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 improve the performance in NLP tasks such as syntactic parsing or sentiment analysis. While useful, they are problematic in representing ambiguous words with multiple meanings, since they keep a single representation for each word in the vocabulary. Constructing separate embeddings for meanings of ambiguous words could be useful for solving the Word Sense Disambiguation (WSD) task.

In this work, we present how a word embeddings average-based method can be used to produce semantic-rich meaning embeddings, and how they can be improved with distance optimization techniques. We also open-source a WSD dataset that was created for the purpose of evaluating methods presented in this research.

Keywords: word embeddings, word sense disambiguation

1 Introduction

Word embedding methods, that map the vocabulary words to low-dimensional continuous space, have been widely applied to various natural language processing (NLP) problems. They are commonly used as the input representation of words, replacing high-dimensional one-hot encodings, and have been shown to improve the performance in tasks such as syntactic parsing[1] and sentiment analysis[2].

In word embedding methods, such as word2vec[3] or GloVe[4], each word in the vocabulary has exactly one representation. While it is enough for most words, it is problematic for ambiguous words, which can contain more than one meaning. For example, consider the following examples with the word *tree*, extracted from Wikipedia articles:

(a) *The olive, known by the botanical name Olea europaea, meaning "European olive", is a species of small* **tree** *in the family Oleaceae [...]*

(b) *Many theories of syntax and grammar illustrate sentence structure using phrase* **trees***, which provide schematics of how the words in a sentence are grouped and relate to each other.*

(c) *Upon completion of listing all files and directories found,* **tree** *returns the total number of files and directories listed.*

All three sentences mention the word *tree* (or *trees*), but the meaning differs based on context - (a) means tree as a forest plant, (b) tree as a parse tree, (c) tree as a command in Unix systems.

For many applications, such as improving relevance of search engines, anaphora resolution or coherence, identifying which meaning is used, based on context, is important. This task is called Word Sense Disambiguation (WSD) and is an open problem in NLP domain. Word embeddings cannot be applied to WSD out-of-the-box, since they cannot differentiate between multiple meanings of an ambiguous word.

In this work, we suggest to create semantically rich embeddings for each *keyword* (ambiguous word together with meaning, e.g. *tree (structure)*, *pool (computer science)*) (4.1), by averaging embeddings of the ambiguous word and words describing its meaning (4.3). We evaluate this approach on a small WSD task, gathered from Wikipedia articles (3). Word is disambiguated by choosing the closest keyword embedding given the embedding of context, which is an average of word embeddings surrounding the word (4.2). We then propose an optimization method, to improve the quality of embeddings and the result on WSD task as well - we move keyword embeddings closer to embeddings of contexts they appear in, using examples from the training set (4.4). A number of experiments is run to test the proposed method and its optimized version (5, 6). Finally, we discuss our results and propose future work (7).

2 Related work

There have been many methods of creating semantically meaningful word representations. Global matrix factorization methods, such as latent semantic analysis (LSA)[5], use matrix factorization to perform dimensionality reduction on a large term-frequency matrix, that captures statistical information about the corpus. As the result, we receive word and document embeddings, which are parametrized by the number of topics we want to extract from the documents, and which can be used to find similarities between different words and documents.

Other approach to creating word embeddings is to take only local context into account, without using global statistics. Example of this is word2vec[3], where a shallow neural network is trained to either predict context words based on the current word (skig-gram), or predict current word based on context words (continuous bag-of-words). Continuous representations of words are then extracted from the hidden layer of the trained network. Such word embeddings were shown to outperform previous approaches at word analogy and similarity tasks. FastText[6] improves upon skip-gram method, by representing each word as a bag of character n-grams, which provides more flexibility and has an added benefit of the ability to compute word representations for words unseen during training.

Global Vectors (GloVe)[4] combine both global matrix factorization and local context window methods, by training word vectors on co-occurence matrix, so that their differences predict co-occurence ratios. Unlike word2vec[3], GloVe takes advantage of global count statistics instead of only local information, which results in better performance on some word analogy and similarity benchmarks.

Word embeddings can be also extracted from a trained language model[7]. Recently, methods like ELMo[8] or BERT[9] were shown to achieve great results in many NLP tasks. They produce deep contextualized word embeddings by using internal states of a trained language model pre-trained on large corpus of text. Since models used are bidirectional (LSTM for ELMo, Transformer for BERT), the word embedding is conditioned on its left and right context, achieving flexible vector representations that could be used to disambiguate words.

Iacobacci et al.[10] were the first to try to use word embeddings for Word Sense Disambiguation. 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.[11] 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.

The most usual baseline for WSD task is the Most Frequent Sense[12] (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.[13] 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.[14] 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.

