Bidirectional LSTM Recurrent Neural Network for Keyphrase Extraction

Marco Basaldella(✉), Elisa Antolli, Giuseppe Serra, and Carlo Tasso

Artificial Intelligence Laboratory, Department of Mathematics, Computer Science, and Physics, University of Udine, Udine, Italy
{marco.basaldella,giuseppe.serra,carlo.tasso}@uniud.it,
antolli.elisa@spes.uniud.it

Abstract. To achieve state-of-the-art performance, keyphrase extraction systems rely on domain-specific knowledge and sophisticated features. In this paper, we propose a neural network architecture based on a Bidirectional Long Short-Term Memory Recurrent Neural Network that is able to detect the main topics on the input documents without the need of defining new hand-crafted features. A preliminary experimental evaluation on the well-known INSPEC dataset confirms the effectiveness of the proposed solution.

1 Introduction

Keyphrases (herein KPs) are phrases that "capture the main topic discussed on a given document" [31]. More specifically, KPs are phrases typically one to five words long that appear verbatim in a document, and can be used to briefly summarize its content.

The task of finding such KPs is called Automatic Keyphrase Extraction (herein AKE). It has received a lot of attention in the last two decades [11] and recently it has been successfully used in many Natural Language Processing (hence NLP) tasks, such as text summarization [34], document clustering [10], or non-NLP tasks such as social network analysis [23] or user modeling [24]. Automatic Keyphrase Extraction approaches have been also applied in Information Retrieval of relevant documents in digital document archives which can contain heterogeneous types of items, such as books articles, papers etc. [15].

The first approaches to solve Automatic Keyphrase Extraction were based on supervised machine learning (herein ML) algorithms, like Naive Bayes [32] or C4.5 decision trees [31]. Since then, several researchers explored different ML techniques such as Multilayer Perceptrons [2,19], Support Vector Machines [19], Logistic Regression [2,9], and Bagging [14]. Since no algorithm stands out as the "best" ML algorithm, often authors test many techniques in a single experiment, and then they choose as best ML algorithm the best performing one [2,9] and/or even the least computationally expensive one [19].

M. Basaldella and E. Antolli—Equally Contributed.

G. Serra and C. Tasso (Eds.): IRCDL 2018, CCIS 806, pp. 180–187, 2018.
https://doi.org/10.1007/978-3-319-73165-0_18

However, AKE algorithms based on unsupervised approaches have been developed over the years as well. For example, Tomokiyo *et al.* [30] proposed to use a language model approach to extract KPs, and Mihalcea *et al.* [21] presented a graph-based ranking algorithm to find keyphrases. Nevertheless, supervised approaches have been the best performing ones in challenges: for example, [19], a supervised approach, was the best performing algorithm in the SEMEVAL 2010 Keyphrase Extraction Task [16].

In the last years, most attention is devoted to the *features* used in these supervised algorithms. The numbers of features used can range from just two [32] to more than 20 [9]. These features can be divided in categories identified with different kinds of knowledge they encode into the model:

- *statistical knowledge:* number of appearances of the KP in the document, TF-IDF, number of sentences containing the KP, etc.;
- *positional knowledge:* position of the first occurrence of the KP in the document, position of the last occurrence, appearance in the title, appearance in specific sections (abstract, conclusions), etc.;
- *linguistic knowledge:* part-of-speech tags of the KP [14], anaphoras pointing to the KP [2], etc.;
- *external knowledge:* presence of the KP as a page on Wikipedia [6] or in specialized domain ontologies [19], etc.

However, given the wide variety of lexical, linguistic and semantic aspects that can contribute to define a keyphrase, it difficult to design hand-crafted feature, and even the best performing algorithms hardly reach F1-Scores of 50% on the most common evaluation sets [14,16]. For this reason, AKE is still far from being a solved problem in the NLP community.

