BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **18**, No 1
Sofia • 2018 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2018-0012

Neural Network Models for Word Sense Disambiguation: An Overview

Alexander Popov

Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria
E-mail: *alex.popov@bultreebank.org*

***Abstract*: *The following article presents an overview of the use of artificial neural networks for the task of Word Sense Disambiguation (WSD). More specifically, it surveys the advances in neural language models in recent years that have resulted in methods for the effective distributed representation of linguistic units. Such representations – word embeddings, context embeddings, sense embeddings – can be effectively applied for WSD purposes, as they encode rich semantic information, especially in conjunction with recurrent neural networks, which are able to capture long-distance relations encoded in word order, syntax, information structuring.*

***Keywords*: *Word sense disambiguation, neural networks, long short-term memory cells, word embeddings, sense embeddings, context representation.*

1. Introduction

1.1. Task definition and applications

Word Sense Disambiguation (WSD) is defined as the task of making automatic choices with regards to which sense of a word is used in a particular context. This problem in Natural Language Processing (NLP) is a long-standing one and concerted work on it dates back to the beginning of the field, starting with the intensive Machine Translation (MT) research projects in the middle of the last century (in fact, it is to some extent the difficulty of WSD that stalled advances in MT and discouraged researchers).

WSD is important for NLP because of the richness and ambiguity of natural languages. Open-class words often have multiple meanings, either due to polysemy (i.e., the existence of separate but related senses corresponding to the same word) or to homonymy (the phenomenon of semantically unrelated concepts being expressed with the same word). Thus, for instance, “knife” can be used either as a noun (an instrument for cutting) or as a verb as the product of conversion/zero derivation (“to knife somebody to death”); if it is a noun, it can have a more specific meaning (a cutting weapon) or a more general one (“any long thin projection that is transient” – definition taken from WordNet 3.1). Another ambiguous case is illustrated by the classic example from the literature: “bank” can mean the earth on the side of a river, or it can mean a financial institution, etc.

These ambiguities are important in many respects, as the lexicon stands as a sort of interface between various levels of linguistic analysis. Ascertaining the correct sense of a word is obviously linked to tagging it correctly with a part-of-speech category label; different verb senses have different syntactic subcategorization frames (i.e., they take a different number and different types of arguments), they impose selectional restrictions on nouns; noun senses in turn can combine with different adjectives, they can take on different semantic roles, etc. Word senses are also important in that they are situated in complex semantic networks where separate senses are interconnected in a web of lexical relations such as hypernymy, meronymy, antonymy, synonymy, etc.

Thus, word senses provide an entry into world knowledge (in the shape of ontologies, for instance) that can be used to enrich the text and make it, to a certain extent, comprehensible to a machine. Such information is useful when establishing coreference in texts, identifying lexical chains, etc. WSD is therefore, unsurprisingly, also important outside of pure linguistic analysis, i.e., when used in practical applications, such as machine translation, where words and senses across languages are not mapped in one-to-one relations; it is also a rich resource of information for building information extraction systems (rule-based or statistical), for information retrieval, question answering, etc.

There are different ways to precisely define the WSD task. This overview will focus such variations where there is a lexicon available, providing word senses for the different words, and the problem at hand is to pick the correct one to be used in a context. Following the Senseval (**http://www.senseval.org/**) competitions, there are two popular variations of this task: the all-words and the lexical sample tasks. The first one requires that an automatic system disambiguate all open-class words in a context, while the second focuses on a limited number of words sampled in some way from a lexicon (therefore usually target words appear alone in the training and test sentences).

1.2. Traditional approaches to WSD

Despite the relevance of WSD to the automatic analysis of language and the long tradition of research in the field, as of yet it is not a task that can be handled well enough for it to make a vital difference. There are a number of reasons for that, among them: data sparseness (there are too many open-class words with too many associated senses, so that it is very expensive to annotate wide-coverage corpora), the difficulty of constructing good lexicons (word sense distinctions can get too coarse – or fine-grained; consequently, the disambiguation either does not provide enough expressivity, or it is too imprecise), the difficulty of handling long-distance context (sometimes, to correctly disambiguate a word you need to look at the other end of the sentence, or even beyond the sentence boundary).

