Building Language Models with Fuzzy Weights

***Tsan-Jung He[1], †Shie-Jue Lee[2], and Chih-Hung Wu[3]**

[1]Department of Electrical Engineering National Sun Yat-Sen University, Kaohsiung, Taiwan.
[2]Department of Electrical Engineering National Sun Yat-Sen University, Kaohsiung, Taiwan
[3]Department of Electrical Engineering National University of Kaohsiung, Kaohsiung, Taiwan

*Presenting author: zlhe@water.ee.nsysu.edu.tw
†Corresponding author: leesj@mail.ee.nsysu.edu.tw

Abstract

Word2Vec is a recently developed tool for building neural network language models. The purpose of this work is to propose an improvement to Word2Vec by adding fuzzy weights related to the distances in the context to use more information than the way adopted in the original linear bag-of-word structure. In Word2Vec, the same weights are given regardless of different distances between words. We consider that word distances in the context bear certain semantic sense which can be exploited to reinforce connections more effectively for the network model. In order to formalize the influence of different distances in the context, we adopt Gaussian functions to represent fuzzy weights which take part in the training of the connections of network. Various experiments show that our proposed improvement can result in better language models than Word2Vec.

Keywords: word2vec, skip-gram, fuzzy weights, neural network language model, word embedding, natural language processing (NLP).

I. Introduction

In order to deal with many tasks like text processing and near-duplicate detecting easily, an appropriate method to select features is extremely important. In these tasks, one of the most employed method is Term Frequency-Inverse Document Frequency (TF-IDF) [1]-[3] which represents documents as a vector by using word counts in a corpus. However, it would be less efficient when a corpus becomes larger. Worst of all, different words in a corpus are considered as independent objects. The limit of this type of representation is that similar words or synonyms are not taken into account. In the real world, people may use different words to represent the same or similar concepts. When two similar documents contain some synonyms, calculating the similarity between these documents would encounter difficulties. Therefore, if we want to do the calculation in a semantic way, word representation is certainly an issue.

The simplest way of word representation is the one-hot representation [4][5] which is a $1\times N$ vector to map different words in a document. The vector is as large in size as the vocabulary which is the collection of all different words in a corpus of documents. All the word vectors form a matrix. In the matrix, an entry is 1 if the word appears in the underlying document, and is 0 otherwise. However, this type of representation has the problem of the curse of dimensionality, and it is also difficult to express the relationship among different words. Recent researches based on Mikolov et al. [6][7] proposed the Word2Vec model, employed in Neural-network Language Models [8], which transfers the one-hot representation into small dense vectors. Words are expressed as low-dimensional distributed vectors. Word2Vec is also an efficient method to capture complex semantics and word relationships, and can be used to find synonymous words by considering positions in the vector space. Therefore, Word2Vec

has been widely applied in many Natural Language Processing (NLP) tasks such as Part-Of-Speech (POS) Tagging and text classification [9]-[11].

To train a network language model, Word2Vec adopts a linear bag-of-words training. During the training phase, documents are divided into many word sets, usually sentences, and then words in the context windows are selected as targets and surroundings to reinforce the connections. If two words occur simultaneously more frequently, the relationship between these words is more increasingly enhanced. However, all the involved words are treated of equal importance and the information of their positions is not properly exploited in the training phase. In this paper, we overcome this disadvantage by integrating the weights induced from the positions of contextual words in the context window into the Word2Vec training architecture. Intuitively, words nearer to the target in the context window should have a closer relationship to the target and should be more important to the target. So we take advantage of the distance information to improve the semantic accuracy of the resulting language model. We express the importance of words in terms of fuzzy weights. To formalize the influence of different distances in the context, we adopt Gaussian functions to represent fuzzy weights which take part in the training of the connections of network. Various experiments show that our proposed improvement can result in better language models than Word2Vec.

The rest of this paper is organized as follows. Section II briefly introduce the Word2Vec neural network architecture. Section III describes our proposed method to integrate fuzzy weights into the Word2Vec algorithm. In Section IV, we present some experimental results. Finally, a conclusion is given in Section V.

II. Related Work

Word2Vec is a tool released by Google in 2013 [6][7]. It provides a simple and high-quality method to train the representation of words and is used in many natural language processing (NLP) applications. In Word2Vec, it has two main training architectures: continuous bag-of-word (CBOW) and skip-gram (SG). The difference between the two is that CBOW predicts the current word based on the context while SG predicts the surrounding words by the current word.

