Towards semantic-rich word embeddings

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

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

Abstract. In recent years word embeddings have been shown to boost the performance in NLP tasks such as syntactic parsing or sentiment analysis, but also in multimodal systems like image captioning or visual question answering. It is therefore not surprising that word embeddings have been applied to Word Sense Disambiguation (WSD) problem as well. In this work, we present how a simple word embeddings average-based method can be used to produce keyword (meaning) and context embeddings, and how the former can be improved with two embeddings distance optimization techniques. We also open source the dataset that was created for the purpose of this research. It is 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.

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

Typically, there are two kinds of approach for WSD: supervised, which make use of sense-annotated training data, and knowledge-based, which make use of the properties of lexical resources. In supervised approach, the most widely used training corpus used is SemCor[8], with 226,036 sense

annotations from 352 manually annotated documents. Knowledge-based systems usually exploit WordNet[9] or BabelNet[10] as semantic network. Our solution joins two approaches. We use the knowledge-based word embeddings like GloVe[2] as a baseline for our method and then optimize them in supervised fashion.

The most usual baseline is the Most Frequent Sense[11] (MFS) heuristic, which selects for each target word the most frequent sense in the training data. Recent growth of sequence learning techniques using artificial neural networks contributed to WSD research: Raganato et al.[12] propose a series of end-to-end neural architectures directly tailored to the task, from bidirectional Long Short-Term Memory (LSTM) to encoder-decoder models. Melamud et al.[13] also use bidirectional LSTM it their work. They use large plain text corpora to learn a neural model that embeds entire sentential contexts and target words in the same low-dimensional space, which is optimized to reflect inter-dependencies between targets and their entire sentential context as a whole.

Iacobacci et al.[14] were first to try to use word embeddings for WSD. They consider four different strategies for integrating a pre-trained word embeddings as context representation in a supervised WSD system: concatenation, average, fractional and exponential decay of the vectors of the words surrounding a target word. Peters et al.[15] create word representations that differ from traditional word embeddings in that each token is assigned a representation that is a function of the entire input sentence. They use vectors derived from a bidirectional LSTM that is trained with a coupled language model objective on a large text corpus.

Most of the previous neural networks applications to WSD ignore lexical resources like glosses (sense definitions) and rely solely on word's context. In Luo et al.[16] paper, they integrate the context and glosses of the target word into a unified framework, in order to make full use of both labeled data and lexical knowledge.

Entity Linking (EL) and WSD both address the lexical ambiguity of language. The aim of EL is to discover mentions of entities within text and to link them to the most suitable entry in a reference knowledge base. The two tasks are pretty similar, but they differ fundamentally: in EL the textual mention can be linked to a named entity which may or may not contain the exact mention, while in WSD there is a perfect match between the word form and a suitable word sense in the knowledge base. Moro et al.[17] present Babelfy, a unified graph-based approach to EL and WSD based on a loose identification of candidate meanings coupled with a densest subgraph heuristic, which selects high-coherence semantic interpretations. We can find also application of random walks[18] and topic models[19] for knowledge-based WSD.

Developing WSD system requires much effort and as a result, very few open source WSD systems are publicly available. Zhong et al.[20] present an English all-words WSD system, IMS (It Makes Sense), built using a supervised learning approach that is written in Java and completely open source. Following Lee and Ng[21], they adopt support vector machines (SVM) as the classifier and integrate multiple knowledge sources including parts-of-speech (POS), surrounding words, and local collocations as features.

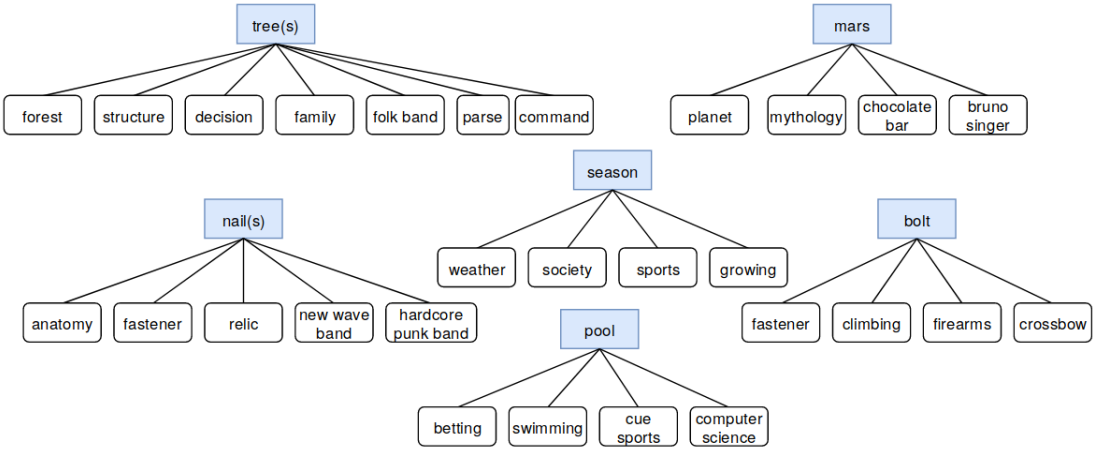
3 Dataset

For the purpose of constructing semantic-rich word embeddings, we manually gathered usage examples for 6 ambiguous words, 4 to 7 meanings each (28 meanings in total) (Table 1). 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 Fig. 1.

