Learning Quality Improved Word Embedding with Assessment of Hyperparameters

Beytullah Yildiz[1(✉)] and Murat Tezgider[2]

[1] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
byildiz@gmail.com
[2] Hacettepe University, Ankara, Turkey
murat.tezgider@hacettepe.edu.tr

Abstract. Deep learning practices have a large impact on many areas. Big data and key hardware developments in GPU and TPU are the main reasons behind deep learning success. The recent progress in the text analysis and classification using deep learning has been significant as well. The quality of word representation that has become much better by using methods such as Word2Vec, FastText and Glove has been important in this improvement. In this study, we aimed to improve Word2Vec word representation, which is also called embedding, by tuning its hyperparameters. The minimum word count, vector size, window size, and the number of iterations were used to improve word embeddings. We introduced two approaches, which are faster than grid search and random search, to set the hyperparameters. The word embeddings were created using documents with approximately 300 million words. A deep learning classification model that uses documents consisting of 10 different classes was applied to evaluate the quality of word embeddings. A 9% increase in classification success was achieved only by improving hyperparameters.

Keywords: Deep learning · Machine learning · Text analysis · Text classification · Word embedding · Word2Vec

1 Introduction

The data produced in the digital world is increasing overwhelmingly. As a result of the development and widespread of the Internet, the data produced and served by internet applications such as social media have given a different impetus to the speed of data production. Text data have a significant share of these vast data. With the increasing volume, tasks performed on the text such as classification [1, 2], clustering, sentiment analysis, information extraction, information retrieval, and searching have become more important. Moreover, the success rate of text processing has significantly increased by deep learning methods in the advent of more data and better computing power.

Text processing requires text representation. Therefore, various methods have been introduced for text representation. One of the important obstacles of text processing has been feature extraction which has been recently eased by deep learning methods. There are different studies in which words, word grams, word roots and bodies, character

U. Schwardmann et al. (Eds.): Euro-Par 2019 Workshops, LNCS 11997, pp. 506–518, 2020.
https://doi.org/10.1007/978-3-030-48340-1_39

grams are used as features to represent text [3–6]. By using a set of word and context pairs extracted from the corpus, vector representations of words can be derived by applying various estimation methods, such as predicting words given their contexts (CBOW), predicting the contexts from the words (Skip-Gram), or factorizing the log of their co-occurrence matrix. Word2Vec [7, 8] implements both Continuous Bag of Words (CBOW) and Skip-Gram (SG) methods. FastText [3] also provides these two models to compute word representations. Although Word2Vec treats each word in corpus like an atomic entity and generates a vector for each word, FastText, which is essentially an extension of word2vec model, considers each word as composed of character n-grams. Therefore, the vector for a word is the sum of these character n-grams. For example, the word vector "orca" is a sum of the vectors of the n-grams such as "or", "orc", "orca", "rca", "ca". Glove [9], on the other hand, factorizes the log of the co-occurrence matrix. In these methods, picking the right context is a critical factor that affects the quality of the resulting vector representations. The most common method for defining this context is to rely on a window positioned around the word. The context window decides which contextual neighbors are taken into consideration to produce the vector representations.

The empirical variations between representation models, which is also called embedding, are basically because of differences in hyperparameters rather than differences in the embedding algorithms [10]. Hence, it is likely that different results are obtained while constructing word embedding with different corpus containing different topics in different dimensions because the size and content of the corpus will cause the words to take on semantically and syntactically different vector values [11]. Additionally, the quality of word representations is significantly affected by hyperparameters such as minimum word count, vector size, window size and the number of iterations.

In this study, we present two approaches to evaluate the important hyperparameters of Word2Vec which are faster than grid search and random search. In general, the quality of the word embedding is not enough because the default hyperparameter values are used to create the word embedding. In addition, unlike well-known deep learning models such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), hyperparameter tuning for word embedding has not been well studied. Moreover, to the best of our knowledge, there isn't a study extensively measuring the accuracy of Word2Vec representations for the Turkish language. 3 million Turkish texts which consist of totaling 300 million words were used to create Word2Vec word embedding.

