Sentiment Classification using Document Embeddings trained with Cosine Similarity

Tan Thongtan
Department of Computer Engineering
Mahidol University International College
Mahidol University
Thailand
tan.thongtan1@gmail.com

Tanasanee Phienthrakul
Department of Computer Engineering
Faculty of Engineering
Mahidol University
Thailand
tanasanee.phi@mahidol.ac.th

Abstract

In document-level sentiment classification, each document must be mapped to a fixed length vector. Document embedding models map each document to a dense, low-dimensional vector in continuous vector space. This paper proposes training document embeddings using cosine similarity instead of dot product. Experiments on the IMDB dataset show that accuracy is improved when using cosine similarity compared to using dot product, while using feature combination with Naïve Bayes weighted bag of n-grams achieves a new state of the art accuracy of 97.42%. Code to reproduce all experiments is available at https://github.com/tanthongtan/dv-cosine.

1 Introduction

In document classification tasks such as sentiment classification (in this paper we focus on binary sentiment classification of long movie reviews, i.e. determining whether each review is positive or negative), the choice of document representation is usually more important than the choice of classifier. The task of text representation aims at mapping variable length texts into fixed length vectors, as to be valid inputs to a classifier. Document embedding models achieve this by mapping each document to a dense, real-valued vector.

This paper aims to improve existing document embedding models (Le and Mikolov, 2014; Li et al., 2016a) by training document embeddings using cosine similarity instead of dot product. For example, in the basic model of trying to predict - given a document - the words/n-grams in the document, instead of trying to maximize the dot product between a document vector and vectors of the words/n-grams in the document over the training set, we'll be trying to maximize the cosine similarity instead.

The motivation behind this is twofold. Firstly, cosine similarity serves as a regularization mechanism; by ignoring vector magnitudes, there is less incentive to increase the magnitudes of the input and output vectors, whereas in the case of dot product, vectors of frequent document-n-gram pairs can be made to have a high dot product simply by increasing the magnitudes of each vector. The weights learned should be smaller overall.

Secondly, as cosine similarity is widely used to measure document similarity (Singhal, 2001; Dai et al., 2015), we believe our method should more directly maximize the cosine similarity between similar document vectors. The angle between similar documents should be lower, and that may encode useful information for distinguishing between different types of documents. We'll compare the performance of our model on the IMDB dataset (Maas et al., 2011) with dot product and to determine if our model serves anything beyond simple regularization, we'll also compare it to dot product using L2 regularization.

2 Related Work

Here we review methods of text representation, in which there are two main categories: **bag of words models** and **neural embedding models**.

The **bag of words** model (Joachims, 1998) represents text as a fixed length vector of length equal to the number of distinct n-grams in the vocabulary. **Naive Bayes - Support Vector Machine (NB-SVM)** (Wang and Manning, 2012) utilizes naïve bayes weighted bag of n-grams vectors for representing texts, feeding these vectors into a logistic regression or support vector machine classifier.

The first example of a **neural embedding model** is **word embeddings** which was proposed by Bengio et al. in 2003, while objective functions utilizing the negative sampling technique for efficient training of word embeddings were proposed in 2013 by Mikolov et al. (word2vec). The aim of word embeddings is to map each word to a real vector, whereby the dot product between two vectors represents the amount of similarity in meaning between the words they represent. There are two versions of word2vec: continuous bag of words (CBOW), in which a neural network is trained to predict the next word in a piece of text given the word's context, and skip-gram, where it will try to predict a word's context given the word itself.

In a 2017 paper Arora et al. produce **Sentence Embeddings** by computing the weighted average of word vectors, where each word is weighted using smooth inverse frequency, and removing the first principle component.

Paragraph Vector (Le and Mikolov, 2014) may be seen as a modification to word embeddings in order to embed as vectors paragraphs as opposed to words. Paragraph vector comes in two flavors: the Distributed Memory Model of Paragraph Vectors (PV-DM), and the Distributed Bag of Words version of Paragraph Vector (PV-DBOW). PV-DM is basically the same as CBOW except that a paragraph vector is additionally averaged or concatenated along with the context and that whole thing is used to predict the next word. In the PV-DBOW model a paragraph vector alone is used/trained to predict the words in the paragraph.