Training data can rely on the unlabeled text data as the input [12], and the model produces the vectors of the words as the output. The vocabulary is first constructed from the input and the vector representation of words is learned. SG and CBOW work the same in the early steps of the training phase, picking a word as a center and selecting contextual words according to the window size. Then CBOW takes contexts as input and SG takes a center word to infer contexts, as presented in Fig 1. After training, each word is represented as a vector in the vector space. These word vectors can then be used as features in the succeeding classification applications.

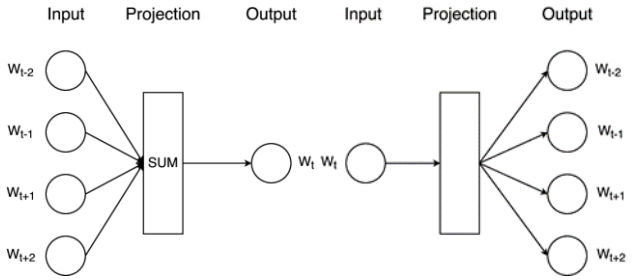


Figure 1: Word2Vec training architectures.

In this work, we focus on the skip-gram architecture. The SG proposed in the original Word2Vec is re-drawn in Fig 2. The input is a token of word w_t, which is mapped to its word vector $v(w)$, and is then used to predict the word vectors of its neighboring words. Given a word sequence w_1, w_2, ..., w_N, the training process maximizes the following objective function.

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{-C\le j\le C, j\neq 0}\log P(w_{i+j} \mid w_i) \tag{1}$$

where is the softmax function defined as

$$P(w_O \mid w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W}\exp({v'_w}^T v_{w_i})}. \tag{2}$$

Note that v_w and v'_w are the input and output vector representations, respectively, of w, and W is the number of words in the vocabulary. As shown in Fig 2, SG models do not take into account the word order to predict the output. Therefore, these methods allow us to find out close vectorial representation of a given word or context, but are not optimal in predicting word sequences based on grammatical and syntactic properties of words.

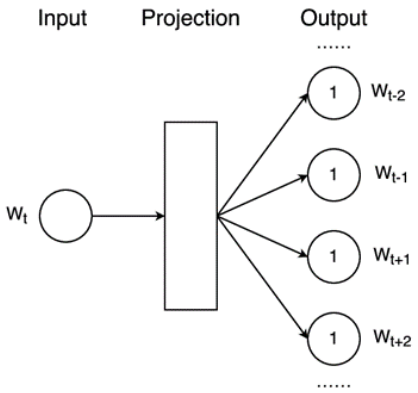


Figure 2: Original Skip-gram model.

III. Proposed Method

In the bag-of-words structure, distances from target words in the context window are not taken into account. In our proposed method, we try to make use of the distances in terms of fuzzy weights in the training phase. Our method can basically be divided into two main processes. The first process is to define adaptive fuzzy weights which denote sensibly the importance of different words according to their distances to the target. The second process is to integrate the fuzzy weights into the SG model of Word2Vec.

Two algorithms are proposed and implemented in our method, skip-gram and negative sampling. Fuzzy weights are integrated in both algorithms. Negative sampling selects a

positive sample (target word w_t) and some negative samples, trying to make these negative samples differ from the target word in the training phase. Although negative sampling may lower the accuracy a bit, but it improves the efficiency in model training.

Word2Vec works as follows. Before beginning to train a network model, as shown in Fig. 3, a vocabulary is built depending on the corpus under consideration. Then random vectors are generated to initialize all the word vectors in the vocabulary, named $syn0$, and set zero to the other synonyms, named $syn1$, which connect word vectors to the output.

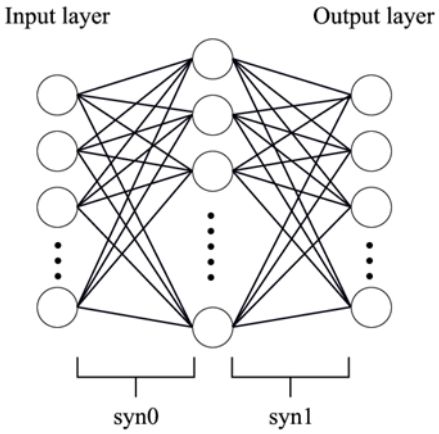


Figure 3: Connections in the network model.

