Evaluating Word Similarity Measure of Embeddings Through Binary Classification

A. Aziz Altowayan and Lixin Tao
Pace University
Computer Science Department
New York, USA
{aa10212w, ltao}@pace.edu

Abstract

We consider the following problem: given neural language models (embeddings) each of which is trained on an unknown data set, how can we determine which model would provide a better result when used for feature representation in a downstream task such as text classification or entity recognition? In this paper, we assess the word similarity measure through analyzing its impact on word embeddings learned from various datasets and how they perform in a simple classification task. Word representations were learned and assessed under the same conditions. For training word vectors, we used the implementation of Continuous Bag of Words described in [1]. To assess the quality of the vectors, we applied the analogy questions test for word similarity described in the same paper. Further, to measure the retrieval rate of an embedding model, we introduced a new metric (Average Retrieval Error) which measures the percentage of missing words in the model. We observe that scoring a high accuracy of syntactic and semantic similarities between word pairs is not an indicator of better classification results. This observation can be justified by the fact that a domain-specific corpus contributes to the performance better than a general-purpose corpus. For reproducibility, we release our experiments scripts and results.[1]

1 Introduction

Language modeling is the crux of the problem in Natural Language Processing (NLP. Recently, neural language models have outperformed the traditional language model approaches such as *n*-gram. The superiority of the neural methods lies in their capability to overcome the curse of the dimensionality problem while, simultaneously, capturing different similarities between words [1].

Neural language models learn distributed representations for each word in the form of real-numbers-value vectors, which allows similar words to have similar vectors. Such sharing is an important characteristic that enables the learning model to treat related words similarly and, hence, gives the model the ability to generalize. These word representations are usually known simply as *Word Embeddings*.

Nowadays, word embedding is the standard approach for feature representation in many NLP tasks. Traditional feature representation methods, such as bag-of-words and term frequency–inverse document frequency (TFIDF), rely on hand-crafted feature extractor, and are time-consuming and domain-specific. Hence, embedding based techniques provide a better alternative for automating many tasks in language modeling and NLP.

Among these techniques context-predicting semantic vectors have distinctly proven their superiority to the count-based ones [2]. While count-based vectors are more about the frequency of the word, context-based vectors make more emphasis on the word and its context.

Popular word vector learning methods are introduced in [[3], [4], [5]] and have gained great attention since then. From these methods, learning continuous word embeddings using skip-gram and negative sampling is the most common approach for

[1]https://github.com/iamaziz/w2v-eval

building word vectors [6]. This method was introduced and described in [3].

However, since vectors training occurs in an unsupervised fashion, there is no accurate way to estimate the quality of the vector representations objectively. Several extrinsic and intrinsic evaluation methods have been discussed in [7]. Nonetheless, at the time of this writing, there is still no reliable method for comparing the quality of different embedding models. So, this is still an open question. Commonly, the quality can be assessed using the word similarity task, which is a test with a set of similarity analogy questions [3].

Nevertheless, with the current word similarity evaluation method, word similarity accuracy and having more vocabulary in the model do not result in better performance in the downstream task.

From experiments, we show that scoring well on word similarity measure questions does not imply better performance in the downstream task. Our findings are in line with the observations of [8]. Therefore, we observe that the accuracy of word similarity measure is not, necessarily, an indicator for the usefulness of the word embedding model. In this paper, we explain and justify this claim based on the observation of our experimentation results.

For instance, we show that the GoogleNews embedding model has the following two advantages over IMDB model. First, it scores better results in word similarity accuracy (74.26%) in comparison to IMDB's (23.71%); second, GoogleNews contains 3 million vocabulary words while IMDB contains around 19,000. Despite these advantages of GoogleNews, the classifiers' performance was worse with GoogleNews than with IMDB.

The rest of the paper is structured into the following parts: related work, our experiments, discussion, future work, and finally, the conclusion.

2 Related Work

We approached related work in the following manner: first, we investigated what it takes to build quality embedding models and which components to consider. We then analyzed similar work for evaluating word embeddings using extrinsic and intrinsic methods. We also reviewed the available current work on building domain-specific embeddings. And finally, we look into work that focuses on the syntactic and semantic similarities between words.

