Comparing Unsupervised Machine Translation Strategies using Word2Vec

II2202, Fall 2015

Bram Leenders
KTH, Royal Institute of Technology
Brinellvägen 8, 114 28 Stockholm
Sweden
b.c.leenders@gmail.com

Marc Romeyn
KTH, Royal Institute of Technology
Brinellvägen 8, 114 28 Stockholm
Sweden
marc.romeyn@gmail.com

Keywords
Word2Vec, Machine Translation

1. INTRODUCTION
The Internet offers a vast amount of natural language that can be used in natural language processing, for a very low price. A study by Buck et al. [3] estimated that each of the top 10 most frequently used languages on the internet has at least 250 GiB worth of text publically available online. For English (the most frequent), they even found 23 TiB of text.

Such amounts of text have a big potential to be used for the training of language models, but only if building these models can be done in an unsupervised fashion. Manually curating, marking and tagging text is far too much work to be feasible. With unsupervised algorithms, however, the structure in language can be exploited to let computers build language models.

This research will focus on unsupervised training of computer models to translate words. Specifically, we focus on the use of word2vec [7] to provide translations.

WE COULD EXPAND A BIT HERE, AND REVISIT AFTER WRITING MORE OF THE OTHER PARTS

1.1 Background and Related Work
Since the introduction of word2vec [7, 9] in 2013, the algorithm has seen a wide variety of usecases. In the initial paper [7], Mikolov et. al describe interesting relations between vectors corresponding to words. A famous example of how word2vec models relations between words as mathematical equations is $king - man + woman = queen$. The sematic relationships between man/woman and king/queen are preserved in the transformation of words to vectors, and can be expressed with basic algebra.

Subsequent papers have improved the algorithm both in terms of accuracy ([6]), performance, parallelization and extended the initial scope of applications. A good example of the latter is a paper by Boycheva [2], which uses word2vec outside the natural language processing (NLP) domain but to generate playlists. Based on a set of playlists, their word2vec-based algorithm can suggest new playlists with artists that go well together.

One of the applications of word2vec inside the NLP domain, is exploiting similarities in languages for assistance in machine translation [11]. Mikolov et al. [8] released a subsequent paper on word2vec, in which they describe similarities between models of different languages. An example they give, is how the usage of the numbers one to five in English is very similar to the usage in Spanish, and likewise for the names of animals. Figure 1 shows a graphical representation of word vectors in English and Spanish.

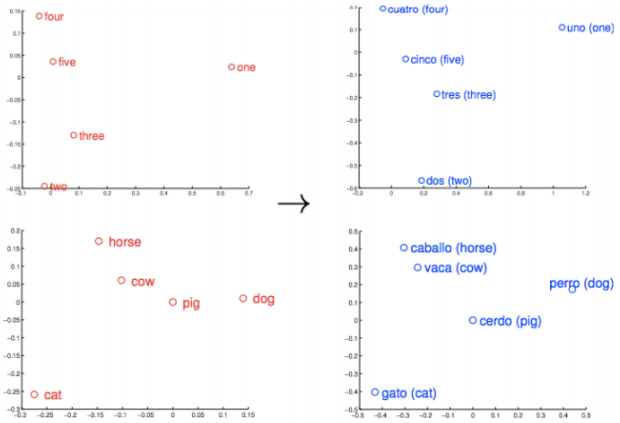

Figure 1: Vector representations of English and Spanish words, after dimensionality reduction and rotation. Notice the high level of similarity between both languages. Reprinted from Mikolov et al. [8]

The similarities between languages can be used to predict translations for words without any human interaction or labeled input data. Using only unsupervised machine learning techniques, a computer could learn how to translate English to for instance Spanish and vice versa. The only requirement is a large amount of text in both languages to train the word2vec models on.

In this research, we will focus on this specific application of word2vec: using similarities in languages to provide translations of words.

