Scaling up Analogical Learning

Philippe Langlais
Université de Montréal / Dept. I.R.O.
C.P. 6128, Québec, H3C3J7, Canada
felipe@iro.umontreal.ca

François Yvon
Univ. Paris Sud 11 & LIMSI-CNRS
F-91401 Orsay, France
yvon@limsi.fr

Abstract

Recent years have witnessed a growing interest in analogical learning for NLP applications. If the principle of analogical learning is quite simple, it does involve complex steps that seriously limit its applicability, the most computationally demanding one being the identification of analogies in the input space. In this study, we investigate different strategies for efficiently solving this problem and study their scalability.

1 Introduction

Analogical learning (Pirrelli and Yvon, 1999) belongs to the family of lazy learning techniques (Aha, 1997). It allows to map forms belonging to an *input* space $\mathcal{I}$ into forms of an *output* space $\mathcal{O}$, thanks to a set of known observations, $\mathcal{L} = \{(i, o) : i \in \mathcal{I}, o \in \mathcal{O}\}$. $I(u)$ and $O(u)$ respectively denote the projection of an observation u into the input space and output space: if $u \equiv (i, o)$, then $I(u) \equiv i$ and $O(u) \equiv o$. For an incomplete observation $u \equiv (i, ?)$, the inference of $O(u)$ involves the following steps:

1. building $\mathcal{E}_{\mathcal{I}}(u)$ the set of analogical triplets of $I(u)$, that is $\mathcal{E}_{\mathcal{I}}(u) = \{(s, v, w) \in \mathcal{L}^3 : [I(s) : I(v) = I(w) : I(u)]\}$
2. building the set of solutions to the target equations formed by projecting source triplets: $\mathcal{E}_{\mathcal{O}}(u) = \{t \in \mathcal{O} : [O(s) : O(v) = O(w) : t], \forall (s, v, w) \in \mathcal{E}_{\mathcal{I}}(u)\}$
3. selecting candidates among $\mathcal{E}_{\mathcal{O}}(u)$.

where $[x : y = z : t]$ denotes an *analogical proportion*, that is a relation between these four items, meaning that "*x* is to *y* as *z* is to *t*", in a sense to be specified. See (Lepage, 1998) or (Stroppa and Yvon, 2005) for possible interpretations.

Analogical learning has recently regained some interest in the NLP community. Lepage and Denoual (2005) proposed a machine translation system entirely based on the concept of *formal analogy*, that is, analogy on forms. Stroppa and Yvon (2005) applied analogical learning to several morphological tasks also involving analogies on words. Langlais and Patry (2007) applied it to the task of translating unknown words in several European languages, an idea investigated as well by Denoual (2007) for a Japanese to English translation task.

If the principle of analogical learning is quite simple, it does involve complex steps that seriously limit its applicability. As a matter of fact, we are only aware of studies where analogical learning is applied to restricted tasks, either because they arbitrarily concentrate on words (Stroppa and Yvon, 2005; Langlais and Patry, 2007; Denoual, 2007) or because they focus on limited data (Lepage and Denoual, 2005; Denoual, 2007).

In this study, we investigate different strategies for making step 1 of analogical learning tractable. We propose a data-structure and algorithms that allow to control the balance between speed and recall. For very high-dimensional input spaces (hundreds of thousand of elements), we propose a heuristic which reduces computation time with a limited impact on recall.

2 Identifying input analogical relations

2.1 Existing approaches

A brute-force approach for identifying the input triplets that define an analogy with the incomplete observation $u = (t, ?)$ consists in enumerating triplets in the input space and checking for an ana-

logical relation with the unknown form t:

$$\mathcal{E}_{\mathcal{I}}(u) = \{\ \langle x,y,z\rangle\ :\ \begin{array}{l}\langle x,y,z\rangle \in \mathcal{I}^3,\\ [x:y=z:t]\ \}\end{array}$$

This amounts to check $o(|\mathcal{I}|^3)$ analogies, which is manageable for toy problems only.

