Experiments in Learning to Solve Formal Analogical Equations

Rhouma, Rafik and Langlais, Philippe

RALI,
Université de Montral, DIRO
C.P. 6128, Succ Centre-Ville
H3C3J7 Montréal
Québec, Canada
felipe@iro.umontreal.ca
http://rali.iro.umontreal.ca

Abstract. Analogical learning is a lazy learning mechanism which maps input forms (e.g. strings) to output ones, thanks to analogies identified in a training material. It has proven effective in a number of Natural Language Processing (NLP) tasks such as machine translation. One challenge with this approach is the solving of so-called analogical equations. In this paper, we investigate how structured learning can be used for learning to solve formal analogical equations. We evaluate our learning procedure on several test sets and show that we can improve upon fair baselines.

Keywords: Natural Language Processing, Formal Analogy, Solving Analogical Equation

1 Introduction

A *proportional analogy* (or analogy for short) — noted $[x : y :: z : t]$ — is a 4-uple of entities which reads "x is to y as z is to t", as in $[Paris : France :: Roma : Italy]$. In this work, we concentrate on *formal analogies*, that is, analogies at the formal or graphemic level, such as $[weak : weaker :: clean : cleaner]$.

Identifying proportional analogies is one core element of *analogical learning*, a learning strategy that can be explained as follows. Given a training set of pairs of input and output forms $\mathcal{D} \equiv \{(x, x')\}$, and an unknown input form u, analogical learning produces output entities u' by searching input elements in $\mathcal{D}$ that define with u an analogy $[x : y :: z : u]$. Those analogies are assumed to carry over the output space; that is, u' should be a solution of a so-called *analogical equation* $[x' : y' :: z' :?]$, where x', y', z' are output forms corresponding in the training material to x, y and z respectively.

Let us illustrate those concepts by an example taken from [11] where the authors applied analogical learning to translate terms of the medical domain. Assume we have a training set of terms in Finnish along with their translation into English: $\mathcal{D} = \{$*(beeta-agonistit, adrenergic beta-agonists)*, *(beetasalpaajat, adrenergic beta-antagonists)*, *(alfa-agonistit, adrenergic alpha-agonists)*$\}$.

We might translate the (unseen at training time) Finnish term *alfasalpaajat* into English by:

1. identifying the input analogy:
   [*beeta-agonistit* : *beetasalpaajat* :: *alfa-agonistit* : *alfasalpaajat*]
2. projecting it into the equation:
   [*adrenergic beta-agonists* : *adrenergic beta-antagonists* ::
   adrenergic alpha-agonists : ?]
3. and solving it: *adrenergic alpha-antagonists* is one solution.

This learning paradigm has been tested in a number of NLP tasks, including grapheme-to-phoneme conversion [21], machine translation [13, 10, 15], transliteration [5, 9], unsupervised morphology acquisition [19], as well as parsing [14, 2]. It has also been used with some success to inflate training material, in tasks where we lack training data, as in [1] for hand-written character recognition and in machine translation [20].

One essential component of an analogical device is an algorithm for solving analogical equations. In [18], the authors observed that learning embeddings of words on large quantities of texts captures analogical regularities (both semantic and formal) that can be used for solving an analogical equation. One distinctive characteristic of the solvers we consider in this study is that they can produce forms never seen at training time, while the approach of [18] can only propose forms for which an embedding has been trained (by exploiting huge quantities of data). On the other hand, our solvers can only deal with formal analogies, which is a limitation.

There are several operational analogical solvers on forms. Notably, in [12], Lepage proposes an algorithm which aligns two by two (like an edit-distance alignment) the forms of an equation. Those alignments are in turn used to guide the production of a solution. In [19], the authors propose a definition of formal analogy which lends itself to a solver that may be implemented by a finite-state automaton. Both solvers have the advantage that no training is required for the solver to be deployed. On the other hand, they both produce several solutions, among which typically only one is valid.

In this paper, we study the averaged structured perceptron algorithm [3] for learning to solve analogical equations on forms. We present in Section 2, two very different state-of-the-art solvers we compare against. We describe in Section 3 the structured learning framework we deployed. We present in Section 4 two datasets we used for training and testing solvers. We report our results in Section 5 and conclude in Section 6.

2 Reference Solvers

2.1 Mikolov et al. (`word2vec`)

In [18], the authors discovered that the vector space induced by `word2vec`, a popular toolkit for computing word embeddings, has the interesting property of

preserving analogies, that is, the difference of the vector representation of two words x and y in an analogy $[x : y :: z : t]$ is a vector which is close to the difference of the vectors associated to the other two words z and t. Therefore, their approach to solve an analogical equation consists in computing:

$$\hat{t} = \underset{t \in V}{\operatorname{argmax}} \cos(t, z - x + y) \tag{1}$$

While this solver can handle both semantic and formal analogies, it can only produce solutions that have been seen at training time, which is of low interest in our case. It is also very slow to run since it requires to go over the full vocabulary of the application V in order to find the word with the best match. This would for instance hardly scale to a vocabulary composed of word sequences in a given language. Nevertheless, the ability of embedding methods to capture analogies, has received a lot of traction recently, leading to performances we can reproduce and compare against. We implemented Equation 1, making use of embeddings trained by the authors[1]. See Figure 1 for solutions produced by this solver.