To train the model, the first step is to select the target word, or positive word, $pword$, and contextual words, or last words, $lword$, according to the window size. The skip-gram holds a target word to train a last word in the window one at a time. The model adopts sigmoid function as activate function and expects the output of similar words to be 1. In negative sampling mode, the label of the target word would be set as 1 and the other samples would be 0. The accumulated error err_h is calculated as

$$output = \frac{1}{1+e^{-syn0^{(l)} \cdot syn1_{(l)}^{(p)}}} \tag{3}$$

$$err_{output} = (label - output) \times \alpha \tag{4}$$

$$err_h = err_h + err_{output} \cdot syn1_{(l)}^{(p)} \tag{5}$$

where α is a learning rate, $syn0^{(l)}$ is the vector of the $lword$ in the context window and $syn1_{(l)}^{(p)}$ is the connections from vector $lword$ to vector $pword$, and $syn1$ is updated by

$$syn1_{(l)}^{(p)} = syn1_{(l)}^{(p)} + err_{output} \cdot syn0^{(l)} \tag{6}$$

After training the positive sample and negative samples, the accumulated errors err_h is for back-propagation to fix $syn0$ by

$$syn0^{(l)} = syn0^{(l)} + err_h. \tag{7}$$

Our proposed method allows fuzzy weights into play with the updating of $syn1$ and $syn0$.

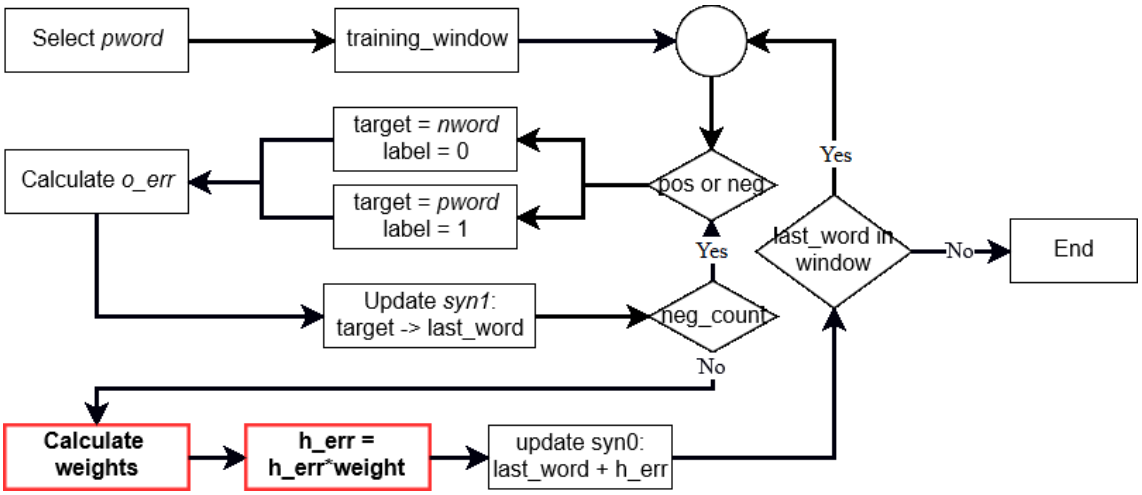


Figure 4: Improved skip-gram algorithm.

As shown in Fig. 4, we calculate the fuzzy weights which are transformed from distances by applying Gaussian functions. For the first process, to take advantage of the relationship of distances between the contextual words and the target word, we replace the original binary weighting of output vectors with Gaussian function values. Words in context are weighted with $G(w_i)$ for word w_i:

$$G(w_i) = \frac{-d(w_i)^2}{2 \cdot \sigma^2} \tag{8}$$

where $d(w_i)$ is the distance between a training word and the target word, i is the order of words in context, and the σ is the radius of Gaussian function. Fig 5 shows the value of each position as a function of the distance from the center of the context window. This function is then used to evaluate the distance relationship of the contextual words. If the position of a contextual word is closer to the target word in the center, the target word would be more influential in the training phase. Consequently, we adopt the following modifications:

$$new\_err_h = G(w_i) * err_h \tag{9}$$

$$syn0^{(l)} = syn0^{(l)} + new\_err_h. \tag{10}$$

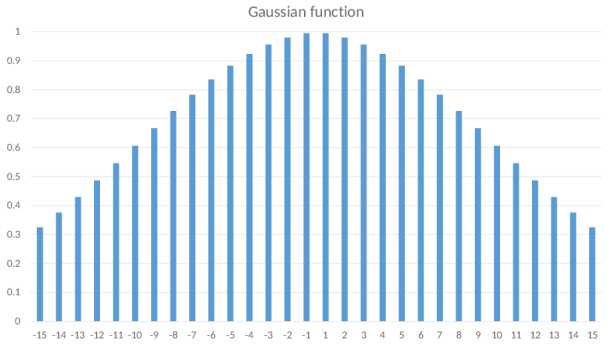


Figure 5: A Gaussian function with $\sigma^2 = 100$.

Fig 6 presents the skip-gram model which incorporates the context weighting approach. The skip-gram model refers to a word to predict its surrounding words. The training process selects a center target word and one of contextual words to form a pair, and then does the training pair by pair. If the weights were applied on word vector directly, the new vector would be regarded as a different word, so we propose to apply the weights in the phase of updating errors. Updating the accumulated error affects the most important learning steps of skip-gram, so we adjust the weights to make similar words closer easily.

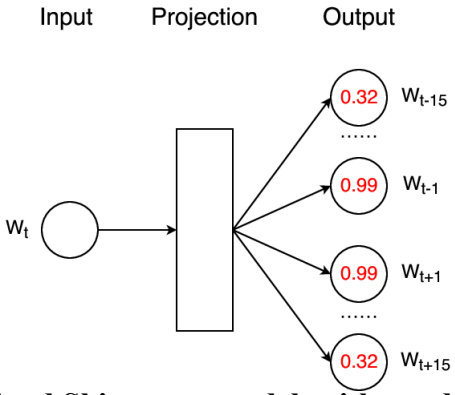


Figure 6: Fuzzy Weighted Skip-gram model, with and context window size .

Example
Assume in an article we have the following sentence:

Lions and jaguars hunt zebras, giraffes and gazelles in the savanna.

Let the context window size be 5, $\sigma^2 = 100$, and the target word be *giraffes*. Table 1 presents the involved fuzzy values. In this table, the first row lists the contextual words, the second row shows the relationship according to the position of the target word, and fuzzy weights are listed in the third row.

Table 1. Fuzzy weights of words

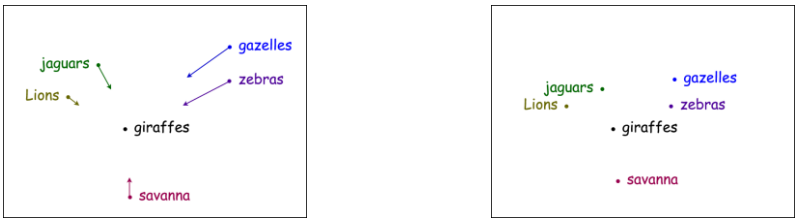
contextual	**Lions**	**and**	**jaguars**	**hunt**	**zebras**
position	-5	-4	-3	-2	-1
weights	0.29	0.45	0.64	0.82	0.95
contextual	**and**	**gazelles**	**in**	**the**	**savanna**
position	+1	+2	+3	+4	+5
weights	0.95	0.82	0.64	0.45	0.29

The following are training pairs:

(Lions, giraffes), (and, giraffes), (jaguars, giraffes), (hunt, giraffes), (zebras, giraffes), (and, giraffes), (gazelles, giraffes), (in, giraffes), (the, giraffes), (savanna, giraffes).

When a weight is greater, the error feedback caused is more effective. In this example, the vectors of 'zebras' and 'gazelles' may have more chance to get closer to 'giraffes' than to 'Lions' and 'jaguars'.

Fig. 7(a) shows the distribution of the words in the vector space. In this figure, the length of arrows stand for the strength of training weights. Note that the strengths are different due to their positions. After updating the vectors, 'zebras' and 'gazelles' take a bigger step to approach 'giraffes', but 'Lions', 'jaguars', and 'savanna' are less effective, as shown in Fig. 7(b).



(a) Before. (b) After

Figure 7: Words in the vector space: (a) Before updating; (b) after updating.