Document Vector by predicting n-grams (DV-ngram) (Li et al., 2016a) trains paragraph vectors to predict not only the words in the paragraph, but n-grams in the paragraph as well. **Weighted Neural Bag of n-grams (W-Neural-BON)** (Li et al., 2016b) uses an objective function similar to the one in DV-ngram, except that each log probability term is weighted using a weighing scheme which is similar to taking the absolute values of naive bayes weights.

In (Li et al., 2017), they introduce three main methods of **embedding n-grams**. The first is context guided n-gram representation (CGNR), which is training n-gram vectors to predict its context n-grams. The second is label guided n-gram representation (LGNR), which is predicting given an n-gram the label of the document to which it belongs. The last is text guided n-gram representation (TGNR), which is predicting given an n-gram the document to which it belongs.

Embeddings from Language Models (ELMo) (Peters et al., 2018) learns contextualized word embeddings by training a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) on the language modelling task of predicting the next word as well as the previous word. **Bidirectional Encoder Representations from Transformers (BERT)** (Devlin et al., 2018) uses the masked language model objective, which is predicting the masked word given the left and right context, in order to pre-train a multi-layer bidirectional Transformer (Vaswani et al., 2017). BERT also jointly pre-trains text-pair representations by using a next sentence prediction objective.

For the rest of this section we'll look at other research which replaces dot product with cosine similarity. In the context of fully-connected neural networks and convolutional neural networks, (Luo et al., 2017) uses cosine similarity instead of dot product in computing a layer's pre-activation as a regularization mechanism. Using a special dataset where each instance is a paraphrase pair, (Wieting et al., 2015) trains word vectors in such a way that the cosine similarity between the resultant document vectors of a paraphrase pair is directly maximized.

3 Proposed Model

In learning neural n-gram and document embeddings, a dot product between the input vector and the output vector is generally used to compute the similarity measure between the two vectors, i.e. 'similar' vectors should have a high dot product. In this paper we explore using cosine similarity instead of dot product in computing the similarity measure between the input and output vectors. More specifically we focus on modifications to the PV-DBOW and the similar DV-ngram models. The cosine similarity between a paragraph vector and vectors of n-grams in the paragraph is maximized over the training set.

3.1 Architecture

A neural network is trained to be useful in predicting, given a document, the words and n-grams in the document, in the process of doing so learning useful document embeddings. Formally, the objective function to be minimized is defined as follows:

$$\sum_{d \in D} \sum_{w_o \in W_d} -\log p(w_o|d) \tag{1}$$

where d is a document, D is the set of all documents in the dataset, w_o is an n-gram and W_d is the set of all n-grams in the document d. $p(w_o|d)$ is defined using softmax:

$$p(w_o|d) = \frac{e^{\alpha \cos \theta_{w_o}}}{\sum_{w \in W} e^{\alpha \cos \theta_w}} \tag{2}$$
$$= \mathrm{softmax}(\alpha \cos \theta_{w_o}) \tag{3}$$

We have $\cos \theta_w$ defined as follows:

$$\cos \theta_w = \frac{\boldsymbol{v}_d^T \boldsymbol{v}_w}{\|\boldsymbol{v}_d\| \|\boldsymbol{v}_w\|} \tag{4}$$

where $\boldsymbol{v}_d$ and $\boldsymbol{v}_w$ are vector representations of the document d and the word/n-gram w respectively and are parameters to be learned. α is a hyperparameter. W is the set of all n-grams in the vocabulary.

Normally, the softmax of the dot product between the input and output vector is used to model the conditional probability term as follows:

$$p(w_o|d) = \frac{e^{\boldsymbol{v}_d^T \boldsymbol{v}_{w_o}}}{\sum_{w \in W} e^{\boldsymbol{v}_d^T \boldsymbol{v}_w}} \tag{5}$$

Whereas dot product ranges from negative infinity to positive infinity, since cosine similarity ranges from -1 to 1, using the cosine similarity term alone as an input to the softmax function may not be sufficient in modeling the conditional probability distribution. Therefore, we add a scaling hyperparameter α to increase the range of possible probability values for each conditional probability term.