It is important to note that word2vec only uses information

of co-occurrences to model words. It does not learn grammatical concepts other than by statistical analysis. This limits our translation to single words; although the translator might be able to translate each word individually, it cannot learn that each finite verb must has a subject, that "we" is plural, etc. It will learn that "swim" is to "swimming" as "walk" is to "walking", but will not know that "swimming" is a gerund. Note that word2vec can be extended to sentences or whole documents as proposed by Le et al. [5] but this will be out of scope for our research.

1.2 Our Contribution

This research aims to improve the capabilities of machines to learn translations with minimal human intervention. Previous research [8, 11] has already shown potential for word2vec in the context of automatic translation, as discussed in section 1.1.

However, we found no practical implementations using Word2Vec and no further research on different setups for word2vec based machine translation.

1.3 Outline

BRIEFLY EXPLAIN STRUCTURE OF PAPER

2. WORD2VEC

Word2vec is an algorithm that computes vector representations of words based on co-occurrences. The spatial distance between two word-vectors corresponds to word similarity. In order to achieve this there two different algorithms: skip-gram and continuous bag of words (CBOW). Skip-gram is the most popular choice because it scales better to bigger datasets and therefore also chosen for our research.

Skip-gram model is an method for learning distributed vector representations that capture a large number of syntactic and semantic word relationships [9]. Given a a word w, the skip-gram model predicts the n neighboring words.

Word2vec is an example of so-called "shallow" learning and can be trained as a simple neural network. This neural network has a single hidden layer with no non-linearities and no unsupervised pre-training of layers is needed [10].

Since word2vec is simplified, its scope of application is more limited than deep neural networks [1]. However, training is far more efficient due to the lower amount of layers and simpler functions.

3. MULTI-MODEL TRANSLATIONS

In this section, we explain a technique for translating words using seperate models for the input and output language. The techniques described here are originally published by Mikolov et al. [8].

3.1 Translation

Given a word in language A that we want to translate to language B, the first step is to find the vector representation of the word. We do this by looking up the word in a word2vec model trained on language A.

Translating is now a matter of mapping vector representations from model A to corresponding vectors in model B. We do this by multiplying the vector with a translation matrix, which gives an expected vector in model B.

The last step is to convert the "translated" vector representation back to a word. We do this by looking up which word in language B which has the vector representation closest to the translated vector. The criterium used for this is simply nearest neighbour with a Euclidean distance.

3.2 Training the Models

This way of translation requires three models: word2vec models for both language A and B and a translation matrix from A to B.

The two language models are trained using the default word2vec algorithm. We refer to section 2 and previous papers for the specifics on this [7, 9].

The translation matrix is trained by solving the following expression:

$$\underset{T}{\operatorname{argmin}} \Sigma_{i=1}^{n} ||T \cdot a_i - b_i||^2$$

where T is an n by n matrix, a_i and b_i are n dimensional vector representations of words in languages A and B, such that b_i is the translation of a_i.

The quality of the translation largely depends on the accuracy of the mapping from vectors in model A to model B. Training language models using word2vec is unsupervised, and can therefore use hundreds of gigabytes or even terabytes of training data. Training the translation matrix is a supervised process (you have to provide correct translations), which makes it impractical to provide more than a few thousand words.

An advantage of using this method, is that it not only provides a word translation but also gives a distance between the translated vector and its nearest neighbour. If this distance is large, it indicates a higher level of uncertainty in the matrix. As such, one can search for translations where the algorithm is unsure what to choose and provide targeted training data to improve a next iteration of the model.

3.3 Training models of different sizes

One of the parameters when training word2vec models is the number of dimensions to train on. Typically, the number of dimensions is between 100 and 400[7].

Large datasets contain many relations, and can be trained on high dimensions (400 or more), but small datasets (i.e. under 50 million words) tend to be trained with lower dimensionality. However, these numbers are not set in stone: a study by Pennington et al. [4] indicates that 300 dimensions sufficiently capture relations -for their dataset. Since training complexity scales linearly with the number of dimensions, for very large datasets a high dimensionality might even simply be too costly.