`word2vec` ($d = 300$)	unreadable *0.574*, illegible *0.496*, scrawled *0.496*, scribbled *0.496*, executor *0.475*
`alea` ($\rho = 10$)	undabloe *4*, undableo *3*, unabldoe *2*, undoeabl *2*, unodable *2*
`alea` ($\rho = 50$)	**undoable** *63* undabloe *45*, undaoble *27*, dunoable *27*, unadoble *22*
`early` ($beam = 100$)	**undoable** *510.9*, undaoble *488.9*, undabole *488.9*, undabloe *488.9*, unadbloe *488.94*

Fig. 1. 5-first solutions produced by different solvers to the equation [*reader* : *unreadable* :: *doer* : *?*]. See the text for the details about the configurations. Note that `word2vec` ranks words in the vocabulary, while other solvers generate their solutions.

2.2 Langlais et al. (`alea`)

In [22], the authors proposed a very different solver which relies on the following theorem:

Theorem 1 *t is a solution to $[x : y :: z :?]$* ***iff*** $t \in \{y \circ z\} \setminus x$

where:

[1] Over 3 million vectors of dimension 300 for words seen at least 5 times; trained with the skip-gram model on the large Google news corpus.

$\boldsymbol{w} \circ \boldsymbol{v}$ the *shuffle* of w and v, is the regular language of the forms obtained by selecting (without replacement) alternatively in w and v, sequences of characters in a left-to-right manner. For instance, both strings *unreadodableer* and *dunoreaderable* belong to *unreadable* $\circ$ *doer*.

$\boldsymbol{w} \backslash \boldsymbol{v}$ the *complementary set* of w with respect to v, is the set of strings formed by removing from w, in a left-to-right manner, the symbols in v. For instance, *unodable* and *undoable* both belong to *unreadodableer* $\setminus$ *reader*.

This theorem states that we can built a finite-state machine that recognizes the set of all the solutions of an analogical equation. However, building such an automaton may face combinatorial issues, which makes this approach practical for analogies involving short enough forms. The algorithm described in [11] that we implemented in this work consists in sampling randomly ρ elements of the shuffle language, then computing the complementary operation. This way, the automaton is never constructed, leading to a very time efficient algorithm. On the other hand, the solver may fail to deliver the correct solution if the number of shuffles considered (ρ) is to small. Since several combinations of shuffling and complementing operations may lead to the same solution, we can rank solutions in decreasing order of their frequency. See Figure 1 for the solutions produced by two configurations of this solver.

3 Structured Learning

Given a function $g : \mathcal{I} \times \mathcal{O} \rightarrow \mathbb{R}$ which evaluates a fit between an object i in an input domain $\mathcal{I}$, to an object in a structured output domain $\mathcal{O}$, we seek to find:

$$\hat{t} = \underset{t \in \mathcal{O}}{\operatorname{argmax}}\, g(i, t) \tag{2}$$

In this work, input objects are triplets of strings $i \equiv (x, y, z)$ over an alphabet $\mathcal{A}$ and output objects are strings over this alphabet. We assume a linear model for $g = \langle \mathbf{w}, \Phi(i, t) \rangle$ parametrized by a feature vector $\mathbf{w}$ in $\mathbb{R}^K$ and a feature map $\Phi(i, t)$ decomposed into K binary feature fonctions $\phi_k : (i, t) \rightarrow \{0, 1\}$ controlled by the scalar $\mathbf{w}_k$. The vector $\mathbf{w}$ defines the parameters of the model we seek to adjust in order to maximise the quality of predictions made over a training set $\mathcal{D} = \{((x, y, z), t)\}$. In this work, we use variants of the averaged structured perceptron algorithm [3] for doing so, that we sketch hereafter.