Number of ambiguous words	6
Number of meanings per ambiguous word	4 to 7
Number of keywords (ambiguous word + meaning)	28
Number of training examples per keyword	5
Number of test examples per keyword	10

Table 1. Statistics of the dataset collected for the purpose of this paper. Training examples can be used by the model to optimize semantic-rich word embeddings (Section 4.4). Test examples are used to evaluate the embeddings and cannot be trained on.

Fig. 1. Ambiguous words with their meanings that are present in the dataset. Each word-meaning pair is called a keyword, e.g. *tree (forest)*, *pool (computer science)*.



We chose ambiguous words based on the number and variety of meanings it had. Meanings themselves were chosen to cover a range of topics (e.g. *tree (forest)*, *tree (family)*, *trees (folk band)*, *tree (command)*). We also tried to look for meanings that are semantically related and can occur in similar context (and in turn be difficult for the model to differentiate between), e.g. *tree (structure)*, *tree (parse)*, *tree (decision)* or *nails (new wave band)*, *nails (hardcore punk band)*. Lastly, we added some keywords, that we suspected to be really underrepresented in the word embedding of the ambiguous word, e.g. *Mars* as the pop singer Bruno Mars (*mars (bruno singer)*) or *pool* as the computer science term (*pool (computer science)*).

Usage examples for keywords were gathered mostly from Wikipedia, using *What links here* utility, which lists all Wikipedia pages that link to a specific article. We used these links to search for usages of our keywords in context. We found that *What links here* utility has some limitations. Many articles linked to the keyword do not use that keyword in text at all or just list it in "See also" section, which does not provide good context around the keyword for the model to improve on. Moreover, some keywords do not have enough usage examples that can be found on Wikipedia alone. In such cases, other websites were used to find proper usage examples.

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. For simplicity, only one word is marked in each text, even if more ambiguous word usages can be found. In case we wanted to mark another word in the same text, we could just add the same example twice, with different words marked each time.

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 (Figure 2).

Dataset, together with the code to execute experiments from this paper, can be found on our GitHub repository [22].

4 Our method

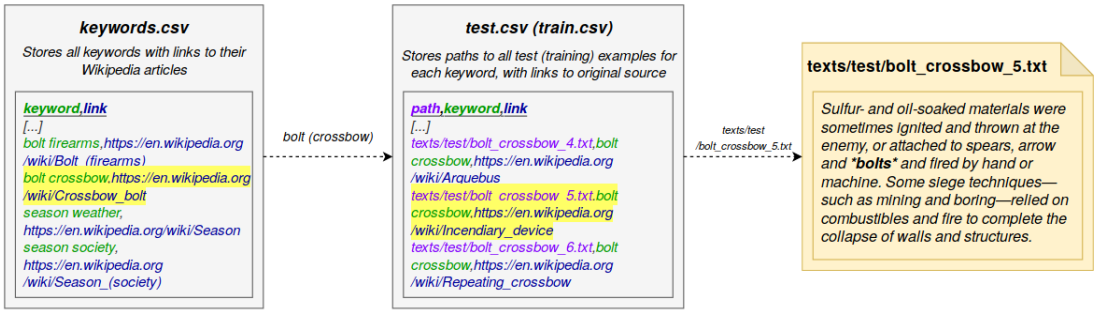
4.1 Keyword embedding

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 average embeddings of all the words in the keyword:

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

Fig. 2. Dataset format with an example for *bolt (crossbow)* keyword (highlighted text and arrows). Usage examples are stored in plain text, with ambiguous word marked with "*" (right), and are divided into training and test set. Paths to all test (training) examples can be read from *test.csv* (*train.csv*), together with the links to where they were taken from (middle). All keywords are stored in *keywords.csv*, along with the link to their Wikipedia articles (left).



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.

Example for keyword *tree (forest)*:

$$\boldsymbol{k}_{tree(forest)} = e(tree, forest) = \frac{e(tree) + e(forest)}{2} \tag{2}$$

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 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. Below is an example taken from the dataset (3):

Example for keyword *pool (computer science)* with context length $l = 3$

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*
Context embedding: $\boldsymbol{c} = \frac{e(kept)+e(ready)+e(use)+e(pool)+e(rather)+e(allocating)+e(destroying)}{7}$

4.3 Basic model

The basic approach is to use keyword (4.1) and context embeddings (4.2) to find the closest keyword given some context, using cosine distance as a metric.

