Delta^*) - \{\epsilon/\epsilon\}$.

As a baseline, we will choose the simplest possible PFSTG: a *bracketing* PFSTG. The bracketing PFSTG has only one nonterminal symbol and one preterminal symbol. As such it relies solely on lexical information to find translation correspondences. The lexical information is surprisingly powerful when both sentences are given.

To assign scores to the sentence pairs, we need a weighting function for the rules, making it a *weighted* PFSTG. We will go one step further, and require the weighting function to define proper probability distributions such that:

$$p(\psi \rightarrow \varphi) \equiv Pr(\varphi \mid \psi)$$

which makes the grammar *stochastic*. These kind of conditional probabilities over rules can be tuned towards a corpus of examples (a training corpus) through expectation maximization, or EM. Expectation maximization can only be used to tune existing parameters, so to have something to tune, we initialize the grammar using the training corpus. The structural rules will have uniform probability, and the lexical rules will have a portion of the probability mass relative to their cooccurrence in the training corpus. Specifically, we will initialize a stochastic bracketing FSTG such that:

$$\begin{aligned}
p(S \rightarrow A) &= 1,\\
p(A \rightarrow QA) &= 0.5,\\
p(A \rightarrow \epsilon/\epsilon) &= 0.5,\\
p\left(Q \rightarrow e/f\right) &= \frac{c\left(e/f\right)}{\sum_{e' \in \Sigma, f' \in \Delta} c\left(e'/f'\right)}
\end{aligned}$$

where $c\left(e/f\right)$ is the cooccurrence count for the biterminal $e/f \in ((\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\})) - \{\epsilon/\epsilon\}$.

To calculate the expectations for the expectation maximization, we will parse the training corpus with the grammar we have, using an adaptation of the parsing algorithm described in Saers et al. (2010). The adaptation consists of allowing multiple categories (not simply bracketing grammars), and to handle preterminals. Although this algorithm was designed for the linear family of transduction grammars, the PFSTGs are merely a restricted form of PLITGs, which means that the algorithm applies to them as well.

Training this grammar on IWSLT07 ChineseEnglish data (Fordyce, 2007) gave us a sentence-level cross-entropy of 110.2. The results of all training runs can be found in Table 1, where the baseline model is designated `fstg`.

3 Chunking helps

We can induce new lexical rules by allowing lexical entities to combine in order to form larger lexical entities. The end results are similar to phrase-based machine translation, but the method of arriving at the segments (chunks or “phrases”) is very different.

To apply chunking to our PFSTG, we use the method described in Saers and Wu (2011). The method was originally developed for PLITGs, which is a superset of PFSTGs, so it can be applied as is. The gist of the method is to allow two bitokens that are observed next to each other to combine into a new, larger bitoken. The process is thus limited to produce bitokens of twice the length of existing bitokens. There is, however, nothing stopping us from applying the method several times, getting larger and larger chunks. The maximum bitoken length of the baseline grammar is 2 (one input token and one output token), which we can double by chunking.

By applying chunking to the baseline PFSTG, and training with the chunks (model `fstg_c` in Table 1), we get a cross-entropy of 73.0, which is significantly better than the 110.2 that the baseline grammar (model `fstg`) scored. If we repeat the chunking (model `fstg_c_c`), we go all the way down to 45.0. A third round of chunking and training (model `fstg_c_c_c`) gets us a small improvement down to 43.9. This is interesting, since going from segments of length 1 to segments of length 2 helps tremendously, and proceeding to segments of length 4 goes a long way indeed. Pushing it up to segments of length 8 (which is close to the default “maximum phrase length” of Moses: 7), does surprisingly little. Either way, chunking was responsible for cutting the cross-entropy in more than half.

4 Splitting helps

We can introduce new nonterminal or preterminal symbols by splitting off some of the probability mass of an existing symbol to one or more new symbols. This gives us the possibility of diversifying the categories used by the grammar, which is necessary in order to move away from bracketing grammars to more interesting types of grammars.