3.1 Average Structured Perceptron

The standard version (`standard`) of the averaged structured perceptron algorithm is depicted in Figure 2. The algorithm is based on the assumption that the inference (argmax) can be computed exactly, which is often impractical. In [6], the authors demonstrate that in cases of inexact search (our case), we should only make *valid updates*, that is, updates where the 1-best hypothesis has a higher model score than the correct sequence. There are several strategies

$\mathbf{w}, \mathbf{w}_a \leftarrow \mathbf{0}$
$e \leftarrow 0$
repeat
  $e \leftarrow e + 1$
  for all example $(i, t) \in D$ **do**
    $\hat{t} = \text{argmax}_y \, \mathbf{w}^T \Phi(i, t)$
    if $\hat{t} \neq t$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \Phi(i, t) - \Phi(i, \hat{t})$
      $\mathbf{w}_a \leftarrow \mathbf{w}_a + \mathbf{w}$
until converged
return $\mathbf{w}_a / e.|\mathcal{D}|$

Fig. 2. Standard averaged structured perceptron algorithm.

for this. One solution (`safe`) consists in conducting the update after checking it is valid. While this variant is guaranteed to converge, it typically throws too many training examples. Another solution initially suggested in [4] consists in updating whenever the reference solution is not anymore attainable from the current search space, in which case the hypothesis with the largest score so far is being used for the update. This variant known as early update (`early`), has the drawback that only a fraction of an example is concerned by the update, leading to longer training times. Other alternatives are proposed in [6]; notably a variant (`late`) which selects the deepest node in the search space which is a valid update (the hypothesis with the largest prefix p). This way, the update is conducted on larger strings.

3.2 Search

In what follows, $x[i]$ stands for the ith symbol of string x,[2] $|x|$ is the length (the number of symbols) of x, $x[i:]$ designates the suffix of x starting at the ith symbol; and $x.y$ designates the concatenation of x with y.

For solving an equation $[x : y :: z :?]$, our structured solver explores a search space in which a state is represented by a 5-uple $< s, i, j, k, p >$, meaning that $x[i:]$, $y[j:]$ and $z[k:]$ are yet to be visited, that p is the current prefix of a solution, and that s is the sequence of the shuffle being considered. The initial state of the search space is $< \epsilon, 0, 0, 0, \epsilon >$ (where ϵ designates the empty string) and goal states are of the form $< \epsilon, |x|, |y|, |z|, sol >$, where sol is a solution to the equation. There are three actions X, Y, and Z that can be applied to a given state and which are described in Figure 3. It is worth noting that action X, the action which implements the complementation operation, is the only one which contributes to add symbols to the solution being generated. This is rather different from typical search spaces, where most actions do impact immediately the solution produced, as for instance in machine translation.

To further illustrate the singularity of the search procedure, let us consider nodes $n_1 = < uu, 0, 1, 1, \epsilon >$ that is generated from the initial one after two

[2] The first valid index is 0.

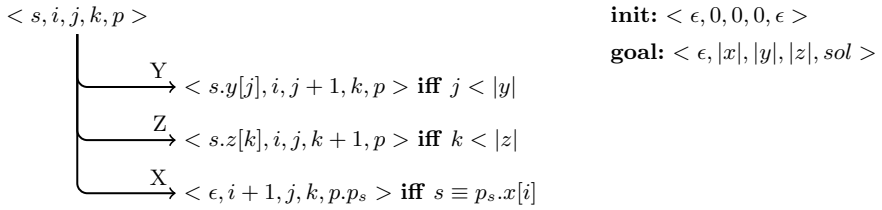

Fig. 3. The three operations defining the search space of the structured learning solvers.

consumption operations took place (Y and Z, in whatever order), and node $n_2 = <$ *undou*$, 0, 4, 1, \epsilon >$, that is reached after four Y and one Z operations. Three operations may expand n_1, that are illustrated in Figure 4 (left part), while two expansions are possible from n_2 (right part). The X and Y operations are just reading one symbol in either y or z respectively, adding the read symbol to the shuffle. Since the y string has been entirely read in the n_2 configuration, only Z is considered. Because in both n_1 and n_2 the shuffle ends with u the symbol in $x[0]$, a complementary operation X is possible from both nodes. In the second configuration, for instance, the shuffle is *undou*, which complementation with u leads to the sequence *undo* being generated in the prefix of the resulting node. Since X is the only operation that generates symbols of the solution, the search space is populated with a lot of nodes with an empty prefix (5 out of the 7 nodes in Figure 4).

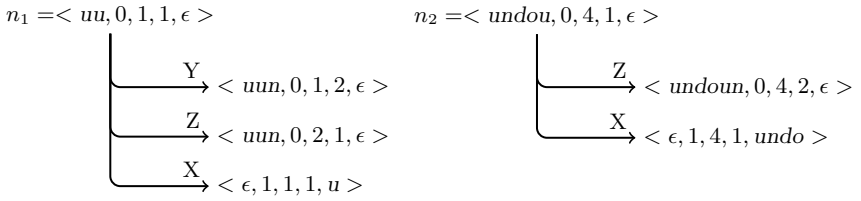

Fig. 4. Expansions of two nodes belonging to the search space built to solve the equation [*unread* : *undo* :: *unreadable* : *?*].