Several broad families of methods for WSD have been explored: supervised, unsupervised and knowledge-based (semi-supervised methods like bootstrapping have also been explored with some good results). Unsupervised models can make use of large amounts of data, but they are constrained by the fact that they can merely cluster distinct usages of words – they cannot employ precompiled lexicons with sense distinctions. Knowledge-based methods are attractive in that they do not require any data (annotated or raw) other than a lexical resource, such as a dictionary or a computational lexicon (e.g., WordNet [1]); for instance, the Lesk algorithm [2] only needs the word sense glosses to compute word overlap with the words in the context (or with the words in their own glosses, depending on the version of the algorithm), while in [3] Agirre and Soroa employ a random walk algorithm to iteratively traverse part of the semantic graph of a lexical resource in order to determine which of the possible senses in a context are the most probable to appear together.

Supervised methods have achieved the best results in WSD tasks. They perform better than knowledge-based ones but need significant amounts of data. The best-performing system in recent years has been IMS, which uses an SVM to do the disambiguation, looking at a window-bounded context around the target word and extracting rich features from it [4]. This system has been used in conjunction with distributed representations of words derived via neural networks – Taghipour and Ng [5] demonstrate that the addition of such features increases its accuracy; a later study [6] explores different ways for more sophisticated integration of distributed representations as features to IMS (such as *fractional weighting* of the vectors and *exponential decay weighting*, the farther they are from the target word). Those yield even higher results: F1 scores of 69.9, 75.2 and 89.4 for Senseval 2, 3 and 7, respectively, whereas the original IMS numbers are 65.3, 72.9, and 87.9. Other machine learning algorithms that have been explored for WSD include: decision lists and trees, naïve Bayes classifiers, k-nearest neighbors (for a detailed survey see [7]).

In the next two sections, neural network approaches to supervised WSD will be explored as alternatives to the ones already mentioned. Neural networks have the advantage of being able to naturally incorporate distributed representations of words, to learn on their own hidden features and to handle long contexts. The overview will focus both on ways of representing words and senses via neural networks and on ways to directly perform classification, given an input context.

2. Embedding words, contexts and word senses for WSD

Before moving to neural models that are designed specifically for the WSD task, the overview will summarize some of the popular neural network language models for obtaining distributed representations of linguistic units, since those have contributed to a wave of significant advances in NLP in recent years. Such distributed representations, also called embeddings (because the original space is transformed to a lower-dimensional one), usually of words, are especially appropriate as input features to a WSD algorithm, because they encode in a tightly-packed way a host of semantic and grammatical distinctions to which this task should be sensitive.

2.1. Feedforward neural network language models

The idea of word embeddings has been around since [8]. Word embeddings are in many ways similar to count-based Language Models (LMs), which have been very successful in NLP. The latter operate by obtaining maximum likelihood estimations of the occurrence of particular sequences of words in texts, but are restricted by the so-called *curse of dimensionality*: word sequences of larger length are unlikely to occur often or at all, therefore it is impossible to train LMs based on higher order n-grams (5-grams is a typical choice). Smoothing is usually applied to fight this problem, but it is a difficult task in itself. Neural network LMs overcome this obstacle by performing dimensionality reduction, which allows for the clustering of words; therefore finding the next most probable word no longer relies on having seen this exact sequence, but rather on having seen similar ones (and similarity is defined along the reduced dimensions of the embedding space).

[9] used a multilayer feedforward neural network to train such a model and obtain distributed representations. The models proposed achieved significantly better results on the perplexity measure than n-gram models, after being trained on corpora of 1 and 15 million words. However, training in this way takes a lot of time and makes experimental work very challenging. In a later work Ronan and Weston [10] obtained word embeddings using a convolutional neural network, which was also trained to perform a host of other NLP tasks such as POS tagging, chunking, named entity recognition, semantic role labeling. Fig. 1 provides a visual example of a feedforward network for obtaining embeddings.

