Super awesome embeddings

Grzegorz Beringer, Mateusz Jabłoński, Piotr Januszewski, and Julian Szymański

Faculty of Electronic Telecommunications and Informatics
Gdańsk University of Technology, Gdańsk, Poland

Abstract. Abstract

Keywords: word sense disambiguation, word embeddings

1 Introduction

Word Sense Disambiguation (WSD) is an open problem of natural language processing (NLP) and ontology. WSD is identifying which sense of a word (i.e. meaning) is used in a sentence based on the word context. Difficulty is when the word has multiple meanings (e.g. a decision tree, a tree data structure, a tree in a forest). The problem requires two inputs: a dictionary to specify the senses which are to be disambiguated and a corpus of language data to be disambiguated. WSD task has two variants: "lexical sample" and "all words" task. The former aim to disambiguate the occurrences of a small sample of selected target words, while in the latter all the words in a piece of running text need to be disambiguated. Our solution targets the former one, but could be extended to the latter variant. The solution to WSD would be useful in many NLP related problems as: relevance of search engines, anaphora resolution, coherence, inference, etc.

Word embeddings are a product of feature learning techniques in NLP, where words from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a dimensionality reduction from a space with one dimension per word to a continuous vector space with a much lower dimension. Methods to generate this mapping include artificial neural networks[1][2][3], dimensionality reduction on the word co-occurrence matrix[4] and probabilistic models[5]. Word embeddings are commonly used as the input representation. They have been shown to boost the performance in NLP tasks such as syntactic parsing[6] and sentiment analysis[7].

Word embeddings cannot distinguish between different meanings of ambiguous words by themselves. By definition, there is only one embedding for each word e.g. for word "tree" there is a single real-valued vector. What can be done, is to try to distinguish the meaning based on the context, in which the word was used. Then, we treat each meaning as a separate keyword, which has its own embedding. We propose a simple method to infer the word meaning: an average of the context and the word embeddings and number of improvements to this approach in the chapter "Our method". Experiments with our solution are presented in the chapter "Experiments". To conduct those experiments we have created the dataset composed of 6 ambiguous words, with 4 to 7 meanings each, and collected real-world usage examples of those meanings, with tagged words to be disambiguated. We describe our dataset in the chapter "Dataset". In the chapter "Related work" we present other approaches to WSD.

2 Related work

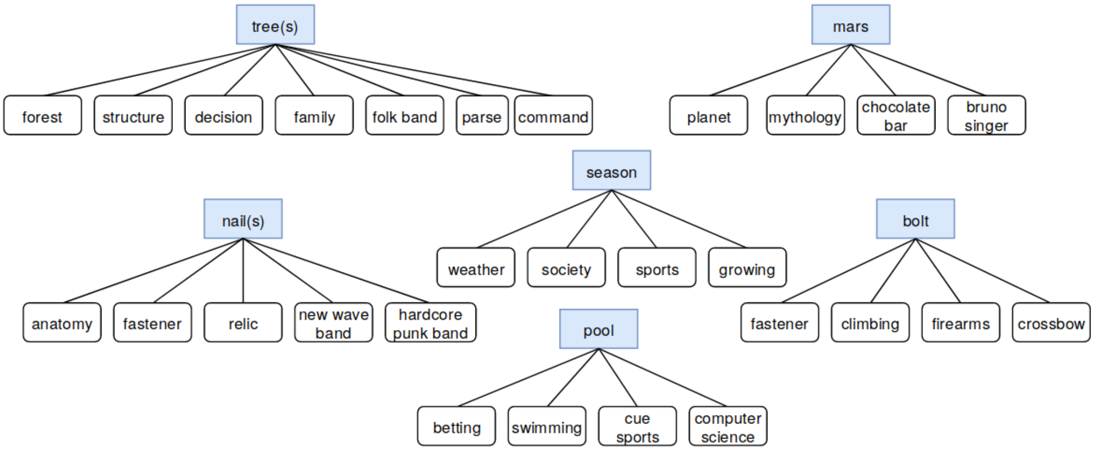
3 Dataset

For the purpose of testing word embeddings as a method to differentiate between different meanings, we gathered examples for 6 ambiguous words, 4-7 meanings each (28 meanings in total). Ambiguous word together with its meaning constitues a *keyword*, which we use as a separate class when identyfing the closest meaning given some context. All keywords can be seen on Figure 1.

Examples were mostly gathered from Wikipedia, using *What links here* utility for each keyword. If usage examples from Wikipedia were not enough, other websites were used (or even the Wikipedia article on specific keyword itself).

The dataset is split into training and test set, with 5 training and 10 test examples for each keyword. Each example is stored in plain text, with the ambiguous word marked with "*" on both sides.