In other words, given an input text and marked ambiguous word within, we extract the context and compute its embedding $\boldsymbol{c}$ (see example in the frame above). The keyword, whose embedding is closest to $\boldsymbol{c}$ w.r.t. cosine distance, is chosen as the ambiguous word's meaning.

We evaluate this model based on top-k accuracy, which checks, if the embedding for correct keyword is among closest k keyword embeddings (5.1).

4.4 Optimization

We assume, that the performance of the basic model (4.3) (and the quality of keyword embeddings) can be improved, if we provide examples of contexts that the specific keyword appears in. To this end, we move keyword embeddings closer to embeddings of contexts they appear in, using examples taken from the training set (3).

For each training example, we shift the correct keyword embedding by a factor of α, in the direction of the context embedding, which describes said keyword.

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

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.

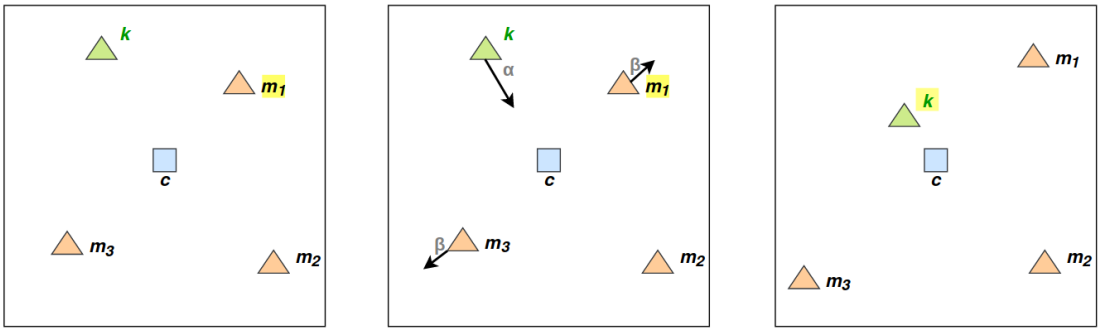
We try to optimize the system even further, by also moving top-k closest keywords that are incorrect given the same context.

$$\boldsymbol{m}^{(i+1)} = \boldsymbol{m}^{(i)} - \beta(\boldsymbol{c} - \boldsymbol{m}^{(i)}) \tag{4}$$

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.

Both optimization methods can be seen on Fig. 3. The performance of these techniques is evaluated in 5.2 and 5.3.

Fig. 3. Optimization. Square is the context embedding $\boldsymbol{c}$, triangles are the closest keyword embeddings, with $\boldsymbol{k}$ being the correct keyword. Without the optimization, the closest keyword embedding w.r.t to context is $\boldsymbol{m}_1$, which is incorrect (a). We move the correct keyword embedding in the direction of the context embedding by the factor of α (Eq. 3) and the closest N incorrect keyword embeddings (here $N = 2$) away from the context embedding by the factor of β (Eq. 4) (b). After optimization, $\boldsymbol{k}$ is the closest keyword (c).



4.5 Limitations

We are aware that our method has some limitations. First of all, it may be impossible to achieve the optimal solution, as we can only optimize keyword embeddings, leaving context embeddings fixed in the multidimensional space. Therefore, it is possible that contexts for specific keyword overlap on contexts for other keyword.

Secondly, the average context embedding may be ambiguous, with a high possibility of two different context being mapped to a similar point in space, especially for longer context legths. In future work (6), we suggest experimenting with different sequence embedding techniques, that might be better suited for this purpose than a flat average.

Finally, we run experiments for a very small number of ambiguous words and meanings (Table 1). Our method could have problems with a bigger dataset, since it would be much more difficult to separate different keywords.

5 Experiments

The goal of following experiments is to check how embedding approach (4.1, 4.2) works for the dataset we collected (3), both out-of-the-box (baseline) and after applying optimization techniques (4.4). We use a pretrained embedding model from spaCy - *en_vectors_web_lg*, which contains 300-dimensional word vectors trained on Common Crawl with GloVe[23].

