Interspeech 2018
2-6 September 2018, Hyderabad, India

Term Extraction via Neural Sequence Labeling
A Comparative Evaluation of Strategies Using Recurrent Neural Networks

Maren Kucza, Jan Niehues, Thomas Zenkel, Alex Waibel, Sebastian Stüker

Karlsruhe Institute of Technology

maren.kucza@student.kit.edu,
{jan.niehues|thomas.zenkel|alexander.waibel|sebastian.stueker}@kit.edu

Abstract

Traditionally systems for term extraction use a two stage approach of first identifying candiate terms, and the scoring them in a second process for identifying actual terms. Thus, research in this field has often mainly focused on refining and improving the scoring process of term candidates, which commonly are identified using linguistic and statistical features. Machine learning techniques and especially neural networks are currently only used in the second stage, that is to score candidates and classify them.

In contrast to that we have built a system that identifies terms via directly performing sequence-labeling with a BILOU scheme on word sequences. To do so we have worked with different kinds of recurrent neural networks and word embeddings.

In this paper we describe how one can built a state-of-the-art term extraction systems with this single-stage technique and compare different network types and topologies and also examine the influence of the type of input embedding used for the task. We further investigated which network types and topologies are best suited when applying our term extraction systems to other domains than that of the training data of the networks.
Index Terms: term extraction, recurrent neural networks, LSTM

1. Introduction

At KIT we have a developed the *Lecture Translator* a system for the simultaneous transcription and translation/interpretation of lectures [1]. Further, the system also offers an archive of the recorded lectures that contains the video stream of the projector signal together with the lecturer's speech, enriched by the transcription of the speech and its translation as subtitles to the video. Through the transcription and translation a 90 minute lecture can now be searched for specific topics. At the same time we also want to enhance the learning experience of the students by automatically cross-linking the content of the lecture with additional external resources. In order to do so, we first need to automatically detect or extract relevant terms in the transcript for which we can then, in a second step, link to relevant external resources.

In this paper we describe our research into this first step, the automatic extraction of relevant terms. We approached the problem as a sequence labeling problem, in which the word sequence of the transcript is the sequence which is labeled with labels indicating relevant terms in the transcript, e.g., via a BILOU labeling scheme [2]. We approached this sequence labeling problems with neural networks, especially recurrent neural networks, which have shown state-of-the-art performance on several sequence labeling tasks in natural language processing in the past [3, 4, 5, 6, 7]. Optimizing network structure and several other parameters experimentally we were able to design a single stage system that gives state-of-the-art performance on a common term extraction task.

1.1. Term Extraction

Term extraction (sometimes also called term recognition) is the task of automatically identifying all domain-specific terminology from a given corpus. In this, term extraction is a sub-task of information retrieval, which can aid various other fields, such as machine translation, ontology creation, knowledge management or document indexing. Contrary to *keyword extraction*, which aims at determining only the most representative words for a given input document, the number of identified terminology is not limited to a few words. Instead it is intended to extract terminology in its entirety. Thereby, the terms extracted can be single words or can be sequences of several words.

Traditionally many term extraction systems rely on a hybrid approach: First linguistic filtering is applied to identify syntactically plausible term candidates; then, in a second step, the candidates are scored and classified using statistical features, special measures or machine learning.

In contrast to that we applied a direct sequence-labeling approach by making use of recurrent-neural networks (RNNs). In our work we researched and compared several strategies in doing so, focusing on aspects such as the importance of the embeddings of the words in the word sequence, different network architectures, and also on the generalization capabilities of the system to data that is from a different domain then the training data.

2. Related Work

Most recent term extraction systems in literature use a two step approach. First candidates are extracted by either noun-phrase POS patterns or by simply taking all possible ngrams, which do not contain stop words, into account. Stop words are frequent words, which hold no deep semantic content, such as conjunctions, articles or prepositions. These candidates are then classified by scoring them using various machine learning techniques.

[8] proposes a method starting by tokenizing and splitting the corpus into 1-5 grams. To reduce the number of ngrams, they are filtered by stop words. For each of these ngrams ten common term extraction features such as total term frequency, c-value, weirdness, etc. are selected. The selected features are classified using either a Random Forest, a Linear Support Vector Machine, a Multinomial Naive Bayes classifier, Logistic Regression or and SGD classifier. While there is no algorithm, which performs best on all test sets, Random Forest achieves the highest scores most often.