3 Dataset

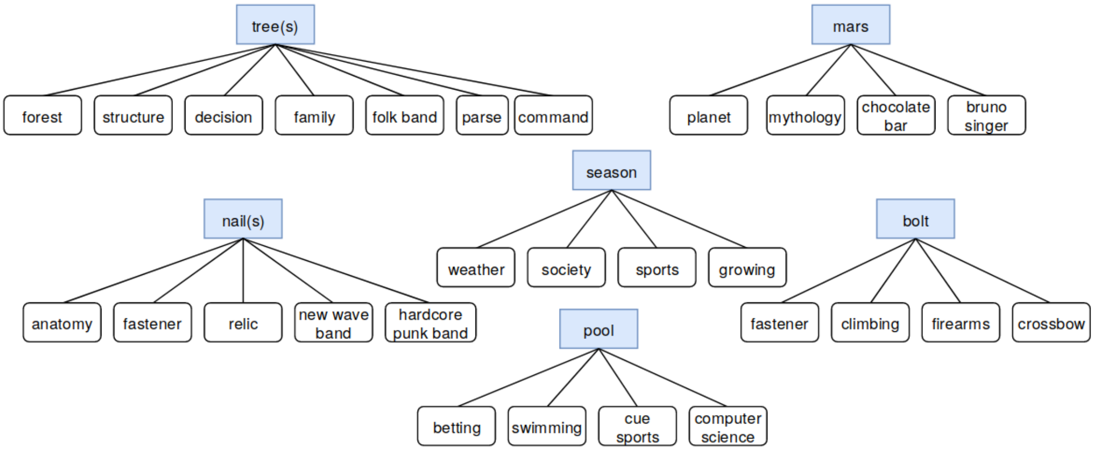
For the purpose of constructing semantic-rich word embeddings, we manually gathered usage examples for 6 ambiguous words, 4 to 7 meanings each (28 meanings in total) (Table 1). Ambiguous word together with its meaning constitues a *keyword*, which we use as a separate class when identyfing the closest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

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

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)*.



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[1].

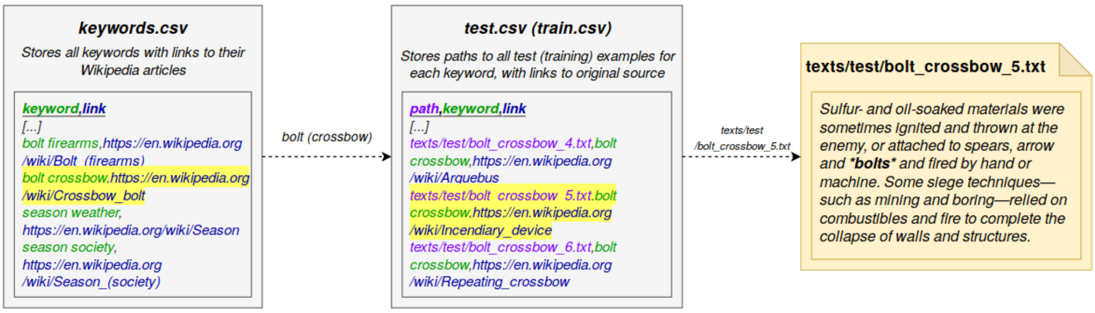
4 Our method

4.1 Keyword embedding

Keyword is a sequence of words that is composed of the ambigous word and words describing its specific meaning, e.g. *tree (forest)* that represents tree as a plant (ambiguous word: *tree*, meaning: *forest*) and *tree (structure)* which represents tree as a mathematical structure (ambiguous word: *tree*, meaning: *structure*).

[1] https://github.com/gberinger/automatic-wiki-links

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).


keywords.csv
Stores all keywords with links to their Wikipedia articles
keyword,link
[...]
bolt firearms,https://en.wikipedia.org/wiki/Bolt_(firearms)
bolt crossbow,https://en.wikipedia.org/wiki/Crossbow_bolt
season weather,https://en.wikipedia.org/wiki/Season
season society,https://en.wikipedia.org/wiki/Season_(society)
bolt (crossbow)
test.csv (train.csv)
Stores paths to all test (training) examples for each keyword, with links to original source
path,keyword,link
[...]
texts/test/bolt_crossbow_4.txt,bolt crossbow,https://en.wikipedia.org/wiki/Arquebus
texts/test/bolt_crossbow_5.txt,bolt crossbow,https://en.wikipedia.org/wiki/Incendiary_device
texts/test/bolt_crossbow_6.txt,bolt crossbow,https://en.wikipedia.org/wiki/Repeating_crossbow
texts/test/bolt_crossbow_5.txt
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.

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

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

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, extracted from some text, which contains an ambiguous word and words surrounding it in text. It is parametrized by **context length** l, which specifies how many words from both sides of the ambiguous word are 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* $\underline{\textit{pool}}$ *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 Baseline 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 baseline 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.