As often in problems of interest, the search space is too huge for a systematic exploration, and heuristics have to be applied in practice. First, the search space is organized as a graph, which avoids developing an hypothesis twice. This safely reduces the search space, without sacrifying optimality. For instance, the node resulting from n_2 in Figure 4 after an X operation may also be generated from the initial state by applying in that order the sequence of operations: Z, Y, X, Y, Y, and Y. Second, we deploy a beam-search strategy to prune less promising hypotheses. Because of the specificity of the search space, the comparison of

hypotheses that lead to the same prefix p is difficult, and we had to resort to a sophisticated beam policy (controlled by a metaparameter *beam*) which details are beyond the scope of this paper. But suffices it to say that in order to avoid filtering too many hypotheses, we have to resort to a third filtering strategy which consists in enforcing that at most η actions Y or Z happen before an action X occurs. The metaparameter η controls the number of hypotheses that can grow without generating a symbol of the solution. We experiment with values of this metaparameter in Section 5.

3.3 Features

As mentioned earlier, the feature map is defined by a number of binary feature functions that apply to any information available in a search node $< s, i, j, k, p >$ created while solving the equation $[x : y :: z :?]$. In order to guide the search we rely on 3 families of features:

language model (14 features) because our solver produces a subset of permutations of the same form as a solution, ranking those solutions with a language model will favor hypotheses with a prefix that is likely in a given language. We compute the likelihood of a prefix p according to an n-gram language model trained on a large set of external data. Binary features check that the likelihood falls into specific predetermined range of probabilities. We also have features of the form $prob(p_i|p_{i-1} \dots p_{i-n+1}) < \delta$ that fire whenever a symbol of p is predicted with less probability than δ. We also deploy similar features for the shuffle s under consideration.

edit-distance (20 features) A solution to a formal equation $[x : y :: z :?]$ typically shares sequences of symbols of y and z. For instance in [*reader* : *unreadable* :: *doer* : *?*], the solution *undoable* shares with *doer* the affix *do*. We compute the edit-distance between the prefix p and the forms y and z with the intuition that solutions that most ressemble one of those strings are more likely to be good ones. Edit-distances are transformed into binary fonctions (binning into 10 intervals).

search-based (20k features) We compute features specific to the search space visited. We measure the percentage of consumption in each form x, y, and z. We also have features to capture the last operation taken, thus providing a first-order Markovian information. We also compute the total number of consecutive shuffling actions (Y or Z) taken so far. This feature might help the learning mechanism to favor a complementarity operation if too many shuffling operations took place. We also have features that record the value of each index (we have a binary feature for each possible value).

On top of this, we also compute a number of binary features for capturing whether specific configurations of symbols have been observed in the search space visited when enforcing the production of the reference solution to an equation (forced-decoding). We compute the following features:

- a binary feature for each possible 3-tuple $(x[i], y[j], z[k])$,

– a binary feature for each bigram at the end of the shuffle s,
– a binary feature for each bigram at the end of the prefix p.

The description of those features is not intended for others to reproduce our experiments precisely, but instead to provide the intuition behind each feature family. We have not conducted a systematic analysis of the usefulness of each feature, but have noticed that removing one family of features leads invariably to a significant loss in performance.

4 Datasets

4.1 Word Equations

In [17], the authors designed a comprehensive task for evaluating the propension of word embeddings to preserve analogical relations. It contains 19 544 analogies, categorized into 14 categories, including *capital-world* (*e.g.*, [*Dublin* : *Ireland* :: *Jakarta* : *Indonesia*]). Roughly 55% of those analogies are actually syntactic ones that capture various morphological phenomena in English (see Figure 5). Many of those syntactic analogies, are actually not formal ones. For example, [*rare* : *rarely* :: *happy* : *happily*] is not formal according to the definition of Yvon et al. [22] we use in this study,[3] because of the commutation of y in *happy* into i in the adverbial form. Therefore, we removed non formal analogies to build a corpus of 4977 analogies named `google` hereafter. We also used the `msr` dataset of 8k syntactic analogies[4], 3664 of which being formal ones.

Examples of analogies of both datasets are reported in Figure 5. Most analogies involve short word forms (6 to 7 characters on average) and are actually rather simple to solve (but see Section 5). We strengthen that because we filtered out all non formal analogies, we place ourselves in an optimistic scenario where the expected solution to an equation is reachable by our solvers, which only makes senses as a case study. We come back to this point later on.

4.2 Phrasal Analogies

Identifying formal analogies on phrases is actually not the kind of task a human would be willing to do extensively and systematically. One might easily produce analogies such as [*She loves Paul* : *He loves Paul* :: *She likes Mark* : *He likes Mark*], but it would rapidly become a daunting task to collect representative analogies, that is, analogies that capture a rich set of linguistic phenomena (such as the fact that the 3rd person of a verb at the present tense should end with a *s* in the example). Therefore, we resorted to an automatic procedure to acquire analogies.