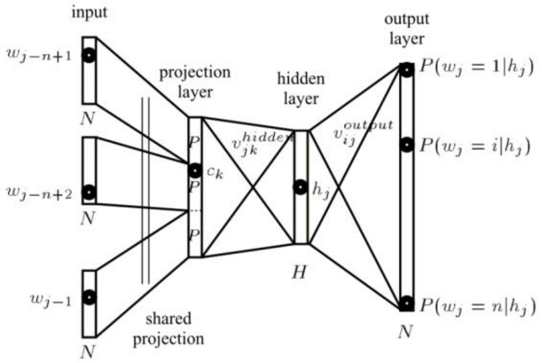

Fig. 1. A feedforward network that projects a context of *n* words before the target one, does computations in the hidden layer and then outputs probabilities for the different possible realizations of the target word (from [11])

2.2. Recurrent neural network language models

Further improvements to neural network language models have been proposed subsequently, such as, most significantly, using a Recurrent Neural Network (RNN) for learning. Such a model has obvious advantages over a feedforward architecture: it avoids the restriction of a window-based approach, has the capability of keeping an unlimited history, as well as of a short-term memory. Another advantage, particularly useful in the case of WSD, is the ability to compress whole contexts into low-dimensional vectors, not merely separate words.

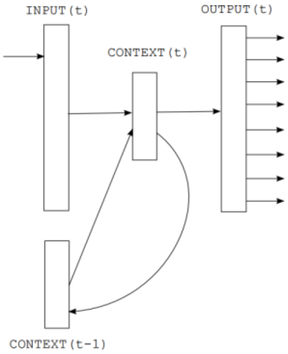

Fig. 2. Graphic representation of a recurrent neural network (from [11])

An RNN operates via a hidden layer that has connections to itself, i.e., it has access to its own memory, which is being updated as the time series progresses. This is analogous to having a very deep feedforward neural network with one hidden layer per time slice (Fig. 2 gives a visual representation of the idea). Additional modifications have been proposed – in order to fight the problem of the *vanishing gradients*, whereby the backpropagated error gradients grow too small with longer sequences and learning becomes impossible. There are two popular modifications of RNNs proposed in the literature that tackle this issue: Long Short-Term Memory (LSTM) cells and Gated Recurrent Units (GRUs). An LSTM cell [12] is internally more complex than the vanilla hidden layer of a simple RNN. It has a cell state that is modified selectively, according to which past and present input information is considered the most relevant by the cell. This is accomplished by special *gates* that regulate the network's focus on the current input and its decisions to forget unnecessary past information; there is also an output gate that controls what is being passed forward. Here is a variant of the equations that define the gates and the cell state (taken from [13]; for a visual representation, see Fig. 3).

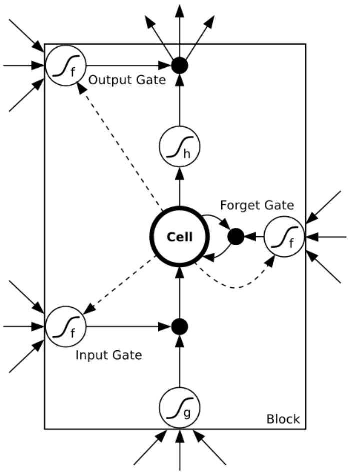

Fig. 3. A graphic representation of an LSTM cell (from [13])

(1) $$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i),$$
(2) $$f_t = \sigma(W_{xi}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f),$$
(3) $$c_t = f_tc_{t-1}\ i_t\tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c),$$
(4) $$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o),$$
(5) $$h_t = o_t\tanh(c_t),$$
where i is the *input gate*, f is the *forget gate*, c is the *cell state*, o is the *output gate* and h is the next state that the hidden layer passes along (as output or as memory); σ is the sigmoid activation function and tanh is the hyperpolic tangent.

GRUs [14] are similar to LSTMs, but are conceptually simpler in that they have fewer parameters and lack an output gate.