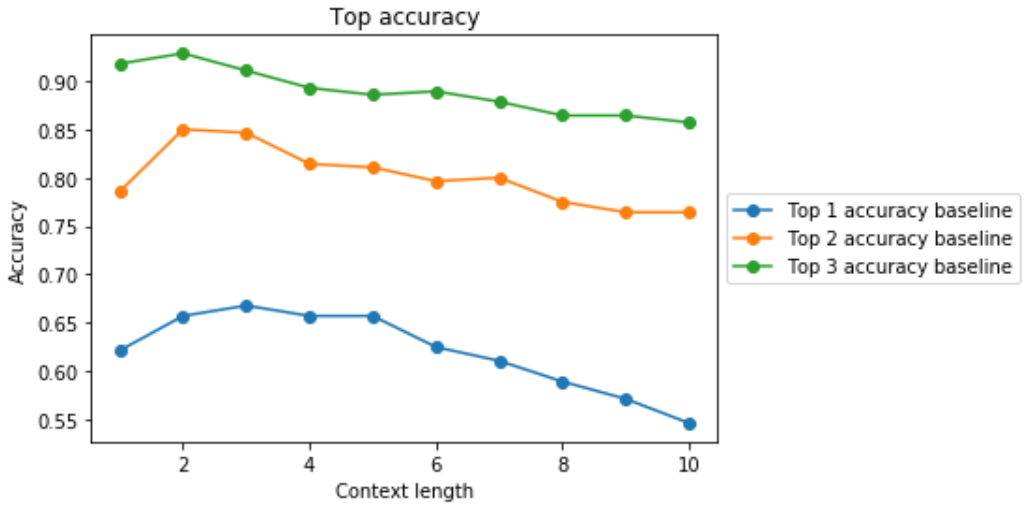
Due to the high impact of training data order on test results, we take the average score of 30 runs (each with a random order of training data) for each optimization experiment. The best configuration of each experiment is taken as a baseline for the next experiment. The goal of these experiments is to show how each of parameters influences on model's accuracy after training.

We compare results on the test set with two metrics: top-1 accuracy, where we are only interested, if the correct keyword is the closest one to a specific context describing it, and top-3 accuracy, where we check if we are in the closest three keywords.

5.1 Baseline

For the baseline model, we check how out-of-the-box keyword (4.1) and context embeddings (4.2) work without any extra optimizations. We evaluate the performance of the model with different context lengths, to see how it affects top-1 and top-3 accuracies (Fig. 4).

Fig. 4. Top-1 and top-3 accuracies of the baseline model with different context lengths.



We can see, that the model does relatively well, even unoptimized. Top-3 accuracy is about 85-90%, which is probably caused by a low number of meanings for each ambiguous word. Top-1 accuracy for shorter context lengths can go as high as 65% but decreases with longer contexts. As suspected (4.5), this is most likely due to the fact, that the average of many word embeddings may make some contexts similar to each other, therefore making it harder to distinguish between some meanings.

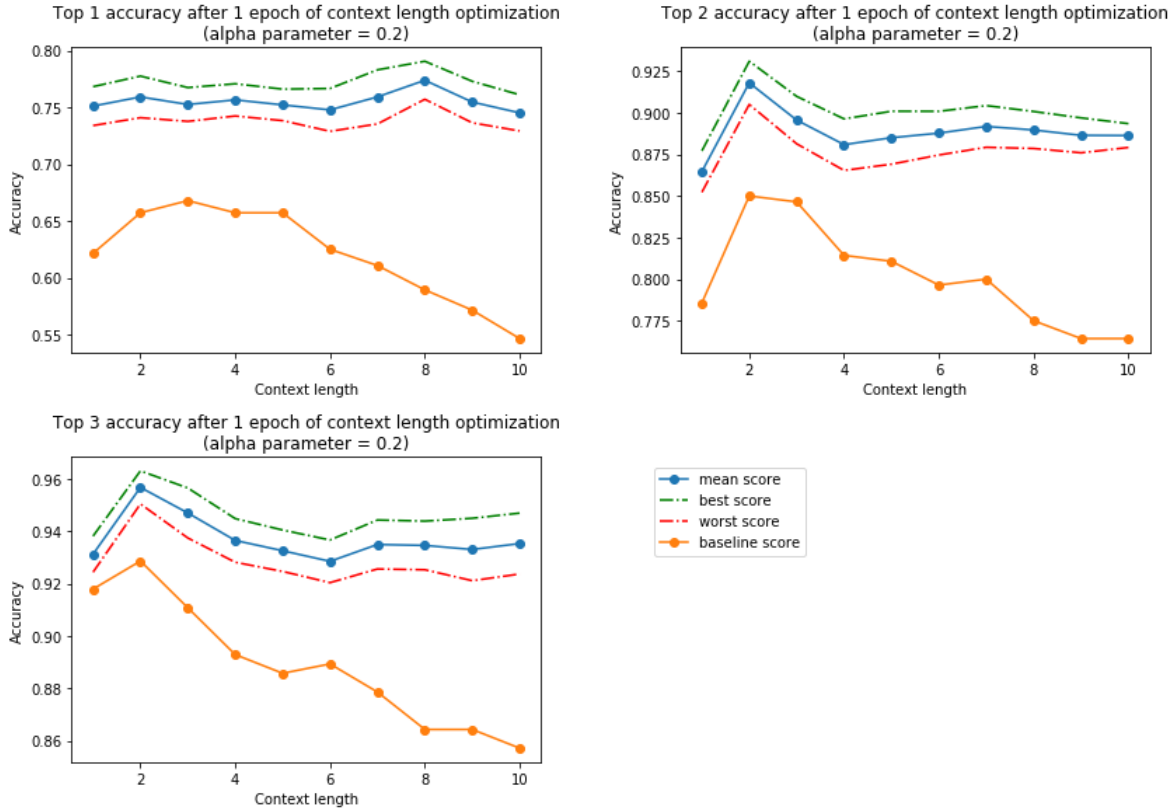
The best result w.r.t. top-1 accuracy was achieved with $l = 3$, which is why we choose this context length as a starting point for next experiments.