Training elements such as the model, the corpus, and the parameters have been analyzed in detail in [9]. They observed that the corpus domain is more important than its size. This explains our results where the smaller domain-specific corpus (IMDB) achieved better results than the much larger general-purpose corpus (GoogleNews).

We reviewed papers on evaluating word vectors' quality and model accuracies. Existing evaluation methods fall into two types: intrinsic and extrinsic evaluation. In the intrinsic evaluation, the goal is to directly assess the quality of word vectors in hopes that it will reflect on the performance of the downstream tasks. So, synthetic metrics are proposed to test the semantic and syntactic similarities between words.

For example, a pre-selected set of query terms is used to estimate words' relationships. Each query denotes two pairs of "analogically" similar words. For example, relation *big* to *bigger* in the same way as as *small* to *smaller* is called "syntactic similarity", while relating *Tokyo* to *Japan* in the same way as *London* to *England* is called "semantic similarity"). Then, such queries can take the form of questions, for instance, "What is the word similar to *small* in the same sense as *bigger* is similar to *big*?". To query the model, a question is formulated in an algebraic expression as follows: *answer = vector('bigger') - vector('big') + vector('small').* This method was first proposed in [3]; and published with a set of around 20 thousand syntactic/semantic questions. It is fast and computationally inexpensive, however, there are problems associated with this technique [8]. Further, other evaluation techniques have been proposed to reduce bias [10]. In such methods, they directly compare embeddings with respect to specific queries.

While in the extrinsic evaluation, we indirectly evaluate word embeddings. In other words, we use the embeddings as input features to a downstream task and measure the performance metrics specified to that task [10]. For instance, when the task is text classification, we would use the embeddings to represent words in the text. In some approaches, they applied extrinsic evaluations to learn task-specific embeddings [11].

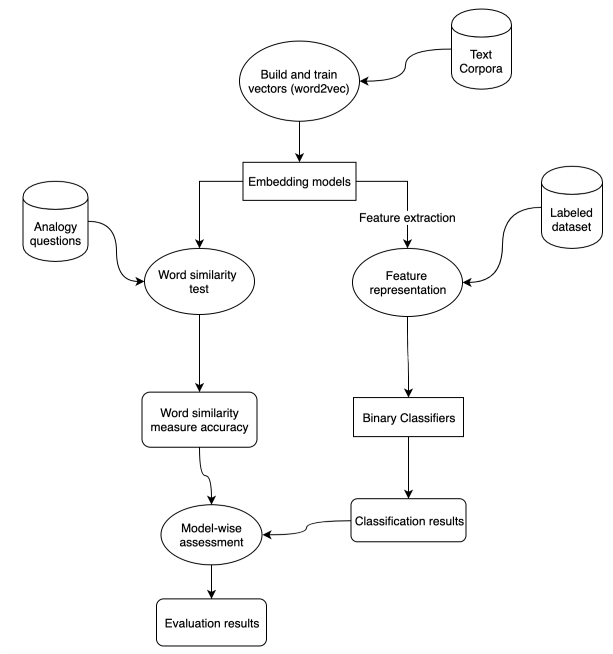

Figure 1: The approach design and workflow

Finally, a thorough investigation and survey covering the current evaluation methods have been discussed in [7].

3 Building Word Embeddings

3.1 Data Collection and Exploration

The diagram in fig. 1 illustrates the overall workflow of our experiment.

In this section, we describe the data sources and texts we used for training the embedding models. We started with two well-known corpora. The first one is text8, a standard corpus[2] used in NLP community which has around 100MB of cleaned English text of a Wikipedia dump from 2006, and the second one is the Large Movie Review Dataset (or IMDB[3]). IMDB contains 100 thousand movie reviews prepared for sentiment classification problems. Later on, we will use this same dataset in our classification experiment; we are aware this may cause bias in the datasets, further discussion to follow later.

As a way to augment our data, we created a new hybrid corpus by concatenating the above two corpora; we call it *text8-imdb*. This allows us to compare the results of two models and their hybrid to

[2]http://mattmahoney.net/dc/textdata

[3]http://ai.stanford.edu/ amaas/data/sentiment/

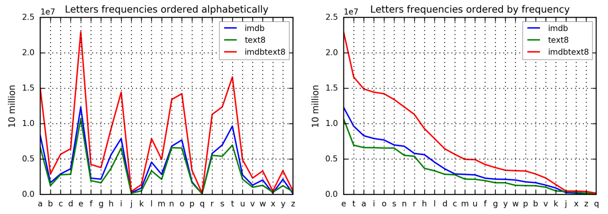

Figure 2: Letters frequencies as they appear in text8 and IMDB

see how they may affect one another. Later on, in the classification section, we will see that *imdb* achieved the best among the three. This is a bit surprising, because its *average retrieval error* (1.46) was higher than that of *text8-imdb* (0.99); though it still achieved better results.