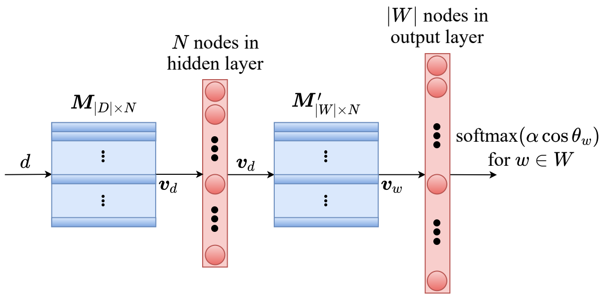
$|W|$ nodes in output layer
N nodes in hidden layer
$\boldsymbol{M}_{|D| \times N}$
$\boldsymbol{M}'_{|W| \times N}$
d
$\boldsymbol{v}_d$
$\boldsymbol{v}_d$
$\boldsymbol{v}_w$
softmax$(\alpha \cos \theta_w)$ for $w \in W$

Figure 1: Proposed Architecture.

Figure 1 shows the architecture of the neural network used in learning the document embeddings. There is a hidden layer with N nodes corresponding to the dimensionality of the paragraph vectors and an output layer with $|W|$ nodes corresponding to the number of distinct n-grams found in the dataset. There are two weight parameter matrices to be learned: $\boldsymbol{M}$, which may be seen as a collection of $|D|$ document vectors each having N dimensions, and $\boldsymbol{M'}$, which is a collection of $|W|$ n-gram vectors each also having N dimensions.

An input document id d is used to select its vector representation $\boldsymbol{v}_d$ which is exactly output through the N nodes of the first hidden layer. The output of each node in the output layer represents the probability $p(w|d)$ of its corresponding n-gram w, and is calculated as in (2) using softmax.

3.2 Negative Sampling

Since the weight update equations for minimizing (1) implies that we must update each output vector corresponding to each feature in the feature set W, with extremely large vocabularies, this computation is impractical. In (Mikolov et al., 2013), the negative sampling technique is introduced as a means to speed up the learning process and it can be shown that the updates for the negative sampling version of (1) as shown in (6) approximates the weight updates carried out in minimizing (1). Therefore in practice, the document embeddings are obtained by minimizing the following objective function with stochastic gradient descent and backpropagation (Rumelhart et al., 1986):

$$\sum_{d \in D} \sum_{w_o \in W_d} \Big[-\log \sigma \left(\alpha \cos \theta_{w_o}\right) - \sum_{w_n \in W_{neg}} \log \sigma \left(-\alpha \cos \theta_{w_n}\right) \Big] \tag{6}$$

where W_{neg} is a set of negatively sampled words; the size of the set or the negative sampling size as well as the distribution used to draw negatively sampled words/n-grams are hyperparameters. σ is the sigmoid function.

By contrast, in the case of dot product the objective function is:

$$\sum_{d \in D} \sum_{w_o \in W_d} \Big[-\log \sigma \left(\boldsymbol{v}_d^T \boldsymbol{v}_{w_o}\right) - \sum_{w_n \in W_{neg}} \log \sigma \left(-\boldsymbol{v}_d^T \boldsymbol{v}_{w_n}\right) \Big] \tag{7}$$

while in the case of L2R dot product, the objective function used is:

$$\sum_{d \in D} \sum_{w_o \in W_d} \Big[-\log \sigma \left(\boldsymbol{v}_d^T \boldsymbol{v}_{w_o}\right) + \frac{\lambda}{2} \|\boldsymbol{v}_d\|^2 + \frac{\lambda}{2} \|\boldsymbol{v}_{w_o}\|^2 - \sum_{w_n \in W_{neg}} \left(\log \sigma \left(-\boldsymbol{v}_d^T \boldsymbol{v}_{w_n}\right) + \frac{\lambda}{2} \|\boldsymbol{v}_{w_n}\|^2 \right) \Big] \tag{8}$$

Features	Dot Product (DV-ngram) (%)	Dot Product with L2R (%)	Cosine Similarity (%)
Unigrams	89.60	87.15 (-2.45)	**90.75 (+1.15)**
Unigrams+Bigrams	91.27	91.72 (+0.45)	**92.56 (+1.29)**
Unigrams+Bigrams+Trigrams	92.14	92.45 (+0.31)	**93.13 (+0.99)**

Table 1: Experimental Results.

where λ is the regularization strength.

