A BOW-Based Sentence Embedding Method for Chinese Event Identification

Lizheng Gao\textsuperscript{1*}, Gang Zhou\textsuperscript{1}, and Junyong Luo\textsuperscript{1}

\textsuperscript{1}State key of Laboratory of Mathematical Engineering And Advanced Computing, Zhengzhou, China

\textsuperscript{*}Corresponding author's e-mail: gao1440429064@foxmail.com

Abstract. Event extraction is an important branch of information extraction. It involves two challenging issues: event identification and argument identification. Most approaches of event identification are trigger-based, which suffer a lot from sample imbalance, word ambiguity, low scalability, etc. To solve these problems, we attempt to explore sentence-based event identification, whose main issue is sentence representation. With the success of word embedding, it has attracted much attention to generate semantic embeddings of sentences. In this paper, we propose a simple BOW-based sentence embedding method to represent event sentences for Chinese event identification. We compute embeddings on the dependency tree and map different relations to simple arithmetic operations. We evaluate our method with a dataset of Chinese event identification and compare the result with other BOW-based methods. The results show that our approach significantly outperforms other BOW-based methods.

1. Introduction

Event identification (EI) is the basis of event extraction aiming to identify event instances of specified types in text. Automatic Content Extraction (ACE) provides a comprehensive definition of almost all aspects related to event extraction and also provides an annotated corpus. As a result, most current approaches of event identification are based on ACE. ACE defines the event trigger as a main word that most clearly expresses an event occurrence and considers event identification as trigger identification. The assumption, however, has a few limitations. First, trigger itself contains limited information about the whole event, which may lead to low precision of event identification and trigger ambiguity. Second, treating every word in a sentence as a candidate trigger brings a lot of negative samples which results in sample imbalance. Third, since the amount of triggers in the train data is limited, it's a problem to expand the vocabulary of triggers. As a result, more and more researchers turn their attention to sentence-based event identification, which suppose the type of an event is determined by the whole event instance.

As event sentences are composed of triggers and arguments, it is necessary to represent them in a unified model before event identification. The success of word embedding motivates the studies of sentence embedding which represent sentences as continuous vectors. Some excellent approaches have been proposed and applied successfully in NLP tasks (e.g., textual similarity, sentiment analysis, etc). We believe sentence embedding can also be a good solution to event representation. The methods of sentence embedding range from simple BOW-based models to sophisticated architectures. Although BOW-based methods are usually simpler, some of them even outperform more complicated ones. What's more, as events in the real word are massive and diverse, the method of event representation should be scalable and easily obtained. This paper presents a simple BOW-based sentence embedding...
method which computes embeddings on the dependency trees of sentences and maps different relations to different vector operations. Our method pay more attention to grammatical information and words relations which are ignored by other BOW-based models. We test our method on the task of Chinese event identification. The experimental results demonstrate that our approach significantly outperforms other BOW-based methods in Chinese event identification.

2. Related Work

2.1 Event Identification
ACE is an information extraction program that has very important influence on the development of event identification. Most studies of event identification follow the ACE standard. Early researches on event identification were based on event patterns formulated by hand. Although performing well in certain situations, it takes a large amount of human efforts to construct patterns and suffers a lot from poor scalability. Researchers (e.g., [1], [2], [3], [4], [5]) then made use of machine learning approaches to build models automatically with complicated feature engineering. These methods can somewhat reduce the labor costs but still need to analyze features manually. In recent years, with the widely application of word embedding in NLP tasks, researchers (e.g., [6], [7], [8], [9], [10], [11]) started to represent event features with word embeddings and build models with sophisticated neural networks. Their methods showed the state-of-the-art performance.

However, approaches discussed above are mostly trigger-based which exist intrinsic problems hard to overcome. Some researchers, therefore, considered the sentence-based model as a substitute. However, it developed slowly and suffered from the difficulty of sentence representation until sentence embedding was applied.