IV. Experimental Results

In this section, we evaluate the effectiveness of our proposed method. We also compare our method with the original Word2Vec method. We conduct our experiments with the Google-One-Billion-Word-Language-Modeling Benchmark containing around 1-billion words for training. After training, there are 553,402 words in the vocabulary. Testing words are also taken from Google, a file named question-word, containing 14 topics: capital-common-countries, capital-word, currency, city-in-state, family, and 9 grammars. The Semantic-syntactic Word relationship test is made of 19,558 questions. Its main objective is to verify if a distributed representation of words captures complex syntactic and semantic relations between words. A question is made of two pairs of words sharing the same relation. For example, for the following question

'woman' 'man' = 'queen' 'king',

The model has to infer from 'queen' to 'king' through the relationship between 'woman' and 'man'. A question in the capital-common-countries topic is something like

Athens Greece Baghdad Iraq.

All the test questions contain the rules of inference that people do in daily life. In other words, if a model gets a large score in this test, the model is accurate in the semantic sense.
We perform experiments with different settings of the parameters for the window size, dimension of vector space, and radius of Gaussian functions. The results with different window sizes are shown in Table 2.

Table 2. Accuracy results with different window sizes

dimension = 100	**w5**	**w10**	**w20**	**nMAX**
capital-common-countries:	363	369	364	462
capital-world:	941	984	976	1295
currency:	7	5	7	40
city-in-state:	418	466	476	1700
family:	260	249	218	342
gram1-adjective-to-adverb:	191	215	235	930
gram2-opposite:	109	112	104	462
gram3-comparative:	1060	986	915	1332
gram4-superlative:	546	515	416	756
gram5-present-participle:	677	668	617	870
gram6-nationality-adjective:	1073	1088	1104	1160
gram7-past-tense:	876	868	844	1560
gram8-plural:	696	689	722	992
gram9-plural-verbs:	411	375	327	650
Total	**7628**	7589	7325	12551

Note that dimension = 100 for this table. Three window sizes are set, namely window size = 5, 10, and 15, represented by w5, w10, and w15, respectively. From this table, we can see that a larger window may not improve the semantic accuracy, since the words far away the target may act as noise for training. In many situations, further words may be less related to the central word syntactically, but for some sentences in which an object mentioned far away may have more influence for a setting with long context window. In Table 2, the topics like 'capital-world' and 'capital-in-state' contain more terms when the window size increases. As a result, accuracy increases as the window size increases. The experimental results with different dimensions are shown in Table 3.

Table 3. Accuracy results with different dimension sizes

window size = 5	**d100**	**d200**	**d400**	**nMAX**
capital-common-countries:	371	392	420	462
capital-world:	932	1044	1103	1295
currency:	5	8	10	40
city-in-state:	400	724	916	1700
family:	260	277	274	342
gram1-adjective-to-adverb:	191	183	167	930
gram2-opposite:	115	165	189	462
gram3-comparative:	1069	1171	1191	1332
gram4-superlative:	573	606	578	756
gram5-present-participle:	678	711	690	870
gram6-nationality-adjective:	1070	1090	1112	1160
gram7-past-tense:	871	920	948	1560
gram8-plural:	698	744	784	992
gram9-plural-verbs:	416	491	460	650
Total	7649	8526	**8842**	12551

For this table, window size is set to be 5, and three dimensions are set, namely dimension = 100, 200, and 400, represented by d100, d200, and d400, respectively. From this table, we can see that the bigger the dimension of the vector space is, the higher accuracy is obtained. As the dimension increases, more information can be stored in the hidden layer and the semantic relationship among words can be more completely maintained. The experimental results with different radii of Gaussian functions are shown in Table 4.