5.2 Alpha optimization

We check, how a simple optimization method of moving keyword embeddings closer to the contexts they appear in (Eq. 3), affects the accuracy of the model. To do this, we arbitrarily choose some α value (here 0.2), test the results of improved keyword embeddings with different context lengths and compare them to the baseline (5.1). Since we average the results from 30 runs, we mark the measurement uncertainty on the graph (Fig. 5).

Independently of some noise caused by the random order of training data, the optimized model is much better than the base model (12-36% relative, 10-20% absolute improvement w.r.t. top-1

Fig. 5. Comparison of the baseline model (5.1) and a model optimized with $\alpha = 0.2$ for different context lengths.



accuracy on the test set). Improvement is most visible for longer context lengths, which achieved relatively poor results in the baseline experiment. It seems, that a simple optimization may help alleviate the problems with ambiguous embeddings of long contexts (4.5).

Overall, results suggest, that for an optimized model, context length is not as important as with the baseline model, at least with this particular α value.

Next, we check the influence of the α parameter value on the top-1 and top-3 accuracies for $l = 3$ (Fig. 6).

Top-1 accuracy grows up to $\alpha = 0.25$ and then starts to drop. It suggests, that bigger values of α may cause the movement on the multidimensional plane to be too chaotic, moving the correct keyword embedding too close to a specific context, causing overfitting to a specific example. Similarly, small values of α may update the keyword embedding too little, ending up in only slightly better position.

5.3 Beta optimization

We proved that moving keyword embeddings closer to their contexts is beneficial for the accuracy on the test set (5.2). We theorize, that the results could be further improved, if we also increase the distance between closest incorrect keyword embeddings and the context embedding for a specific example (Eq. 4).

In following experiment, we therefore check the influence of β parameter value on the accuracy on the test set, for $l = 3, \alpha = 0.25$ (Fig. 7). Since we compare the results to an average value from previous experiments (Fig. 6), we decide to only plot an average result achieved by this optimization.

Low values of β seem to slightly improve the performance of our optimization method w.r.t. top-1 accuracy (about 1% absolute and relative improvement). High values of β suffer from a similar problem to high values of α, causing too much disruption to keyword embeddings.

We chose to move only top-3 closest keywords per each training example. In future work, it might be interesting to see, how does the number of keyword embeddings moved affects the results.

Fig. 6. Influence of α parameter value on top-1 and top-3 accuracies for $l = 3$. Blue line denotes the average result from 30 runs, dotted lines denote worst and best result. The value for $\alpha = 0$ is the result achieved by the baseline model (5.1).