To evaluate the quality of the word embedding, a deep learning model developed for text classification was used. The classification model and the data are kept unchanged to examine the effect of Word2Vec hyperparameters on the quality of word embedding. Text classification with deep learning model was performed by using different word embeddings created by changing Word2Vec hyperparameters. According to the accuracy of the classification process, the quality of the word embeddings has been measured. We used multi-core and CUDA-enabled GPU environments to create and evaluate word embeddings. Cloud base TPU and GPU can be used to accelerate word embedding, as the number of documents used and the number of unique words requires more processing power.

In the second section, related works are explained. Information about Word2Vec word embedding and used environment will be given in Sect. 3. Section 4 consists of measurements and evaluation. We conclude and give "rules of thumb" in Sect. 5.

2 Related Work

There are several studies investigating the hyperparameters of word embedding methods. Caselles-Dupré et al. investigated the importance of hyperparameters through large hyperparameter grid searches on various datasets [12]. The results revealed that optimizing the hyperparameters significantly improved the performance of a recommendation task.

Levy et al. claimed that most of the word embedding performance gains were due to specific system design choices and hyperparameter optimizations rather than embedding algorithms [10]. Although it is advisable to adjust the entire hyperparameters for the task at hand, this approach may be expensive in terms of calculation. Therefore, they provided some "thumb rules" for the solution.

In general, the quality of word representation was measured using either a model or an analogy and similarity datasets. The quality of Word2Vec word embeddings is assumed to affect the accuracy of the classification model. Embedding models often associate each word with a single vector representing its properties. Therefore, evaluation methods should analyze the accuracy and completeness of these properties. Multi-label classification is a convenient way to carry out this evaluation [13]. Nooralahzadeh et al. conducted evaluations of both general and domain-specific embeddings [14]. Evaluation of embedding models was provided by the task of domain-specific sentence classification.

Analogy and similarity datasets were often used to measure the quality of word representations, consisting of questions and answers that query the semantic and syntactic relations of words. Mikolov et al. used 8869 semantic, 10675 syntactic questions in total, consisting of 5 semantic question types and 9 syntactic question types to measure the quality of word vectors [7]. Lia et al. experimented on WordSim353 and the TOEFL dataset to measure semantic and syntactic relationships [11]. In their study, they compare the methods used for representing words as vectors.

3 Word2Vec Word Embedding and Used Environment

Word2Vec was proposed by Mikolov et al. in 2013 to represent a word as a vector [8]. It takes a large corpus as input and usually produces vectors of several hundreds of dimensions. Word2Vec represents words in vector space based on the unsupervised prediction. Word2Vec aims to minimize the distance value of words that are the same or semantically and syntactically close to maximize the similarity value. CBOW and SG are commonly used methods. The CBOW method estimates the center word by using adjacent words. Rather, the SG attempts to predict neighboring words using the center word. The SG model consists of input, hidden and an output layer. The input layer uses a one-hot encoding. In one-hot encoding, an index is assigned to each word.

The value corresponding to the word index in the vector is set to 1, and the others to 0. The output layer uses a softmax classifier. The number of neurons in the hidden layer determines the size of the Word2Vec vector because weights in the hidden layer are used to represent words as vectors.

Deep learning models benefit from parallel processing. However, it can be argued that no learning algorithm is really embarrassingly parallel, but some are almost embarrassingly parallel. As with previous parallel applications [15–17], procedures such as parallel processing, pipelining and orchestration should be used in the best way. In addition, deep learning, word embedding as well, is basically an optimization problem. In other words, optimizing hyperparameter is one of the most important functions.

We investigated important hyperparameters that affect the quality of Word2Vec word representation. Minimum word count, vector size, window size and number of iterations were the hyperparameters on which experimented. The quality of Word2Vec word embeddings is assumed to affect the accuracy of the classification model in which word embeddings are used. From this hypothesis, a deep learning classification model was used to evaluate the word embeddings. The classification was performed with the word embeddings created with different hyperparameter sets. The classification model and the data were kept unchanged and only the Word2Vec hyperparameters were adjusted. Based on the success of the classification model, we concluded the successful hyperparameters to create better word embeddings. Keras library and TensorFlow infrastructure were utilized for the classification. Gensim library [18] was used to construct Word2Vec word vectors. We created the word vectors by choosing combinations of 5, 10, 15, 20, 25 values as window size; 1, 2, 5, 10, 20, 30 values as the minimum word count; 50, 100, 150, 200, 250, 300 values as the vector size; 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60 values as the number of iterations.