2.3. Shallow neural network language models

The most significant breakthrough in producing distributed representations of words came with [15]. That work introduced simpler neural architectures that are trainable in reasonable time, due to the fact that they do not have hidden non-linear layers (i.e., no non-linear activation functions are used on the single hidden layer of the network). These shallow networks are less powerful than deep feedforward neural nets, let alone RNNs, but they compensate with much greater speed. It turns out that by training on large data (on the order of billions of words; e.g., the pre-trained Google vectors (Downloadable from: **https://code.google.com/archive/p/word2vec/**) distributed freely online are trained on 100 billion words) these simple networks can achieve good representations for individual words. This is demonstrated through experiments on word similarity and relatedness datasets, as well as on analogy making. Simple arithmetical operations on the vectors give impressive and immediately tangible results that suggest the representations do encode particular semantic features in the shared space (e.g., “France” – “Paris” + “Berlin” = “Germany”, where the names stand for the respective word vectors).

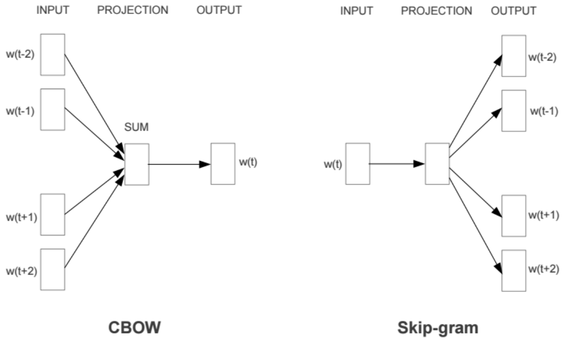

Fig. 4. The CBOW and Skip-gram architectures (from [15])

The aforementioned work introduced two simple architectures for obtaining word embeddings: CBOW and Skip-gram. CBOW projects a number of words around the target word to the lower-dimensional space and sums their vectors, then it maps the context to a vocabulary-sized vector and obtains a probability distribution over it (via a softmax function); the training procedure aims to optimize the network with respect to the correct target word. The Skip-gram model is similar, only it attempts the reverse – to predict the surrounding context on the basis of a single word in its center (the two architectures are graphically represented in Fig. 4). The final step in these models – the softmax function – is very expensive with large vocabularies (those can grow up to hundreds of thousands or even millions of words when training on large corpora). One way to alleviate this is by using a *hierarchical softmax* [15], which represents the vocabulary as a Huffman binary tree, or *negative sampling* [16], which compares the probability for the correct target word with a small number of sampled incorrect ones.

2.4. Embedding word senses

Word embeddings, even though they are powerful features, are not perfectly suited to all NLP tasks. On one hand, in some cases similar distributed information on other levels of analysis could be more useful: morpheme-wise or character-wise, for example. On the other, word embeddings in many cases bundle together information about multiple word senses that can encode subtly or widely divergent meanings. Regarding the first observation, there is a number of works that have explored embedding words on the character level (e.g., [17]) or on the level of individual morphological elements (e.g., [18]). There have also been a number of attempts to derive *sense embeddings*, i.e., distributed representations of word senses rather than of word forms.

Producing sense embeddings in the same way as word embeddings is difficult, as not enough annotated data is available and the construction of such resources is prohibitively expensive in large quantities. Here are discussed a few methods for the creation of sense embeddings that attempt to get around this problem. In the next section some further strategies are introduced that make use of recurrent neural networks.

One way to approach the task is to automatically annotate large enough corpora with word senses (SemCor [19], the largest annotated lexical resource, has around 200K annotated word senses, while word embeddings are usually trained on billions of words). [20] presents one such attempt, in which the BabelNet (**http://www.babelnet.org/**) sense inventory (a merger of several resources, most notably WordNet and Wikipedia) is used to tag a training corpus: a dump of Wikipedia. The Babelfy (**http://www.babelfy.org/**) tool, which employs a random walk on graph algorithm, is used to perform the disambiguation stage; the senses annotated with a high enough confidence score are selected as final, otherwise the procedure backs off to the most frequent sense. The senses annotated with high confidence account for about half of the content words in the sense inventory. This automatically generated corpus of several billion words is fed into the word2vec tool (**https://code.google.com/archive/p/word2vec/**) (more specifically to the CBOW algorithm) and sense embeddings are produced from it. The vector representations are tested on word and relational similarity datasets, giving state-of-the-art results, thus suggesting that sense embeddings can indeed offer more nuanced information than word embeddings.