Having discussed the cost/benefit of higher dimensions, we remark that translations with multiple models also allows

for models trained with a different number of dimensions. If both models have the same dimensions, the translation matrix will be square. If the dimensions are different, one can either use dimensionality reduction on the larger matrix, or use a non-square translation matrix to scale down the dimensionality.

4. SINGLE-MODEL TRANSLATIONS

4.1 Translation using Relations

As a simplest form of translation, one could say that "king" is to "koning" (the Dutch word for king) as "queen" is to "koninging" (Dutch for queen). Intuitively, this means we consider relations between two languages, instead of relations within a single language.

Thus, if we train a single model with two languages in it, one could expect to find these relations and use them to translate Dutch to English and vice versa.

This method is so simplistic, we do not expect it to give good translations. There is no guarantee the trained model models the Dutch-English relation in a consistent manner. A proper English sentence will not contain the word "koning", since it is not an English word. As such, it will be difficult (if not impossible) for word2vec to learn that "koning" and "king" are related.

Despite our doubts about its effectiveness, we choose to include this method as a baseline for performance and simplicity. Any more complex method performing not at least as good as this is not worth the additional complexity.

4.2 Translation matrix in single space

A mix between the two methods described above is using a single model trained on multiple languages (as in the previous section), but using a translation matrix (as in section 3).

This method would allow the language model to be trained without having to take into account the languages it is being trained on. As such, one could feed it text without determining the language that text is written in. This is a significant advantage for applications where the input is unlabelled.

5. EXPERIMENTS

5.1 Datasets

The datasets used to train the word2vec models are freely available online. We used the wikipedia datasets containing a snapshot of all articles on the English and Dutch Wikipedia and a copy of all Reddit comments.

The Go program used for cleaning the Reddit data is published on GitHub[1]. The wikipedia data was parsed with a slightly modified version of Wikipedia Extractor[2], which is also available on our GitHub page.

The characteristics of the cleaned datasets can be seen in table 5.1.

[1] https://github.com/bcleenders/autoTranslate
[2] https://github.com/bwbaugh/wikipedia-extractor

Name	Items	Words
Reddit comments [3]	1,325,482,268	38,177,224,313
English Wikipedia [4]	4,929,936	1,707,791,444
Dutch Wikipedia [5]	1,831,031	209,095,532

Table 1: Dataset statistics. For Wikipedia, "Items" refers to the number of articles. For Reddit, it refers to the number of comments.

5.2 Tests

We tested each translation method in multiple configurations, most notably with different dimensions for the language models.

The accuracy of the translation is measured in percentages how often the algorithm gave the expected answer. To account for answers that are almost correct, we keep the following statistics:

- **Accuracy @1**: the algorithm gave the correct answer.
- **Accuracy @5**: the correct answer was in the top 5.
- **Accuracy @10**: the correct answer was in the top 10.

If the answer is in the top 5 it is likely to be a synonym or very related word, since word2vec tends to group words by meaning. For example, the translation may be "buy" or "acquire" where the correct translation is "purchase".

6. RESULTS

ADD MORE RESULTS (MATTER OF RUNNING MORE EXPERIMENTS, MOSTLY)

6.1 Multi-model Translations

Figure 2 shows the accuracy of the multi-model translations, both for models trained on 100 and 400 dimensions. We used the same training set of translations to train the matrix on, and both language models are trained on respectively the Dutch and English Wikipedia.

An interesting observation is that the accuracy at all levels (top 1, top 5 and top 10) starts off lower for models trained at 400 dimensions. However, at the biggest training set, each level is better than the corresponding level at 100 dimensions. This suggests a form of overfitting; the translation matrix might be overfitted on the translation samples.