Grid search and random search are among the well-known parameter optimization methods. In both methods, the processing can take a very long time to determine the hyperparameter values. For the grid search, when the hyperparameter values mentioned above are used, the 2160 combinations must be tried. A single word embedding can take more than a day depending on the hyperparameters. Therefore, we used two approaches to set up Word2Vec hyperparameters.

Table 1. Default values of Word2Vec model hyperparameters.

Minimum word count	5	Window size	5
Vector size	100	Number of iterations	5

In the first approach, we initially started with the default values in Table 1. In each step, only one of the Word2Vec hyperparameters was updated to create word embeddings. For a single hyperparameter, we created as many word embeddings as the number of values of the hyperparameter. After evaluating each hyperparameter, the most appropriate word embedding was obtained with the hyperparameter set using the best hyperparameter values.

The second approach similarly starts with the default hyperparameters. In each step, a hyperparameter value that produces the best result is determined and used in place of the default value of that parameter in the next steps. This model continues progressively. At the last stage, parameters that produce the best results are saved as the best hyperparameter set.

3.1 Classification Model

To investigate Word2Vec word representation using text classification, we constructed a model by using CNN, which is a deep learning model. Two convolutional layers, two maximum pooling layers, one flatten layer, two fully connected (dense) layers were used. Like the input layer, the embedding layer has dimensions of 160 x "vector size". The maximum text length was set to 160 words. For shorter texts containing fewer than 160 words, a vector of zeroes was added. In the first CNN layers, 64 filters with ReLU activation function were used. Kernel size was set to 5 for the first CNN layer, and 2 for the second CNN layer. Maximum pooling size was set to 2 for the first maximum pooling layer and was set to 3 for the second maximum pooling layer. Stride size was set to 1 for both layers. 0.5 value for the dropout layer was used to prevent overfitting. A fully connected layer of 128 units was used with ReLU activation function. Softmax activation function was used in the output layer consisting of 10 units. Adam was used as the optimization function and categorical cross entropy was used as the loss function.

3.2 Dataset

3 million Turkish texts with approximately 300 million words were used to create Word2Vec word embeddings. Corpus consists of about 2.8 million unique tokens. From the texts used for Word2Vec creation, 149504 text documents for classification were selected. The documents were labeled with 10 different classes. 104448 documents corresponding to 70% of the total documents were used for training, 22528 documents corresponding to 15% of the total documents were used for verification and 22528 documents corresponding to 15% of the total documents were used for testing purposes.

3.3 Hyperparameters

Words that appear only very few times in the hundreds of million words corpus are probably uninteresting typos or mistakes. Moreover, there is insufficient data to make a reasonable training on these words, so it is best to throw them away. Minimum word count hyperparameter is used to remove words by the number of appearance in the documents. For example, if the minimum word count parameter is set to 5, which is the default value of the Gensim library used, words that are presented less than 5 times will be discarded.

The vector size hyperparameter defines the vector dimension of Word2Vec. This hyperparameter also specifies the number of units in the hidden layer of the Word2Vec model. Therefore, the vector size also affects the cost of computation. Increasing the

vector size will also increase the cost. However, the larger vector size may lead to better and more accurate models even though it may require more training data.

The window size indicates how many words to use for prediction from the left and right of the input word. The window size is the most noteworthy hyperparameter associated with the context. When it is set to 5, which is the default value of the Gensim library, 5 words will be used to the left and right of the input word for content prediction. Larger window size tends to capture more topic and domain information while smaller window size tends to capture more about 'functional' and 'synonymic' models, which may lead to better performance on similarity measurements [19, 20].