In recent years, Deep Learning techniques have shown impressive results in many Natural Language Processing tasks, e.g., Named Entity Recognition, Automatic Summarization, Question Answering, and so on [18,25,27,29]. In Named Entity Recognition, for example, researchers have proposed several Neural Network Architectures

To best of our knowledge, only recently some first attempts to address AKE task with Deep Learning techniques, has been presented [20,33]. In [33], the authors present an approach based on Recurrent Neural Networks, specifically designed for a particular domain, i.e., Twitter data. On the other hand, in [20] the authors use more datasets to evaluate their RNN for keyphrase extraction, and they propose a study of the keyphrases *generated* by their network as well.

In this paper, we present a Deep Learning architecture for AKE. In particular, we investigate an approach based on based on Bidirectional Long Short-Term Memory RNN (hence Bi-LSTM), which is able to exploit previous and future context of a given word. Our system, since it does not require specific features carefully optimized for a specific domain, can be applied to a wide range of scenarios. To evaluate the proposed method, we conduct experiments on the well-known INSPEC dataset [14]. The experimental result showed the proposed solution performs significantly better than competitive methods.

2 Proposed Approach

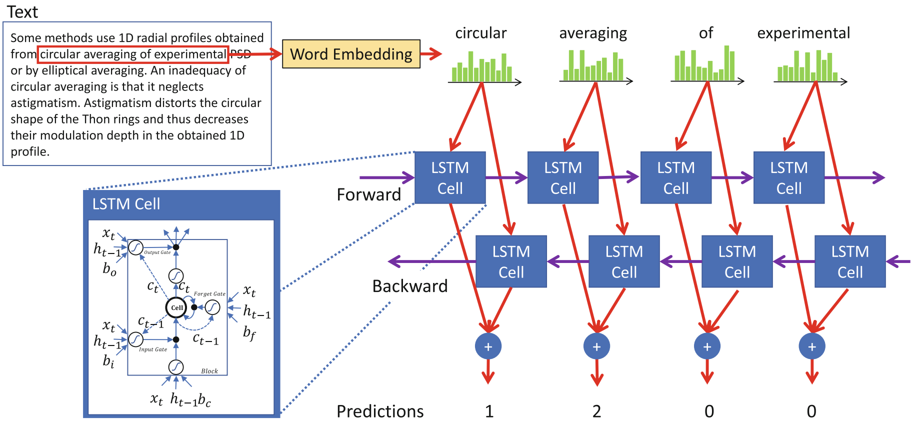
To extract KPs we implemented the following steps, as presented in Fig. 1. First, we split the document into sentences, and then we tokenize the sentences in words using NLTK [3]. Then, we associate a word embedding representation that maps each input word into a continuous vector representation. Finally, we feed our word embeddings into a Bi-LSTM units, which it can effectively deal with the variable lengths of sentences and it is able to analyze word features and their context (for example, distant relation between words). The Bi-LSTM is connected to a fully connected hidden layer, which in turn is connected to a softmax output layer with three neurons for each word. Between the Bi-LSTM layer and the hidden layer, and between the hidden layer and the output layer, we use dropout [28] to prevent overfitting.

As in the techniques used for Named Entity Recognition, the three neurons are mapped to three possible output classes: `NO_KP`, `BEGIN_KP`, `INSIDE_KP`, which respectively mark tokens that are *not* keyphrases, the *first* token of a keyphrase, and the other tokens of a keyphrase.

For example, if our input sentence is "*We train a neural network using Keras*", and the keyphrases in that sentence are "*neural network*" and "*Keras*", the tokens' classes will be `We/NO_KP train/NO_KP a/NO_KP neural/BEGIN_KP network/INSIDE_KP using/NO_KP Keras/BEGIN_KP'`.

2.1 Word Embeddings

The input layer of our model is a vector representation of the individual words contained in input document. Several recent studies [5, 22] showed that such representations, called word embeddings, are able to represent the semantics of words better than an "one hot" encoding word representation, when trained on large corpus. However, the datasets for AKE are relatively small, therefore



Some methods use 1D radial profiles obtained from circular averaging of experimental PSD or by elliptical averaging. An inadequacy of circular averaging is that it neglects astigmatism. Astigmatism distorts the circular shape of the Thon rings and thus decreases their modulation depth in the obtained 1D profile.