Splitting into new symbols is a problem that can be formulated as splitting the probability mass of one symbol to several existing symbols. This requires the new symbols to be inserted into the grammar before the splitting takes place, which is a reasonable prerequisite. The assumption that the probability mass of the symbol being split could end up in any of the existing symbols makes it necessary to have some mechanism to control the destination of the probability mass. Furthermore, it is desirable to have some mechanism to introduce perturbations into the splitting of the probability mass, since exactly splitting it uniformly will inhibit learning. In addition to this wish list, we also wanted to differentiate between the case when the nonterminal was on the left-hand side of a rule, and when it was on the right-hand side. Remember that we are dealing with preterminalized grammars, and being able to have different splitting policies depending on where in the rule the symbol occurs is important. We want preterminals to be heavily perturbed on the left-hand side, but lightly perturbed on the right-hand side. When the preterminal is on the left-hand side, it is assigning a category to the terminal pair it rewrites to; by having larger differences in probability mass, we are essentially assigning a category at random, but keeping some of the mass in the other category, just in case the decision was wrong. When the preterminal is on the right-hand side, it determines the contexts in which a particular category can occur, which we do not want to randomize too much; relying on training is better. It turns out that all these desiderata can be addressed in a very simple way.

To split a nonterminal x, we give a set of target nonterminals $y_0, y_1, ..., y_k$ paired up with a kind of pseudo counts $a_0, a_1, ..., a_k$ that represent "how often x was split into y_i". The as are used as parameters to a Dirichlet distribution, from which a categorical distribution is randomly drawn every time x has to be split.

If any of the ys do not exist in the grammar, they are added to it. If no pseudo count is given for a nonterminal, it is assumed to be zero. This means that the Dirichlet distribution will be well-defined over the set of nonterminal symbols in the grammar. The expected categorical distribution drawn from the Dirichlet distribution will be "relative frequency" over the pseudo counts, but since we are drawing the categorical distribution at random, there will be some perturbation to it. The magnitude of the perturbation will be inversely proportional to the total number of pseudo counts in the Dirichlet distribution. Intuitively, the more pseudo counts we have, the more certain we can be that they are correct. Conversely, with few pseudo counts, there is a lot of uncertainty, and therefore a lot of variation in the categorical distributions we draw.

Once we have a categorical distribution over the set of nonterminals, we can distribute the probability mass of a rule containing x, into the expected portion according to the categorical distribution. To keep x as a symbol, it needs to retain some of the probability mass, and the simplest way to do this is to have non-zero pseudo counts for it; this is the approach we took.

To allow for different behavior depending on where in the rule x occurs, we simply supply two Dirichlet distributions: one to be used when x is on the left-hand side of a rule, and one to be used when it is on the right-hand side.

The pseudo code for the algorithm is give below.

4.1 General nonterminal (and preterminal) splitting algorithm

To refresh memory, a rule consists of a single nonterminal (or preterminal) symbol (i) on the left-hand side, and a sequence of nonterminal (or preterminal) and biterminal symbols on the right-hand side (ϕ). Furthermore, there is a permutation over the right-hand side (π). The rule form is thus $i \rightarrow \phi; \pi$. There is also a probability function that assigns a probability to each rule (p).

Input: a set of rules R, a nonterminal to split x, a probability function p over R, a left-hand side Dirichlet distribution over nonterminals D_l, a right-hand side Dirichlet distribution over nonterminals D_r.

Output: a new set of rules R' and a new probability function p' over R'.

```
$R' \leftarrow ()$
**for all** $i \rightarrow \phi; \pi \in R$ **do**
  $R'' \leftarrow ()$
  **if** $i = x$ **then**
    Draw the parameters to a categorical distribution over nonterminals $\alpha \sim D_l$
    **for all** nonterminal $y$ **do**
      $R'' \leftarrow (R'', [y \rightarrow; \pi, \alpha_y p(i \rightarrow \phi; \pi)])$
    **end for**
    else
        $R'' \leftarrow ([i \rightarrow; \pi, p(i \rightarrow \phi; \pi)])$
    end if
    for all $0 \leq k < \left|\phi\right|$ do
        $R''' \leftarrow ()$
        for all $[z \rightarrow \psi_{0..k}; \pi, p''] \in R''$ do
            if $\phi_k = x$ then
                Draw the parameters to a categorical distribution over nonterminals $\alpha \sim D_r$
                for all nonterminals $y$ do
                    $R''' \leftarrow (R''', [z \rightarrow \psi_{0..k}y; \pi, \alpha_y p''])$
                end for
            else
                $R''' \leftarrow (R''', [z \rightarrow \psi_{0..k}\phi_k; \pi, p''])$
            end if
            $R'' \leftarrow (R'', R''')$
        end for
    end for
    for all $[r, p''] \in R''$ do
        $R' \leftarrow (R', r)$
        $p'(r) += p''$
    end for
end for
```

We consistently used pseudo counts of 1000 (both for left- and right-hand side) when splitting nonterminals, and 0.5 (for left-hand side) and 1000 (for right-hand side) when splitting preterminals.