THERE'S ALSO A DIP TOWARDS THE END. CHECK IF THAT'S STILL THERE IF WE RUN IT WITH DIFFERENTLY SPLIT TEST/TRAINING DATA SETS.

6.2 Single-model Translations

In this section, we use two models: both are trained on the text of the English Wikipedia and the Dutch Wikipedia

[3] http://academictorrents.com/details/7690f71ea949b868080401c749e878f98de34d3d
[4] https://dumps.wikimedia.org/enwiki/20150901/
[5] https://dumps.wikimedia.org/nlwiki/20150901/

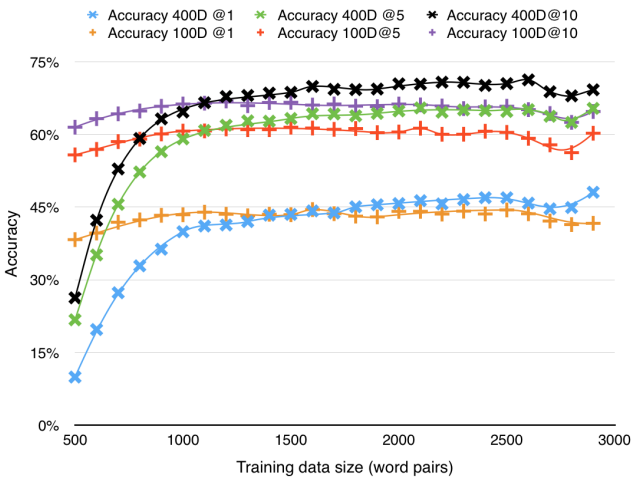

Figure 2: Accuracy for multi-model translations, trained on Dutch and English wiki. 'x' marks denote dimensionality 400, '+' marks dimensionalty 100.

combined. Since the Dutch Wikipedia is much smaller than its English counterpart (by a factor of 8, see table 5.1), we ran it eight times over the Dutch text. This artificially increases the weight given to Dutch text, so both are equally well represented.

The only difference between the two models is the number of dimensions: one is trained at 100, and the other at 400 dimensions.

6.2.1 Using Relations

The method described in section 4.1 is tested by randomly selecting 100 translations from our testset of correct translations. These are used as a known translation, to derive other translations from. For each of these translations, we then select 500 different translations to test against. This process is done both for a model trained at 100 dimensions, and one trained at 400 dimensions, and both for Dutch to English and vice versa.

For example, given "koning → king" as base pair, we then test whether the algorithm can translate "koningin" to "queen", "kopen" to "buy", et cetera.

Table 6.2.1 shows the results of these experiments.

An interesting note (which is not shown in the table), is that for every scenario, the lowest percentage of correct translations is 0.00%. This means that there are some very bad base translations. In the next section, we will give some reasons what may cause this.

6.2.2 Using Translation Matrix

Check figure 3 for the results.

Again: weird dip towards the end. **MARC, DO YOU KNOW WHY? JUST BAD LUCK CHOOSING TEST SAMPLES?**

7. DISCUSSION

		100 dim		400 dim	
		NL→EN	EN→NL	NL→EN	EN→NL
Avg.	@1	4.29%	3.92%	1.91%	2.02%
	@5	8.71%	8.28%	4.89%	4.97%
	@10	11.4%	10.9%	6.96%	7.00%
Max.	@1	14.6%	12.4%	8.60%	8.60%
	@5	23.4%	23.8%	15.8%	16.4%
	@10	28.4%	28.6%	21.2%	22.0%
Stdev.	@1	3.50%	3.24%	1.97%	2.19%
	@5	6.10%	6.15%	4.11%	4.55%
	@10	7.51%	7.60%	5.42%	5.85%

Table 2: Translation accuracy, using a single model without translation matrix and 400 dimensions. The minimum is left out, because it is 0.00% for all scenarios.