Another possible approach is via utilizing already existent word embeddings and inferring the sense embeddings from them. For instance, Johansson and Pina [21, 22] use lemma embeddings produced via the Skip-gram method and interpret those as convex combinations of the different possible senses per each lemma. The method minimizes a *neighborhood* metric between sense nodes, based on the relational structure of lexical resources (e.g., the hypernymy relations in WordNet). Thus, only word/lemma embeddings and a computational lexicon are needed to create the sense embeddings under this strategy.

Still another way for obtaining sense embeddings is AutoExtend [23]. This work adds to the paradigm of word embeddings the concepts of synset and lexeme emeddings (synsets being sets of interchangeable synonyms and lexemes being pairings of words and synsets). It builds on the idea that a lexical resource can be used to impose constraints on embeddings and therefore to allow a system to extend word embeddings to embeddings of other data types. Words are interpreted as the sums of their lexemes, and synsets, analogously, as the sums of *their* attendant lexemes. In this way, words, lexemes and synsets are situated in the same embedding space. The learning of the embeddings itself is accomplished via an autoencoding framework where the input and output layers of the autoencoder are the word embeddings and the hidden layer gives the synset embeddings. Lexeme embeddings are defined by the transition from word to synset embeddings. Additional constraints are defined via the WordNet relations in order to force similar synsets to stay close in the embedding space.

Finally, there is at least one more line of work that offers an alternative opportunity for producing training corpora for word2vec-style embedding creation. [24] employs the PageRank algorithm [25] in order to traverse the relational structure of WordNet and simultaneously with that to emit series of artificial “sentences” – i.e., the algorithm hops along connected vertices and emits their IDs, until the hops are terminated (a probability variable ensures this), then it initiates a new sequence-hopping (each one is a new “sentence”). This particular work replaces the vertex IDs with associated lemmas for the synsets and uses the corpus to train lemma embeddings, which are shown to be competitive with “regular” word embeddings on the word similarity and relatedness tasks. However, nothing precludes the analogous creation of synset embeddings in the exact same manner. This approach is attractive in that it requires no manually-annotated data and because it allows different graph topologies to be tested for the creation of the corpora (which should force the resultant embeddings to encode different semantic relations; this is demonstrated in [26]).

3. Recurrent neural networks for WSD

As discussed previously, RNNs are powerful mechanisms for modeling sequential data, which is especially relevant when analyzing language. Being able to keep a trace of arbitrarily long contexts is crucial for more complex NLP tasks where long-distance dependencies need to be modeled. The variants already mentioned of RNNs – LSTMs and GRUs – make this possible in practice by allowing the network to learn how to selectively focus on relevant knowledge seen at separate time steps. A relatively simple modification to RNNs can render them even more powerful with regards to language processing – making them *bidirectional*. This means using two RNNs in tandem – one reading the input sequence (e.g., sentence) in natural order, the other doing it in reverse. Thus, at each time step the representations of the *forward* and the *backward* RNNs can be concatenated, which in effect combines information about preceding and following context. Bidirectional LSTMs have been successfully applied to different NLP tasks, such as POS tagging [27], chunking [28], NER [29], dependency parsing [30], etc.

The following subsections summarize some ways in which RNNs have been applied to WSD.

3.1. Using RNNs for context embedding

One of the usages of RNNs for WSD is a simple extension of the ideas presented in Section 2: To transform input contexts into vectors in an embedding space. The context vectors can then be compared to the possible word sense vectors for the target word that is to be disambiguated. Context embedding can be accomplished in different ways, the simplest of them being the bag-of-words approach, which simply averages the word embeddings in a particular context. This strategy loses any information about word order and is thus not very effective (See, however, [31], where word embeddings are trained with the specific task of sentence representation – the network tries to predict neighboring sentences on the basis of the context. This gives competitive results and is very fast compared to more sophisticated methods.).

Word-order-information-preserving context representations that use embeddings include weighting the word embeddings [32] or combining vectors in an order dictated by the parse trees of context sentences [33]. Le and Mikolov [34] in turn use a network that learns word embeddings and simultaneously with that learns paragraph embeddings, where the latter are used as additional features to context words for the prediction of target words; paragraph embeddings in this work operate as a sort of global memory that influences the classification.