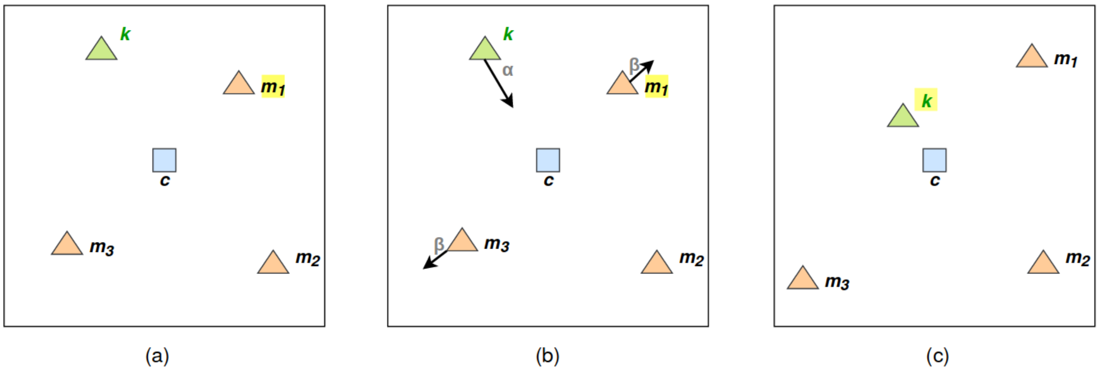
We try to optimize the system even further, by also moving top-k closest keywords that are incorrect given the same context.

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

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).



5 Experiments

The goal of following experiments is to check how semantic-rich keyword embeddings (4.1, 4.2) perform for the dataset we collected (3), both for the basic approach (4.3) 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[2].

[2] https://spacy.io/models/en#section-en_vectors_web_lg

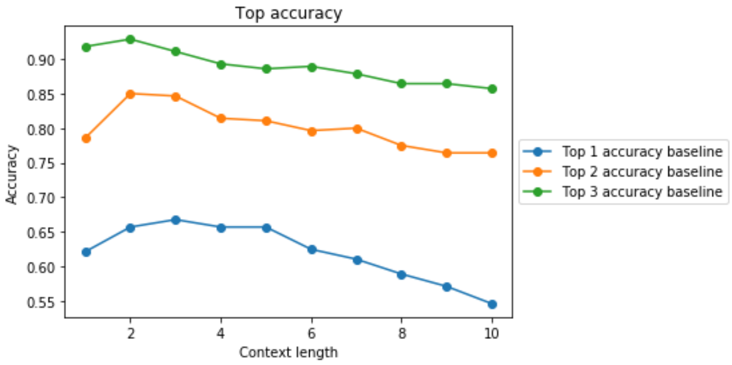
We compare results on the test set with top-k metrics ($k \in 1, 2, 3$), where we check, if the correct keyword is in closest k keywords given a specific context describing it (4.3). We focus mostly on top-1 accuracy, since we are interested if the word is correctly disambiguated.

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.

5.1 Baseline

As the baseline for optimization experiments, we check how keyword and context embeddings perform without any extra optimizations (4.3). We evaluate the performance of the baseline model with different context lengths, to see how it affects top-k accuracies (Fig. 4).

Fig. 4. Top-k 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 (6.2), 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 an 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 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 (6.2).

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-k accuracies for $l = 3$ (Fig. 6). Top-1 accuracy improves 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.