[3] The definition immediately follows from Theorem 1.
[4] `http://www.marekrei.com/blog/linguistic-regularities-word-representations/`

`msr`	# of analogies: 3664	word's avr. length: 6
JJ-JJR	[*high* : *higher* :: *wild* : *wilder*]	
JJR-JJ	[*greater* : *greatest* :: *earlier* : *earliest*]	
JJS-JJ	[*low* : *lowest* :: *short* : *shortest*]	
NN-NNPOS	[*problem* : *problems* :: *program* : *programs*]	
VB-VBP	[*take* : *takes* :: *run* : *runs*]	
VB-VBD	[*prevent* : *prevented* :: *consider* : *considered*]	
NNPOS-NN	[*days* : *day* :: *citizens* : *citizen*]	
VBZ-VBD	[*believes* : *believed* :: *likes* : *liked*]	

`google`	# of analogies: 4977	word's avr. length: 7
adjective-adverbe	ADJ-ADV	[*amazing* : *amazingly* :: *serious* : *seriously*]
opposite	OPP	[*certain* : *uncertain* :: *competitive* : *uncompetitive*]
comparative	COMP	[*fast* : *faster* :: *bright* : *brighter*]
superlative	SUP	[*warm* : *warmest* :: *strange* : *strangest*]
present-participle	PP	[*code* : *coding* :: *dance* : *dancing*]
nationality-adverb	NAT	[*Australia* : *Australian* :: *Croatia* : *Croatian*]
past-tense	PAST	[*decreasing* : *decreased* :: *listening* : *listened*]
plural	PLUR	[*eye* : *eyes* :: *donkey* : *donkeys*]
plural-verbs	PL-VB	[*listen* : *listens* :: *eat* : *eats*]

Fig. 5. Main characteristics of the `msr` and the `google` datasets, and examples of formal analogies of each category.

For this, we first trained a phrase-based machine translation (SMT) system on the English-Spanish Europarl corpus,[5] using the Moses toolkit [8]. We collected millions of phrase associations such as those in Figure 6, and filtered in, those with a good association score ($prob \geq 0.1$). A subset of phrase pairs was elected as a reference $\mathcal{R}$, the remaining pairs being kept as a translation memory $\mathcal{M}$. Then, we applied an analogical learning translation device very similar to the one described in [13] for translating Spanish phrases of pairs $(u, u') \in \mathcal{R}$ into English. For a given form to translate, u, this system identifies $(x, x'), (y, y'), (z, z') \in \mathcal{M}$ such that $[x : y :: t : u]$, and solves equations $[x' : y' :: z' :?]$. Whenever a solution to such an equation produces u', we consider $[x' : y' :: z' : u']$ a useful analogie. We could have identified analogies in the English part directly, but we would have ended up with many *spurious* analogies, that is, true formal analogies that are simply fortuitous as [*croyons* : *crons* :: *montroyal* : *montreal*]. The assumption here is that while a spurious source analogy might be identified, it is very unlikely that its projection into the target language (English) leads to an equation for which the reference translation is a solution.

[5] `http://statmt.org`

```
a actualizar los acuerdos ||| to update the agreements ||| 1 0.00474847
a cambiar la base ||| to change the basis ||| 0.5 0.0035545
basado en el trabajo de ||| based on the efforts of ||| 1 2.02579e-05
```

Fig. 6. Phrases pairs collected by an SMT engine trained on the Spanish-English Europarl corpus. The format shows the source phrase (Spanish), the target phrase (English) and the first two scores estimating their likelihood of being in translation relation.

By applying this procedure, we collected many analogies: 10 000 of them where elected `simple` (we kept 1000 for training purposes, and the remaining ones for testing). For the remaining analogies, we solved with the `alea` solver the equations built by removing the forth form. We then split equations according to the rank of the reference translation (the forth term) among the ranked list of solutions proposed. We collected 400 analogies ranking in each of the following intervals : [1-5], [6-10], [11-20], [21-50] and [51-100], leading to a total of 2000 analogies (1000 for training, 1000 for testing). We qualify this dataset as `hard` in the sequel. As a matter of fact, 80% of analogies in the `simple` dataset have a degree[6] of 2 or 3, while this rate is only 20% for the `hard` dataset.

`simple` phrase's avr. length: 16

▷ [*international investigation* : *international democracy* :: *an international investigation* : *an international democracy*]
▷ [*young girls* : *training of young girls* :: *young girls and* : *training of young girls and*]
▷ [*political situation is viable* : *political situation is still* :: *the political situation is viable* : *the political situation is still*]

`hard` phrase's avr. length: 17

▷ [*adopted recently by* : *recently adopted by* :: *study published recently by* : *study recently published by*]
▷ [*competition and the* : *competition and against* :: *competition and of the* : *of competition and against*]
▷ [*their governments to* : *their governments are* :: *their governments and to* : *and their governments are*]

Fig. 7. Exemples of phrasal analogies automatically identified.