4 Experiments

The models are benchmarked on the IMDB dataset (Maas et al., 2011), which contains 25,000 training documents, 25,000 test documents, and 50,000 unlabeled documents. The IMDB dataset is a binary sentiment classification dataset consisting of movie reviews retrieved from IMDB; training documents in the dataset are highly polar. For labeled documents, there is a 1:1 ratio between negative and positive documents. The document vectors are learned using all the documents in the dataset (train, test and unlabeled documents). The dataset consists of mainly long movie reviews.

In order to train the document vectors on unigrams to trigrams, the reviews are preprocessed in such a way that tokens representing bigrams and trigrams are simply appended to the original unigrams of the review itself. An L2-regularized logistic regression (LR) classifier is used to classify the documents at the end of each epoch using the predefined train-test split. However, the results reported in this paper include only the accuracy obtained from classifying documents in the final epoch. For any java implementations of the LR classifier we use the LIBLINEAR library (Fan et al., 2008) while for python implementations we use Sci-kit learn (Pedregosa et al., 2011). Code to reproduce all experiments is available at https://github.com/tanthongtan/dv-cosine.

4.1 Optimal Hyperparameters

Grid search was performed using 20% of the training data as a validation set in order to determine the optimal hyperparameters as well as whether to use a constant learning rate or learning rate annealing. Table 2 shows the optimal hyperparameters for the models on the IMDB dataset. We did not tune the N hyperparameter or the negative sampling size and left it the same as in (Li et al., 2016a) and (Lau and Baldwin, 2016). We did however tune the number of iterations from [10, 20, 40, 80, 120], learning rate from [0.25, 0.025, 0.0025, 0.001] and α from [4, 6, 8]. A sensible value of α should be around 6, since looking at the graph of the sigmoid function, for input values greater than 6 and less than -6, the sigmoid function doesn't change much and has values of close to 1 and 0, respectively. In the case of using L2 regularized dot product, λ (regularization strength) was chosen from [1, 0.1, 0.01].

Hyperparameter	Dot Prod.	L2R Dot Prod.	Cos. Sim.
N (dimensionality)	500	500	500
Neg. Sampling Size	5	5	5
Iterations	10	20	120
Learning Rate	0.25	0.025	0.001
α	-	-	6
λ	-	0.01	-
LR annealing	true	false	false

Table 2: Optimal Hyperparameters.

The optimal learning rate in the case of cosine similarity is extremely small, suggesting a chaotic error surface. Since the learning rate is already small to begin with, no learning rate annealing is used. The model in turn requires a larger number of epochs for convergence. For the distribution for sampling negative words, we used the n-gram distribution raised to the 3/4th power in accordance with (Mikolov et al., 2013). The weights of the networks were initialized from a uniform distribution in the range of [-0.001, 0.001].

4.2 Results

Each experiment was carried out 5 times and the mean accuracy is reported in Table 1. This is to account for random factors such as shuffling document and word ids, and random initialization. From here we see that using cosine similarity instead of dot product improves accuracy across the board. The results are most apparent in the case of unigrams + bigrams. However it is not to suggest that switching from dot product to cosine similarity alone improves accuracy as other minor ad-

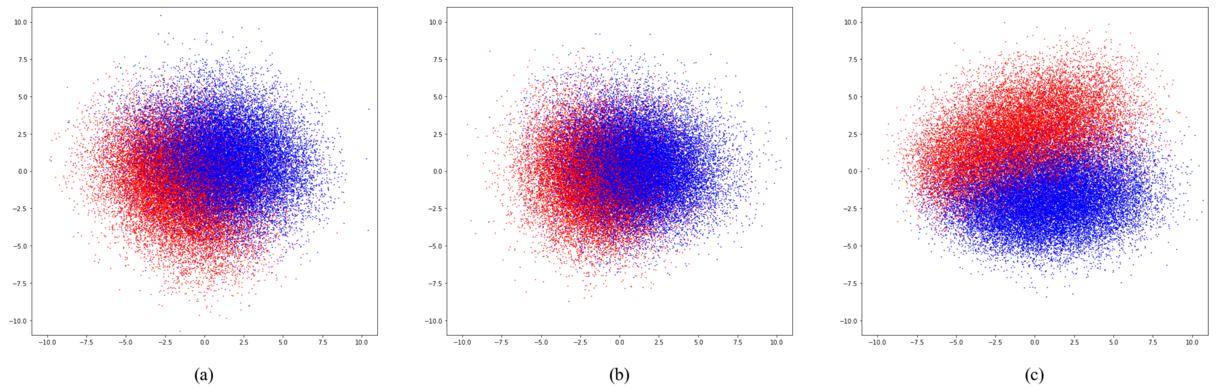

Figure 2: PCA visualization of embeddings trained with (a) dot product, (b) L2R dot product and (c) cos. similarity.

justments and hyperparameter tuning as explained was done. However it may imply that using cosine similarity allows for a higher potential accuracy as was achieved in these experiments.