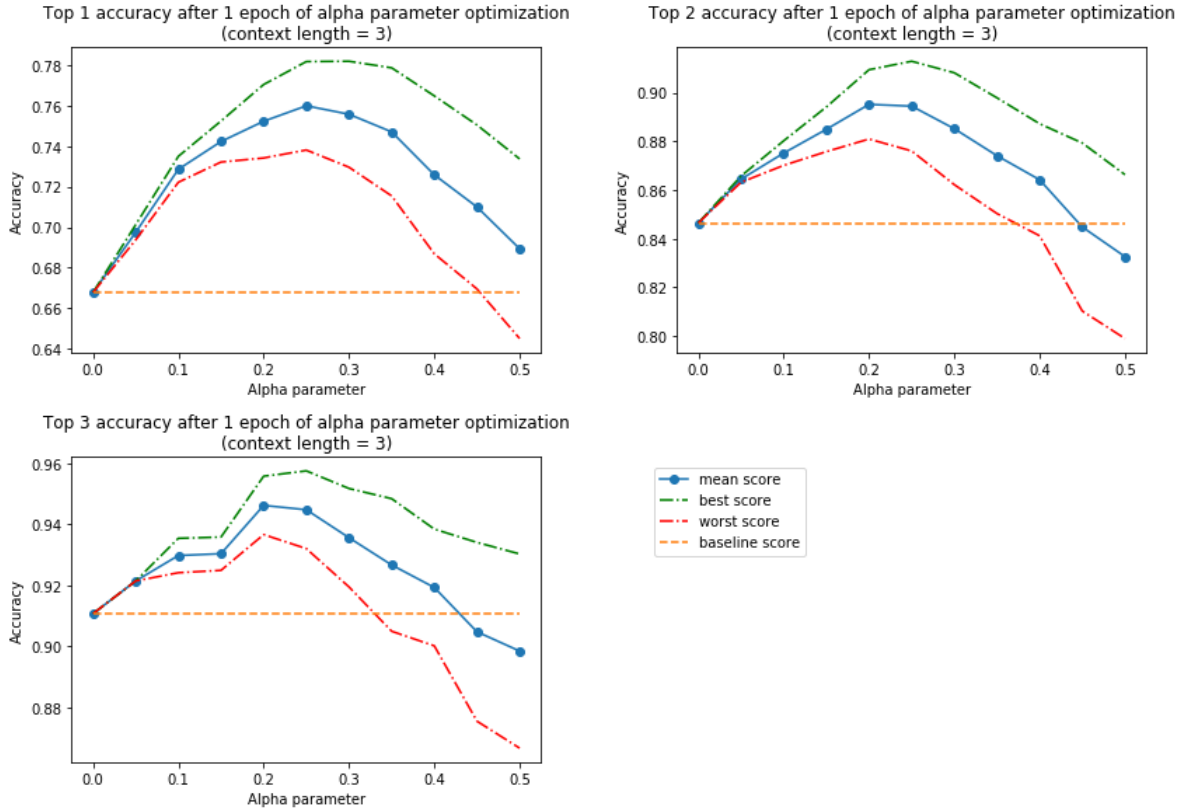
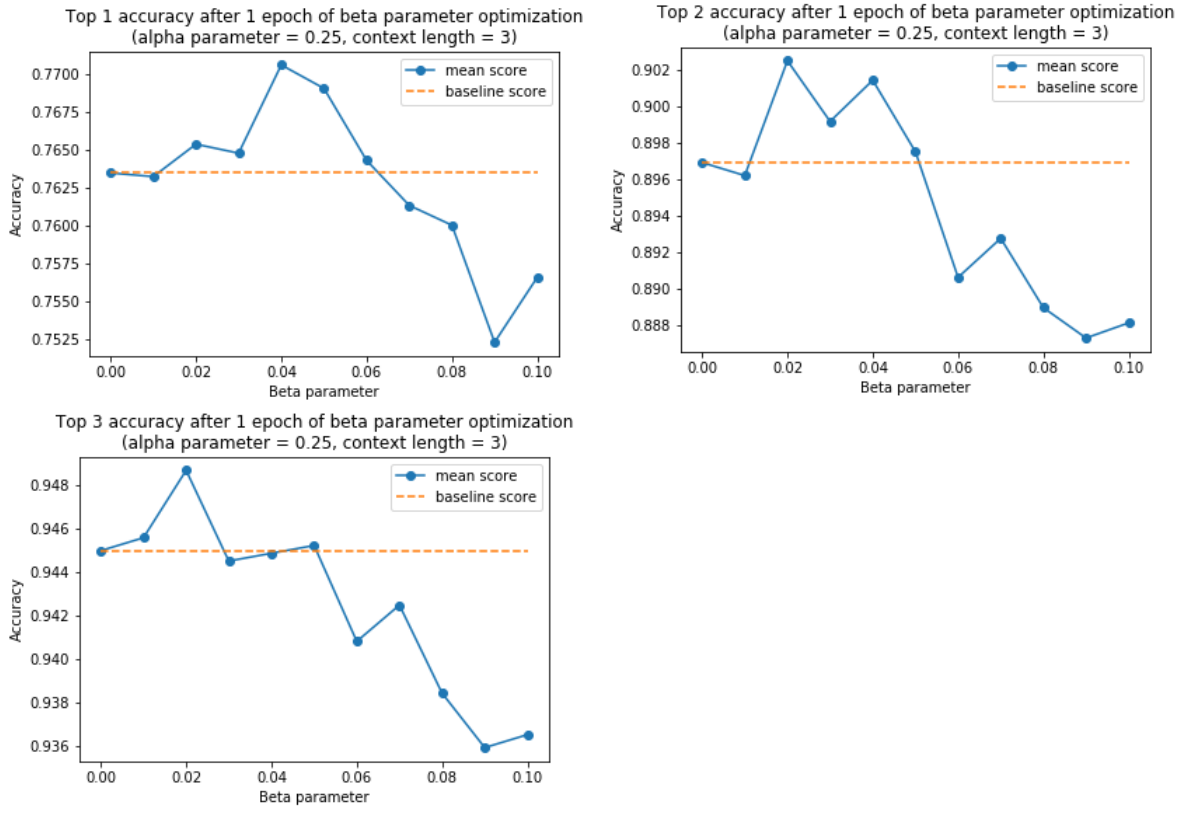


Fig. 7. Influence of β parameter value on top-1 and top-3 accuracies for $l = 3$ nad $\alpha = 0.25$. The value for $\beta = 0$ is the best result achieved by the model from previous experiment (Fig. 6).