Another machine learning approach is given in [9]. Before selecting candidate terms, the system standardizes word vari-

10.21437/Interspeech.2018-2017

ations from the text and annotates POS tags to it. Numbers, punctuation, single characters, stop words and conjugations of "to be" are removed. The remaining words are considered possible terms. Features from statistical and linguistic knowledge and contrastive corpora are calculated for each unigram. The most representative features for term classification are selected by an algorithm based on either correlation or consistency and used with JRip, Naive Bayes, J48 or SMO from WEKA, a java based text classifier. Though none of the algorithms performs best on all of the test sets, the approach achieved state of the art scores on unigram extraction in Brazilian Portuguese.

[10] proposes a method enabling co-training using neural networks. Before candidate terms are selected, plurals of nouns are removed and all words are converted to lowercase. An identifier based on a noun phrase POS pattern and an identifier based on ngram chunking and stop word delimiters select possible terms. Co-training aims to build a stable classificator with only limited annotated data and a majority of unannotated data. It requires two different views on the data, which is achieved by putting the candidate terms in an LSTM and a CNN. After each iteration the most confident term predictions are added to the annotated data.

In contrast, in our work we do not perform any prior candidate selection, but perform sequence labeling via recurrent networks directly on the word sequence of the texts. To our knowledge there is no other system of this kind described in literature.

3. Data

We performed our experiments on the GENIA 3.02 corpus [11] and the ACL RD-TEC 2.0 corpus [12]. The GENIA corpus was used as primary training corpus, while the RD-TEC corpus was mainly used for testing performance on an out-of-domain corpus.

GENIA contains 1,999 Medline abstracts from PubMed. In this collection of biomedical literature terms are assigned to several categories of terms, e.g., *DNA domain or region*. For the sake of simplicity, we did not differentiate between these categories of terms since they are all specific to the biomedical domain. We also removed all sentences from the corpus consisting of only one word, as they do not hold any substantial content. This resulted in a total of 18,427 sentences with 437,307 words and 76,463 annotated terms.

The ACL RD-TEC 2.0 corpus consists of 300 abstracts from the ACL Anthology Corpus. It contains 1,384 valid sentences with 29,921 words and 2,104 annotated terms. Again, we removed sentences with only one word from the corpus.

For training, the corpus data was split into sentences and then partitioned randomly into 80% training data, 10% validation data and 10% test data. For the GENIA corpus, the training set contained 14,741 sentences, the validation set 1,842 and the test set 1,844. Since the batches for the neural networks required sequences of equal length, the sentences were sorted by length in each of these sets. Shorter sequences were padded at the end with special tokens.

Parallel to the input sequence, a BILOU label sequence was generated. Each of the labels represents a word's role in the sequence. Single-word terms were tagged with *U (unit)*, multi-word terms with *B I* L (begin inside* last)* and non-terms with *O (outside)*.

When the network classified a sentence on character level, the BILOU label sequence had to be altered. It was necessary to assign a label to each character in a word, this means the same label was assigned to all characters in the respective word.

4. Term Extraction with Recurrent Neural Networks

Term extraction can be seen as a sequence labeling problem. Whether a word qualifies as a term depends on its syntactic and semantic features. Syntactic features depend on the features of the surrounding words. This characteristic makes sequence labeling a plausible choice.

As RNNs, and among them LSTMs [13] and gated recurrent units networks (GRUs) [14], are especially suited for performing sequence labeling, we focused on LSTMs and also contrasted them against GRUs. We experimentally evaluated several architectures for the task at hand and optimized several meta-parameters incrementally.

4.1. Embedding

Finding a good feature representation for sequences of words is a common problem when processing natural language with neural networks, as one-hot encodings lead to high input dimensionalities due to large vocabulary sizes. It is therefore common practice to perform word embeddings that project the discrete one-hot-encoding vector of words to a lower dimensional but continuous vector [15]. These embeddings are usually trained by building a classificator for an auxillary classification task, such as skip-grams [16]. Since test data will always feature words that have not been seen during training, the proper handling of unknown words is important when performing embedding. There are two general methods for dealing with out-of-vocabulary words:

1. A pretrained word embedding with an *UNK* token, which is used for all out-of-vocabulary words.
2. A character-level embedding, which is independent from the vocabulary size.

In our experiments we contrasted these two ways of performing embedding.

Also, since embeddings are already trained to perform certain classification tasks we also investigated their classification power for the term extraction task by performing classification on word embeddings using a single softmax classification layer.