Recurrent neural networks fit naturally in this research paradigm – they preserve word order information without requiring any additional information about the context, such as syntactic parses; their major disadvantage is their relatively slow speed. [35] presents *context2vec* (Fig. 5), which uses a bidirectional LSTM to learn word and context embeddings simultaneously. Instead of merely averaging the word vectors in the context like in CBOW, the Bi-LSTM encodes two contexts – one to the left and one to the right of the target word – and concatenates them, feeds them to a MultiLayer Perceptron (MLP) and finally uses the output of the MLP to obtain an objective function with regards to the embedding of the target word.

The word-centered contexts obtained in this way can be used for WSD as outlined above – context embeddings are produced for each of the tagged instances per sense of the target word; then the context embedding for the target word is compared via a similarity metric to the different example contexts (essentially an application of the *k*-nearest-neighbor algorithm with *k*=1). The authors evaluated *context2vec* on the Senseval-3 lexical sample dataset and obtained results almost on par with the best-scoring systems at the time (see also [36] for an evaluation on the Senseval all-words lexical tasks, which puts *context2vec* almost as high as the IMS system, the best-scorer in that study). [37] is similar to *context2vec*, but produces generic sentence representations that are not related to a specific target word.

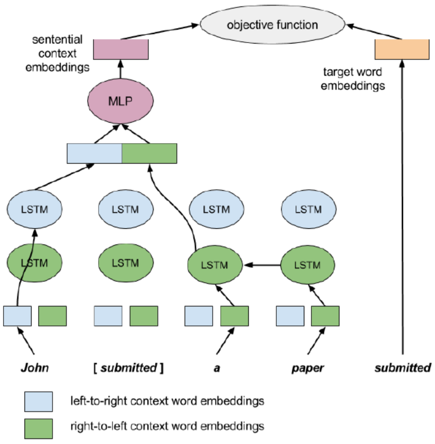

Fig. 5. The *context2vec* architecture (from [35])

Another similar study directly related to WSD is [38] (Fig. 6). The authors train an LSTM language model to predict a held-out word. The last state of the hidden layer encodes the context of the word. The context representations are again compared to the possible sense embedding for the target word (obtained by running the example sentences for the senses through the LSTM and then averaging them) – via cosine similarity. This study improves its results additionally by enriching in a semi-supervised manner the number of example sentences, thus better approximating the decision boundary between different senses (as the nearest neighbor algorithm assumes a spherical shape for the sense clusters, which often hurts its performance due to the low number of available examples).

Note that with all of the discussed approaches, it would be possible to use sense embedding obtained in different ways – for instance, one of those outlined at the end of the previous section.

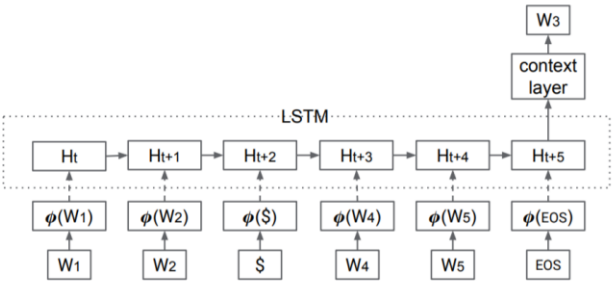

Fig. 6. A neural network with an LSTM hidden layer that encodes a context with regards to a hidden word (represented via the ϕ sign). The final hidden state is the context representation, which is then used to choose the hidden word (from [38])

3.2. Direct WSD with RNNs

Kageback and Salomonsson [39] (Fig. 7) also use LSTMs for WSD, but do not calculate similarity between a context and pre-calculated sense embedding. Rather, their system directly classifies synsets. The architecture uses a Bi-LSTM to encode a representation of the target word – the forward and backward RNNs provide the left and right contexts, which are concatenated and put through a non-linearity, and finally a softmax function carries out the classification phase. This approach is obviously designed to deal with a single target word per pass. All words share the Bi-LSTM parameters, so that the model can be improved after the processing of each training case, but separate words have their own models for compressing the representations to vectors of the necessary length (to match the number of synsets) and for the softmax classification. The study reports state-of-the-art results on the Senseval-2 and 3 lexical sample tasks. It uses a number of regularization techniques, most notably *dropword* – randomly hiding words in the input sequences, so as to make the model more robust to overfitting.

