GBD-NER at PARSEME Shared Task 2018: Multiword Expression Detection Using Bidirectional Long-Short-Term Memory Networks and Graph-Based Decoding

Tiberiu Boros
Adobe Romania
boros@adobe.com

Ruxandra Burtica
Adobe Romania
burtica@adobe.com

Abstract

This paper addresses the issue of multiword expression (MWE) detection by employing a new decoding strategy inspired after graph-based parsing. We show that this architecture achieves state-of-the-art results with minimum feature-engineering, just by relying on lexicalized and morphological attributes. We validate our approach in a multilingual setting, using standard MWE corpora supplied in the PARSEME Shared Task.

1 Introduction

Multiword expression (MWE) detection is a challenging Natural Language Processing (NLP) task that consists in finding isolated tokens or sequences of tokens that form high-level structures (i.e. multiword expressions). Naively, this can be regarded as a sequence labeling task, which in fact influenced legacy methods for MWE detection to use this strategy. In fact, this is not far fetched, since this approach has yielded highly accurate results so far. However, we have to point a couple of task-specific details:

- **Sparsity**: Inside an utterance there are only a couple of tokens that must be labeled as named entities or multiword expressions. This actually yields data sparsity and could bias the model towards not identifying any of the tokens as part of a multiword expression;
- **Overlapping**: MWE corpora contains many examples where high-level entities share tokens among each other. There are several ways in which this issue can be mitigated, for instance training several different classifiers for each type of label;
- **Long spans**: It is often the case that MWE tokens are distantly distributed across a single utterance. Classical classifiers and even modern LSTM-based approaches are unable to detect such long range dependencies (though the later mentioned approach should), mainly because these cases are rare inside the training data and there is insufficient statistical evidence to support them.

In what follows, we propose a new methodology for learning dependencies within MWE tokens which mitigates the impact of the previously mentioned issues. Particularly, we use Bidirectional Long-Short-Term Memory (BDLSTM) (Graves and others, 2012) networks, but we do not employ a naive tagging approach as it is done in similar related work (see section 2). Instead, we use a strategy inspired by the graph-based parser of Kiperwasser and Goldberg (2016), with the following major differences: (a) the network is trained to produce fully connected subgraphs (not parsing trees); (b) we use a different graph-decoding strategy that is tailored for MWEs; (c) our feature-set is composed of morphological and lexicalized features and we apply a two-tier dropout methodology (explained in Section 3) in order to increase the generalization capability of our models.

The proposed methodology was evaluated during the PARSEME Shared Task on Verbal Multiword Expression Detection (Ramisch et al., 2018) and the full evaluation details for all languages are available in our blog[1].

[1] http://opensource.adobe.com/NLP-Cube/blog/posts/1-gbd/results.html

Proceedings of the Joint Workshop on
Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG-2018), pages 254–260
Santa Fe, New Mexico, USA, August 25-26, 2018.

2 Related work

There is a strong resemblance between NER and MWE detection, and we feel that the related work section should cover both tasks, whenever the underlying methodology and results can be shared between the two. Aside from classical approaches to NER and MWE detection that employ Conditional Random Fields (CRFs), Support Vector Machines (SVMs), Hidden Markov Models (HMMs), Perceptrons etc., there are several neural inspired methods, on which we will currently focus, given that we also use a deep learning approach.

Chiu and Nichols (2015) introduce a NER system that uses stacked BDLSTM layers over word-encodings that are obtained by concatenating word-embeddings with manually engineered word-lever features. The character level embeddings that are computed using convolutional neural networks (CNNs). Their system is trained to distinguish between "PERSON", "ORGANIZATION", "LOCATION" and "MISC" entity types and they employ a classical BIOES tagging strategy (Begining, Inside, Outside, Ending and Single)

Shao et al. (2016) compare (a) a "window-based" feed-forward network, (b) a standard BDLSTM network and (c) a "window-based" BDLSTM network. They use word embeddings combined with word-level features as input for their networks and they also rely on IOB labeling strategy. As expected, the feed-forward neural network is easier to train but it is outperformed in accuracy by the standard BDLSTM model; (b) the "window-based" BDLSTM is robust whenever fewer features are employed, but it is outperformed by other BDLSTM models, such as the one presented in Chiu and Nichols (2015).