4.1.1. Word-level Embedding

For word level embeddings, we used a pretrained embedding, namely the GloVe 6B token embedding with 300 dimensional vectors[17]. This word representation is trained in an unsupervised manner on the non-zero entries of a word-word cooccurence matrix. Each entry represents the frequency of which two words occur along with each other. Several different versions of GloVe embedding can be downloaded from the Stanford website. These word vectors were set to untrainable so that they could not be altered during the training of the network. We manually added an UNK token, a *start of sentence token* (SOS) and an *end of sentence token* (EOS). The EOS token was used to padd shorter sequences.

The vocabulary comprises 400,000 lowercase words, so before each word was translated into its respective index, sentences were transformed into lower case and all punctuation was removed.

4.1.2. Character-level Embedding

The character embedding input vocabulary consists of Python's *string.printable* characters as well as an UNK token. The UNK

token is included, as we do not want the program to crash if there are non-ASCII characters in the input sentences when we do not apply any preprocessing. Shorter sequences are padded with a '.' instead of a special EOS token. Thus, the vocabulary size is 101. The character embedding was either trained end-to-end with the input sequences or as part of a language model (intending to predict the next most likely character) and the results compared against each other. In the latter case, the vectors were set to untrainable so that they could not be altered during the training of the sequence classifier. Various tests indicated good results for an embedding dimension of 50.

For our tests with character embedding, we applied different grades of preprocessing, i.e. no preprocessing at all, transformation to lowercase characters and removal of all punctuation combined with only lowercase characters. To ensure that these different grades of preprocessing do not lead to different number of labels in the same sentence, which may distort comparability of the scores, an additional word list and word label list was generated. During the label calculation, the character sequence was aligned with the word list. The labels were then calculated over the characters of each word by either majority voting or score summation.

4.2. Network Topology

The general structure of our model consists of three segments: first an embedding layer, secondly a sequence classifying layer and lastly a linear layer mapping to the five output classes of BILOU encoding. For the second segment we analyzed different configurations:

a) one softmax layer with input window sizes 1, 3 or 5.

b) unidirectional LSTM or GRU

c) bidirectional LSTM or GRU

d) multi-layer LSTM or GRU

4.3. Network Training

The neural network was trained using stochastic gradient descend to minimize cross-entropy loss with a new bob scheduler for the learning rate. I.e., if the validation loss did not decrease by more than a defined threshold, the learning rate was via exponential decay. Training was finished when after the switch to exponential decay the validation loss again did not decrease by more than the defined threshold.

5. Experimental Results

5.1. Metric

To measure the performance of the network we calculated precision, recall and f-score. The scores are calculated separately for every label and batch and then arithmetically averaged.

We defined the baseline as the score achieved, when the most common label (O) is chosen for every word.

5.2. Results

As the test, validation and training set as well as the batches were selected randomly for every test run, the test results for each individual run differed slightly. All given results are averaged over the number of tests.

To test out the capabilities of the network, we experimented with different batch sizes, hidden sizes and number of layers. We did not adjust the learning rate for every experiment as we wanted to determine the stability of the configuration. Stability in this context means whether the training converges for all test runs and how large the deviation of a single test run to the established mean is.