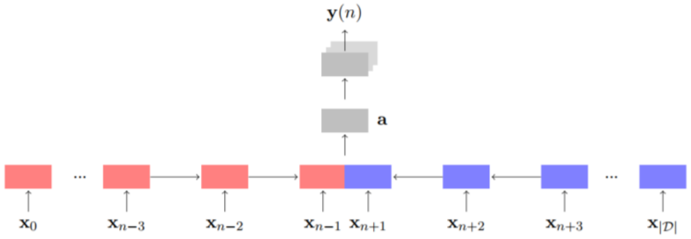

Fig. 7. A Bi-LSTM centered on a word at position *n*. The accumulated left and right contexts are concatenated and the result is used for classification. The final layer is a softmax activation function that is parameterized with regards to separate words (from [39])

4. Conclusion

This article has surveyed recent advances in artificial neural network architectures, with regards to their use for solving the word sense disambiguation task. It has provided an overview of different ways for obtaining distributed word representations and on how to extend this idea to representing other elements of text, such as word senses and contexts. It has also examined a number of successful applications of recurrent neural networks to the disambiguation task. RNNs are uniquely well-suited to deal with the rich and long-reaching dependencies that determine and reflect the use of word senses; therefore they are rightly being explored as one of the most promising approaches to making advances in the field.

References

1. Miller, G. A. WordNet: A Lexical Database for English. – Communications of the ACM, Vol. **38**, 1995, No 11, pp. 39-41.
2. Lesk, M. Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. – In: Proc. of 5th Annual International Conference on Systems Documentation, ACM, 1986.

3. Agirre, E., A. Soroa. Personalizing Pagerank for Word Sense Disambiguation. – In: Proc. of 12th Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2009.
4. Zhong, Z., H. T. Ng. It Makes Sense: A Wide-Coverage Word Sense Disambiguation System for Free Text. – In: Proc. of ACL 2010 System Demonstrations. Association for Computational Linguistics, 2010.
5. Taghipour, K., H. T. Ng. Semi-Supervised Word Sense Disambiguation Using Word Embeddings in General and Specific Domains. – HLT-NAACL, 2015.
6. Iacobacci, I., M. T. Pilehvar, R. Navigli. Embeddings for Word Sense Disambiguation: An Evaluation Study. – ACL, Vol. **1**, 2016.
7. Navigli, R. Word Sense Disambiguation: A Survey. – ACM Computing Surveys (CSUR), Vol. **41**, 2009, No 2, p. 10.
8. Hinton, G. E., J. L. Mcclelland, D. E. Rumelhart. Distributed Representations, Parallel Distributed Processing. – Explorations in the Microstructure of Cognition, Vol. **1**, Foundations, 1986.
9. Bengio, Y., R. Ducharme, P. Vincent, C. Jauvin. A Neural Probabilistic Language Model. – Journal of Machine Learning Research, Vol. **3**, February 2003, pp. 1137-1155.
10. Ronan, C., J. Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. – In: Proc. of 25th International Conference on Machine Learning, ACM, 2008.
11. Mikolov, T., M. Karafiát, L. Burget, J. Cernocký, S. Khudanpur. Recurrent Neural Network Based Language Model. – Interspeech, Vol. **2**, 2010.
12. Hochreiter, S., J. Schmidhuber. Long Short-Term Memory. – Neural Computation, Vol. **9**, 1997, No 8, pp. 1735-1780.
13. Graves, A., A.-R. Mohamed, G. Hinton. Speech Recognition with Deep Recurrent Neural Networks. – IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13), IEEE, 2013.
14. Cho, K., B. V. Merriënboer, D. Bahdanau, Y. Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. – arXiv preprint arXiv:1409.1259, 2014.
15. Mikolov, T., K. Chen, G. Corrado, J. Dean. Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781, 2013.
16. Mikolov, T., I. Sutskever, K. Chen, G. Corrado, J. Dean. Distributed Representations of Words and Phrases and Their Compositionality. – In: Advances in Neural Information Processing Systems, 2013.
17. Chen, X., L. Xu, Z. Liu, M. Sun, H. B. Luan. Joint Learning of Character and Word Embeddings. – In: IJCAI, 2015.
18. Thang, L., R. Socher, C. D. Manning. Better Word Representations with Recursive Neural Networks for Morphology. – In: CoNLL, 2013.
19. Miller, G. A., C. Leacock, R. Tengi, R. T. Bunker. A Semantic Concordance. – In: Proc. of Workshop on Human Language Technology, Association for Computational Linguistics, 1993.
20. Iacobacci, I., M. T. Pilehvar, R. Navigli. SensEmbed: Learning Sense Embeddings for Word and Relational Similarity. – ACL, Vol. **1**, 2015.
21. Johansson, R., L. N. Pina. Embedding a Semantic Network in a Word Space. – HLT-NAACL, 2015.
22. Johansson, R., L. N. Pina. Combining Relational and Distributional Knowledge for Word Sense Disambiguation. – In: Proc. of 20th Nordic Conference of Computational Linguistics, NODALIDA 2015, 11-13 May 2015, Vilnius, Lithuania, No 109, Linköping University Electronic Press, 2015.
23. Sascha, R., H. Schütze. Autoextend: Extending Word Embeddings to Embeddings for Synsets and Lexemes. – arXiv preprint arXiv:1507.01127, 2015.
24. Josu, G., A. Soroa, E. Agirre. Random Walks and Neural Network Language Models on Knowledge Bases. – In: HLT-NAACL, 2015.
25. Page, L., S. Brin, R. Motwani, T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab, 1999.