So far, we only have four PFSTGs, with different segment lengths. We did not carry out all possible experiments, but the ones we did carry out all show that splitting helps (all models in Table 1 with an `s` followed by some number have had all their symbols split into the given number of symbols). Sometimes it helps as little as when moving from 73.0 for a bracketing PFSTG with segment length 2, to 72.5 for the same grammar with its pre- and nonterminal split in two (model `fstg_c` to `fstg_c_s2`). The same modest improvement was observed when splitting the grammar with segment length 4, where we moved from 45.0 cross-entropy to 44.5 (model `fstg_c_c` to `fstg_c_c_s2`). However, splitting again gave 42.9 (moving to `fstg_c_c_s2_s2`), and a third time gave 39.5 (moving to `fstg_c_c_s2_s2_s2`). Beyond the first split, we see a better improvement than chunking an extra time.

5 Moving to linear transductions helps

The *preterminalized finite-state transduction grammar* is very closely related to the *preterminalized linear inversion transduction grammar*. The only difference is that the latter has more structural rules. As a reminder, the PFSTG has rules on the following forms:

$$A \rightarrow QB, \quad A \rightarrow \epsilon/\epsilon, \quad Q \rightarrow e/f$$

Whereas the PLITG has rule on the following forms:

$$A \rightarrow [QB], \quad A \rightarrow [BQ], \quad A \rightarrow \langle QB \rangle, \quad A \rightarrow \langle BQ \rangle, \quad A \rightarrow \epsilon/\epsilon, \quad Q \rightarrow e/f$$

Where the square and angled brackets have their customary interpretation as straight and inverted permutations respectively. The difference is in the structural rules, and each of the structural rules in a PFSTG corresponds to four structural rules in a PLITG. To move from a PFSTG to a PLITG, we merely have to add these rules, and distribute the probability mass. We distribute the probability mass uniformly.

Another possibility that PLITGs have that PFSTGs do not is to canonize the singletons. In the PLITGs we have used so far, there is always some ambiguity due to the fact that singletons are empty in one of the languages, and there is no way to tell whether the empty string should attach to the front or the back of the string in that language. This ambiguity can be eliminated by insisting that it attach to the same end that the known string attaches to in its language. This enforces a canonical form for singletons, and reduces complexity.

To canonize a PLITG, start by creating two new preterminal symbols, one for input singletons and one for output singletons for every existing preterminal. Then move all the singleton realizations of that preterminal to the newly created symbols. Consider the preterminal P. We will split it into $P^{e/f}$ (non-singletons), P^{f} (input singletons) and P^{e} (output singletons), and divide the set of rules so that the singletons must be produced by the singleton symbols, and the non-singletons must be produced by the non-singleton symbol.

To produce the new preterminal symbols, we also need to account for the case when P is on the right-hand side of a structural rule, and withholding some of the possible new rules will make the grammar singleton canonical. In a PLITG in normal form there may be at most one preterminal symbol on the right-hand side. Consider the four kinds of rules where P could occur in a PLITG:

$$A \to [PB], A \to [BP], A \to \langle PB \rangle, A \to \langle BP \rangle$$

To keep the PLITG canonical, we would like to have the following rules:

$$\begin{gathered} A \to \left[P^{e/f}B\right], A \to \left[BP^{e/f}\right], A \to \langle P^{e/f}B \rangle, A \to \langle BP^{e/f} \rangle, \\ A \to [P^{e}B], A \to \langle P^{e}B \rangle, A \to \left[P^{f}B\right], A \to \langle BP^{f} \rangle \end{gathered}$$

The probability mass is divided to reflect the probability of P rewriting to non-singletons and the singletons respectively.

Whether we use a canonical PLITG or not, it allows us to move from any of the grammar we have trained so far to a PLITG and resume training with this more expressive grammar formalism. This gives a consistent improvement in cross-entropy (this can be seen in Table 1, where the PLITG models end in `_ltg`, and are meaningful to compare to the models preceding them).

6 Moving to inversion transductions helps

Moving from a PFSTG or PLITG into an ITG is a much more complicated process than moving from a PFSTG to a PLITG. Since we have that first step covered, we will focus on moving from a PLITG to an ITG. Once this step is in place, we can move from a PFSTG to an ITG *via* a PLITG (with or without training at the PLITG stage).