The number of iteration determines how many times the data is to be trained. Increasing the number of iterations generally improves the quality of word representation, but also significantly increases the duration of training.

4 Measurement and Evaluation

We investigated the quality of the word embedding by using a classification model. Word2Vec word embeddings created with the hyperparameters using the SG method were evaluated by using the classification model and dataset that were kept both unchanged. The classification was repeated with the word embeddings obtained by changing the value of one of the hyperparameters at a time. The optimum values of the hyperparameters were determined by evaluating the accuracy and loss values of the classification.

The two approaches mentioned in the methodology section were used for the classification to determine the hyperparameters. The results obtained by the first and second approaches will be explored below. Since the experiment of the minimum word count hyperparameter is the same for both approaches, the results are given only in the first approach. The second approach is continued using this result.

4.1 Decisive Approach

In Decisive Approach (DA), only the value of hyperparameter examined was changed. We started evaluating Word2Vec's minimum word count because it influenced the number of words of the Word2Vec dictionary. We created six models by setting the minimum word count hyperparameter to 1, 2, 5, 10, 20 and 30. The remaining parameters are fixed to the values given in Table 1.

We did not take any action to correct the misspelled words or typos because we want to make sense of the words that people wrote incorrectly. Depending on the geographical region, there are also some local forms of words that can be seen as a misspelled word. Therefore, we will use "token" instead of "word" to indicate any original word form in the documents. Because a token contains a regular word as well as a prefixed, affixed or misspelled word, the number of unique tokens may significantly exceed the number of words in a language. We saw this situation in our Word2Vec dictionary because there were 2.895.675 unique tokens. When the tokens used once or twice were examined, it was found that the vast majority of these tokens were misspelled. Very few tokens were very rarely used words.

The statistical details of the tokens for Word2Vec models using different values of minimum word count are given in Table 2. When the value of minimum word count is set to 1, all 2.895.675 unique tokens are used and the total number of tokens in the dictionary is 297.149.774. When the minimum word count is given as 2, 1.537.529 tokens that repeat once are removed from the vocabulary. The remaining 1.358.146 unique tokens correspond to about 47%. However, the total number of tokens does not change significantly. Less than 1% of all tokens are removed in the dictionary. This is better seen when the minimum word count is 30; while only 7,72% of the unique tokens remain, the percentage of the remaining total tokens is 96,97%. The sudden drop in the number of unique tokens is primarily due to misspelled words. The frequency of a word in documents used when training the Word2Vec model affects its correct positioning in the vector space. Less repetitive words in the corpus are thought to be not positioned correctly in the vector space. Accuracy and loss values of the classification model trained using the word vectors are significantly affected by the correct placement in this space. Therefore, the removal of very few repetitive words has a positive effect on classification success. When the value of minimum word count is increased, the accuracy of the classification model increases and the value of loss reduces, shown in Fig. 1.

Table 2. Word2Vec vectors and statistics

Minimum word count	Removed unique tokens	Remaining unique tokens	Percentage of remaining unique tokens	Removed total tokens	Remaining total tokens	Percentage of remaining total tokens
1	0	2.895.675	100	0	297.149.774	100
2	1.537.529	1.358.146	46,9	1.582.536	295.567.238	99,46
5	2.236.456	659.219	22,76	3.002.271	294.147.503	98,98
10	2.493.595	402.080	13,88	4.916.341	292.233.433	98,34
20	2.611.046	284.629	9,82	7.138.619	290.011.155	97,59
30	2.672.127	223.548	7,72	8.994.314	288.155.460	96,97

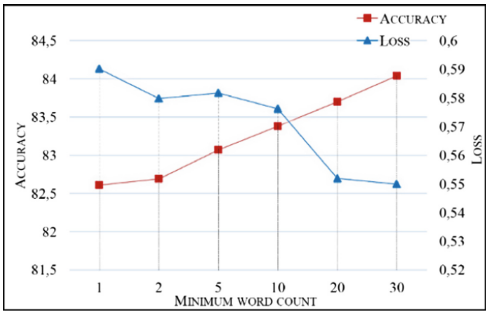

Fig. 1. Classification accuracy and loss for the various values of minimum word count