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

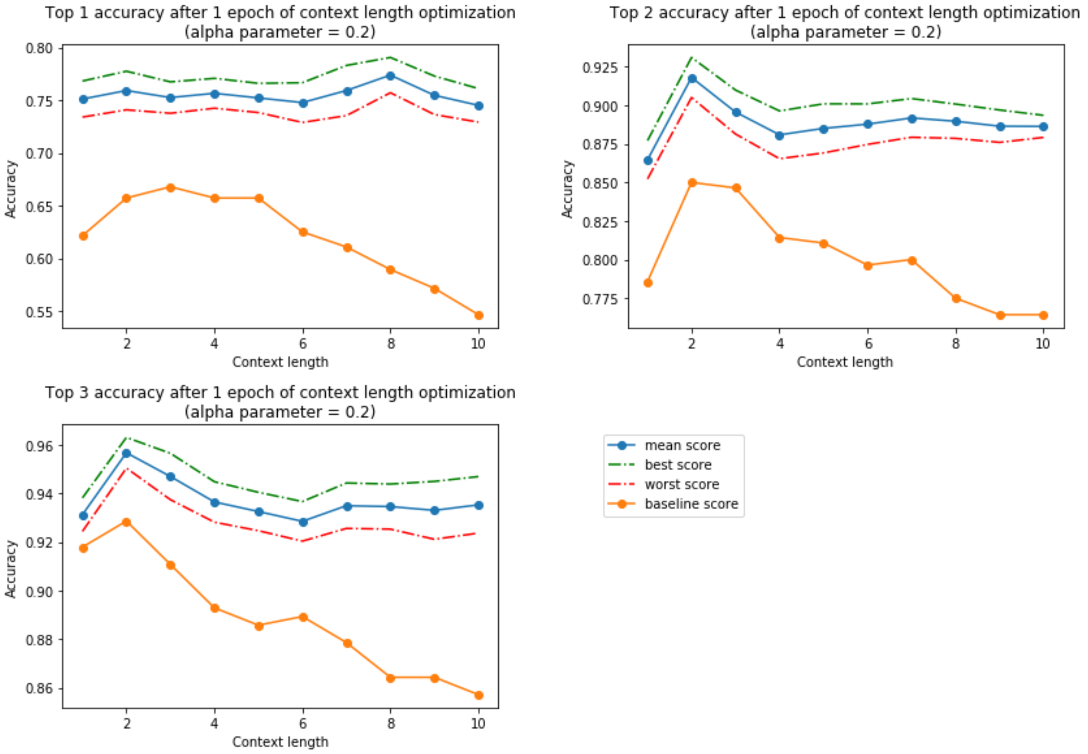
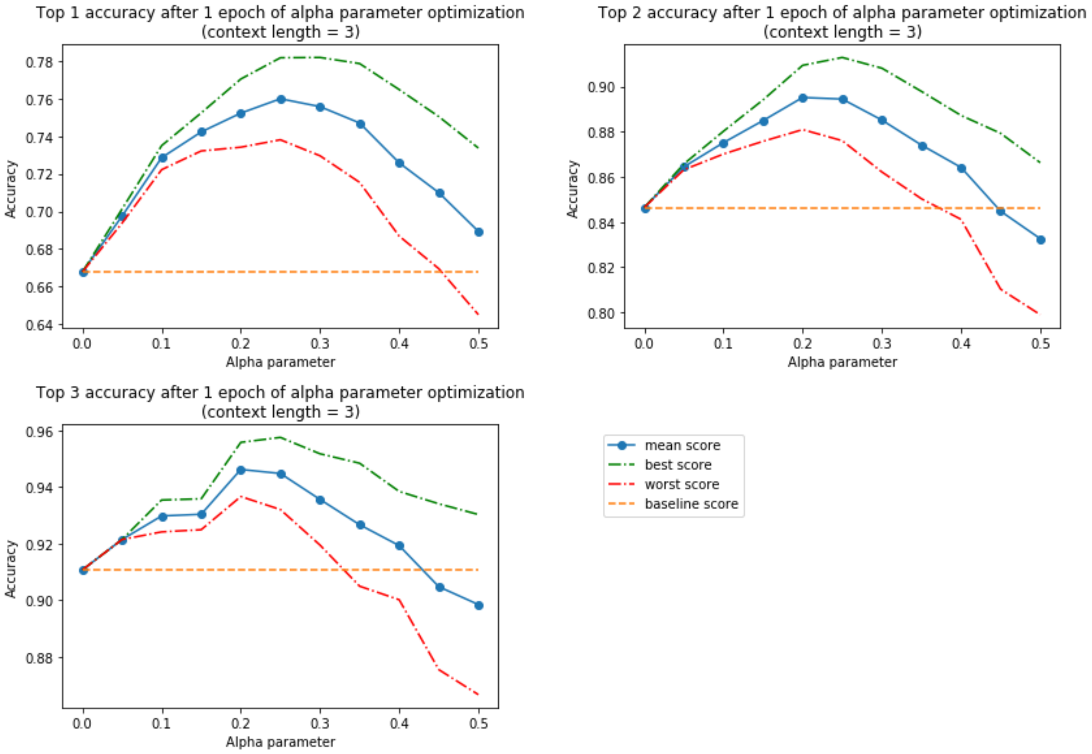


Fig. 6. Influence of α parameter value on top-k accuracies for $l = 3$. Blue line denotes the average result from 30 runs, dotted lines denote worst and best result. The value for $\alpha = 0$ is the result achieved by the baseline model (5.1).