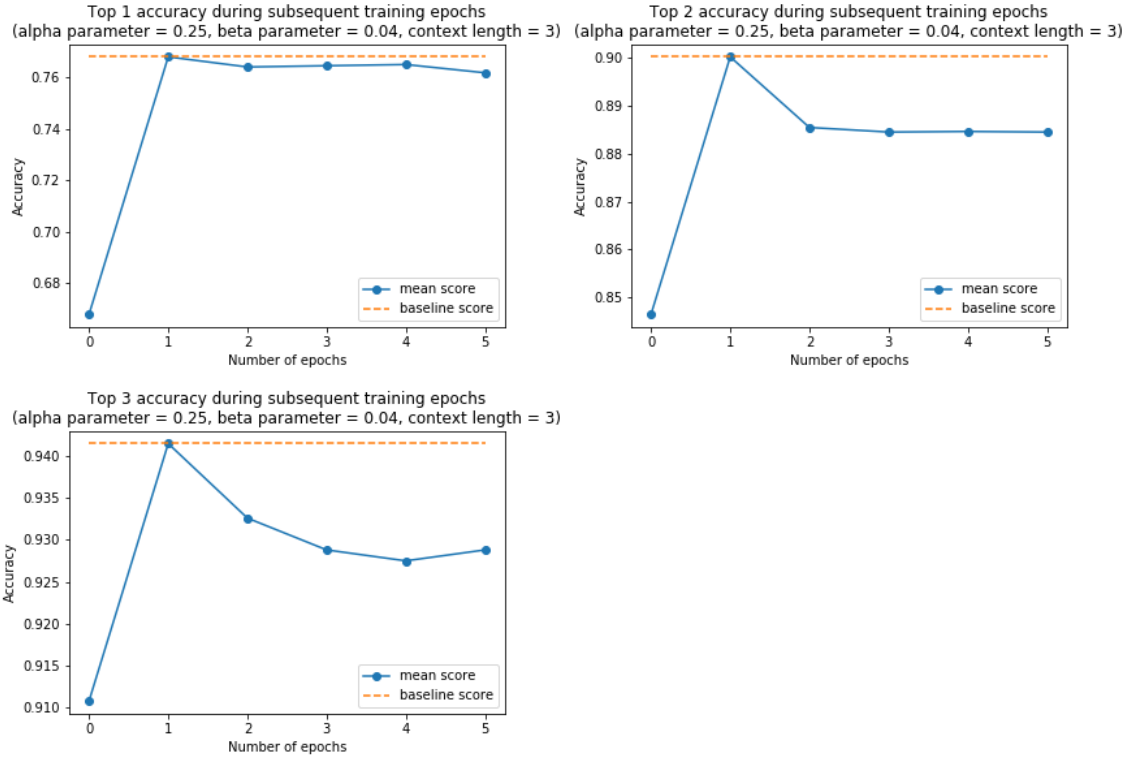


We also theorize, that influence of this particular optimization could be bigger for trainings with more than 1 epoch.

5.4 Accuracy in time

In previous experiments, we passed through the training data only once, which may not be optimal. The goal of this experiment is to check how a number of epochs (full passes through training data) affects the final accuracy. We run 5 epochs of training, using best parameters from previous experiments: $l = 3, \alpha = 0.25, \beta = 0.05$ (Fig. 8).

Fig. 8. Top-1 and top-3 accuracy for model with $l = 3, \alpha = 0.25, \beta = 0.05$ trained for 5 epochs.



We see that there is almost no improvement after the first epoch of training. The possible problem is that the values of α and β are too high, causing the updates to keyword embeddings to be too big. We therefore check what happens, when we choose lower values of these parameters (Fig. 9).