2.2 Word Embedding
Word embedding represents words as continuous vectors and maps the semantic similarity of words to the proximity of vectors. It has made the most exciting breakthrough of NLP in recent years. There are a few famous word embedding models. Bengio [12] proposed to learn a distributed representation for words with neural networks, which was the first attempt to neural-based word embedding. Mikolov [13] proposed the famous word2vec which included two effective models named CBOW and Skip-gram for computing word embeddings from very large data sets. In another paper [14], they presented several extensions that improved both the quality of the vectors and the training speed. Their studies have great contribution to the widely research and application of word embedding. Pennington [15] combined local context window and global matrix factorization to simultaneously capture the information of local context and the relationship of words. Their method is said to outperform word2vec consistently. Motivating by the success of English word embedding, many researchers tried to explore the approach of Chinese word embedding. In this paper, we leverage the word embeddings provided by Li [16] to compute Chinese sentence embedding.

2.3 BOW-based Sentence Embedding
BOW-based sentence embedding represents a sentence as a bag of words and computes embeddings of sentences with simple arithmetic operation on word vectors. Mikolov [13] found that the simple average of embeddings can be surprisingly good at capturing sentence-level properties. Kusner [17] and Brokos [18] assigned the TFIDF weights to words in the BOW. Arora [19] applied the so-called “smooth inverse frequency” as weights and removed the projections of the average embeddings on their first principal component. Their method outper form some sophisticated models in many NLP tasks. Rücklé [20] suggested to replace average word embeddings with power mean word embeddings and took the concatenation of different types of power mean word embeddings as the sentence embedding.

BOW-based sentence embedding usually ignores the grammatical information and words relations. In this paper, we propose a simple but effective method that overcome those limitations by computing
the embedding of a sentence on its dependency tree and mapping dependencies to simple vector operations. Our method is proved to be outperformed than other BOW-based methods in Chinese event identification.

3. Our Approach
An event is composed of a trigger and several arguments. A trigger is a word which mostly describes the occurrence of an event. Arguments consist of time, location, participants, etc. For example, in the sentence “An American fighter attacked a terrorist training camp in Afghanistan.”, the trigger is “attacked”, which describes an “Attack” event. The arguments are “an American fighter”, “a terrorist training camp” and “Afghanistan”, which are participants and location of the attack. Although containing the most information of event type, many triggers may be semantically ambiguous as the polysemy of human languages, for example, the trigger “beat” in the sentence “Trump beat Hillary”. Generally, common words are more likely to be ambiguous than uncommon ones. So the ambiguity of triggers are inevitable. As a result, it’s better to determine the type of an event by the whole sentence but not only a trigger.

To introduce our approach, we firstly introduce the most common structure of event sentences: <subject, predicate, object> (<s, p, o> for short), whose corresponding dependency tree is shown in Figure 1a. In this structure, subject and object are arguments and predicate is a trigger. The structure can be rewritten as <argument, trigger, argument> (<arg., tri, argo.> for short), whose dependency tree is shown as Figure 1b. We thus break down the task of computing the embedding of <arg., tri, argo.> into three sub-tasks:

- Argument embedding: computing the embedding of an argument phrase.
- Trigger embedding: computing the embedding of a trigger phrase.
- Argument-trigger embedding: computing the embedding of <arg., tri, argo.>.

3.1 Argument Embedding
An argument in an event sentence is always a phrase which consist of a core word c and its modifier m. Define the structure of the argument as <c, m>. Core word c contains main semantic of the argument and is confined by its modifier m. Suppose \( V(x) \) as the embedding of x (\( x \) can be word, phrase and even sentence). According to the finding of Mikolov [13] that vector addition records the information of all constituent vectors, we can compute the embedding of <c, m> as \( V(c)+V(m) \). As shown in Figure 2a, \( V_0 \) is equal to \( V(c)+V(m) \) based on the rule of vector addition.

However, the semantic of an argument is determined jointly but unequally by the core word and its modifier. Since the direction of \( V(x) \) represents the semantic of x, we wish \( V(<c, m>) \) to be closer to \( V(c) \) rather than \( V(m) \). So we shorten \( V(m) \) to \( \omega V(m) \) (\( 0<\omega<1 \)) and compute \( V(<c, m>)=V(c)+\omega V(m) \) (\( 0<\omega<1 \)), as shown in Figure 2b. It’s clearly that \( V' \) is much closer to \( V(c) \) than \( V_0 \).