(Lample et al., 2016) present two network architectures for NER: the CRF-LSTM architecture and the transition-based LSTM strategy, which, just like our strategy, is also inspired after parsing. The input features are based on word-embeddings and character-level embeddings and the results show that, in most cases, the two network architectures obtain similar results, with the CRF-LSTM model always a bit more accurate.

The transition-based strategy is also exploited by Al Saied et al. (2017), but this time for multiword expression detection using SVMs. They evaluate their system during the PARSEME Shared Task (Savary et al., 2017) and, according to the official results, this methodology achieves state-of-the-art results.

3 Proposed methodology

3.1 Graph-based encoding with neural networks

As opposed to (a) local classifiers which work by statically estimating the output label (t_i) probability from localized features (f_i): $\boldsymbol{P(t_i|f_i)}$ and from (b) graph or sequence classical classifiers which work by using a localized set of features (f_i) and by performing decoding using limited-context dependencies : $\boldsymbol{argmax(P(t_i|\{t_j \in \{dependencies\}\}, f_i))}$, Long-Short-Term Memory (LSTM) networks are way better at computing each word's label (t_i) based on features extracted from the entire sentence: $\boldsymbol{P(t_i|f_{i=\overline{1,n}})}$.

This could in theory mean that, if we are not dealing with overlapping expression, one could employ LSTM models for sequence labeling, regardless of the span between the tokens that are part of the same high-level entity. The capacity of the model to detect long-range dependencies is bound to the number of stacked LSTM layers and the number of LSTM cells inside each layer. However, adding more layers and increasing the number of cells is also a recipe for over-fitting the training data, and there is a limit of the extent to which mechanisms to prevent over-fitting mechanisms are able to help.

As such, we propose a graph-based encoding/decoding strategy that is able to cope with overlapping high-level entities and large spans between tokens, by **forcing the model to focus** on the dependencies between tokens that belong to the same expression.

Figure 1 contains a rolled-out version of our network architecture. We start off with a sequence of words as input, and convert it into a sequence of fixed-sized vectors (details will follow). After this, we use two stacked Bidirectional LSTM (BDLSTM) layers (Graves and Schmidhuber, 2009), obtaining a list of vector embeddings for each word inside the original sequence. Next, we take each vector from the internal representation and we split it using three parallel $tanh$ layers into 3 separate "projections".