Fig. 1. Overview of the proposed system.

it is difficult to train word embeddings to capture the word semantics. Hence, we adopt Stanford's GloVe Embeddings, which are trained on 6 billion words extracted from Wikipedia and Web texts [26].

2.2 Model Architecture

Let $\{x_1, \ldots, x_n\}$ the word embeddings representing the input tokens, a Recurrent Neural Network (hence RNN) computes the output vector y_t of each token x_t by iterating the following equations from $t = 1$ to n:

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

$$y_t = W_{hy}h_t + b_y \tag{2}$$

where h_t is the hidden vector sequence, W denotes weight matrices (for example W_{xh} is the matrix of the weights connecting the input layer and the hidden layer), b denotes bias vectors, and H is activation function of the hidden layer. Equation 1 represents the connection between the previous and the current hidden states, thus RNNs can make use of previous context.

In practice however, the RNN is not able to use effectively the all input history due to the *vanishing gradient* problem [12]. Hence, a better solution to exploit long range context is the Long Short-Term Memory (LSTM) architecture [13]. The LSTM is conceptually defined like an RNN, but hidden layer updates are replaced by specific units called memory cells. Specifically, a LSTM is implemented by the following functions [7]:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{4}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{5}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{6}$$

$$h_t = o_t \tanh(c_t) \tag{7}$$

where σ is the logistic sigmoid function, i, f, o, and c are the input gate, forget gate, output gate and cell activation vectors, and all b are learned biases.

Another shortcoming of RNNs is that they consider only previous context, but in AKE we want to exploit future context as well. For example, consider the phrase "John Doe is a lawyer; he likes fast cars". When we first encounter "*John Doe*" in the phrase, we still don't know whether he's going to be an important entity; then, we find the word "*lawyer*" and the pronoun "*he*", which clearly refer to him, stressing his importance in the context. "*Lawyer*" and "*he*" are called *anaphoras* and the technique to find this contextual information is called *anaphora resolution*, which has been exploited to perform keyphrase extraction in [2].

In order to use future context, in our approach we adopt a Bidirectional LSTM network [8]. In fact, with this architecture we are able to make use of both past context and future context of a specific word. It consists of two separate hidden layers; it first computes the forward hidden sequence $\overrightarrow{h_t}$; then, it computes the backward hidden sequence $\overleftarrow{h_t}$; finally, it combines $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ to generate the output y_t. Let the hidden states h be LSTM blocks, a Bi-LSTM is implemented by the following functions:

$$\overrightarrow{h_t} = H(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \tag{8}$$

$$\overleftarrow{h_t} = H(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}) \tag{9}$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y \tag{10}$$

3 Experimental Results

We present experiments on a well-known keyphrase extraction dataset: the INSPEC dataset [14]. It is composed by 2000 abstract papers in English extracted from journal papers from the disciplines Computer and Control, Information Technology. It consists of 1000 documents for training, 500 for validation and the remaining 500 for testing. We choose this dataset since it's well known in the AKE community, so there are many other available results to compare with; moreover, is much bigger than the dataset used in the SEMEVAL 2010 [16] competition, which contains only 144 documents for training, 40 for validation, and 100 for testing.

In order to implement our approach, we used Keras with Theano [1] as back end, which in turn allowed us to use CUDA to train our networks using a GPU. Experiments are run on a GeForce GTX Titan X Pascal GPU. The network is trained to minimize the Crossentropy loss. We train our network using the Root Mean Square Propagation optimization algorithm [17] and batch size 32. After trying different configurations for the network, we obtained the best results with a size of 150 neurons for the Bi-LSTM layer, 150 neurons for the hidden dense layer, and a value of 0.25 for the dropout layers in between.