Langlais and Patry (2007) deal with an input space in the order of tens of thousand forms (the typical size of a vocabulary) using following strategy for $\mathcal{E}_{\mathcal{I}}(u)$. It consists in solving analogical equations $[y : x = t : ?]$ for some pairs $\langle x,y\rangle$ belonging to the neighborhood[1] of $I(u)$, denoted $\mathcal{N}(t)$. Those solutions that belong to the input space are the z-forms retained.

$$\mathcal{E}_{\mathcal{I}}(u) = \{\ \langle x,y,z\rangle\ :\ \begin{array}{l}\langle x,y\rangle \in \mathcal{N}(t)^2,\\ [y:x=t:z]\ \}\end{array}$$

This strategy (hereafter named LP) directly follows from a symmetrical property of an analogy ($[x : y = z : t] \Leftrightarrow [y : x = t : z]$), and reduces the search procedure to the resolution of a number of analogical equations which is quadratic with the number of pairs $\langle x,y\rangle$ sampled.

2.2 Exhaustive tree-count search

The strategy we propose here exploits a property on character counts that an analogical relation must fulfill (Lepage, 1998):

$$[x:y=z:t] \Rightarrow |x|_c+|t|_c = |y|_c+|z|_c \quad \forall c \in \mathcal{A}$$

where $\mathcal{A}$ is the alphabet on which the forms are built, and $|x|_c$ stands for the number of occurrences of character c in x. In the sequel, we denote $\mathcal{C}(\langle x,t\rangle) = \{\langle y,z\rangle \in \mathcal{I}^2 \ :\ |x|_c + |t|_c = |y|_c + |z|_c \quad \forall c \in \mathcal{A}\}$ the set of pairs that satisfy the count property with respect to $\langle x,t\rangle$.

The strategy we propose consists in first selecting an x-form in the input space. This enforces a set of necessary constraints on the counts of characters that any two forms y and z must satisfy for $[x : y = z : t]$ to be true. By considering all forms x in turn,[2] we collect a set of candidate triplets for t. A verification of those that define with t a analogy must then be carried out. Formally, we built:

$$\mathcal{E}_{\mathcal{I}}(u) = \{\ \langle x,y,z\rangle\ :\ \begin{array}{l}x \in \mathcal{I},\\ \langle y,z\rangle \in \mathcal{C}(\langle x,t\rangle),\\ [x:y=z:t]\ \}\end{array}$$

This strategy will only work if (i) the number of quadruplets to check is much smaller than the number of triplets we can form in the input space (which happens to be the case in practice), and if (ii) we can efficiently identify the pairs $\langle y,z\rangle$ that satisfy a set of constraints on character counts. To this end, we propose to organize the input space thanks to a data structure called a *tree-count* (see Section 3), which is easy to built and supports efficient runtime retrieval.

2.3 Sampled tree-count search

As shown in (Langlais and Yvon, 2008), using tree-count to constraint the search allows to *exhaustively* solve step 1 for reasonably large input spaces. Computing analogies in very large input space (hundreds of thousand forms) however remains computationally demanding, as the retrieval algorithm must be carried out $o(\mathcal{I})$ times. In this case, we propose to sample the x-forms:

$$\mathcal{E}_{\mathcal{I}}(u) = \{\ \langle x,y,z\rangle\ :\ \begin{array}{l}x \in \mathcal{N}(t),\\ \langle y,z\rangle \in \mathcal{C}(\langle x,t\rangle),\\ [x:y=t:z]\ \}\end{array}$$

There is unfortunately no obvious way of selecting a good subset $\mathcal{N}(t)$ of input forms, as analogies does not necessarily entail the similarity of "diagonal" forms, as illustrated by the analogy [*une pomme verte* : *des pommes vertes* = *une voiture rouge* : *des voitures rouges*], which involves singular/plural commutations in French nominal groups. In this situation, randomly selecting a subset of the input space seems to be a reasonable strategy (hereafter RAND).

For some analogies however, the first and last forms share some sequences of characters. This is obvious in [*dream* : *dreamer* = *dreams* : *dreamers*], but can be more subtle, as in our first example [***Th****is guy dri****nk****s too much* : *This boat sinks* = *These guys drank too much* : ***Th****ese boats sa****nk***] where the diagonal terms share some n-grams reminiscent of the number (***Th****is*/***Th****ese*) and tense (*dri****nk***/*dra****nk***) commutations involved.

We thus propose a sampling strategy (hereafter EV) which selects x-forms that share with t some sequences of characters. To this end, input forms are represented in a vector space whose dimensions are frequent character n-grams, retaining the `k`-most frequent n-grams, where $n \in [\texttt{min};\texttt{max}]$. A form is thus encoded as a binary vector of

[1] The authors proposed to sample x and y among the closest forms in terms of edit-distance to $I(u)$.