Table 4. Accuracy results with different radii

	Word2Vec	**r100**	**r125**	**r150**	**r175**	**r200**	**r225**
capital-...:	392	**412**	408	406	403	404	403
capital-...:	1078	1081	**1089**	1087	**1089**	1085	1087
currency:	8	9	9	9	**10**	9	9
city-in-...:	920	**1001**	968	972	971	957	962
family:	246	**269**	267	264	260	257	259
gram1-...:	216	227	229	222	221	221	**237**
gram2-...:	**186**	171	178	178	174	174	172
gram3-...:	1120	1121	1108	1123	1112	1125	**1129**
gram4-...:	518	533	540	543	546	**547**	542
gram5-...:	691	698	690	696	698	693	**703**
gram6-...:	**1129**	1128	1128	1126	1128	1126	1128
gram7-...:	938	966	970	**974**	965	969	955
gram8-...:	784	786	794	795	**800**	791	787
gram9-…:	410	418	418	415	418	**427**	423
Total	8636	**8820**	8796	8810	8795	8785	8795

For this table, six values are set to the radius, namely σ^2 = 100, 125, 150, 175, 200, and 225, denoted by r100, r125, r150, r175, r200, and r225, respectively. As can be seen, all the versions of our proposed method perform better than the original Word2Vec in terms of semantic accuracy. Our method can improve the semantic accuracy. However, more execution time is needed. The computer system we have used for the experiments is shown in Table 5. Table 6 shows the execution times required for Word2Vec and our method.

Table 5. Computing environment

Experimenting equipment	
CPU	Intel(R) Core(TM) Core i7-6700 @ 3.40GHz.
RAM	2133 GHz, 8 GB
OS	Ubuntu 14.04 LTS x64
Develop tools	Python 3.5
Tool Library	gensim-0.13.2

Table 6. Comparison on execution time

	w5	Time:(s)	**w10**	Time:(s)	**w15**	Time:(s)
Word2Vec	8769	11837	8718	18574	8636	23576
r100	8798	11981	8796	18698	**8820**	23849
r125	8757	12345	8784	19134	8796	24481
r150	8782	12407	8801	19327	8810	24579
r175	8764	12504	**8817**	19709	8795	25032
r200	**8808**	12304	8801	19157	8785	24386
r225	8751	11993	8794	18768	8795	23845

As can be seen, our method runs slower than Word2Vec, but only slightly.

V. Conclusion

There are many situations where the words with similar meanings appear at the same time in the same sentence. Word2Vec adopts the bag-of-words model to train a type of word representation with the idea of context window. It makes senses that relevant words may appear simultaneously in a context window. However, Word2Vec ignores the distance information and treats each word equally in the context. In our work, we offer an alternative method to weight the contextual words at different positions. Fuzzy values are exploited to express the degree of importance a context word imposed on the target due to the distance between them. The derived fuzzy values take part in the training of network language models. From the experimental results, we have seen that our method can improve the semantic accuracy for the Google-One-Billion-Word-Language-Modeling Benchmark dataset.

References

[1] Salton, G.and Lesk, M. E. (1968) Computer evaluation of indexing and text processing, *Journal of the ACM (JACM)* **15**, 8–36.

[2] Sparck Jones, K. (1972) A statistical interpretation of term specificity and its application in retrieval, *Journal of documentation* **28**, 11–21.

[3] Salton, G. and Buckley, C. (1988) Term-weighting approaches in automatic text retrieval, *Information processing & management* **24**, 513–523.

[4] Turian, J., Ratinov , L. and Bengio, Y. (2010) Word representations: a simple and general method for semi-supervised learning, *Proceedings of the 48th annual meeting of the association for computational linguistics*, 384–394.

[5] Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C. (2003) A neural probabilistic language model, *Journal of machine learning research* **3**, 1137–1155.

[6] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013) Distributed representations of words and phrases and their compositionality, *Advances in Neural Information Processing Systems* **26**, 3111–3119.

[7] Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013) Efficient estimation of word representations in vector space, *CoRR*, **abs/1301.3781**.

[8] Kim, Y. (2014) Convolutional neural networks for sentence classification, *arXiv preprint arXiv:1408.5882*.

[9] Socher, R., Bauer, J., Manning, C. D. and Ng, A. Y. (2013) Parsing with compositional vector grammars. *ACL (1)*, 455–465.

[10] Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., Potts, C. et al., (2013) Recursive deep models for semantic compositionality over a sentiment treebank, *Proceedings of the conference on empirical methods in natural language processing (EMNLP)* **1631**, 1642

[11] Xue, B., Fu, C. and Shaobin, Z. (2014) A study on sentiment computing and classification of sina weibo with word2vec, *IEEE International Congress on Big Data (Big Data Congress)*, 358–363.

[12] Nigam, K., McCallum, A. K., Thrun, S. and Mitchell, T. (2000) Text classification from labeled and unlabeled documents using EM, *Machine learning* **39**, 103–134.