[6] The degree of an analogy roughly correlates with the number of commutations among strings involved; the higher the degree, the harder it is to solve the analogy.

5 Experiments

5.1 Metrics

Each solver produced solutions to the equations we built by removing the last form of each analogy in the datasets presented in the previous section. We evaluated their *accuracy* at identifying this form in first position. Since some variants may fail to retrieve a solution (search failure), we also report — when pertinent — *silence* as the ratio of test equations for which no solution is being generated.

5.2 Word Equations

In this experiment, we take benefits of the analogies of the `google` and `msr` datasets. We trained on one corpus, and tested on the other. Because both `alea` and our structured solvers require a metaparameter (ρ or the beam size), we report in Figure 8 the performance of different variants as a function of this metaparameter (that we varied from 1 to 20). We did not use the η metaparameter in this experiment, since equations involve short enough forms (6 characters on average), leading to search spaces manageable to visit entirely.

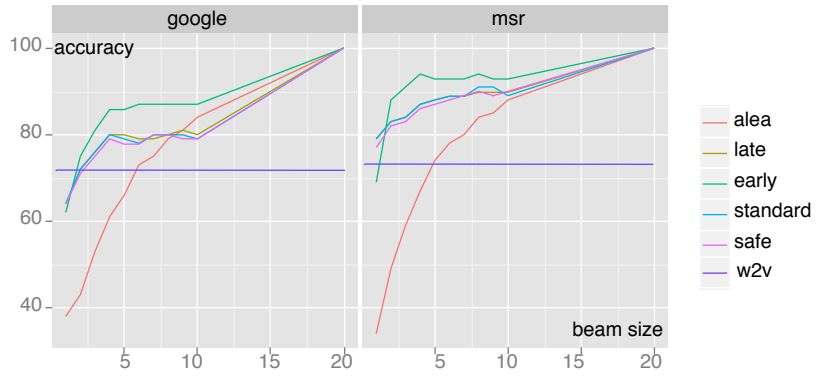

Fig. 8. Accuracy on `google` and `msr`, as a function of the beam size (or ρ for `alea`).

We observe that for both solvers, larger values of the metaparameter are preferable. For a value of 20, all solvers respond perfectly to all equations. Still, we observe that learning to solve equations leads to better accuracy overall, especially for low values of the metaparameter. That formal solvers produce the expected solution to all (formal) equations in the first place confirms that those equations are indeed very simple. Actually, most of them involve simple prefixation/suffixation operations (see Figure 5). It is worth pointing that the `word2vec` solver is registering an accuracy of 78% and 71% on `msr` and `google` respectively.

This is not a bad level of performance considering that the embeddings were not trained to capture such an information. But it also indicates that `alea` and the structured solvers are actually doing very well. Recall however, that we only consider formal equations here, which solutions are reachable by our solvers, while non formal ones are not. Last, we observe that solving the `msr` equations when training on the `google` analogies leads to slightly better results overall.

Figure 9 compares the accuracy of the `early` and the `word2vec` solvers on each category of equations for both datasets. The former solver systematically outperforms the latter, especially for a few categories such as NN-NNPOS in the `msr` dataset or ADJ-ADV in the `google` one. There are mainly two reasons for this. First, most nouns in English can be verbs as well, but each form of the vocabulary receives only one embedding, leading to some errors. Second, `word2vec` very often outputs terms that are semantically related to the solution expected. For instance, it produces the form *fantastic* to the equation [*cold* : *colder* :: *great* : *?*].

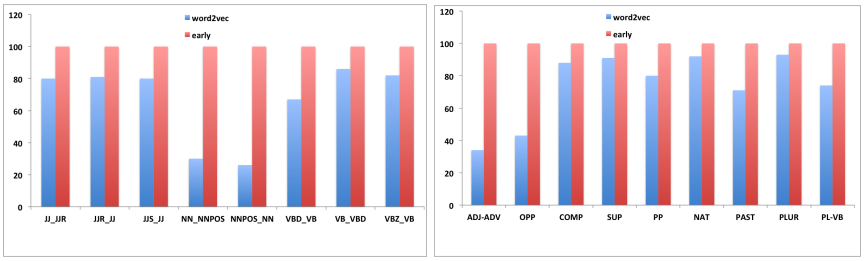

Fig. 9. Accuracy of the `word2vec` and the `early` solvers on formal analogies in the `msr` (left part) and the `google` (right part) datasets, detailed by categories of equations.