26. Simov, K., P. Osenova, A. Popov. Comparison of Word Embeddings from Different Knowledge Graphs. – In: International Conference on Language, Data and Knowledge. Springer, Cham, 2017.
27. Wang, P., Y. Qian, F. K. Soong, L. He, H. Zhao. Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network. – arXiv preprint arXiv:1510.06168, 2015.
28. Wang, P., Y. Qian, F. K. Soong, L. He, H. Zhao. A Unified Tagging Solution: Bidirectional LSTM Recurrent Neural Network with Word Embedding. – arXiv preprint arXiv:1511.00215, 2015).
29. Huang, Z., W. Xu, K. Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. – arXiv preprint arXiv:1508.01991, 2015.
30. Wang, W., B. Chang. Graph-Based Dependency Parsing with Bidirectional LSTM. – ACL, Vol. **1**, 2016.
31. Kenter, T., A. Borisov, M. de Rijke. Siamese Cbow: Optimizing Word Embeddings for Sentence Representations. – arXiv preprint arXiv:1606.04640, 2016.
32. Chen, X., Z. Liu, M. Sun. A Unified Model for Word Sense Representation and Disambiguation. – EMNLP, 2014.
33. Socher, R., C. C. Lin, C. Manning, A. Y. Ng. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. – In: Proc. of 28th International Conference on Machine Learning (ICML'11), 2011.
34. Le, Q., T. Mikolov. Distributed Representations of Sentences and Documents. – In: Proc. of 31st International Conference on Machine Learning (ICML'14), 2014.
35. Melamud, O., J. Goldberger, I. Dagan. Context2vec: Learning Generic Context Embedding with Bidirectional LSTM. – CoNLL, 2016.
36. Raganato, A., J. Camacho-Collados, R. Navigli. Word Sense Disambiguation: A Unified Evaluation Framework and Empirical Comparison. – Proc. of EACL, 2017.
37. Kiros, R., Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, S. Fidler. Skip-Thought Vectors. – Advances in Neural Information Processing Systems, 2015.
38. Yuan, D., J. Richardson, R. Doherty, C. Evans, E. Altendorf. Semi-Supervised Word Sense Disambiguation with Neural Models. – arXiv preprint arXiv:1603.07012, 2016.
39. Kågebäck, M., H. Salomonsson. Word Sense Disambiguation Using a Bidirectional LSTM. – arXiv preprint arXiv:1606.03568, 2016.

Received 01.07.2017; *Second Version 06.10.2017*; *Accepted 23.10.2017*