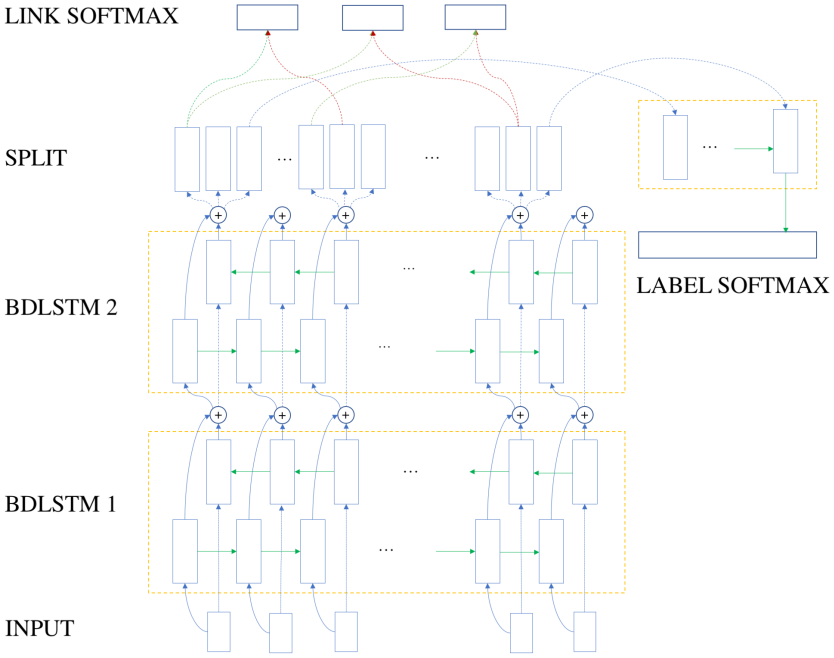

Figure 1: The rolled-out sparse tagging network

Until now, this architecture is similar to the one described in Kiperwasser and Goldberg (2016), except that we use 3 projections instead of one. As such, for each word w_i, we obtain three vectors which we will refer to as v_i^0, v_i^1 and v_i^2. Finally, for every pair of words w_i and w_j with $i \neq j$ we use single unit sigmoid activation, outputting the probability of the two words being part of the same high-level expression or not. The input for the sigmoid unit is obtained by concatenating the first vector from the split operation for word w_i (i.e. v_i^0) with the second vector of the split operation for word w_j (i.e. v_j^1). We store the resulting values in an "adjacency matrix" a (Equation 3).

$$a_{i,j} = \sigma(W_s \cdot (v_i^0 \oplus v_j^1) + b_s) \tag{1}$$

During training, we establish if words w_i and w_j are part of the same high-level entity and, for every pair (i, j) we infer as loss $-\log a_{i,j}$ if the words are constituents of the same expression or $-log(1.0 -$

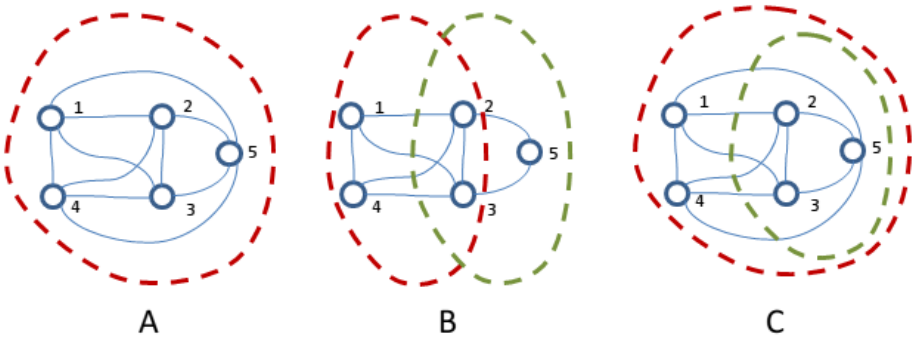

Figure 2: Hard decoding cases in entity extraction

$a_{i,j}$) otherwise.

Because a is an adjacency matrix, we expect it to be symmetrical on the main diagonal and reduce the number of computational steps by adding the condition $j > i$, when we compute the loss. Also, when training, we perform back-propagation and parameter update only after we compute and sum all individual losses.

The feature-set used in our approach is composed of the concatenation of two vectors, one for lexicalized features and the other for morphological features. The lexicalized feature vector is obtained by adding two specialized embeddings, one for the surface-form and one for the word's lemma[2] . Morphological features are obtained by adding three vectors representing embeddings for the three morphological tiers provided as input (see McDonald et al. (2013) for more details): universal part-of-speech (UPOS), language-specific part-of-speech (XPOS) and attributes (ATTRS).

$$a_{i,j} = \frac{\exp(y^0_{i,j}))}{\sum_{k=0}^{1} \exp(y^k_{i,j}))}, y^k_{i,j} \in Y_{i,j} \tag{2}$$

$$Y_{i,j} = w \cdot (v^0_i \oplus v^1_j) + b \tag{3}$$

During training, we use a two-tier dropout methodology: (a) independently drop embedding vectors for word, lemma, UPOS, XPOS and ATTRS, scaling the other vectors to cope with missing values and (b) independently drop one of the lexical or morphological vectors, also using scaling. Given word w_i with its corresponding lemma (l_i), UPOS (u_i), XPOS (x_i) and ATTRS (t_i), we define the input vector for the BDLSTM (Equation 4).

$$f(w_i, l_i, u_i, x_i, t_i) = (((E_{w_i} + E_{l_i}) \cdot s_{1_l}) \oplus (E_{u_i} + E_{x_i} + E_{t_i}) \cdot s_{1_m})) \cdot s_2 \tag{4}$$

where E is used to generically represent an embeddings lookup table and s_{1_l}, s_{1_m} and s_2 are scalars used as scaling factors in the two-tier dropout technique.