Regardless, we believe the comparisons are fair since each model is using its own set of optimal hyperparameters, but for the full sake of comparison, leaving everything the same and only switching out dot product for cosine similarity ($\alpha = 1$) as well as switching it out and using a sensible value of α at $\alpha = 6$ both achieve an accuracy of around 50%. This is because our model fails whenever the learning rate is too large. As seen during grid search, whenever the initial learning rate was 0.25, accuracy was always poor.

Introducing L2 regularization to dot product improves accuracy for all cases except a depreciation in the case of using unigrams only, lucikily cosine similarity does not suffer from this same depreciation.

4.3 Discussion

From table 3, the mean Euclidean norm of embeddings trained with cosine similarity is lower than that of L2R dot product which is in turn lower than in the case of using dot product; this suggests that the method employing cosine similarity acts as a regularization mechanism, preventing the weights from getting too large. Large magnitudes of document vectors may be harder for the end classifier to fit in such a way that generalizes well, which may be why cosine similarity and L2R dot product perform better than dot product on the IMDB dataset.

As predicted, the mean cosine similarity between all pairs of vectors in the same class (Same Mean Cos. Sim.) is higher in the case of cosine similarity than the other two models. Unfortunately, the mean for all pairs in different classes (Diff. Mean Cos. Sim.) is higher as well. Further analysis and hopefully some formalism as to why cosine similarity performs better is a planned future work.

Embedding Statistic	Dot Prod.	L2R Dot Prod.	Cos. Sim.
Same Mean Cos. Sim.	0.23	0.20	0.35
Diff. Mean Cos. Sim.	0.21	0.17	0.32
Mean Norm	8.91	6.30	5.35

Table 3: Embedding statistics.

Figure 2 shows the projection of the embeddings along their first two principle components, different colors corresponding to different classes. Cosine similarity shows slightly better seperability between the two classes, while dot product and L2R dot product are quite similar.

4.4 Feature Combination

Another contribution of this paper is demonstrating the effectiveness of concatenating naive bayes weighted bag of n-grams with DV-ngram, L2R dot product, or document vectors trained with cosine similarity, the last achieving state of the art accuracy on the IMDB dataset. We note that all models utilize unigrams to trigrams and additional unlabeled data if possible. Table 4 shows a comparison between our proposed models (shown in bold) and previous state of the arts and other published results.

5 Conclusion and Future Work

Our proposed model trains document embeddings using cosine similarity as the similarity measure and we show that sentiment classification performance on the IMDB dataset is improved when

Model	IMDB Dataset Accuracy (%)
NB-SVM Bigrams (Wang and Manning, 2012)	91.22
NB-SVM Trigrams (Mesnil et al., 2015)	91.87
DV-ngram (Li et al., 2016a)	92.14
Dot Product with L2 Regularization	**92.45**
Paragraph Vector (Le and Mikolov, 2014)	92.58
Document Vectors using Cosine Similarity	**93.13**
W-Neural-BON Ensemble (Li et al., 2016b)	93.51
TGNR Ensemble (Li et al., 2017)	93.51
TopicRNN (Dieng et al., 2017)	93.76
One-hot bi-LSTM (Johnson and Zhang, 2016)	94.06
Virtual Adversarial (Miyato et al., 2016)	94.09
BERT large finetune UDA (Xie et al., 2019)	95.80
NB-weighted-BON + DV-ngram	**96.95**
NB-weighted-BON + L2R Dot Product	**97.17**
NB-weighted-BON + Cosine Similarity	**97.42**

Table 4: Comparison with other models.

utilizing these embeddings as opposed to those trained using dot-product. Cosine similarity may help reduce overfitting to the embedding task, and this regularization in turn produces more useful embeddings. We also show that concatenating these embeddings with Naïve bayes weighed bag of n-grams results in high accuracy on the IMDB dataset.