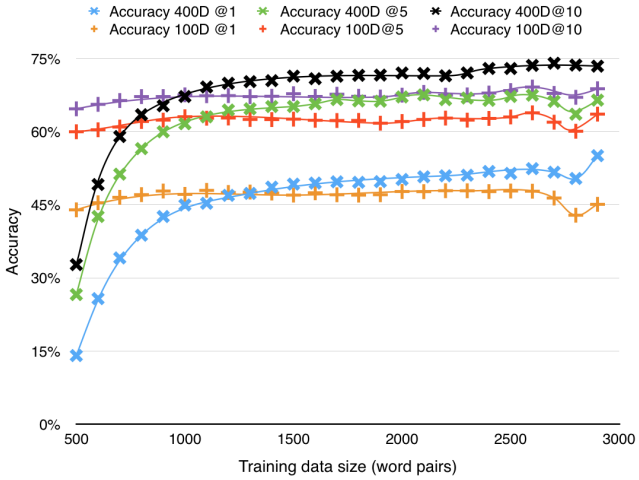

Figure 3: Accuracy for multi-model translations, trained on Dutch and English wiki. 'x' marks denote dimensionality 400, '+' marks dimensionalty 100.

The results for translations using a single table show that the accuracy is lower than what we might expect. One reason for this, is that it is difficult to provide a one-to-one mapping from one language to another. This especially goes for:

- Words that are spelled the same in both languages, but mean something different, e.g. "beer" (Dutch for bear). Word2vec maps both entities to one token.
- Words that are spelled the same in both languages and mean the same, e.g. "wild". If these are used as training data for translation with a single model and no matrix (section 4.1), word2vec will think Dutch and English are the same.
- Words that translate to more than one word, e.g. "wake" (EN) → "wakker worden" (NL). Because word2vec splits based on spaces (in our setup), the Dutch has two tokens that together correspond to a single English token. The translation algorithms we described are not sophisticated enough to handle this.

The accuracy of translations can be improved by targeting

these potential problems. The following three solutions address these problems:

- Label words in training data with their language (e.g. "nl_koning", "en_king") to distinguish words spelled the same, but with a different language. By labelling words, there can be no collision between English and Dutch tokens. One can, however, still say that words that are spelled the same are likely to be translations, and use this as prior knowledge (i.e. proper nouns are usually spelled the same in various languages).
- Use word2phrase to combine fixed combinations of words into a single token (e.g. "San Francisco" should be one token). This way we can extract more information out of a cooccurence of two tokens, and possibly even cope with combinations like "lopen" → "to walk".
- Handle ambiguous translations better (e.g. the English homonym "arm" translates the Dutch words "wapen" or "arm"). This requires understanding of the context, which we think is more complicated and outside the scope of word2vec.

The first two should be relatively easy to implement, but the third one would be quite difficult. Instead, a deep neural network may be worth consideration to provide the flexibility to handle the full complexity in language.

8. CONCLUSION

METHOD X IS BEST. REFER TO SECTION ?? FOR FURTHER IMPROVEMENTS. section 7

Acknowledgements

The authors thank SICS Swedish ICT for the resources they provided.

9. REFERENCES

[1] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

[2] N. Boycheva. Distributional similarity music recommendations versus spotify: A comparison based on user evaluation. 2015.

[3] C. Buck, K. Heafield, and B. van Ooyen. N-gram counts and language models from the common crawl. In *Proceedings of the Language Resources and Evaluation Conference*, 2014.

[4] R. JeffreyPennington and C. Manning. Glove: Global vectors for word representation. 2014.

[5] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[6] O. Levy, Y. Goldberg, and I. Ramat-Gan. Linguistic regularities in sparse and explicit word representations. *CoNLL-2014*, page 171, 2014.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] H. Wang. Introduction to word2vec and its application to find predominant word senses. 2014.

[11] L. Wolf, Y. Hanani, K. Bar, and N. Dershowitz. Joint word2vec networks for bilingual semantic representations. *International Journal of Computational Linguistics and Applications*, 5(1):27–44, 2014.