Notes:

1. We observed that **character-level features are better suited for standard NER tasks and not for MWE detection**. Likely, this is related to the fact that character-level features such as casing, letter n-grams or appurtenance to the symbol and/or number class are useful in identifying proper names, dates, location etc., but they are not informative for semantics (at least not for most languages);

2. External word embeddings would have rendered our system into the "open" track, which we did not consider in our evaluation, but future development plans include this option.

3.2 Expression decoding and label selection

The first part of the algorithm trains the network to output half of an adjacency matrix inside an undirected graph. Whenever two or more words form a high-level entity we expect them to project into a complete subgraph, because the model is taught that all words inside that entity are adjacent in the graph structure. In order to extract the entities, we use an algorithm to compute all complete subgraphs from the adjacency matrix.

In our experiments we used backtracking to perform the extraction of complete subgraphs. The algorithm starts by building a mini-graph with two nodes that are adjacent. Then, it recursively adds new nodes that satisfy the property of being adjacent with all the nodes already inside the subgraph. Whenever no new nodes can be added to a subgraph, the subgraph is added to the solution list and the recursion is stopped. Partial solutions are ignored and, at the end of the process, we remove duplicates inside the solution list. This naive backtracking implementation works for us mainly because high-level entities only contain a few tokens and the $O(2^n)$ complexity is actually manageable.

To select the correct output label for every sequence of words inside an high-level entity. Let E^k be a detected entity and I^k_l the ordered set of word indices belonging to this entity. In order to predict the

[2]We only compute embeddings for frequently-occurring words and lemmas – we used ($f_{tok} \geq 2$)

L	MWE			#	Tok			#	L	MWE			#	Tok			#
	P	R	F		P	R	F			P	R	F		P	R	F	
BG	79.41	56.42	65.97	1	82.10	55.72	66.39	1	**HR**	51.52	37.35	43.31	4	64.41	42.22	50.69	2
DE	44.13	33.94	38.37	4	64.24	43.49	51.87	1	**HU**	85.47	76.55	80.76	5	90.54	77.09	83.27	6
EL	64.74	49.10	55.85	1	74.02	52.22	61.24	1	**IT**	49.52	41.73	45.30	2	62.46	45.59	52.71	2
EN	10.52	57.68	17.79	8	11.86	60.81	19.85	8	**LT**	07.11	44.80	12.27	4	08.63	49.60	14.70	4
ES	22.22	36.80	27.71	5	30.38	46.17	36.65	7	**PL**	75.06	62.52	68.22	1	78.30	62.18	69.32	1
EU	82.07	71.40	76.36	1	83.12	71.79	77.14	1	**PT**	66.59	55.88	60.77	3	69.78	57.58	63.09	2
FA	81.96	75.25	78.46	1	88.60	77.54	82.70	1	**RO**	88.35	83.70	85.96	1	89.52	83.52	86.41	1
FR	72.76	42.37	53.55	2	81.85	44.66	57.79	2	**SL**	61.07	52.40	56.40	2	68.57	55.08	61.09	2
HE	74.19	09.16	16.31	7	75.76	08.66	15.54	10	**TR**	68.07	54.35	60.44	1	70.06	54.64	61.40	1
HI	61.90	72.80	66.91	5	65.18	71.38	68.14	6	**AVG**	**60.36**	**53.38**	**56.65**	**1**	**66.28**	**55.78**	**60.57**	**1**

Table 1: Results obtained on the PARSEME VMWE identification Shared Task

label for this entity we build the corresponding list of internal embeddings vectors from the third split $v^3_{I^k_l}$. We fed the list in natural order into a single layer LSTM trained in a many-to-one fashion and use output of the final state as input for a Softmax layer trained to output the correct label for the sequence.