Model	**cross-entropy**
fstg	110.2
fstg_ltg	108.9
fstg_ltg_itg	95.5
fstg_itg	93.2
fstg_c	73.0
fstg_c_ltg	72.5
fstg_c_ltg_itg	60.7
fstg_c_itg	60.7
fstg_c_s2	72.5
fstg_c_s2_ltg	70.5
fstg_c_s2_itg	60.8
fstg_c_c	45.0
fstg_c_c_ltg	44.5
fstg_c_c_itg	36.5
fstg_c_c_s2	44.5
fstg_c_c_s2_ltg	42.5
fstg_c_c_s2_itg	35.8
fstg_c_c_s2_s2	42.9
fstg_c_c_s2_s2_ltg	39.5
fstg_c_c_s2_s2_itg	33.3
fstg_c_c_s2_s2_s2	39.5
fstg_c_c_s2_s2_s2_ltg	39.5
fstg_c_c_s2_s2_s2_itg	**32.8**
fstg_c_c_s3	44.5
fstg_c_c_s3_itg	36.3
fstg_c_c_c	43.9
fstg_c_c_c_itg	35.9

Table1: The cross-entropy scores on the data for various models.

6.1 Definition 3

A syntax-directed transduction grammar (SDTG) in normal form is a tuple $\langle N, \Sigma, \Delta, S, R\rangle$ where N is a finite nonempty set of nonterminal symbols, Σ is a finite nonempty set of input language symbols, Δ is a finite nonempty set of output language symbols, $S \in N$ is the designated start symbol, and R is a finite nonempty set of syntax-directed transduction rules on the forms:

$$S \to A, \quad A \to \varphi ; \pi, \quad A \to e/f$$

where $A \in N$, $\varphi \in NNN^*$, π is a permutation vector over φ, and $e/f \in (\Sigma^* \times \Delta^*) - (\epsilon/\epsilon)$.

6.2 Definition 4

An inversion transduction grammar (ITG) in normal form is an SDTG in normal form, where the number of nonterminals allowed on the right-hand side is exactly two for all nonterminals except the start symbol (which may only have one). The rules are thus on the forms:

$$S \rightarrow A, \quad A \rightarrow BC; \pi, \quad A \rightarrow e/f$$

Since there are only two possible permutation vectors (straight and inverted), the rules are typically expressed as:

$$S \rightarrow A, \quad A \rightarrow [BC], \quad A \rightarrow \langle BC \rangle, \quad A \rightarrow e/f$$

where the square brackets represent straight order, and the angled brackets represent inverted order.

To make a PLITG in normal form into an ITG in normal form, all that has to be done is to eliminate the rules where a nonterminal is allowed to go to nothing (empty rules). This does, however, leave us with an ITG that generates the exact same transduction that the PLITG generated, which is not necessarily what we want. It is more likely that we want a grammar that is more expressive than the grammar we had. To this end, we will promote the preterminals to full nonterminals. Since the removal of the empty rules entails having the nonterminals behave as preterminals, and promoting the preterminals to full nonterminals entails having the preterminals behave as nonterminals, there is no longer any point in making the distinction between the two classes of symbols. Indeed, an ITG only has nonterminal symbols.

The theory of promoting preterminals to nonterminals is to insert unary rules, where the preterminal rewrites into exactly one nonterminal symbol. For every preterminal that is to be promoted, one such rule is generated for each nonterminal. To control how much of its "preterminal-ness" the preterminal retain, we employ a hyperparameter α. The probability mass of the preterminal is redistributed so that α of it is retained in its original rules, whereas $1-\alpha$ is redistributed to the new unary rules. The last thing that needs to be determined is the preterminals affinity to specific nonterminals. We use the function $\beta\left(y\right)$ for this, which gives the probability of the nonterminal y given the preterminal that is being promoted. In this round of experiments, β was uniform over all nonterminals. Naturally, we want to keep the new ITG in normal form, so we will actually not insert the unary rules, but rather anything the nonterminal can expand into. This gives us the following new probability function (p') over the new and old rules:

$$p'(x \rightarrow \phi; \pi) = \alpha p(x \rightarrow \phi; \pi) + (1-\alpha) \sum_{y \in N} \beta\left(y\right) p(y \rightarrow \phi; \pi)$$

Where x is the preterminal being promoted, N is the set of nonterminals, ϕ is a sequence of nonterminals and biterminals, and π is a permutation over ϕ.