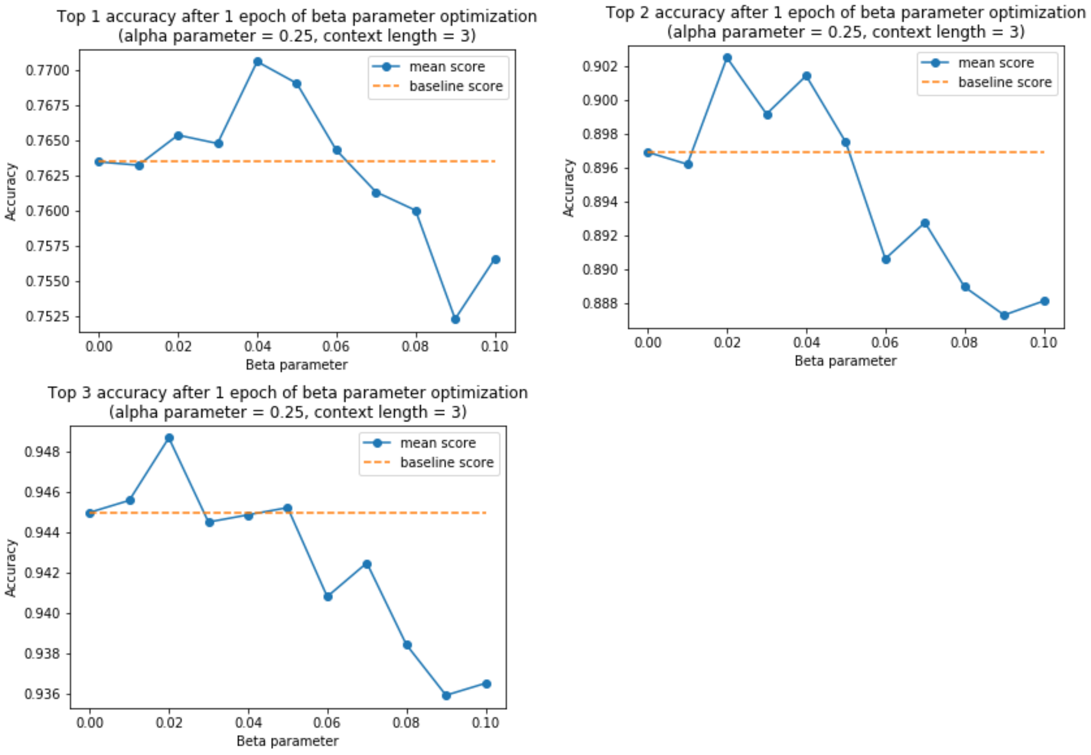


5.3 Beta optimization

We proved that moving keyword embeddings closer to their contexts is beneficial for the accuracy on the test set (5.2). We assume, 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.

Fig. 7. Influence of β parameter value on top-k 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).



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

5.4 Accuracy in training epochs

In previous experiments, we performed only one epoch of training, i.e. we used each training example only once for keyword embeddings optimization. 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).

It can be seen, 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, similar to what happens, if we choose a big learning rate when training neural networks. We therefore check what happens, when we choose lower values of these parameters (Fig. 9).

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

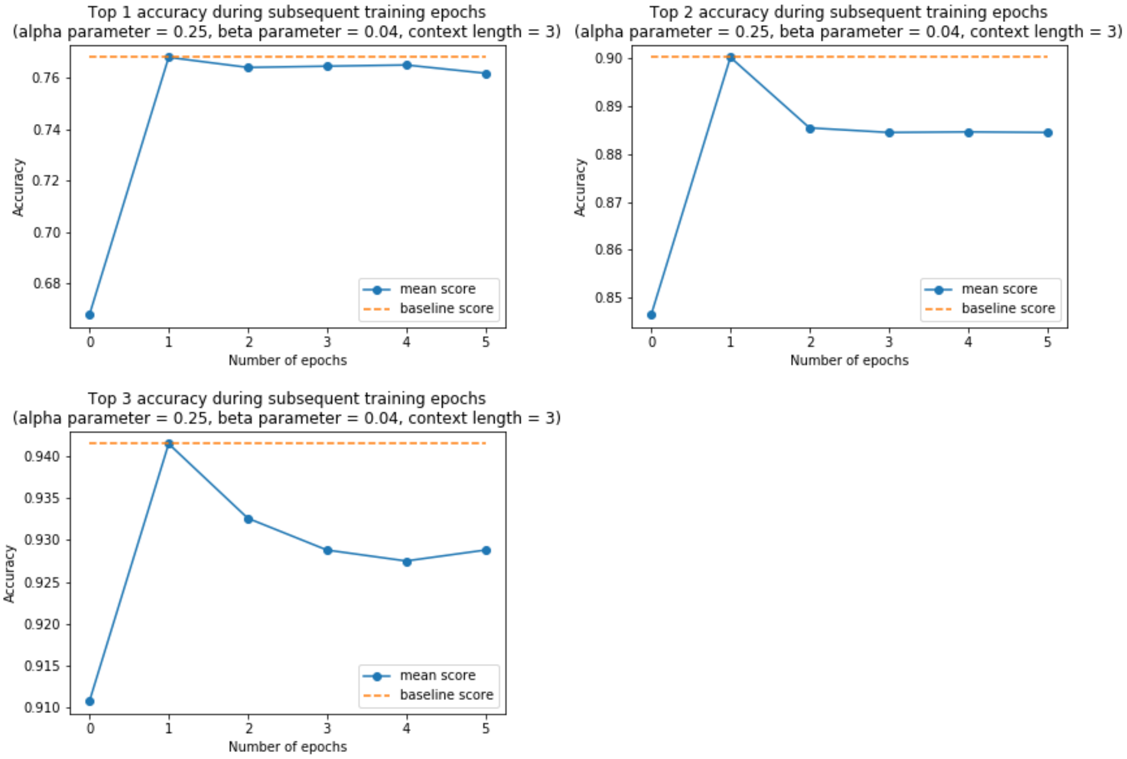
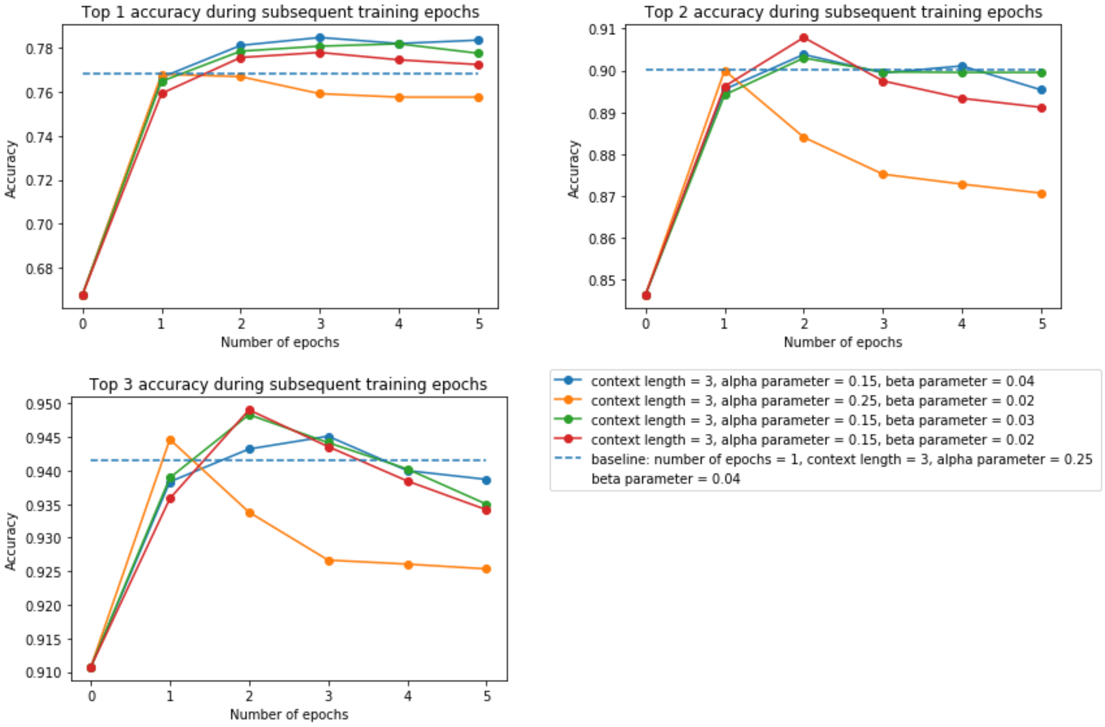