For additional insights about the data, we explored each corpus for statistical information "metadata" such as number of the unique words, the total count of characters, and the total count of words. See table 1 for more details on these metrics.

We also wanted to get a better sense of the characters' usage and their frequency in each corpus. Figure 2 illustrates some visualization of the usages. It shows the frequency of the 26 English letters usage in each of the three corpora.

3.2 Model Training and Parameters

Following [3] approach for learning vector representations of words, we trained three models using three various corpora. In the first one, we merged the entire set of 100,000 movie reviews [12] into one big text file, we will refer to the vectors "model" generated from this text as **imdb**. And for the second model, as mentioned in the previous section, we used a 100 MB of cleaned Wikipedia English text known as *text8*, we will call the model from this corpus: **text8**. The third "hybrid" model is the combination of the two above files (as one big text file). We refer to this model as **imdb-text8**. The fourth model, in our experiment, is **GoogleNews**. A pretrained model published in [3].

With the exception of GoogleNews[4], all the models were trained using CBOW architecture with the same hyper-parameters. We used the original (C language) implementation of word2vec toolkit[5]. After

[4]It was not clear to us which exact parameters were used. See: Mikolov's response in word2vec-gmail-group

[5]https://github.com/tmikolov/word2vec

Corpus	char count	word count	unique words
imdb	125,882,839	23,573,192	144,841
text8	100,000,000	17,005,207	253,854

Table 1: Statistics of the training text (corpus).

compiling and building the software locally, we use the following command to train the models:

```
$ ./word2vec -train $CORPUS \
             -output $OUT \
             -cbow 1 \
             -size 300 \
             -window 10 \
             -negative 25 \
             -hs 0 \
             -sample 1e-4 \
             -threads 20 \
             -binary 1 \
             -iter 15
```

3.3 Exploring Models

After we built the models, we decided to evaluate their response to the analogy question test sets. Table 2 displays the number of the learned "vectorized" vocabulary in each model. The table also shows the number of questions seen in the model, along with their average similarity accuracy. These results were obtained based on `$ ./word2vec/compute-accuracy` script in the same toolkit. For faster approximate evaluation, we used the recommended threshold of 30,000 to reduce vocabulary.

3.4 Determining Models Accuracy

To conduct a fair comparison between models, we introduce the *Average Retrieval Error* "AVG_ERR" as a way to estimate the vectors' availability in the given model. It is the total number of *missed words* (i.e. words that queried but not available in the embedding model) over the *total words* queried. See formula 1 below:

$$\frac{\sum_{i=1}^{n} Q(t_i - r_i)}{n} \tag{1}$$

Where, Q is a query to the model which returns the vectors for a set of given tokens (words), n is the total number of the queries made, t is the number of tokens in query i, and r is the number of retrieved (found) vectors for query i.

For simplicity, we can rewrite 1 as:

$$Avg.\ Retrieval\ Error = \frac{\sum_{i=1}^{n} m_i}{n} \tag{2}$$

And m is the number of missed (not found) vectors for query i.

In figure 3, we show the number and percentage of analogy questions seen in the model (with a threshold of 30K) for word similarity task.

We also recorded the accuracy for each topic of the 14 question type categories. Instead of using a huge table with many numbers, we decided to illustrate the result in figure 4 to quickly grasp the topics' results.

Finally, in figure 5, we show the overall accuracy results for every model; such as the average score for all topics and density of topics' results.

Despite the scored word similarity accuracy of the IMDB model, its classification result is quite impressive. We will see that in the next section; where the learned word representations reflect a great deal of the actual semantics.