Fig. 1. Ambiguous words with their meanings (keywords) from the dataset



The correct keyword for each example, together with a path to file and a link, where the original text was taken from, are stored in CSV files: *train.csv* for training set, *test.csv* for test set (columns: path,keyword,link). Keywords themselves, together with links to their Wikipedia articles, are stored in *keywords.csv* file. Dataset, together with the code to execute experiments from this paper, can be found on our GitHub repository [8].

Example for *bolt crossbow* keyword:

Keyword definition: keyword.csv

keyword,link

...

bolt crossbow,https://en.wikipedia.org/wiki/Crossbow_bolt

Test set: test.csv

path,keyword,link

...

texts/test/bolt_crossbow_5.txt,bolt crossbow,https://en.wikipedia.org/wiki/Incendiary_device

Text: texts/test/bolt_crossbow_5.txt

*"Sulfur- and oil-soaked materials were sometimes ignited and thrown at the enemy, or attached to spears, arrow and *bolts* and fired by hand or machine. Some siege techniques—such as mining and boring—relied on combustibles and fire to complete the collapse of walls and structures."*

4 Our method

4.1 Keyword embedding

Keyword is a sequence of words that lets us disambiguate between different meanings of the same ambiguous word, e.g. "tree forest" that represents a tree as a plant and "tree structure" which represents tree as a mathematical structure.

To get the embedding of the keyword, we simply average all the words in the keyword:

$$\boldsymbol{k} = e(w_1, w_2, ..., w_N) = \frac{1}{N}\sum_{i=1}^{N} e(w_i) \tag{1}$$

where $e(\cdot)$ is the embedding function used and $w_1, w_2, ..., w_N$ is a sequence of N words that, in this case, constitutes a keyword.

4.2 Context embedding

Context is a sequence of words that contains an ambiguous word within, usually in the middle. It is parametrized by context length l, which specifies how many of words from both side of the ambiguous word in question were taken into consideration.

Context embedding $\boldsymbol{c}$ is also achieved by taking an average of word embeddings (Equation 1). In this case, $N = 2l + 1$ and $w_1, w_2, ..., w_N$ is the context with ambiguous word inside. For some cases $N < 2l + 1$, since the ambiguous word may occur at the beginning or ending of text example and full context cannot be collected. In this case, we just average average the reduced context.

Text, that we take context from, must be preprocessed. We remove any special characters and stopwords, and use lower-case letters only.

Example with $l = 3$:

Text: *"The object pool pattern is a software creational design pattern that uses a set of initialized objects kept ready to use – a " *pool* " – rather than allocating and destroying them on demand."*

Preprocessed text: *"the object pool pattern is a software creational design pattern that uses a set of initialized objects kept ready to use a *pool* rather than allocating and destroying them on demand"*

Context: *kept ready use pool rather allocating destroying*

4.3 Optimization

As a baseline, we use keyword and context embeddings described above to find the closest keyword given some context, using cosine distance as a metric. To optimize this system, we use texts from the training set to improve the quality of keyword embeddings. For each training example, we shift the correct keyword embedding by a factor of α in the direction of context embedding, which describes said keyword.

$$\boldsymbol{k}^{(i+1)} = \boldsymbol{k}^{(i)} + \alpha(\boldsymbol{c} - \boldsymbol{k}^{(i)}) \tag{2}$$

where $\boldsymbol{k}^{(i)}$ is the embedding of the correct keyword at iteration i, $\boldsymbol{c}$ is the context embedding and α is a scalar that controls the strength of the update.

We try to optimize the system even further, by also moving top-k closest keywords that are incorrect given the same context.

$$\boldsymbol{m}^{(i+1)} = \boldsymbol{m}^{(i)} - \beta(\boldsymbol{c} - \boldsymbol{m}^{(i)}) \tag{3}$$

where $\boldsymbol{m}^{(i)}$ is the embedding of incorrect keyword at iteration i and β is a scalar that controls the strength of the update.

References

1. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. arXiv e-prints (2013) arXiv:1310.4546
2. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: In EMNLP. (2014)
3. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606 (2016)
4. Levy, O., Goldberg, Y.: Neural word embedding as implicit matrix factorization. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS'14, Cambridge, MA, USA, MIT Press (2014) 2177–2185
5. Globerson, A., Chechik, G., Pereira, F., Tishby, N.: Euclidean embedding of co-occurrence data. J. Mach. Learn. Res. **8** (2007) 2265–2295
6. Socher, R., Bauer, J., Manning, C.D., Andrew Y., N.: Parsing with compositional vector grammars. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics (2013) 455–465
7. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics (2013) 1631–1642
8. Beringer, G., Jablonski, M., Januszewski, P., Szymański, J.: https://github.com/gberinger/automatic-wiki-links (2018)