To train at the ITG stage, we use the algorithm presented in Saers et al. (2009), which we generalize to handle multiple nonterminals rather than being restricted to bracketing ITGs.

It is clear from Table 1 that the ITG models explain the data better than any other kind of model, and that more induction steps are better (the best model is the most heavily processed one, with two chunking steps and three splitting steps before moving on to ITGs). In the cases where we move to ITGs via PLITGs, we also see some improvements, which is encouraging.

7 Qualitative analysis and translation assessments

In this section, we present a qualitative analysis of an example to illustrate the nature of alignments learned during various stages in the training. As a step forward towards our goal of purely inducing ITGs in an unsupervised manner for the purpose of translation, we report our initial findings on using these bootstrapped models in translation tasks.

7.1 Bootstrapping improves the alignments

Although the cross-entropy scores indicate an improvement in the quality of the grammars learned after LTGs and ITGs with FSTGs, we were interested in understanding the qualitative nature of generalizations the model was learning. Such an analysis would reveal how new generalizations might emerge as expressivity of models increases. They would also help identify overfitting of the model and how errors propagate from one stage to another.

An important consideration in bootstrapping would be to identify whether or not alignments that could not be learned in less expressive models could be learned by more expressive models after bootstrapping. In order words, whether bootstrapping enables learning of those alignments that would otherwise be impossible to learn with less expressive models. One should also consider the possibility that incorrect alignments in the initial stages might prevent the correct alignments from ever being learned.

With the above question in mind, we decided to investigate one of the simplest and most common manifestations of this scenario in Chinese-English parallel data. Typically, locatives in chinese appear before the verb whereas in English they appear after the verb. Hence, the alignments of bisentences which contain these locative markers require the model to permit a certain degree of reordering. We present below, a qualitative analysis of the translation rules learned for one such locative marker at various stages in our bootstrapping pipeline.

We choose the Chinese locative marker which occurs 67 times in our data set. typically translates to *inside* in English. The following are two sentences that occur in our training data.

Source:
Gloss: inside is what ?
Translation: what is inside ?

Source:
Gloss: inside is what thing ?
Translation: what does it contain ?

In the first example, the correct alignment would require an alignment permutation of $[2, 1, 0, 3]$ which is beyond the expressive power of the FSTG models. As the translation that is present in the training data for the second example is inexact, the locative marker is forced to align to . These examples are representative of the way in which appears in the training corpus.

Table 2 shows some of the best translations (in the decreasing order of probability) of the token learned after the FSTG, FSTG_LTG, FSTG_ITG and FSTG_LTG_ITG stages in our training pipeline, in contrast to directly inducing ITGs. It is not surprising that FSTGs fail to learn the correct translations of the locative as most alignments of sentences containing the

FSTG	**FSTG_LTG**	**FSTG_LTG_ITG**	**ITG**
$P \rightarrow$ /	$P \rightarrow$ /inside	$P \rightarrow$ /inside	$A \rightarrow$ /inside
$P \rightarrow$ /there	$P \rightarrow$ /it	$P \rightarrow$ /	$A \rightarrow$ /in
$P \rightarrow$ /it	$P \rightarrow$ /there	$P \rightarrow$ /it	$A \rightarrow$ /
$P \rightarrow$ /what	$P \rightarrow$ /	$P \rightarrow$ /in	$A \rightarrow$ /it
$P \rightarrow$ /I	$P \rightarrow$ /what	$P \rightarrow$ /I	$A \rightarrow$ /I

Table2: The best translation rules for the Chinese locative marker at various stages in the training pipeline.

Model	**Cross-Entropy**	**BLEU**	**NIST**
FSTG	110.2	7.95	0.4752
FSTG_LTG	108.9	8.34	0.7466
FSTG_LTG_ITG	95.5	8.83	0.8554

Table3: The correlation between the cross-entropy and the BLEU and NIST scores for token based models.

token are non-monotonic. From the second column of the table we can observe that the correct translation is learned to be the best translation after bootstrapping with LTGs alone. However, singleton translation and other synonyms of do not appear in the top five translations. Upon bootstrapping ITG with the FSTG_LTG, we identify that the translation *in* makes it into the top translations. The translations learned at FSTG_LTG_ITG stage are almost comparable to directly inducing an ITG as shown in the rightmost column.

Although locatives represent an extreme case of the nature of generalizations that could be learned from bootstrapping, we find that this kind of learning applies to all alignments at the FSTG stage. The correct alignments tend to get rewarded through successive stages of bootstrapping while the noisy alignments are drowned out. The end result is comparable to that of inducing LTGs and ITGs directly without the overhead of such an induction.