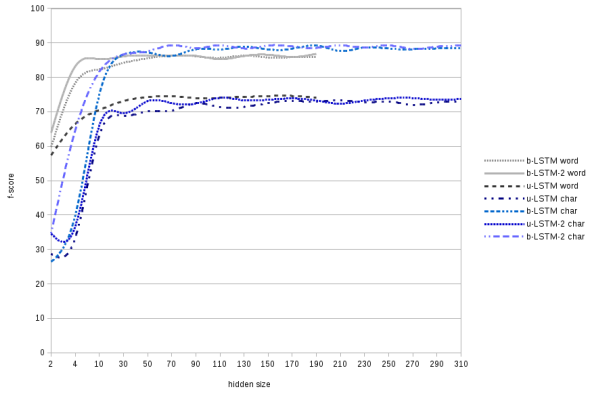

Figure 1: *Impact of different hidden sizes on F-score*

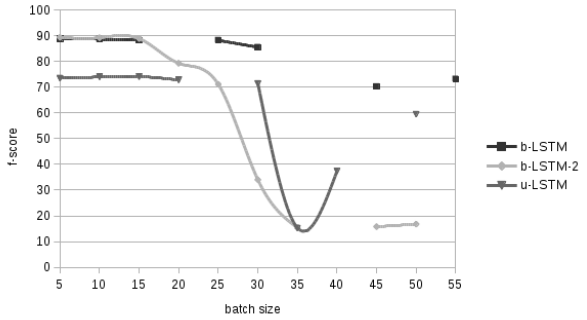

Figure 2: *Testing different batch sizes with the same learning rate of 0.001.*

The results of different hidden sizes in context with either one or two layers is visualized in Figure 1. When word embedding is used, only very small hidden sizes impact the performance. Larger hidden sizes increase the stability of the results, i.e. the deviation between single test runs becomes smaller. It is also shown that two layers have an enhancing effect for small hidden sizes.

Character embeddings are more dependent on a well chosen hidden size. For sizes below 100, the results differ notably between test runs. Like with the word embedding, using two layers has a slightly positive effect for small hidden sizes. For larger hidden sizes, the results are slightly stabler, but not consistently better.

For both kinds of embeddings, there is a kind of threshold. After passing this threshold, an increase of the hidden size mostly affects the stability and does not improve the results further.

Figure 2 shows the result of different batch sizes applied to the same network with the same learning rate. The gaps indicate that the test run did not converge, because the loss became *nan*. With larger batch size, the training becomes increasingly unstable. Some test runs may still converge due to a "well selected" random test set, while for others the learning rate is too high and needs to be adjusted.

	f-score	precision	recall
baseline GENIA	15.99	20.27	13.28
softmax-1	51.78	49.67	54.08
softmax-3	78.56	77.72	79.41
softmax-5	79.87	79.67	80.08
u-LSTM	75.47	75.81	75.13
b-LSTM	86.38	87.06	85.72
b-LSTM-2	86.73	87.03	86.18
u-GRU	74.70	74.65	74.75
b-GRU	86.17	86.71	85.64
b-GRU-2	**86.89**	**87.27**	**86.51**

Table 1: *Results of classification with word embedding.* linear-n *indicates one softmax layer with window size n. In* x-LSTM/GRU-n *x defines uni- or bidirectional, n accounts for the number of layers.*

The results of the network using word embeddings are shown in Table 1. We used the same batch size of 5 and learning rate of 0.001 for all listed configurations. The hidden size was set to 200 for all LSTMs and GRUs.

The listed linear configurations consist of only one softmax classification layer. Different window sizes are used to include a few words surrounding the word, which is currently classified. For a window size of three this means that one word prior and one word subsequent to the respective word are included, for a window size of five, two in both directions. Additionally considering the surrounding words shows a large improvement of the performance.

Furthermore, the results show that the use of a bidirectional recurrent neural network provides a significant improvement compared to a unidirectional one. Combining this knowledge with what was shown from softmax-3 and softmax-5, it is clear that words prior and subsequent to the considered word are important for its classification.

GRUs require fewer weights to be trained, which in some cases improves the results. From the given results this expectation cannot be confirmed. For a single layer GRU the results are slightly below those of a single layer LSTM, for a two-layered GRU slightly better. There is no consistent improvement recoginzable.

	unidirectional		bidirectional	
	LSTM	LSTM-2	LSTM	LSTM-2
end2end	74.46	75.05	88.97	89.22
pretrain	73.84	74.88	89.04	**89.75**

Table 2: *F-scores of different configurations with end-to-end trained and pretrained character embeddings. No text preprocessing is applied to the sentences.*

For the networks utilizing character embedding, the hidden size of the LSTMs was set to 400.

The results seen in Table 2 show that adding multiple layers provides little improvement. As with word embeddings, classifying sentences bidirectionally gives a considerable advantage. For unidirectional LSTMs the results from word embedding are slightly better. Using bidirectional LSTMs, character embeddings show better results.

The character embeddings trained as part of a language model show nearly no improvement. This indicates that the model trained end-to-end can calculate a good representation fitting the classification purpose for each character.

	end-to-end		pretrained	
	precision	recall	precision	recall
no pre-processing	89.69	88.75	**90.30**	**89.25**
lowercase	88.70	88.58	88.95	88.55
lower case and no punctuation	87.44	86.82	86.37	84.40

Table 3: *Results for different grades of preprocessing. For all tests, the same number of layers and hidden size is used.*

Table 3 shows the results of different grades of preprocessing applied to the data. Both LSTMs have two layers with a hidden size of 400. Batch size and learning rate are identical.