![Figure 1](image1.png)

![Figure 2](image2.png)
An argument may have a few sibling modifiers, whose structure can be defined as \(<c, m_1, m_2, ..., m_n>\) (\(m_i\), \(1\leq i\leq n\), is a modifier of the core word \(c\)). Since these modifiers are siblings, we assign the same weights to them and compute the embedding as \(V(c) + \omega \times V(m_1) + \omega \times V(m_2) + ... + \omega \times V(m_n)\) \((0<\omega<1)\). For instance, "a Chinese student", whose dependency tree is shown in Figure 3a, has two sibling modifiers: "a" and "Chinese", and its embedding is computed as \(V("student") + \omega \times V("a") + \omega \times V("Chinese")\). We set the weights of all modifiers as \(\omega\) and assign the same value to them (the same below). The experimental result proves that this setting doesn't affect the performance, but improves the generalization of our method.

There may be nested modifiers in an argument, namely the modifier of an argument may have its own modifiers, such as \(<c_1, <c_2, m_2>>\), in which \(<c_2, m_2>\) is the modifier of \(c_1\) and \(m_2\) is the modifier of \(c_2\). The further a modifier is away from the core word, the smaller its weight should be. To calculate \(V(<c_1, <c_2, m_2>>)\), we firstly compute the value of \(V(<c_2, m_2>)\) as \(V(c_2) + \omega \times V(m_2)\) and then compute \(V(<c_1, <c_2, m_2>) = V(c_1) + \omega \times V(<c_2, m_2>)\). For example, "Chinese high school student" is an argument in which "Chinese high school" is the modifier of "student" and "Chinese" is the modifier of "high school". The embedding of this phrase is \(V("student") + \omega \times V("high school") + \omega \times V("Chinese")\). Arguments may be formed by a few child arguments and have neither core words nor modifiers. We calculate their embedding as the sum of the embeddings of their child arguments, for example \(V("Tom and Jerry") = V("Tom") + V("Jerry")\).

### 3.2 Trigger Embedding

The "trigger" in Figure 1b is always a phrase which contains not only a trigger but also its adverbial modifier. We define the structure of the trigger phrase as \(<t, ma>>\) and compute the embedding as \(V(t) + \omega \times V(m_a)\). If the trigger phrase is composed of a few coordinate triggers (defined as \(t_1, t_2, ..., t_n\)), the embedding is the sum of their vectors \(V(t_1) + V(t_2) + ... + V(t_n)\). Adverbial modifiers are sometimes prepositional phrases which consist of prepositions and objects. Since prepositions contain very little information, we neglect them when compute the embedding of prepositional phrases. Time and place in an event are always parts of adverbial modifiers, we compute their embeddings just like other arguments.
3.3 Argument-Trigger Embedding
The type of an event sentence is decided by its trigger and arguments. Although trigger is the main word that most clearly expresses an event occurrence, arguments always take a much larger proportion in the sentence, which may lead to information imbalance and semantic preference. In this paper, we divide \(<\text{arg}_s, \text{tri}, \text{arg}_o>\) into sub-structures: \(<\text{tri}, \text{arg}_s>\) and \(<\text{tri}, \text{arg}_o>\), and calculate the embedding of \(<\text{arg}_s, \text{tri}, \text{arg}_o>\) as \(V(<\text{tri}, \text{arg}_s>)+V(<\text{tri}, \text{arg}_o>)\). If there are no objects in an event sentence (e.g. "She cried"), the embedding is exactly \(V(<\text{tri}, \text{arg}_s>)\). If there are more than one object in an event sentence, the sentence embedding is the sum of all \(V(<\text{tri}, \text{arg}_s>)\). We take the calculations of \(V(<\text{tri}, \text{arg}_s>)\) and \(V(<\text{tri}, \text{arg}_o>)\) as the same and compute \(V(<\text{tri}, \text{arg}_o>)=V(\text{tri})+V(\text{arg})\). In this way, the weight of the trigger will change with the number of arguments, which reduce the lost of trigger information to a certain degree when the arguments information is rather richer in the event sentence.