For clarity, we have included a couple of decoding examples (Figure 2) which we will further discuss. Note that the indexes used for the nodes have nothing to do with the actual indexes of the words that form the graph. We only marked them with numbers from 1 to 5, to be able to easily reference them.

The simplest **Case (A)** shows 5 interconnected graph nodes, which form a single high-level entity. Most of training data contains only this type of example, where we have to mark all the discovered tokens with the same label.

Case (B) is more complex, as we have two overlapping high-level entities which share nodes 2 and 3. As such, the decoding algorithm can still easily spot the two distinct complete subgraphs, [1, 2, 3, 4] and [2, 3, 5] and mark them as two distinct entities (even if they are, or not, of the same type).

Case (C) is by far the hardest. Here we have two distinct entities, with one entity [1, 2, 3, 4, 5] completely including the second entity [2, 3, 5]. We have not encountered this case in any of the training data but we feel obliged to discuss it as well. Obviously, given that the input data is correct, the decoding algorithm will always find a single entity, because all the nodes are adjacent and there is no reason to perform a split into two subgraphs. One solution to this issue is to check if any partial subgraph yields a different label from the full subgraph. This solution is only valid for entities which belong to different classes, but we feel that the two expressions sharing the same label would be highly unlikely to overlap. However, this solution may not always hold, and in the absence of real data to support Case (C).

Finally, we need to select the correct output label for every sequence of words inside an high-level entity. Let E^k be an detected entity and I^k_l the ordered set of word indices belonging to this entity. In order to predict the label for this entity we build the corresponding list of internal embeddings vectors from the third split $v^3_{I^k_l}$. We keep the list in natural order, meaning the order in which the words appear inside the utterance. This list is sequentially feed into a single layer LSTM trained in a many-to-one fashion. This means that we only use the final state of the cells as input for a Softmax layer. The Softmax layer is designed to output the probability distribution over all possible labels. During training we only compute loss for detected entities that are inside the gold standard and we do this even if the algorithm did not detect them correctly at that state. We did not use a bidirectional layer or attention mainly because the input list is usually short and a larger model could easily over-fit the training data.

4 Experimental validation

We validated our approach using the multilingual corpora provided by the PARSEME corpus. The data covers 19 languages and comes in CUPT format. The CUPT format is similar to CONLLU, but adds a new annotation layer for VMWE expressions. For each entry inside a sentence contains the word, it's lemma, UPOS, XPOS and ATTRs, which are automatically labeled using UDPipe (Straka et al., 2016). Table 1 summarizes the results. The "AVG" entry refers to the macro-average computed over all languages. This means that the precision and recall are computed for all languages and then F-score is recalculated using the standard equation. For every language, as well as for the overall result, we include

the absolute ranking ('#') of our system based on the results from the official runs. During the official runs. our software was affected by a bug that caused it to ignore any lexical information in the CUPT files. The results shown in the table refer to the corrected version of our system and we do provide full access to the source-code[3] for anyone interested.

As shown, this methodology achieves overall higher results, but it does not always produce state-of-the-art results for certain languages. The lower scores for languages such as EN, HE and LT can be explained by the fact that those languages were only provided with small training sets and no development data. It is somewhat expected that deep learning methods require a critical mass of input data in order to provide good generalization. Also, the proportion of MWEs that were previously unseen in the training data for EN is 92%, 37% for HE and 94% for LT. Without using external data such as word embeddings our intuition is that it is hard to learn patterns that have to do with "word semantics" (the case of MWEs) with so few training examples.