To put those figures in perspective, we report in Table 1 the performance of a solver that would consider the output of the structured perceptron (the `early` variant) whenever we face an equation that has a formal solution, and the output of `word2vec` otherwise. Since in practice, we cannot know wether an equation admits a formal solution, this simulation only provides a point of comparison with other works that report the performance of embedding-based approaches on the full data sets. This is for instance the case of the work of [16] where Levy et al. compare many variants of distributional approaches, the best performing one recording a slightly better accuracy than our combination (third line of Table 1). That we compare to state-of-the-art results without much adjustments suggests that our structured solver is indeed very apt at solving formal equations.

5.3 Phrasal Equations

We trained variants of the structured perceptron using a beam size of 100, and a value of the metaparameter η of 7. Since a phrase has an average of 16 symbols,

Table 1. Accuracy of different solvers on the full `google` and `msr` datasets. The last line is the best performance reported by the authors of [16].

	`msr`	`google`
`word2vec`	67%	63%
`early+word2vec`	72%	71%
Levy et al., 2015	72.9%	75.8%

it roughly means that we enforce the solver to consume no more of 20% of the shuffle of y and z before a complementary operation (X) takes place. While we initially trained our solver on the training set the most similar to the test material, we also considered variants training of `hard` and testing on `simple`, or the reverse. Of course, we took care at construction time to ensure no overlap between training and test sets (see Section 4). Those configurations are compared to the `alea` solver where ρ was set to 1000, a conservative setting that leads the solver to always propose a solution. Results are presented in Table 2.

Table 2. Accuracy and silence rate (in parenthesis) of different configurations of structured solvers ($\eta = 7$) compared to the `alea` solver ($\rho = 1000$) on phrasal equations. Structured solvers have been trained on 10 epochs.

test	`simple`		`hard`	
train	`simple`	`hard`	`simple`	`hard`
`standard`	30.6 (7)	30.1 (10)	**19.6** (17)	**24.0** (56)
`early`	**38.4** (8)	27.6 (10)	**26.0** (6)	**22.9** (7)
`late`	26.9 (9)	20.3 (11)	**18.9** (9)	**20.1** (6.7)
`safe`	25.4 (8)	25.6 (13)	14.6 (18.7)	**21.0** (56)
`alea`	33 (0)		18 (0)	

On the `simple` test set, only one configuration managed to outperform `alea`: equations are easier to solve than those of the `hard` test set, and the latter solver already achieves a decent job. Still the `early` variant managed to outperform `alea`, which is encouraging. It is also clear that it is far much preferable to train our models on `simple`. We observe that at best, we could solve correctly at rank 1 only 38% of the equations of the `simple` test set: obviously, solving equations on longer sequences is more challenging, than solving equations on words, as typically done in the literature. This is also consistent with [7] in which the authors trained a neural network for solving analogies, but reported a failure of their model to solve analogies involving long forms.

On the `hard` test set, the structured solvers deliver a much better accuracy than the `alea` solver ($\rho = 1000$) which accuracy platoes at 18%. Expectedly, the accuracy of solvers on `hard` is much lower than the one measured on `simple`. We also observe that it is overall preferable to train our solvers on `hard`.

That the best configurations overall are recorded when training on data sets similar (in terms of difficulty) to the test material is disappointing, although we anticipated it. This means in practice that care must be taken to prepare an adequate training set. Understanding good practices for doing so is an open issue.

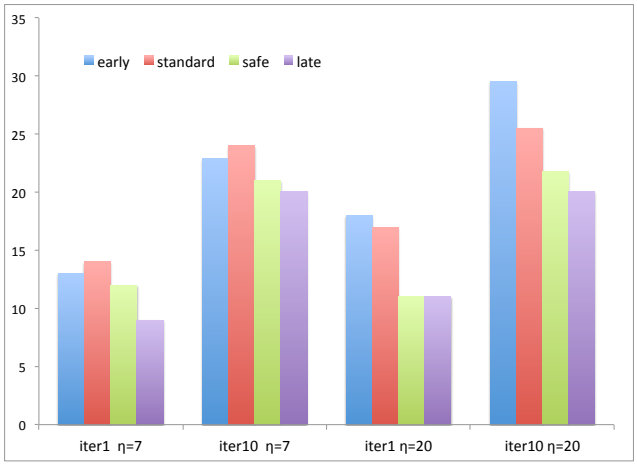

Fig. 10. Accuracy on the `hard` test set of different solvers after 1 and 10 epochs, and for 2 values of the metaparameter η.