[2] Anagram forms do not have to be considered separately.

dimension k, in which ith coefficient indicates whether the form contains an occurrence of the ith n-gram.[3] At runtime, we select the N forms that are the closest to a given form *t*, according to a distance[4]. Figure 1 illustrates some forms selected by this process. For comparison purposes, we also tested a sampling strategy which consists in selecting the *x*-forms that are closest to the source form *t*, according to the usual edit-distance (hereafter ED).

establish a report – order to establish a – has tabled this report – is about the report – basis of the report – other problem is that – problem that arises – problem is that those

Figure 1: The 8 nearest neighbors of *to establish a report* in a vector space computed from an input space of over a million phrases.

3 The tree-count data-structure

A tree-count is a tree which encodes a set of forms. Nodes are labeled by an alphabetical symbol and contain a (possibly empty) set of pointers to forms. A vertice from a node n labeled c to a node m is weighted by the count of c in the forms encoded by m, that is, the set of forms that can be reached from this node and its descendants. Thus, a path in a tree-count represents a set of constraints on the counts of the characters encountered along this path. This structure allows for instance the identification of anagrams in a set of forms: it suffices to search the tree-count for nodes that contain more than one pointer to forms in the vocabulary.

An example of a tree-count is provided in Figure 2 for a small set of forms. The node double circled in this figure is labeled by the character d and encodes the 6 input forms that contain 1 occurrence of *'o'* and 1 occurrence of *'s'*. One form is *os*, referenced by the pointer *m*, the other five forms are found by descending the tree from this node; among which *gods* and *dogs*, two anagrams encoded by the leave which set of pointers is b, k.

3.1 Construction time

The construction of a tree-count from a set of forms only needs an arbitrary order on the characters of the alphabet. This is the order in which we will encounter them while descending the tree. The lack of space prevents us to report the construction algorithm (see (Langlais and Yvon, 2008)), but it is important to note that it only involves a simple traversal of the input forms and is therefore time efficient. Also worth mentioning, our construction procedure only stores necessary nodes. This means that when enumerating characters in order, we only store zero-count nodes as required. As a result, the depth of a tree-count is typically much lower than the size of the alphabet.

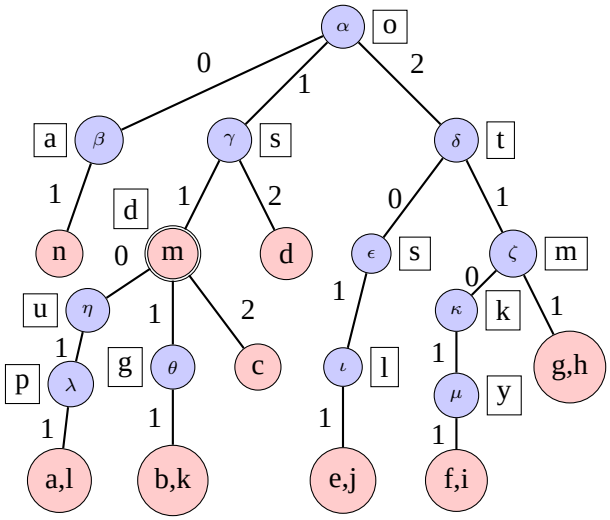

Figure 2: The tree-count encoding the set: {*soup*(a), *gods*(b), *odds*(c), *sos*(d), *solo*(e), *tokyo*(f), *moot*(g), *moto*(h), *kyoto*(i), *oslo*(j), *dogs*(k), *opus*(l), *os*(m), *a*(n)}. The character labeling a node is represented in a box; the counts of each character labels each vertice. Roman letters in nodes represent pointers to input forms; greek symbols label internal nodes.

3.2 Retrieval time

The retrieval of $\mathcal{C}(\langle x, t\rangle)$ can be performed by traversing the tree-count while maintaining a *frontier*, that is, the set of pairs of nodes in the tree-count that satisfy the constraints on counts encountered so far. Imagine, for instance, that we are looking for the pairs of forms that contain exactly 3 occurrences of characters *o*, 2 of characters *s* and 1 character *l*, and no other character. Starting from the root node labelled by *o*, there is only one pair of nodes that satisfy the constraint on *o*: the frontier is therefore $\{(\gamma, \delta)\}$. The constraint on *s* leads to the frontier $\{(m, \iota)\}$ (since the count of *t* must be null). Finally, descending this node yields the frontier $\{(m, (e, j))\}$, which identifies the pairs $(os, solo)$ and $(os, oslo)$ to be the only

[3] Typical values are min=max=3 and k=20 000.
[4] We used the Manhattan distance in this study.

ones satisfying the initial set of constraints.