3.4 Sentence to Embedding
As described above, we classify the dependencies between sentence constituents into four types: modification, coordination, argument-trigger relation and object-preposition relation (mod, coo, a-t, o-p for short). We assign different weights and calculations to them. By default, the order of their calculations is coordination, modification, object-preposition relation, and argument-trigger relation.

Given an event sentence, the process of computing the embedding can be described briefly as follows: First, segment the sentence into words and map words to their embeddings. Second, construct the dependency tree of the sentence whose nodes are mapped to word vectors. Third, merge nodes from leaves to parents with their corresponding calculations until there is only a root node on the tree.

To illustrate the process, we give a simple instance here. Suppose the sentence "The Chinese high school student drew a beautiful picture at home", whose dependency tree is shown as Figure 4a. The process of calculating the embedding is shown as Figure 4a to Figure 4d.

Figure 4. The process of calculating the embedding of the sentence "The Chinese high school student drew a beautiful picture at home".
4. Experiment and Results Analysis

To prove the effectiveness of our method on the task of event identification, we design the following experiments. Firstly, we analyze the influence of \( \omega \) to the embedding of event sentences. Secondly, we compare the performance of different calculation functions of argument-trigger relation. Thirdly, we compare our method with some famous BOW-based sentence embedding methods on the task of event identification.

4.1 Experiment Metric

The basic thought of the experiment can be described as: Firstly, compute the vector of every sentence in the corpus with the sentence embedding approach. Second, select an event sentence of specific type as a seed (defined as \( s \)). Third, choose the top-K event sentences based on their similarity with the seed. Fourth, count the number (defined as \( K_0 \)) of the sentences which have the same type with the seed in the top-K sentences and calculate the precision as \( p = K_0 / K \).

We apply the cosine similarity to measure the possibility that two event sentences are of the same type. In order to make our experimental results more convincing, we randomly select a group of seed sets instead of only one seed for each experiment. We select randomly for \( N \) times and get the seeds sets \( S = \{ S_i | 1 \leq i \leq N \} \), where \( S_i \) is a seed set, \( N \) represents the number of sets. Each set contains seeds of every type. Define \( S = \{ s_j | 1 \leq j \leq T, 1 \leq i \leq M \} \), where \( s_j \) is a seed, \( T \) represents the number of event types, \( M \) represents the number of seeds chosen for every event type. For each seed \( s_j \) in \( S_i \), we can obtain the corresponding precision \( p_{ij} \). We define the final metric macro-\( p \) as the average of \( p_{ij} \):

\[
macro-p = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{T} \sum_{j=1}^{T} \frac{1}{M} \sum_{l=1}^{M} p_{ij}
\]

4.2 Experiment Dataset

The annotated datasets of Chinese event identification are very lacking. ACE provides an annotated dataset designed for trigger-based event identification which is inappropriate to sentence-based event identification. In this paper, we choose the CEC\(^1\) (Chinese Emergency Corpus) dataset as our experimental data. CEC includes news reports about earthquake, fire, traffic accident, terrorist attack and food poisoning. Since CEC doesn’t annotate event sentences with definite types, we annotate them manually with six types (including a non type). The non type consists of not only non events but events that are not interested. Since event sentences are always separated with commas in Chinese, we use comma as the minimum separator and obtain 6882 sentences in total.

In this paper, we use the word embeddings supplied by Li [21]. We assign the same vector in which each coordinate is generated randomly to words that are not in the lexicon. We apply HanLP\(^2\) to segment sentences and construct dependency trees.

4.3 Effect of \( \omega \) on Performance

Parameter \( \omega \) is the weight of the modifier in a phrase. Since core word (including trigger) contains most information of a phrase, we set its weight to be 1 and set \( \omega < 1 \). Meanwhile, modifiers also contains rich information which can be used in event identification, so its weight should be greater than zero. We analyse the influence of \( \omega \) on the performance of our method and plot the curve of \( macro-p \) with \( \omega \) range from 0 to 1 and a granularity of 0.01. We plot three curves of \( macro-p \) which correspond to top-10, top-30 and top-50 in Figure 5. As we can see, \( macro-p \) exhibits a trend of first increase and then decrease with the increase of \( \omega \). What’s more, \( macro-p \) drops with the increase of \( K \). When \( K=10 \), \( macro-p \) gets maximum value 0.7736 at \( \omega=0.62 \). When \( K=30 \), \( macro-p \) gets maximum value 0.6879 at \( \omega=0.56 \). When \( K=50 \), \( macro-p \) gets maximum value 0.6224 at \( \omega=0.55 \).