Fig. 9. Comparison of models optimized with different α and β values with $l = 3$ for 5 epochs.



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 (7) 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 Discussion

6.1 Model analysis

We take a look at how the proposed model performs before (4.3) and after optimization (4.4). For this purpose, we take a model with context length $l = 4$ as the baseline and then optimize it on the training data (3) for 4 epochs (5.4), using $\alpha = 0.15, \beta = 0.04$.

Metric	**Baseline**	**Optimized**
Avg. cosine distance	0.348763	0.264627
Top-1 accuracy	66%	78%
Top-2 accuracy	81%	90%
Top-3 accuracy	89%	93%

Table 2. Results achieved on the test set for the model before (baseline) and after optimization (4 epochs, $\alpha = 0.15, \beta = 0.04$). Cosine distance is measured between the correct keyword and context embeddings.

Performance on the test set can be seen in Table 2. All metrics improved due to the optimization process of moving correct keywords closer to (and incorrect keywords away from) contexts found in the training set. High top-2 and top-3 accuracies suggest, that the correct keyword is usually relatively close to the context describing it, even with the baseline model.

For the purpose of analysing which keywords get confused with each other, confusion matrices were constructed for each ambiguous word, for the baseline and optimized model (due to space constraints, Fig. 10-15 were moved to Appendix A). Model was not limited to choosing between the meanings of the specific ambiguous word, but nevertheless, it relatively rarely chose the meaning of unrelated ambiguous word (especially after optimization). It suggests, that the ambiguous word (which is always a part of context) is usually enough for the context embedding to shift in the direction of meanings connected to this word.

Analysis of confusion matrices also tells us, that some keywords are easier to recognize than other, i.e. the ambiguous word often appears together with its meaning (e.g. *decision tree* or *growing season*, which are coined phrases) or the context in which they appear is often very similar and distinct from the rest (e.g. *tree (command)*). Moreover, keywords of similar meaning are more likely to get confused, e.g. *nails (new wave band)*, *nails (hardcore punk band)* and *trees (folk band)*, or *tree (decision)*, *tree (structure)* and *tree (parse)*.

It is important to note, that the performance might worsen, if we expand the keyword vocabulary to large-scale experiments, where we have much more possible keywords than 28 (Table 1). We suggest checking the proposed solution on large-scale dataset as possible future work (7).

6.2 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 (7), 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.

7 Conclusion and future work

Constructing semantic-rich embeddings for ambiguous words, by taking the average of embeddings of the ambiguous word and words describing its meaning (4.1), and then comparing it with the average embedding of context words describing said keyword (4.2), proved to be a surprisingly good approach for the task of disambiguation on the dataset of 28 keywords we collected (3). The baseline model (4.3) achieved 67% top-1, 85% top-2 and 93% top-3 accuracy (5.1) for context length $l = 3$. Longer context lengths were shown to decrease the accuracy, since the average of many word embeddings may result in similar embeddings for different contexts.