To test the impact of word embeddings, we perform experiments with the pre-trained Stanford's GloVe Embeddings using all the word embedding sizes available, i.e., 50, 100, 200 and 300. The training of the network takes about 30 s to perform a full epoch with all the GloVe Embeddings. To stop the training, we used Keras' own embedded early stopping rule, which halts training when the training loss does not decrease for two consecutive epochs. The number of epochs requested to converge in all the four settings is displayed in Table 1, along with precision, recall and F1-score obtained by our system when trained using different sizes of the word embeddings. We can note that the best results are obtained with embedding size of 100; however, the embedding sizes of 200 and 300 obtain a very close result in term of F1-Score. The scores seem to show an interesting pattern: in fact, looking at the results, we see that the precision increases with embedding size, while recall decreases from size 100 onwards.

Table 2 compares the performances in term of precision, recall, and F-score our approach with other competitive systems, based both on supervised and unsupervised machine learning techniques. The first three systems are the ones

Table 1. Performance with different vector sizes of the GloVe Word Embeddings: 50, 100, 200 and 300 (we called them GloVe-(SIZE), respectively).

Embedding	Size	Precision	Recall	F1-score	Epochs
GloVe-50	50	0.331	0.518	0.404	20
GloVe-100	100	0.340	**0.578**	**0.428**	14
GloVe-200	200	0.352	0.539	0.426	18
GloVe-300	300	**0.364**	0.500	0.421	8

Table 2. Comparison results on INSPEC dataset

Method	Precision	Recall	F1-score
Proposed approach	0.340	**0.578**	**0.428**
n-grams with tag [14]	0.252	0.517	0.339
NP chunking with tag [14]	0.297	0.372	0.330
Pattern with tag [14]	0.217	0.399	0.281
TopicRank [4]	**0.348**	0.404	0.352

presented in [14], with three different candidate keyphrase generation techniques: n-grams, Noun Phrase (NP) chunking, and patterns. The fourth system is TopicRank [4], a graph-based keyphrase extraction method that relies on a topical representation of the document. Our proposed solution achieves best performance in term of F1-score and Recall. Although TopicRank obtains best performance in precision, its recall results are significantly worse than the ones obtained by us; moreover, we have to stress that we're able to obtain better precision when using an embedding size of 200 and 300, albeit with a slightly lower overall F1-Score. Finally, it's worth noting that we perform better than the results presented in [20], which is to the best of our knowledge the only one DL AKE algorithm evaluated on the INSPEC dataset. In fact, we obtain a F1@10 score of 0.422, while the best F1@10 score obtained by [20] is 0.342.

4 Conclusion

In this work, we proposed a Deep Long-Short Term Memory Neural Network model to perform automatic keyphrase extraction, evaluating the proposed method on the INSPEC dataset. Since word representation is a crucial step for success, we perform experiments with different pre-trained word representations. We show that without requiring hand-crafted features, the proposed approach is highly effective and achieves better results with respect to other competitive methods. For the future, we plan to test additional network architectures and to evaluate our algorithms on more datasets, in order to demonstrate its robustness.

References

1. Al-Rfou, R., et al.: Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints abs/1605.02688, May 2016, http://arxiv.org/abs/1605.02688
2. Basaldella, M., Chiaradia, G., Tasso, C.: Evaluating anaphora and coreference resolution to improve automatic keyphrase extraction. In: Proceedings of International Conference on Computational Linguistics (2016)
3. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python, 1st edn. O'Reilly Media Inc., Sebastopol (2009)
4. Bougouin, A., Boudin, F., Daille, B.: Topicrank: graph-based topic ranking for keyphrase extraction. In: Proceedings of International Joint Conference on Natural Language Processing (2013)
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
6. Degl'Innocenti, D., De Nart, D., Tasso, C.: A new multi-lingual knowledge-base approach to keyphrase extraction for the Italian language. In: Proceedings of International Conference on Knowledge Discovery and Information Retrieval (2014)
7. Gers, F.A., Schraudolph, N.N., Schmidhuber, J.: Learning precise timing with LSTM recurrent networks. J. Mach. Learn. Res. **3**, 115–143 (2002)
8. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**(5), 602–610 (2005)
9. Haddoud, M., Abdeddaim, S.: Accurate keyphrase extraction by discriminating overlapping phrases. J. Inf. Sci. **40**(4), 488–500 (2014)
10. Hammouda, K.M., Matute, D.N., Kamel, M.S.: CorePhrase: keyphrase extraction for document clustering. In: Perner, P., Imiya, A. (eds.) MLDM 2005. LNCS (LNAI), vol. 3587, pp. 265–274. Springer, Heidelberg (2005). https://doi.org/10.1007/11510888_26
11. Hasan, K.S., Ng, V.: Automatic keyphrase extraction: a survey of the state of the art. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics (2014)
12. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001)
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
14. Hulth, A.: Improved automatic keyword extraction given more linguistic knowledge. In: Proceedings of Conference on Empirical Methods in Natural Language Processing (2003)
15. Jones, S., Staveley, M.S.: Phrasier: a system for interactive document retrieval using keyphrases. In: Proceedings of International ACM SIGIR Conference on Research and development in Information Retrieval (1999)
16. Kim, S.N., Medelyan, O., Kan, M.Y., Baldwin, T.: Semeval-2010 task 5: automatic keyphrase extraction from scientific articles. In: Proceedings of International Workshop on Semantic Evaluation (2010)
17. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of International Conference on Learning Representations (2014)
18. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. In: Proceedings of International Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (2016)