\(^1\) https://github.com/shijiebei2009/CEC-Corpus
\(^2\) http://hanlp.linrunsoft.com/
4.4 Effect of Argument-trigger Function on Performance

We explore five different argument-trigger functions (named \( f_1, f_2, f_3, f_4 \) and \( f_5 \)) in this paper. Function \( f_1 \) is the simple addition of \( V(\text{arg}) \) and \( V(\text{tri}) \). Function \( f_2 \) represents the subtraction of \( V(\text{arg}) \) and \( V(\text{tri}) \). Since vectors in subtraction have different orders, such as \( V(\text{arg})-V(\text{tri}) \) and \( V(\text{tri})-V(\text{arg}) \), we test the orders in our experiment and only display the best performance one (the same below). Function takes the vertical vector from \( V(\text{arg}) \) to \( V(\text{tri}) \) as \( V(<\text{tri}, \text{arg}>) \). According to the rule of vector subtraction, \( V(<\text{tri}, \text{arg}>) \) can be computed as,

\[
V(\text{tri}, \text{arg}) = V(\text{arg}) - V(\text{tri}) \frac{V(\text{arg})}{V(\text{tri})} \cos \theta
\]

in which \( \theta \) is the angle between \( V(\text{arg}) \) and \( V(\text{tri}) \). Function \( f_3 \) takes the base of a special isosceles triangle as \( V(<\text{tri}, \text{arg}>) \). The vertex angle of the triangle is the angle between \( V(\text{arg}) \) and \( V(\text{tri}) \). \( V(\text{arg}) \) is one leg of the triangle. Another leg is the vector which has the same direction with \( V(\text{tri}) \) and has the same length with \( V(\text{arg}) \). Function \( f_3 \) compute \( V(<\text{tri}, \text{arg}>) \) as:

\[
V(\text{tri}, \text{arg}) = V(\text{arg}) - V(\text{tri}) \frac{V(\text{arg})}{V(\text{tri})}
\]

Function \( f_4 \) replace the subtraction in formula \( (3) \) with vector addition,

\[
V(\text{tri}, \text{arg}) = V(\text{arg}) + V(\text{tri}) \frac{V(\text{arg})}{V(\text{tri})}
\]

We set \( \omega = 0.55 \) and calculate the macro-\( p \) with top-K ranging from 5 to 50. The curves are plotted in Figure 6. As we can see, \( f_1 \) shows the best performance in all functions. It proves that although vector addition is rather simple, it can keep the information of arguments and triggers effectively.
4.5 Performance of Different Sentence Embedding Methods
We compare our approach with four typical BOW-based methods: unweighted, TFIDF-weighted method, SIF, and P-means. Among them, SIF and P-means have showed state-of-the-art performance in many NLP tasks (e.g., textual similarity, sentiment analysis, etc). We tune the parameters of these methods to make them achieve the best performance in our task and set the parameter $\omega$ of our method as 0.55. We plot curves of $\text{macro-}p$ with $K$ ranging from 5 to 50 in Figure 7. As is shown, our method is obviously better than other competitors in the task of event identification. It is interesting that SIF and P-means, which perform well in many NLP tasks, show a poor performance in event identification. By contrast, the simplest unweighted method achieves the best performance among the other four methods. We suppose that there may be two main reasons for this phenomenon. Firstly, sentence embedding is always domain-related, and SIF and P-means may not be adept at event identification. However, since the unweighted method has good portability, it can also perform well in event identification. Secondly, we believe the difference of languages may also affect the performance of sentence embedding methods. There are differences between Chinese and English and it's very difficult to design a method which can perform well both in English and Chinese.
5. Conclusion
This paper provided a simple BOW-based sentence embedding method for Chinese event identification, which compute the embedding of a sentence on its dependency tree and map different dependencies to different weights and calculations. Compared with other BOW-based methods, our method can capture the information of word function and word dependency. To prove the effectiveness of our method, we design a few experiments. The results showed that our approach achieve the best performance among the typical BOW-based methods in Chinese event identification.