To improve the quality of embeddings, an optimization method of moving keyword embeddings closer to contexts they appear in was suggested (4.4). We have shown, that accuracy on the disambiguation task can be improved up to about 79% top-1, 90% top-2 and 95% top-3 accuracy, even with a few training examples per keyword. Importance of choosing parameters α (Eq. 3) and β (Eq. 4) was shown, with α being the most crucial for final accuracy (5.2). Moving incorrect keyword embeddings, given some context embedding, controlled by β parameter, was shown to minimally improve results (5.3) over only using the α parameter. The accuracy was further increased, by performing the optimization over many epochs (i.e. using the training examples more than once, in random order), but with lower values of α than with a single epoch (5.4).

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 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.

It would also be interesting to see, how the suggested approach for constructing semantic-rich embeddings would perform on a large-scale dataset. Such a dataset could be automatically collected from Wikipedia, using disambiguation pages to find ambiguous words and their meanings, and *What links here* utility, to find usage examples for each keyword.

References

1. 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
2. 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
3. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
4. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: In EMNLP. (2014)
5. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE **41** (1990) 391–407
6. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606 (2016)
7. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3** (2003) 1137–1155
8. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. (2018)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2018)
10. 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
11. 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

12. 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
13. 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
14. 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

A Appendix

Fig. 10-15 depict confusion matrices for meanings of ambiguous words, displaying how frequently each keyword was confused with others, before (4.3)(left matrices) and after optimization (4.4)(right matrices). Each confusion matrix was limited to meanings of one ambiguous word, with other keywords labeled as *other* on the graph.

Model with context length $l = 4$ was used. Optimization was done for 4 epochs with $\alpha = 0.15, \beta = 0.04$ on the train set. Both baseline and optimized model were evaluated on the test set of 10 usage examples per keyword (3).

Fig. 10. Confusion matrices for *tree* keywords.

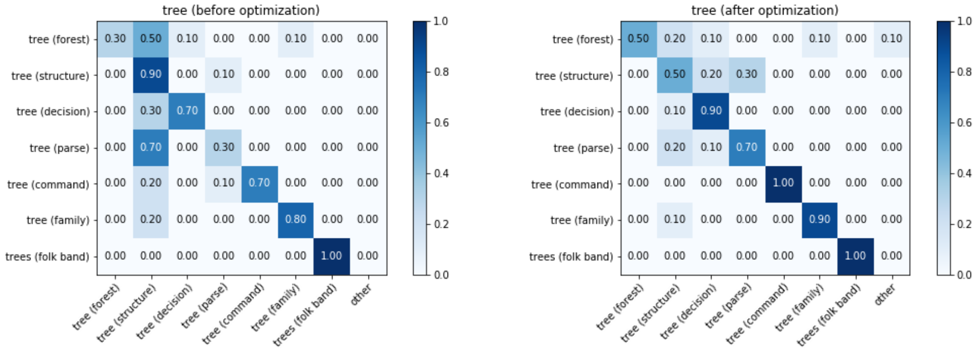


Fig. 11. Confusion matrices for *nail* keywords.

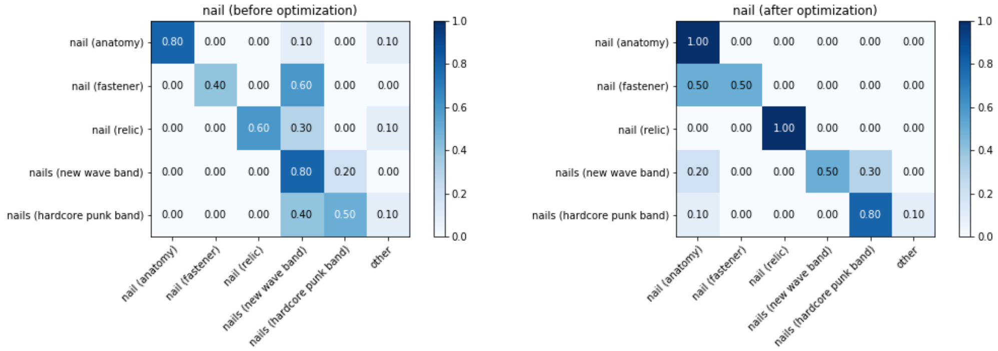


Fig. 12. Confusion matrices for *mars* keywords.

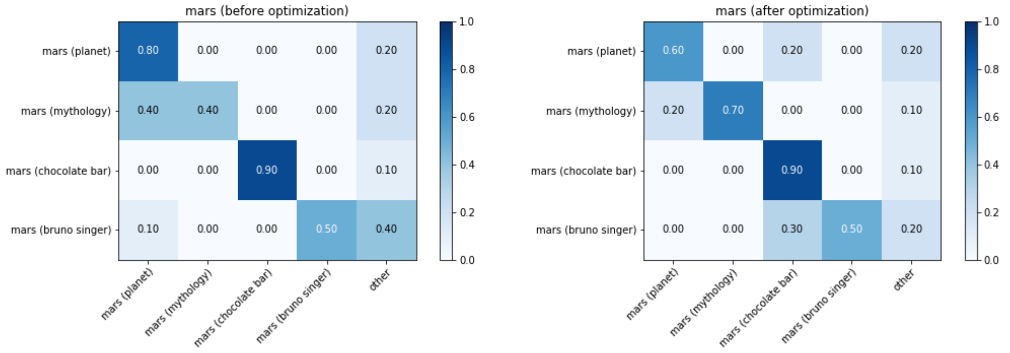


Fig. 13. Confusion matrices for *bolt* keywords.