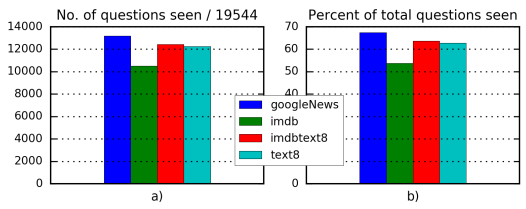

Figure 3: Embeddings results on the word analogy task (out of the total 19544 questions), fig. a. is the number of questions seen and fig. b. is the percentage of the questions seen.

4 Applying Embedding Models for Binary Classification

In this section we evaluate the performance of each embedding model in a downstream task. Our task is a simple binary classification for sentiment analysis problem.

Embeddings	# vocab.	dim.	# quest. seen	avg. sim. acc.
imdb	53,195	300	10,505	33.41%
text8	71,291	300	12,268	53.60%
imdb-text8	94,158	300	12,448	59.89%
GoogleNews	**3M**	300	**13,190**	**76.85%**

Table 2: Statistics of the models' and their results on the analogy questions test.

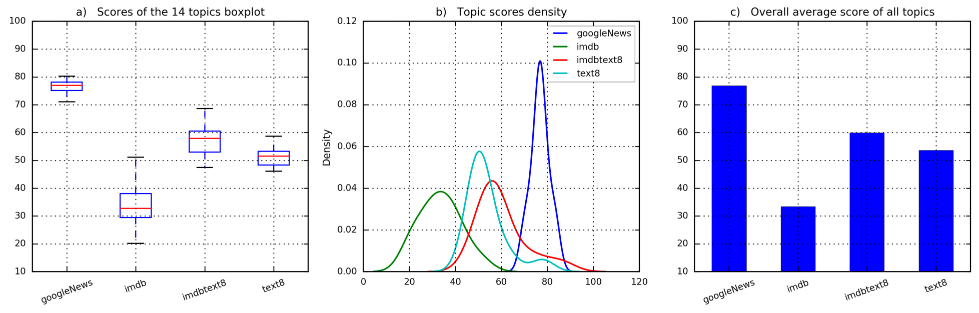

Figure 5: Visualizations of the overall models' performance in the analogy questions test

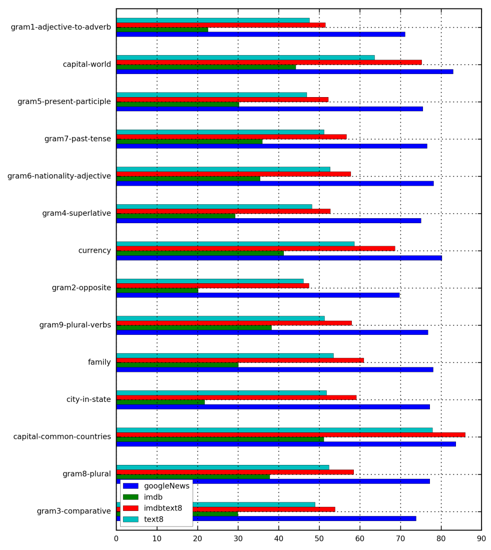

Figure 4: Topic-wise results of the models' accuracy on word analogy task

4.1 Supervised Training Dataset

To train the sentiment classifiers, we used the popular benchmark IMDB-50K movie reviews dataset. It was introduced by [12], and available to download.[6] The dataset, which was prepared specially for binary sentiment classification, contains 25K highly polar movie reviews for training and 25K for testing. The sentiment of reviews is balanced in both data sets, i.e. one half is positive, and the other half is negative.

Additionally, IMDB has another unlabeled dataset contains 50K reviews which we used in training our word2vec models. This dataset, however, was not used for training the binary classifiers.

4.2 Representing Reviews

After preprocessing the review text, the vector representation of each token "word" is then retrieved by querying the embedding model. If a token is not found in the embeddings' vocabulary, its representation will be ignored. That's where the concept of *Average Retrieval Error* comes from. The more tokens missed, the higher the average error will be. When all the review's tokens are processed, the review then will be represented as a fixed size feature vector by averaging the representations of all tokens.

[6] `http://ai.stanford.edu/~amaas/data/sentiment/`

4.3 Training Classifiers Results

We trained five simple binary classification algorithms Perceptron, Support Vector Machines, Stochastic Gradient Descent, Logistic Regression, and Random Forest. We used the built-in implementations of these algorithms provided by the scientific toolkit library "scikit-learn"[7]. As for parameters tuning, we applied the default parameters in scikit-learn.