Since trigger-based event identification suffers a lot from limitations which are hard to overcome, some researchers turn their attention to sentence-based event identification. Our research provides a useful attempt for the sentence-based event identification. We believe that with the application of sentence embedding technology, sentence-based event identification will soon experience a rapid development.

Acknowledgments
This work was supported by National Natural Science Foundation of China (No. 61602508). We thank the authors of (Zhang et al., 2012) for sharing the CEC corpus. The word embeddings and test data used to support the findings of this study are included within the article and from prior studies which has been cited at relevant places within the text as references.

References
[1] D. Ahn. The stages of event extraction. In Proceedings of ACL, pp. 1-8, 2006.
[2] H. Ji and R. Grishman. Refining event extraction through cross-document inference. In Proceedings of ACL, pp. 254-262, 2008.
[3] P. Gupta and H. Ji. Predicting unknown time arguments based on cross-event propagation. In Proceedings of ACL-IJCNLP, pp. 369-372, 2009.
[4] S.S. Liao and R. Grishman. Using document level cross-event inference to improve event extraction. In Proceedings of ACL, pp. 789-797, 2010.
[5] Y. Hong, J.F. Zhang, B. Ma, J.M. Yao, G.D. Zhou, and Q.M. Zhu. Using cross-entity inference to improve event extraction. In Proceedings of ACL-HLT, pp. 1127-1136, 2011.
[6] T.H. Nguyen and R. Grishman. Event detection and domain adaptation with convolutional neural networks. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pp. 365-371, 2015.
[7] Y.B. Chen, L.H. Xu, K. Liu, D.J Zeng and J. Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pp. 167-176, 2015.
[8] T.H. Nguyen, K. Cho and R. Grishman. Joint event extraction via recurrent neural networks. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 300-309, 2016.
[9] T.H. Nguyen, L.S. Fu, K. Cho and R. Grishman. A two-stage approach for extending event detection to new types via neural networks. Proceedings of the 1st Workshop on Representation Learning for NLP, pp. 158-165, 2016.
[10] S.L. Liu, Y.B. Chen, K. Liu and J. Zhao. Exploiting argument information to improve event detection via supervised attention mechanisms. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. pp. 1789-1798, 2017.
[11] T.H. Nguyen and R. Grishman. Modeling skip-grams for event detection with convolutional neural networks. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 886-891, 2016.
[12] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. Journal of Machine Learning Research, pp.1137-1155, 2003.
[13] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems, pp. 3111-3119, 2013.

[14] T. Mikolov, W.-T. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 746-751, 2013.

[15] J. Pennington, R. Socher, and C.D. Manning. Glove: Global vectors for word representation. Proceedings of the Empirical Methods in Natural Language Processing, pp. 1532-1543, 2014.

[16] S. Li, Z. Zhao, R.F. Hu, W.S Li, T. Liu, and X.Y. Du. Analogical reasoning on Chinese morphological and semantic relations. arXiv preprint arXiv:1805.06504, 2018.

[17] M.J. Kusner, Y. Sun, N.I. Kolkin, and K.Q. Weinberger. From word embeddings to document distances. International Conference on Machine Learning, pp. 957-966, 2015.

[18] G.-I. Brokos, P. Malakasiotis, and I. Androutsopoulos. Using centroids of word embeddings and word mover's distance for biomedical document retrieval in question answering. In Proc. Workshop on Biomedical Natural Language Processing, pp. 114-118, 2016.

[19] S. Arora, Y. Liang, and T. Ma. A simple but tough to beat baseline for sentence embeddings. In International Conference on Learning Representations, 2017.

[20] A. Rücklé, S. Eger, M. Peyrard, and I. Gurevych. Concatenated power mean word embeddings as universal cross-lingual sentence representations. arXiv preprint arXiv:1803.01400, 2018.