The complexity of retrieval is mainly dominated by the size of the frontier built while traversing a tree-count. In practice, because of the sparsity of the space we manipulate in NLP applications, retrieval is also a fast operation.

4 Checking for an analogy

Stroppa (2005) provides a dynamic programming algorithm for checking that a quadruplet is an analogy, whose complexity is $o(|x| \times |y| \times |z| \times |t|)$.[5] Depending on the application, a large number of calls to this algorithm must be performed during step 1 of analogical learning. The following property helps cutting down the computations:

$$
\begin{aligned}
&[x : y = z : t] \Rightarrow \\
&\quad (x[1] \in \{y[1], z[1]\}) \ \vee \ (t[1] \in \{y[1], z[1]\}) \\
&\quad (x[\$] \in \{y[\$], z[\$]\}) \ \vee \ (t[\$] \in \{y[\$], z[\$]\})
\end{aligned}
$$

where $\bullet[\$]$ denotes the last character of $\bullet$. A simple and efficient trick consists in calling the analogy checking routine only for those triplets that pass this test.

5 Discussion

We investigated the aforementioned search strategies by translating 1 000 new words (resp. phrases) thanks to a translation table populated with pairs of words (resp. pairs of phrases). We studied the scalability of each strategy by varying the size of the transfer table (small, medium, large). Precise figures can be found in (Langlais and Yvon, 2008); we summarize here the main outcomes.

On the *word*-task, we compared the tree-count search strategy to the LP one. On the largest word-set (84 000 input words), the former (exact) strategy could find an average of 746 input analogies for 964 test-words at an average response time of 1.2 seconds per word, while with the latter strategy, an average of 56 analogies could be identified for 890 test-words, in an average of 6.3 seconds.

On the *sequence*-task, where input spaces are much larger, we compared the various sampling strategies presented in Section 2.3. We set N, the number of sampled input forms, to 10^3 for all sampling strategies. On the medium size dataset (293 000 input phrases), both ED and RAND perform badly compared to EV. With the two former filtering strategies, we could at best identify 6 input analogies for 38% of the test-phrases (at an average response time of 9 seconds), while with EV, an average 34 analogies could be identified for 75.2% of the test-phrases (in 3 seconds on average).

Finally, we checked that the approach we proposed scales to very large datasets (several millions of input phrases), which to the best of our knowledge is simply out of the reach of existing approaches. This opens up interesting prospects for analogical learning, such as enriching a phrase-based table of the kind being used in statistical machine translation.

[5] In this study, we used the definition of a formal analogy provided by Stroppa and Yvon (2005). Lepage (1998) proposes a less general definition, which is faster to check.

Acknowledgment

This study has been accomplished while the first author was visiting Télécom ParisTech.

References

Aha, David A. 1997. Editorial. *Artificial Intelligence Review*, 11(1-5):7–10. Special Issue on Lazy Learning.

Denoual, Etienne. 2007. Analogical translation of unknown words in a statistical machine translation framework. In *Machine Translation Summit, XI*, Copenhagen, Sept. 10-14.

Langlais, Philippe and Alexandre Patry. 2007. Translating unknown words by analogical learning. In *EMNLP-CoNLL*, pages 877–886, Prague, Czech Republic, June.

Langlais, Philippe and François Yvon. 2008. Scaling up analogies. Technical report, Télécom ParisTech, France.

Lepage, Yves and Étienne Denoual. 2005. Purest ever example-based machine translation: Detailed presentation and assessment. *Machine Translation*, 29:251–282.

Lepage, Yves. 1998. Solving analogies on words: an algorithm. In *COLING-ACL*, pages 728–734, Montreal, Canada.

Pirrelli, Vitto and François Yvon. 1999. The hidden dimension: a paradigmatic view of data-driven NLP. *Journal of Experimental & Theroretical Artifical Intelligence*, 11:391–408.

Stroppa, Nicolas and François Yvon. 2005. An analogical learner for morphological analysis. In *9th Conf. on Computational Natural Language Learning (CoNLL)*, pages 120–127, Ann Arbor, MI, June.

Stroppa, Nicolas. 2005. *Définitions et caractérisations de modèles à base d'analogies pour l'apprentissage automatique des langues naturelles*. Ph.D. thesis, ENST, Paris, France, Nov.