Also, we have to mention that some training sets contained "single token" multiword expressions, which, as the name suggests, means that a multiword expression contained only one token. Our algorithm was originally designed to work by detecting dependencies between each pair of tokens inside a MWE, which is not the case here because the MWEs only contain one word. In order to do this, we adapted our algorithm by introducing a virtual "ROOT" in every sentence. During training we modified the system to output links between all MWE tokens and the ROOT. This solved the issue with single-token MWEs, but biased the model for multiword tokens. For instance, the single token MWE proportion for HU is 74%, and our f-score on this type of expressions is 84.17% using this tweak. However, if we compare the new f-score for multiword MWEs we get an absolute drop of 7%.

5 Conclusions and future work

We have described a new methodology of extracting MWEs using Bidirectional LSTMS and a graph-based representation. Our code is freely available on GitHub and we plan to later integrated it in our NLP processing framework (NLP-Cube)[4].

In the near future we plan to explore and answer a couple of questions, such as:

1. How externally computed word embeddings influence the performance of this methodology on MWE detection;

2. Will this graph-based decoding strategy have a positive impact on standard or domain-specific NER;

3. What is the source for lower f-scores on languages such as Hungarian, Deutsch and Hindi. that, at first glance, have enough training data to support our approach;

4. How will this method work for other NLP tasks which involve sparse and long-range dependencies between words, one good example being co-reference resolution.

Regarding MWE identification, parsing information could help reduce the ambiguities in classification, given that to our knowledge, in the Universal Dependencies Corpus words that form MWEs are marked using the "fixed" subordination dependency, at least for some languages. Whether or not parsing accuracy is sufficient enough to support MWE identification is a different question. Also, given that our system is inspired from parsing, our intuition is that parsing will not enhance results.

On a related note, NLP-Cube has end-2-end raw text processing to UD format processing capabilities. This means that it can be used for MWE detection without requiring external CUPT files. Anyone interested can check the end-2-end raw text processing capacity of NLP-Cube on this year's shared task on universal dependencies parsing[5].

[3] https://github.com/adobe/NLP-Cube/tree/dev.gbd-ner
[4] https://github.com/adobe/NLP-Cube
[5] http://universaldependencies.org/conll18/

References

Hazem Al Saied, Matthieu Constant, and Marie Candito. 2017. The atilf-llf system for parseme shared task: a transition-based verbal multiword expression tagger. In *Proceedings of the 13th Workshop on Multiword Expressions (MWE 2017)*, pages 127–132.

Jason PC Chiu and Eric Nichols. 2015. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*.

Alex Graves et al. 2012. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer.

Alex Graves and Jürgen Schmidhuber. 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *arXiv preprint arXiv:1603.04351*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 92–97.

Carlos Ramisch, Silvio Ricardo Cordeiro, Agata Savary, Veronika Vincze, Verginica Barbu Mititelu, Archna Bhatia, Maja Buljan, Marie Candito, Polona Gantar, Voula Giouli, Tunga Güngör, Abdelati Hawwari, Uxoa Iñurrieta, Jolanta Kovalevskaitė, Simon Krek, Timm Lichte, Chaya Liebeskind, Johanna Monti, Carla Parra Escartín, Behrang QasemiZadeh, Renata Ramisch, Nathan Schneider, Ivelina Stoyanova, Ashwini Vaidya, and Abigail Walsh. 2018. Edition 1.1 of the parseme shared task on automatic identification of verbal multiword expressions. In *Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG-2018)*, Santa Fe, USA, August. Association for Computational Linguistics.

Agata Savary, Carlos Ramisch, Silvio Cordeiro, Federico Sangati, Veronika Vincze, Behrang Qasemizadeh, Marie Candito, Fabienne Cap, Voula Giouli, Ivelina Stoyanova, et al. 2017. The parseme shared task on automatic identification of verbal multiword expressions. In *Proceedings of the 13th Workshop on Multiword Expressions (MWE 2017)*, pages 31–47.

Yan Shao, Christian Hardmeier, and Joakim Nivre. 2016. Multilingual named entity recognition using hybrid neural networks. In *The Sixth Swedish Language Technology Conference (SLTC)*.

Milan Straka, Jan Hajic, and Jana Straková. 2016. Udpipe: Trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing. In *Language Resources and Evaluation Conference*.