After the minimum word count, the effect of vector size hyperparameter of Word2Vec was examined. Word2Vec vectors were created by using hyperparameters of experiment 2 column in Table 3. When the classification was applied by using the same model and dataset, we obtained the accuracy and loss rates shown in Fig. 2. Keeping the vector size too large or too small affects the success of the classification negatively. It should be aimed to find an optimum vector size according to the available datasets. We observed that vector size, ranging from 50 to 300, has an impact on the classification accuracy of about 2%. The most appropriate value of the vector size for the dataset used in this study was 250. But when the amount of data is increased, using a larger-sized vector would be a more accurate approach [7].

Table 3. Hyperparameters for DA after minimum word count experiment

Parameters	Experiment 2	Experiment 3	Experiment 4
Minimum word count	5	5	5
Vector size	50, 100, 150, 200, 250, 300	100	100
Window size	5	5, 10, 15, 20 ,25	5
Number of iterations	5	5	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60

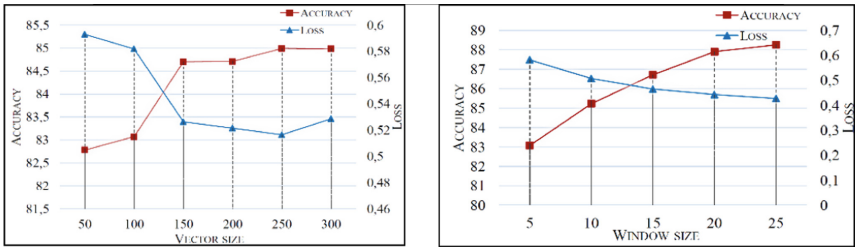

Fig. 2. Effect of vector size and window size on classification accuracy and loss for DA

Window size is an important hyperparameter to detect the context of a word. Therefore, it is expected that it will significantly affect the classification success. Using the same model and dataset, we used Word2Vec embeddings prepared using the hyperparameters of the experiment 3 column in Table 3 for classification. A higher value of window size appears to have a positive effect on the classification success shown in Fig. 2. While the value of window size increases, the classification accuracy also increases. When the window size increases from 5 to 25, a 5% improvement in classification success is achieved. Therefore, it can be deduced that context is important for classification and significantly affects the classification accuracy.

The hyperparameters of the experiment 4 column in Table 3. were used to examine the effect of the number of iterations used to train Word2Vec model. The classification

results obtained using word vectors are given in Fig. 3. While the number of iterations increased from 5 to 60, the success of classification increased by about 6%. This improvement shows that the number of iterations is an important hyperparameter. However, the contribution of the number of iterations to the classification success starts to slow down after 15 iterations. Therefore, an iteration value that gives a certain success can be selected because each iteration requires extra time for training.

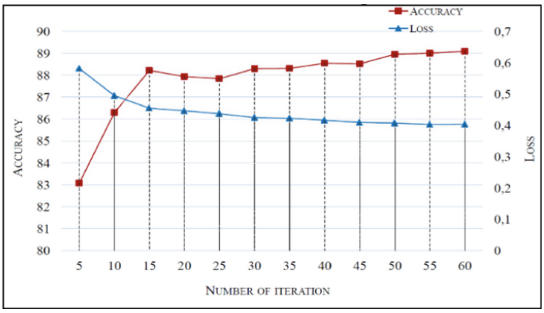

Fig. 3. Effect of number of iteration on classification accuracy and loss for DA

4.2 Progressive Approach

In Progressive Approach (PA), after determining the value of parameters that give the best result for classification, unlike DA, the best value is used instead of the default value for the subsequent steps. Since the minimum word count parameter was investigated in DA, we did not repeat this step and used the results of the minimum word count from DA. Minimum word count will be 30 as the best value for the subsequent experiments.