To know the complete set of parameters for each classifier, one can refer to the log file we included with our project code.

In table 3, we show the performance of each classifier with each of the respective four embedding models. For a better visual comparison of the same scores, see fig. 6. We can see that the classifiers scored better with IMDB embedding model, despite that GoogleNews model has better accuracy in term of analogy query test. We can also notice that IMDB is still better than its hybrid model text8-imdb which intuitively should enrich the model's representation capacity by adding more vocabulary (which can be verified by inspected the average retrieval error decrease from imdb to text8-imdb). Reducing AVG_ERR did not improve the classifiers; but on the contrary, combining text8 degrades imdb's performance.

Avoiding bias in IMDB. The training and testing datasets are initially the same corpus that we use to generate imdb embeddings. Thus, and to make sure that our testing is not biased, we used another sentiment dataset (i.e. other than IMDB reviews) to test the performance of the classifier. The dataset contains 7086 labeled (positive/negative) training sentences and 33052 unlabeled sentences provided for prediction problems. We used the training data for testing our classifiers, as we were not able to acquire the actual labels of prediction set. As expected, the highest scores of the classifiers still achieved with imdb embeddings.

5 Results and Discussion

Results summary. To summarize and aggregate all the results and scores for both the intrinsic and extrinsic evaluations together in one place, we took the average score of all classifiers achieved with each

[7]https://scikit-learn.org

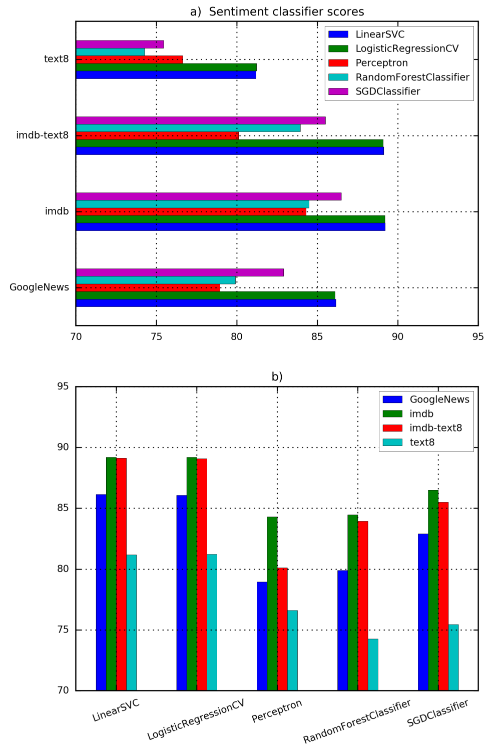

Figure 6: Sentiment classifiers score with each embeddings. a) embedding models wise results, and b) classifiers wise results.

embedding model. These aggregates are displayed in table 4.

Model Accuracy and Classifiers Performance. Why IMDB word embedding model is better than GoogleNews embedding? Learning task-specific vectors through fine-tuning offers further gain in performance. See static vs. non-static representation (section 4.2 of CNN sentence classification [13]).

So, for example, you'd expect words like "amazing" and "awful" to be very far apart whereas in word2vec they'd probably be closer because they can appear in similar contexts [8].

In the accuracy evaluation, IMDB model scored 22.94% on the 8182 test cases found (out of the 19544 test cases); while the GoogleNews model scored 74.26% on the 7614 test cases found. Although the IMDB model scored less, the sentiment classifiers performed better with it in comparison to the other model.

Improving classifiers' performance. Although we were not concerned with improving the over-

[8]see: cnn-text-classification-tf Denny Britz's blog

Model	Vocab.	AVG_ERR	Percept.	SVM	SGD	LogReg	RForest
imdb	53,195	1.46	**84.29%**	**89.20%**	**86.49%**	**89.19%**	**84.39%**
text8	71,291	4.62	76.62%	81.17%	75.44%	81.22%	73.88%
imdb-text8	94,158	**0.99**	80.11%	89.12%	85.50%	89.08%	83.96%
GoogleNews	**3,000,000**	28.04	78.94%	86.14%	82.89%	86.08%	80.16%

Table 3: Vocabulary Size, Average Retrieval Errors, and Classifiers Performance with each model.