To objectively compare the performance, all scores are calculated over the word list. Thus, the enlarged number of characters (due to not removing punctuation) does not influence the label balance of the text. The scores indicate that removing punctuation does not improve performance. Labeling sentences without any kind of preprocessing applied to them yields the best results. We could not determine whether this outcome results from the slightly enlarged amount of training data or of actual content-related information, the punctuation provides.

	precision	recall
baseline ACL	22.27	19.35
character	**44.36**	**52.31**
word	41.79	50.38

Table 4: *Comparison of performance of word and character embedding on an out-of-domain corpus. For both networks, the previously shown best configuration is selected.*

The results in Table 4 show a significant drop in performance for tests run on an out-of-domain corpus. In accordance with previous results, character embedding yields better results than word embedding. These results suggest, that the LSTM adapts to certain properties in the sentence or word structure, which differ for biomedical and computational linguistic diction.

6. Conclusion

Comparing the performance of our approach to existing systems is difficult for two reasons. First of all, we perform sequence-to-sequence classification and do not output a list of terms, which will be compared to a predetermined list for the f-score calculation. Secondly, there is for the moment no possibility to identify *nested terms* (terms, which consist of several words, of which subsets are defined as terms as well), as a word can only be assigned one label at a time. Nevertheless, the scores achieved with this model are clearly state-of-the-art and impressive considering the simplicity of the approach.

Contrary to existing systems, our approach does not rely on the quality of the POS tagger and does not require any linguistic knowledge. This makes it easy to apply it to other languages even without knowledge of their syntactical properties. Our test results show a stable and good performance on in domain sentences, if the training corpus is large enough. This shows that recurrent neural networks can successfully be applied to term extraction.

Finally when planning on applying the trained models to other domains than that of the training data, it is beneficial to use a character embedding instead of a word embedding.

7. References

[1] M. Müller, T. S. Nguyen, J. Niehues, E. Cho, B. Krüger, T.-L. Ha, K. Kilgour, M. Sperber, M. Mediani, S. Stüker, and A. Waibel, "Lecture translator—speech translation framework for simultaneous lecture translation," in *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, San Diego, USA, June 2016.

[2] L. Ratinov and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning CoNLL '09)*, Boulder, CO, USA, June 2009.

[3] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *The ICML '06 Proceedings of the 23rd international conference on Machine learning*, Pittsburgh, PA, USA, June 2006.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[5] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *CoRR*, vol. abs/1601.00770, 2016. [Online]. Available: http://arxiv.org/abs/1601.00770

[6] R. Socher, C. D. Manning, and A. Y. Ng, "Learning continuous phrase representations and syntactic parsing with recursive neural networks," in *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, vol. 2010, 2010, pp. 1–9.

[7] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu, "Recurrent neural networks for language understanding." in *Interspeech*, 2013, pp. 2524–2528.

[8] Y. Yuan, J. Gao, and Y. Zhang, "Supervised learning for robust term extraction," in *International Conference on Asian Language Processing (IALP 2017)*. IEEE, October 2017, © 2017 IEEE. [Online]. Available: http://eprints.whiterose.ac.uk/123210/

[9] M. da Silva Conrado, T. A. S. Pardo, and S. O. Rezende, "A machine learning approach to automatic term extraction using a rich feature set," in *Proceedings of the NAACL HLT 2013 Student Research Workshop*, Atlanta, Georgia, June 2013, pp. 16–23. [Online]. Available: http://www.aclweb.org/anthology/N13-2003

[10] R. Wang, W. Liu, and C. McDonald, "Featureless domain-specific term extraction with minimal labelled data," in *Proceedings of the Australasian Language Technology Association Workshop 2016*, Melbourne, Australia, December 2016, pp. 103–112. [Online]. Available: http://www.aclweb.org/anthology/U16-1011

[11] J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii, "Genia corpusa semantically annotated corpus for bio-textmining," *Bioinformatics (Oxford, England)*, vol. 19 Suppl 1, pp. i180–2, 02 2003.

[12] B. QasemiZadeh and A.-K. Schumann, "The acl rd-tec 2.0: A language resource for evaluating term extraction and entity recognition methods," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, N. C. C. Chair), K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis, Eds. Paris, France: European Language Resources Association (ELRA), may 2016.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[14] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[15] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[16] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *CoRR*, vol. abs/1402.3722, 2014. [Online]. Available: http://arxiv.org/abs/1402.3722

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162