7.2 Successive bootstrapping promises improvements in translation quality

The final goal of our approach is to build a theoretically principled SMT system with well-defined representations. Although cross-entropy is a good measure of gauging how our models explain the data, it is not sufficient to guarantee an SMT system with a good performance. It is important to observe an improvement in the performance on a translation task that correlates with the improvement in cross-entropy of the model.

We approach this promise with caution as our models in their current state, are not optimized to compete with the state of the art SMT systems. We report the results of our preliminary experiments on using our trained models for the task of SMT.

Table 3 shows the BLEU scores obtained using only token based models (without any sort of chunking) on the IWSLT07_CE test set. These scores gives a gist of how translation quality changes with the grammar formalism, but a token-based model will naturally perform poorly compared to the state of the art. We used an in-house decoder for the purpose of decoding using our models. We used a trigram LM trained using SRILM (Stolcke, 2002) on the IWSLT dataset and a part of the gigaword data set.

We can observe significant gains in the BLEU (Papineni et al., 2002) and NIST (Doddington, 2002) scores after training on LTGs and a further improvement after training on ITGs. Further, these scores seem to be reflecting our estimates about the goodness of our models using cross-entropy.

Due to time constraints, we could not perform extensive experiments on the quality of translations produced with other models discussed in the rest of the paper. We also want to emphasize that an exhaustive evaluation of the performance of these models on translation tasks would result in a combinatoric explosion of models where chunking and splitting could be applied at various stages in the training pipeline with different chunking and splitting strategies. Therefore, in order to realize a competitive SMT system using our bootstrapping technique, we would first need to understand the qualitative and quantitative effects of using different models and the respective chunking and splitting strategies. We intend to pursue this task more confidently given our encouraging results on cross-entropy and the BLEU score correlation.

8 Conclusions

We reported a wide range of comparative experiments for a completely unsupervised approach to bilingual grammar induction with early category formation and phrasal chunking in the bootstrapping process up the expressiveness hierarchy from finite-state to linear to inversion transduction grammars. We reported a consistent improvement in the cross-entropy as we move through FSTGs to LTGs and ITGs. We also saw a substantial improvement in cross-entropy scores upon chunking and inducing categories. We reported an illustrative example that indicates generalizations learned from bootstrapping are equivalent to inducing models with higher expressivity directly but at a much lower cost. Finally, we discussed some of the encouraging results of our translation assessment experiments using bootstrapped models.

Acknowledgements

This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under BOLT contract no. HR0011-12-C-0016, and GALE contract nos. HR0011-06-C-0022 and HR0011-06-C-0023; by the European Union under the FP7 grant agreement no. 287658; and by the Hong Kong Research Grants Council (RGC) research grants GRF621008, GRF612806, DAG03/04.EG09, RGC6256/00E, and RGC6083/99E. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the RGC, EU, or DARPA.

References

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 263–270, Ann Arbor, Michigan.

Doddington, G. (2002). Automatic Evaluation of Machine Translation Quality using N-gram Co-occurrence Statistics. In *Proceedings of the 2nd International Conference on Human Language Technology Research (HLT-02)*, pages 138–145, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Fordyce, C. S. (2007). Overview of the IWSLT 2007 evaluation campaign. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 1–12.

Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W., and Thayer, I. (2006). Scalable inference and training of context-rich syntactic translation models. In *Proceedings of* *the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 961–968, Sydney, Australia.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 311–318.

Saers, M., Nivre, J., and Wu, D. (2009). Learning stochastic bracketing inversion transduction grammars with a cubic time biparsing algorithm. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 29–32, Stroudsburg, PA, USA. Association for Computational Linguistics.

Saers, M., Nivre, J., and Wu, D. (2010). Word alignment with stochastic bracketing linear inversion transduction grammar. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 341–344, Stroudsburg, PA, USA. Association for Computational Linguistics.

Saers, M. and Wu, D. (2011). Principled induction of phrasal bilexica. In *Proceedings of the 15th Annual Conference of the European Association for Machine Translation, Leuven, Belgium, May*.

Saers, M., Wu, D., and Quirk, C. (2011). On the expressivity of linear transductions. In *Proceedings of the 13th Machine Translation Summit (MT Summit XIII)*.

Stolcke, A. (2002). SRILM-an Extensible Language Modeling Toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing (Interspeech-02)*.

Wu, D. (1997). Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–403.