Table 4. Hyperparameters for PA after minimum word count experiment

Parameters	Experiment 2	Experiment 3	Experiment 4
Minimum word count	30	30	30
Vector size	50, 100, 150, 200, 250, 300	250	250
Window size	5	5, 10, 15, 20, 25	25
Number of iterations	5	5	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60

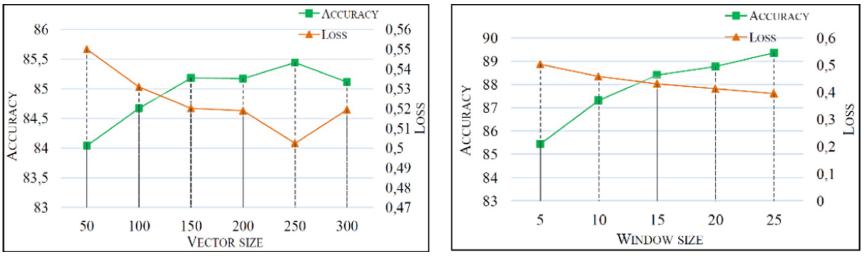

Fig. 4. Effect of vector size and window size on classification accuracy and loss for PA

The effect of vector size was examined using the hyperparameters of experiment 2 in Table 4. The results are given in Fig. 4. The best result is when the vector size is 250, which is the same as in DA. With the best minimum word count value, the vector size provides an almost 1.5% improvement with a value of 250. This was almost 2% for DA. We will use the 250 for the vector dimension parameter for the next steps.

The hyperparameters of experiment 3 in Table 4 were used to measure the effectiveness of the window size. The results are shown in Fig. 4. Increasing the window size value increases the classification success by less than 4%. In DA, the rate of improvement was 5%. As PA uses the best values in the past steps, the success of classification is seen to be increased at a lesser rate. The window size value will be used as 25 for the next steps because the best result is obtained with the value of 25.

The hyperparameters of experiment 4 in Table 4 were used to examine the effect of the number of iterations. The results are shown in Fig. 5. The increase in the number of iterations increases the success of classification by about 1%. In the first method, the contribution of the number of iterations to the classification success was about 6%. Since other parameters contribute to the improvement of classification in the previous steps, the number of iterations in this method seems to be less effective. The best result was taken at 45 iterations. However, the value of 60 also showed a very close success.

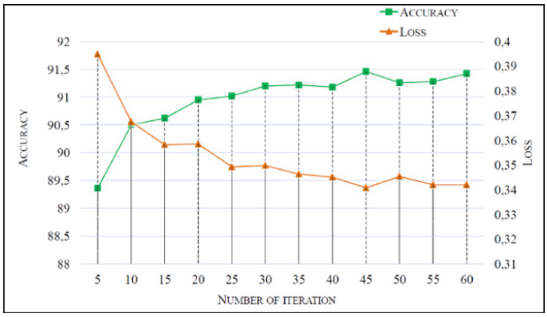

Fig. 5. Effect of iteration number on classification accuracy and loss for PA

4.3 Comparing Word2Vec Models Using the Best and Default Hyperparameters

In PA, the parameters that produce the best result in the final stage are chosen as a successor. The best parameters obtained by DA were the same as the best parameters obtained with the second method, except the number of iterations.

Table 5. Word2vec parameters set for best and worst classification accuracy

Method	Min. word count	Vector size	Window size	Number of iterations	Loss	Accuracy
Default values	5	100	5	5	0,5817	83,07
Decisive approach	30	250	25	60	0,3397	91,42
Progressive approach	30	250	25	45	0,3408	91,46

Table 5 shows the parameters for the best and worst cases in terms of the loss and accuracy of Word2Vec parameter evaluations. When we applied the classification model to classify the documents into 10 classes by using different parameters of Word2Vec, we got about 9% improvement in the classification accuracy.

5 Conclusion

In this study, we evaluated Word2Vec hyperparameters that affect the quality of word representation. A classification model was used to determine the Word2Vec hyperparameters. The results clearly show that Word2Vec hyperparameters affect the classification accuracy and thus the quality of word representation. We observed a 9% increase in the accuracy of our classification model. Considering that the classification process is done in 10 classes, the success rate achieved by setting only Word2Vec hyperparameters cannot be ignored. The Progressive Approach has been observed to offer faster convergence and more efficient performance improvement. Therefore, using the best value of each hyperparameter in the next steps is a wise choice for Word2Vec hyperparameter tuning.