19. Lopez, P., Romary, L.: HUMB: automatic key term extraction from scientific articles in GROBID. In: Proceedings of International Workshop on Semantic Evaluation (2010)
20. Meng, R., Zhao, S., Han, S., He, D., Brusilovsky, P., Chi, Y.: Deep keyphrase generation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers, pp. 582–592. Association for Computational Linguistics (2017). http://aclanthology.coli.uni-saarland.de/pdf/P/P17/P17-1054.pdf
21. Mihalcea, R., Tarau, P.: Textrank: bringing order into texts. In: Proceedings of Empirical Methods on Natural Language Processing (2004)
22. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
23. De Nart, D., Degl'Innocenti, D., Basaldella, M., Agosti, M., Tasso, C.: A content-based approach to social network analysis: a case study on research communities. In: Calvanese, D., De Nart, D., Tasso, C. (eds.) IRCDL 2015. CCIS, vol. 612, pp. 142–154. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41938-1_15
24. De Nart, D., Degl'Innocenti, D., Pavan, A., Basaldella, M., Tasso, C.: Modelling the User Modelling Community (and Other Communities as Well). In: Ricci, F., Bontcheva, K., Conlan, O., Lawless, S. (eds.) UMAP 2015. LNCS, vol. 9146, pp. 357–363. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20267-9_31
25. Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., Ward, R.: Deep sentence embedding using long short-term memory networks: analysis and application to information retrieval. IEEE/ACM Trans. Audio Speech Lang. Process. **24**(4), 694–707 (2016)
26. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: Proceedings of Empirical Methods on Natural Language Processing (2014)
27. Rush, A.M., Chopra, S., Weston, J.: A neural attention model for abstractive sentence summarization. arXiv preprint arXiv:1509.00685 (2015)
28. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
29. Tan, M., Xiang, B., Zhou, B.: LSTM-based deep learning models for non-factoid answer selection. CoRR abs/1511.04108 (2015). http://arxiv.org/abs/1511.04108
30. Tomokiyo, T., Hurst, M.: A language model approach to keyphrase extraction. In: Proceedings of Workshop on Multiword Expressions: Analysis, Acquisition and Treatment (2003)
31. Turney, P.D.: Learning algorithms for keyphrase extraction. Inf. Retriev. **2**(4), 303–336 (2000)
32. Witten, I.H., Paynter, G.W., Frank, E., Gutwin, C., Nevill-Manning, C.G.: KEA: practical automatic keyphrase extraction. In: Proceedings of ACM Conference on Digital Libraries, pp. 254–255 (1999)
33. Zhang, Q., Wang, Y., Gong, Y., Huang, X.: Keyphrase extraction using deep recurrent neural networks on Twitter. In: Proceedings of Conference on Empirical Methods in Natural Language Processing (2016)
34. Zhang, Y., Zincir-Heywood, N., Milios, E.: World Wide Web site summarization. Web Intell. Agent Syst. **2**(1), 39–53 (2004)