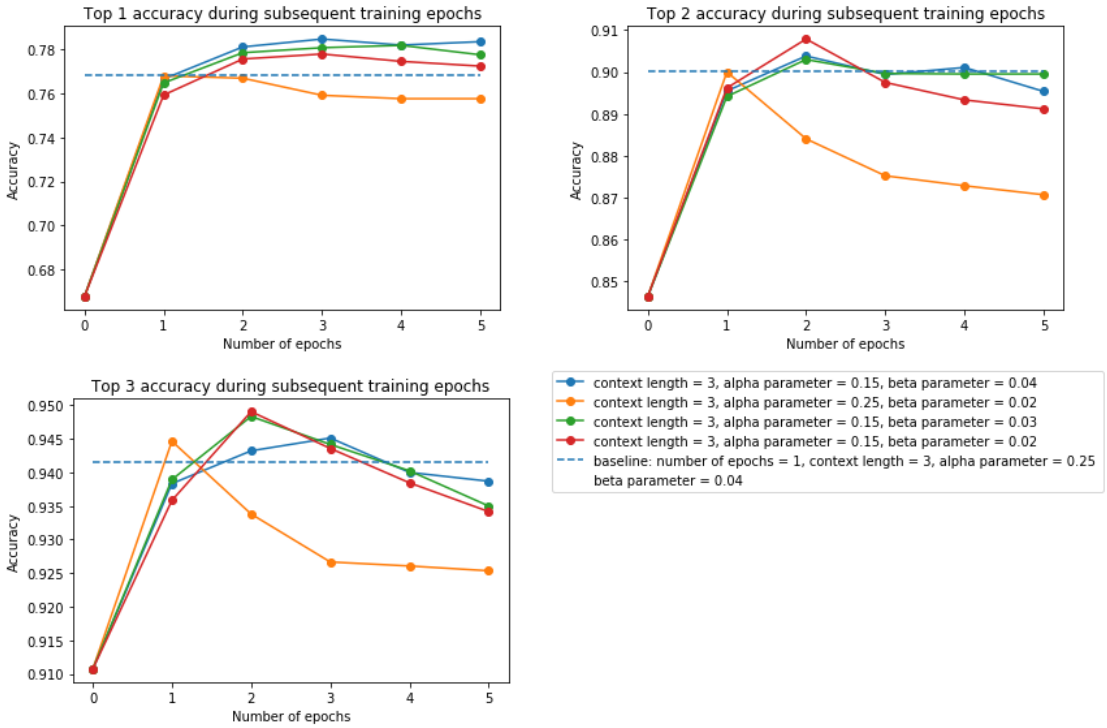
Longer training shows that optimal parameters obtained from previous experiments are not necessarily good for multiple passes through the training data. If we train for a bigger number of epochs, lower values of α and β parameters are better. In future work (6) we suggest, that a decay parameter might be useful for this particular reason, with each update being a bit smaller to fine-tune the results.

6 Conclusion and future work

Average-based keyword and context embeddings proved to be a surprisingly good baseline for the dataset of ambiguous keywords that we collected, achieving over 65% top-1 and 93% top-3 accuracy. We have shown, that you can improve these results up to about 79% top-1 and 95% top-3 accuracy by using a simple optimization method of improving keyword embeddings with even a few training examples per keyword.

Further improvements could be sought by using different keyword and context embedding schemes, e.g. weighted average or by using some sentence embedding method. Optimization method itself could be made more stable by applying decay to alpha and beta parameters and by using a

Fig. 9. Comparison of models optimized with different α and β values with $l = 3$ for 5 epochs.



validation set for early stopping. It could also be bound to cosine distance between the keyword and context - the bigger the difference, the bigger the update. Finally, the dataset could be enhanced to contain more ambiguous words and training/test examples.

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. Mihalcea, R.: Semcor semantically tagged corpus. (1998)
9. : Wordnet, Princeton University (2010)
10. Navigli, R., Ponzetto, S.P.: BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence **193** (2012) 217–250
11. Raganato, A., Camacho-Collados, J., Navigli, R.: Word sense disambiguation: A unified evaluation framework and empirical comparison. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, Association for Computational Linguistics (2017) 99–110

12. Raganato, A., Delli Bovi, C., Navigli, R.: Neural sequence learning models for word sense disambiguation. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics (2017) 1156–1167
13. Melamud, O., Goldberger, J., Dagan, I.: context2vec: Learning generic context embedding with bidirectional lstm. In: Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning, Association for Computational Linguistics (2016) 51–61
14. Iacobacci, I., Pilehvar, M.T., Navigli, R.: Embeddings for word sense disambiguation: An evaluation study. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics (2016) 897–907
15. Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics (2018) 2227–2237
16. Luo, F., Liu, T., Xia, Q., Chang, B., Sui, Z.: Incorporating glosses into neural word sense disambiguation. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics (2018) 2473–2482
17. Moro, A., Raganato, A., Navigli, R.: Entity linking meets word sense disambiguation: a unified approach. Transactions of the Association for Computational Linguistics **2** (2014) 231–244
18. Agirre, E., López de Lacalle, O., Soroa, A.: Random walks for knowledge-based word sense disambiguation. Computational Linguistics **40** (2014) 57–84
19. Chaplot, D.S., Salakhutdinov, R.: Knowledge-based word sense disambiguation using topic models. In: AAAI. (2018)
20. Zhong, Z., Ng, H.T.: It makes sense: A wide-coverage word sense disambiguation system for free text. In: Proceedings of the ACL 2010 System Demonstrations, Association for Computational Linguistics (2010) 78–83
21. Lee, Y.K., Ng, H.T.: An empirical evaluation of knowledge sources and learning algorithms for word sense disambiguation. In: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10. EMNLP '02, Stroudsburg, PA, USA, Association for Computational Linguistics (2002) 41–48
22. Beringer, G., Jablonski, M., Januszewski, P., Szymański, J.: https://github.com/gberinger/automatic-wiki-links (2018)
23. spaCy: https://spacy.io/models/en (2016)