Embeddings	vocab. size	AVG. retrieval err.	AVG. similarity acc.	AVG. sentiment score
imdb	53,195	1.46	33.41%	**86.73%**
text8	71,291	4.62	53.60%	77.74%
imdb-text8	94,158	**0.99**	59.89%	85.55%
GoogleNews	**3M**	28.04	**76.85%**	82.79%

Table 4: Summary on the final results for embedding models' accuracy and classification performance

all performance of the classifiers, there are several things to consider that can improve the classifiers' results.

For example, one can apply the ensemble approach, described in [14], that combine multiple baseline models rather than relying on a single model. Further improvement might be introduced by describing the review feature differently, instead of averaging the vectors [15].

Also, while training the vectors, careful choice and tuning of the hyper-parameters could bring much gain to the model accuracy [16]. Finally, one may consider words dependency instead of relying solely on linear contexts [17].

Missing data. When a given token (of a sentence) is not available in the embedding model, its vector value is ignored. However, it is counted toward the sentence length when we take the overall average. Can we do something else about this? e.g. 1) substitute (compute) its value as the average of other tokens in the same review, or 2) do not count it in review length, or 3) apply other known techniques for handling NaN values.

Average retrieval error. After comparing the models' sensitivity to the average retrieval error, we noticed that word retrieval in a model does not affect the overall performance. Possibly, one way to enrich this metric is by introducing word-wise weights. For example, common words can have low weight while the less common ones can have higher weight.

Extending this work. We can think of three possible ways to further extend this work. Firstly, expand the models range for broader comparison. For instance, one can integrate more (other) pre-trained models such as GloVe, ELMo, BERT to use in both experiments; embedding quality assessment, and binary classifiers. Secondly, and to enrich the procedure of classification comparison, one can try another approach to aggregating the sentence features (other than averaging vectors for sentence representations). Finally, in this work, we introduced the *Average Retrieval Error* "AVG_ERR". We think this measure can be further improved by adding weights to words in the sentences. For example, stop words, and common vocabulary can have less weight than those that are more specific.

6 Conclusion

We discussed the problem of choosing between multiple word embedding models. To this end, we made the following contributions. We built and trained three different embeddings models based on published data sets. We, then, implemented two types of evaluation methods on the models. For the intrinsic evaluation, we applied the word similarity measure method; while we did the extrinsic evaluations through a binary classification problem. We presented the results of performance comparisons over four different embedding models. We also introduced a metric for measuring the model's retrieval rate to the number of queries made. For reproducibility, we released the models, data, and scripts used in our experiments.

We have shown that scoring high accuracy in the Word Similarity Measure test does not imply better performance in the downstream task. In other words, if a model *A* achieves a higher score than model *B* in the analogy question test, this does not mean *A* will perform better than *B* in a downstream task. This finding is in line with observations from related work. We also observed that the model's coverage of vocabulary (i.e. vocabulary size) is not as essential as containing a domain-specific dictionary.

References

[1] Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv.org*, January 2013.

[4] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *EMNLP*, pages 1532–1543, 2014.

[5] Christopher D Manning. Computational linguistics and deep learning. *COLING*, 41(4): 701–707, 2015.

[6] Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, 2015.

[7] Amir Bakarov. A survey of word embeddings evaluation methods. *arXiv preprint arXiv:1801.09536*, 2018.

[8] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. *arXiv preprint arXiv:1605.02276*, 2016.

[9] Siwei Lai, Kang Liu, Shizhu He, and Jun Zhao. How to generate a good word embedding. *IEEE Intelligent Systems*, 31(6):5–14, 2016.

[10] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307, 2015.

[11] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, 2014.

[12] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[13] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[14] Grégoire Mesnil, Tomas Mikolov, Marc'Aurelio Ranzato, and Yoshua Bengio. Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *AAAI Spring Symposium AI Technologies for Homeland Security 200591-98*, cs.CL, 2014.

[15] Rémi Lebret and Ronan Collobert. " the sum of its parts": Joint learning of word and phrase representations with autoencoders. *arXiv preprint arXiv:1506.05703*, 2015.

[16] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3(0):211–225, May 2015.

[17] Edward Grefenstette, Phil Blunsom, Nando de Freitas, and Karl Moritz Hermann. A

deep architecture for semantic parsing. *arXiv preprint arXiv:1404.7296*, 2014.