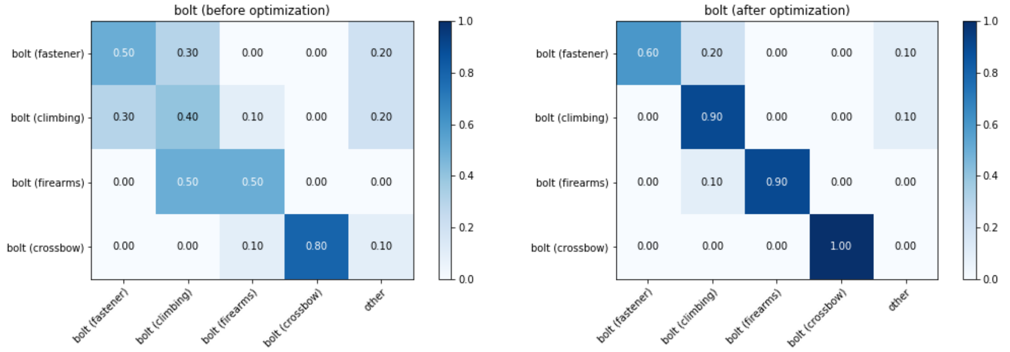


Fig. 14. Confusion matrices for *season* keywords.

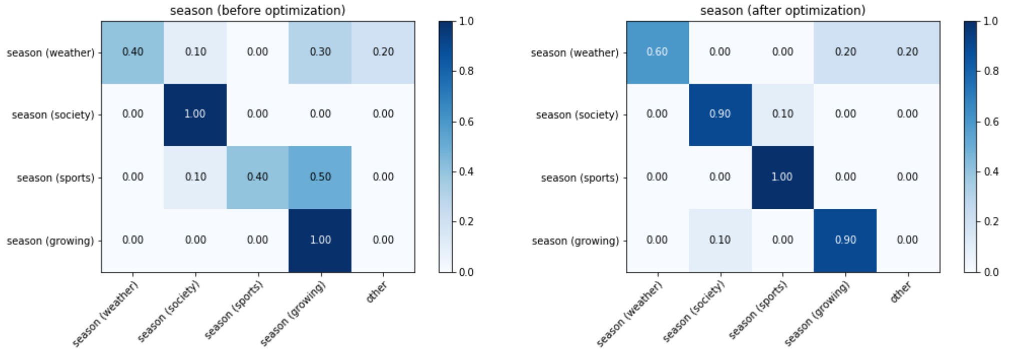
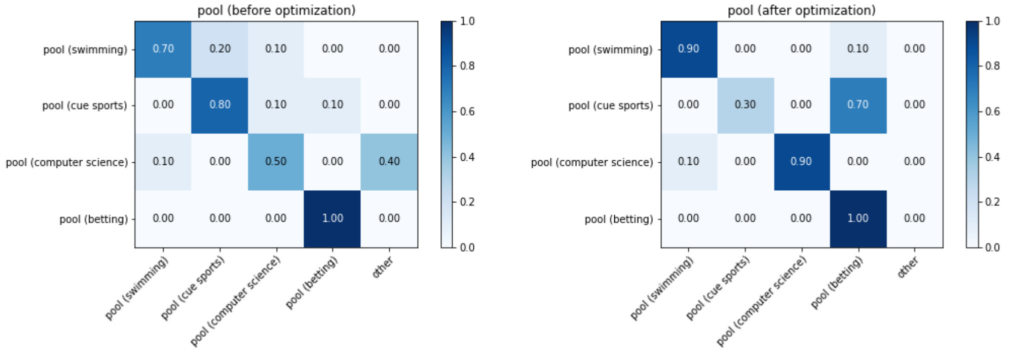


Fig. 15. Confusion matrices for *pool* keywords.


pool (before optimization)
pool (swimming) 0.70 0.20 0.10 0.00 0.00
pool (cue sports) 0.00 0.80 0.10 0.10 0.00
pool (computer science) 0.10 0.00 0.50 0.00 0.40
pool (betting) 0.00 0.00 0.00 1.00 0.00
pool (swimming)
pool (cue sports)
pool (computer science)
pool (betting)
other
pool (after optimization)
pool (swimming) 0.90 0.00 0.00 0.10 0.00
pool (cue sports) 0.00 0.30 0.00 0.70 0.00
pool (computer science) 0.10 0.00 0.90 0.00 0.00
pool (betting) 0.00 0.00 0.00 1.00 0.00
pool (swimming)
pool (cue sports)
pool (computer science)
pool (betting)
other