We must state that there are no rules-of-thumb for a good word embedding applying to every purpose. However, we make the following conclusions for better word embedding. It was observed that the vector size, window size, and iteration number were the main hyperparameters affecting the word representation quality. Setting these parameters too large or too small can adversely affect success. The larger window size captures the topic and semantic better, but the smaller window size is more relevant to the syntactic relationship. The optimum vector size depends on the

size of the datasets. Larger datasets require a larger vector size. Although the higher number of iterations requires more computing time, it is generally better for word representation quality.

The minimum word count determines the number of unique words in the dictionary and affects the amount of memory used. Very few repetitive words are often misspelled or very rare. Training these words in the Word2Vec model does not contribute to the quality of word representation. As can be seen in the results, with the small increase in the minimum word count, the number of the unique word in the dictionary is halved. This helps to remove words that are insignificant for the model. Therefore, performance gains are achieved by decreasing very rare and misspelled words.

The frequency of words affects the optimal value of hyperparameters. More frequent words may not require a larger window size for a good representation of the words. Therefore, the smaller window size value may be as good as the larger values when the dataset grows.

References

1. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. **41**(6), 391–407 (1990)
2. Pang, B., Lee, L.: Opinion mining and sentiment analysis. Found. Trends Inf. Retriev. **2**(1–2), 1–135 (2008)
3. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)
4. Wang, P., Xu, B., Xu, J., Tian, G., Liu, C.-L., Hao, H.: Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification. Neurocomputing **174**, 806–814 (2016)
5. Liu, J., Chang, W.-C., Wu, Y., Yang, Y.: Deep learning for extreme multi-label text classification. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 115–124 (2017)
6. Hinton, G.E.: Learning distributed representations of concepts. In: Proceedings of the 8th of the Cognitive Science Society, vol. 1, pp. 12 (1986)
7. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
8. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing System, pp. 3111–3119 (2013)
9. Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1532–1543 (2014)
10. Levy, O., Goldberg, Y., Dagan, I.: Improving distributional similarity with lessons learned from word embeddings. Trans. Assoc. Comput. Linguist. **3**, 211–225 (2015)
11. Lai, S., Liu, K., He, S., Zhao, J.: How to generate a good word embedding. IEEE Intell. Syst. **31**(6), 5–14 (2016)
12. Caselles-Dupré, H., Lesaint, F., Royo-Letelier, J.: Word2Vec applied to recommendation: hyperparameters matter. arXiv preprint arXiv:1804.04212 (2018)

13. Yaghoobzadeh, Y., Kann, K., Schütze, H.: Evaluating word embeddings in multi-label classification using fine-grained name typing. In: The 3rd Workshop on Representation Learning for NLP (RepL4NLP), Melbourne, Australia, pp. 101–106 (2018)
14. Nooralahzadeh, F., Øvrelid, L., Lønning, J.T.: Evaluation of domain-specific word embeddings using knowledge resources, LREC (2018)
15. Yildiz, B., Fox, G.C.: Toward a modular and efficient distribution for web service handlers. Concurr. Comput.: Pract. Exp. **25**(3), 410–426 (2013)
16. Yildiz, B., Fox, G., Pallickara, S.: An orchestration for distributed web service handlers. In: 3th International Conference on Internet and Web Applications and Services, pp. 638–643 (2008)
17. Aktas, M.S., Kaplan, S., Abacı, H., Kalipsiz, O., Ketenci, U., Turgut, U.O.: Data imputation methods for missing values in the context of clustering. In: Big Data and Knowledge Sharing in Virtual Organizations (IGI Global, 2019), pp. 240–274 (2019)
18. Rehurek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks (2010)
19. Lison, P., Kutuzov, A.: Redefining context windows for word embedding models: an experimental study. arXiv preprint arXiv:1704.05781 (2017)
20. Goldberg, Y.: A primer on neural network models for natural language processing. J. Artif. Intell. Res. **57**, 345–420 (2016)