Figure 10 investigates the impact of the metaparameter η and the number of epochs used to train the solver. Four configurations for each variant we considered are evaluated on the `hard` test set. Training over 10 epochs (as done for the results reported in Table 2) is expectedly preferable to training only on one. Increasing the value of η seems to impact performance positively, but increases the size of the search space, leading to higher time response. It seems overall that `early` and `standard` are the best variants of the structured perceptron, at least on `hard`. The silence rate of the `standard` approach is very high and around 50%. This suggests that our pruning strategy eliminates from the search space hypotheses that should not, which is a problem. The silence rate of the `easy` variant is however much lower: 7% for $\eta = 7$, and 13% for $\eta = 20$. That it is higher for largest values of η simply suggests that there is an interplay between metaparameters that control the search.

6 Discussion

We have presented experiments for learning to solve an analogical equation thanks to structured learning. On formal word equations, our trained solvers achieve perfect performance, as does the `alea` solver of [11]. On phrase equations, the performance of our learning mechanism is lower, but still superior to the `alea` solver. Our approach requires example analogies for training. Therefore we proposed a methodology for acquiring those analogies from a parallel corpus, without supervision, but leveraging a parallel corpus and a statistical phrase-based translation model.

We are currently investigating the impact of training formal solvers on more epochs. We also plan to investigate more systematically the interplay between some metaparameters, as well as the usefulness of the all the features we considered in this work. Preliminary results indicate that better performance can be obtained with less features. Last, we must compare our approach to the one of [7]: while we do report higher results on equations involving long strings, an end-to-end comparison is required.

Acknowledgments. This work has been partly funded by the Natural Sciences and Engineering Research Council of Canada. We thank reviewers for their constructive comments.

References

1. Bayoudh, S., Mouchère, H., Miclet, L., Anquetil, E.: Learning a classifier with very few examples: Analogy based and knowledge based generation of new examples for character recognition. In: 18th ECML. pp. 527–534 (2007)
2. Ben Hassena, A.: Apprentissage analogique par analogie de structures d'arbres. Ph.D. thesis, Univ. de Rennes I, France (2011)
3. Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: EMNLP. pp. 1–8 (2002)
4. Collins, M., Roark, B.: Incremental parsing with the perceptron algorithm. In: 42nd ACL (2004)
5. Dandapat, S., Morrissey, S., Naskar, S.K., Somers, H.: Mitigating problems in analogy-based ebmt with smt and vice versa: a case study with named entity transliteration. In: PACLIC. Sendai, Japan (2010)
6. Huang, L., Fayong, S., Guo, Y.: Structured perceptron with inexact search. In: NAACL. pp. 142–151 (2012)
7. Kaveeta, V., Lepage, Y.: Solving analogical equations between strings of symbols using neural networks. In: Workshop on Computational Analogy at ICCBR 2016. pp. 67–76 (2016)
8. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open Source Toolkit for Statistical Machine Translation. In: 45th ACL. pp. 177–180 (2007), interactive Poster and Demonstration Sessions
9. Langlais, P.: Mapping source to target strings without alignment by analogical learning: A case study with transliteration. In: 51st ACL. pp. 684–689 (2013)

10. Langlais, P., Patry, A.: Translating Unknown Words by Analogical Learning. In: EMNLP. pp. 877–886. Prague, Czech Republic (2007)
11. Langlais, P., Yvon, F., Zweigenbaum, P.: Improvements in Analogical Learning: Application to Translating multi-Terms of the Medical Domain. In: 12th EACL. pp. 487–495. Athens (2009)
12. Lepage, Y.: Solving analogies on words: an algorithm. In: COLING-ACL. pp. 728–733. Montreal, Canada (1998)
13. Lepage, Y., Denoual, E.: Purest ever example-based machine translation: Detailed presentation and assesment. Mach. Translat 19, 25–252 (2005)
14. Lepage, Y., Shin-ichi, A.: Saussurian analogy: A theoretical account and its application. In: 7th COLING. pp. 717–722 (1996)
15. Letard, V., Illouz, G., Rosset, S.: Reducing noise sensitivity of formal analogical reasoning applied to language transfer. In: Workshop on Computational Analogy at ICCBR 2016. pp. 87–97 (2016)
16. Levy, O., Goldberg, Y., Dagan, I.: Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics 3, 211–225 (2015)
17. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013)
18. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: 26th NIPS. pp. 3111–3119 (2013)
19. Stroppa, N., Yvon, F.: An analogical learner for morphological analysis. In: 9th CONLL. pp. 120–127. Ann Arbor, USA (2005)
20. Yang, W., Lepage, Y.: Inflating a small parallel corpus into a large quasi-parallel corpus using monolingual data for chinese–japanese machine translation. Journal of Information Processing (Information Processing Society of Japan) (2017)
21. Yvon, F.: Paradigmatic cascades: a linguistically sound model of pronunciation by analogy. In: 35th ACL. pp. 429–435 (1997)
22. Yvon, F., Stroppa, N., Delhay, A., Miclet, L.: Solving analogies on words. Tech. Rep. D005, École Nationale Supérieure des Télécommuncations, Paris, France (2004)