An important future development is to carry out experiments on other datasets. It is essential that we benchmark on more than one dataset, to prevent superficially good results by overfitting hyperparameters or the cosine similarity model itself to the IMDB dataset. Other tasks and datasets include: (1) sentiment analysis - the Stanford sentiment treebank dataset (Socher et al., 2013), the polarity dataset v2.0 (Pang and Lee, 2004), (2) topic classification - AthR, XGraph, BbCrypt (Wang and Manning, 2012), and (3) semantic relatedness tasks - datasets from the SemEval semantic textual similarity (STS) tasks (Agirre et al., 2015).

References

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, and Janyce Wiebe. 2015. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, pages 252–263.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the 5th International Conference on Learning Representations*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Andrew M. Dai, Christopher Olah, and Quoc V. Le. 2015. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Adji B. Dieng, Chong Wang, Jianfeng Gao, and John Paisley. 2017. Topicrnn: A recurrent neural network with long-range semantic dependency. In *Proceedings of the 5th International Conference on Learning Representations*.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9(Aug):1871–1874.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, pages 137–142.

Rie Johnson and Tong Zhang. 2016. Supervised and semi-supervised text categorization using lstm for region embeddings. In *Proceedings of the 4th International Conference on Learning Representations*.

Jey Han Lau and Timothy Baldwin. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1188–1196.

Bofang Li, Tao Liu, Xiaoyong Du, Deyuan Zhang, and Zhe Zhao. 2016a. Learning document embeddings by predicting n-grams for sentiment classification of long movie reviews. In *Proceedings of the 4th International Workshop on Learning Representations*.

Bofang Li, Tao Liu, Zhe Zhao, Puwei Wang, and Xiaoyong Du. 2017. Neural bag-of-ngrams. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 3067–3074.

Bofang Li, Zhe Zhao, Tao Liu, Puwei Wang, and Xiaoyong Du. 2016b. Weighted neural bag-of-n-grams model: New baselines for text classification. In *Proceedings of the 26th International Conference on Computational Linguistics*, pages 1591–1600.

Chunjie Luo, Jianfeng Zhan, Lei Wang, and Qiang Yang. 2017. Cosine normalization: Using cosine similarity instead of dot product in neural networks. *arXiv preprint arXiv:1702.05870*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 142–150.

Grégoire Mesnil, Tomas Mikolov, Marc'Aurelio Ranzato, and Yoshua Bengio. 2015. Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. In *Proceedings of the 3rd International Workshop on Learning Representations*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, pages 3111–3119.

Takeru Miyato, Andrew M. Dai, and Ian Goodfellow. 2016. Adversarial training methods for semi-supervised text classification. In *Proceedings of the 4th International Conference on Learning Representations*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, page 271.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2227–2237.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Amit Singhal. 2001. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5998–6008.

Sida Wang and Christopher D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 90–94.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. In *Proceedings of the 4th International Conference on Learning Representations*.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. 2019. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*.

A Obtaining the weight update equations in the case of cosine similarity

To obtain the weight update equations for the input and output vectors of our model in each iteration of stochastic gradient descent, we must find the gradient of the error function at a given training example, which may be considered a document, n-gram pair.

Let:

$$\begin{aligned} E = &- \log \sigma\left(\alpha \cos \theta_{w_o}\right) \\ &- \sum_{w_n \in W_{neg}} \log \sigma\left(-\alpha \cos \theta_{w_n}\right) \end{aligned} \tag{9}$$

where:

$$\cos \theta_w = \frac{\boldsymbol{v}_d^T \boldsymbol{v}_w}{\|\boldsymbol{v}_d\| \|\boldsymbol{v}_w\|} \tag{10}$$

be the objective function at a single training example (d, w_o). Then, to find the gradient of E first differentiate E with respect to $\cos\theta_w$:

$$\frac{\partial E}{\partial \cos\theta_w} = \alpha\left(\sigma\left(\alpha\cos\theta_w\right) - t\right) \tag{11}$$

where $t = 1$ if $w = w_o$; 0 otherwise. We then obtain the derivative of E w.r.t. the output n-gram vectors:

$$\frac{\partial E}{\partial \boldsymbol{v}_w} = \frac{\partial E}{\partial \cos\theta_w} \cdot \frac{\partial \cos\theta_w}{\partial \boldsymbol{v}_w} \tag{12}$$

$$\frac{\partial E}{\partial \boldsymbol{v}_w} = \alpha\left(\sigma\left(\alpha\cos\theta_w\right) - t\right) \cdot \left(\frac{\boldsymbol{v}_d}{\|\boldsymbol{v}_d\|\|\boldsymbol{v}_w\|} - \frac{\boldsymbol{v}_w\left(\boldsymbol{v}_d^T\boldsymbol{v}_w\right)}{\|\boldsymbol{v}_d\|\|\boldsymbol{v}_w\|^3}\right) \tag{13}$$

This leads to the following weight update equation for the output vectors:

$$\boldsymbol{v}_w^{(new)} = \boldsymbol{v}_w^{(old)} - \eta\frac{\partial E}{\partial \boldsymbol{v}_w} \tag{14}$$

where η is the learning rate. This equation needs to be applied to all $w \in \{w_o\} \cup W_{neg}$ in each iteration.

Next, the errors are backpropagated and the input document vectors are updated. Differentiating E with respect to $\boldsymbol{v}_d$:

$$\frac{\partial E}{\partial \boldsymbol{v}_d} = \sum_{w\in\{w_o\}\cup W_{neg}} \frac{\partial E}{\partial \cos\theta_w} \cdot \frac{\partial \cos\theta_w}{\partial \boldsymbol{v}_d} \tag{15}$$

$$= \sum_{w\in\{w_o\}\cup W_{neg}} \alpha\left(\sigma\left(\alpha\cos\theta_w\right) - t\right) \cdot \left(\frac{\boldsymbol{v}_w}{\|\boldsymbol{v}_d\|\|\boldsymbol{v}_w\|} - \frac{\boldsymbol{v}_d\left(\boldsymbol{v}_d^T\boldsymbol{v}_w\right)}{\|\boldsymbol{v}_d\|^3\|\boldsymbol{v}_w\|}\right) \tag{16}$$

Thus, we obtain the weight update equation for the input vector in each iteration:

$$\boldsymbol{v}_d^{(new)} = \boldsymbol{v}_d^{(old)} - \eta\frac{\partial E}{\partial \boldsymbol{v}_d} \tag{17}$$

B Weight update equations in the case of dot product

This section contains the weight update equations for the input and output vectors of the dot product model in each iteration of stochastic gradient descent.

The following weight update equations for the output vectors:

$$\boldsymbol{v}_w^{(new)} = \boldsymbol{v}_w^{(old)} - \eta\left(\sigma\left(\boldsymbol{v}_d^T\boldsymbol{v}_w\right) - t\right) \cdot \boldsymbol{v}_d \tag{18}$$

where $t = 1$ if $w = w_o$; 0 otherwise, needs to be applied to all $w \in \{w_o\} \cup W_{neg}$ in each iteration.

The following weight update equation needs to be applied to the input vector in each iteration:

$$\boldsymbol{v}_d^{(new)} = \boldsymbol{v}_d^{(old)} - \eta\sum_{w\in\{w_o\}\cup W_{neg}}\left(\sigma\left(\boldsymbol{v}_d^T\boldsymbol{v}_w\right) - t\right) \cdot \boldsymbol{v}_w \tag{19}$$

C Weight update equations in the case of L2R dot product

This section contains the weight update equations for the input and output vectors of the L2R dot product model in each iteration of stochastic gradient descent.

The following weight update equations for the output vectors:

$$\boldsymbol{v}_w^{(new)} = \boldsymbol{v}_w^{(old)} - \eta\left(\sigma\left(\boldsymbol{v}_d^T\boldsymbol{v}_w\right) - t\right) \cdot \boldsymbol{v}_d - \eta\lambda\boldsymbol{v}_w \tag{20}$$

where $t = 1$ if $w = w_o$; 0 otherwise, needs to be applied to all $w \in \{w_o\} \cup W_{neg}$ in each iteration.

The following weight update equation needs to be applied to the input vector in each iteration:

$$\boldsymbol{v}_d^{(new)} = \boldsymbol{v}_d^{(old)} - \eta\sum_{w\in\{w_o\}\cup W_{neg}}\left(\sigma\left(\boldsymbol{v}_d^T\boldsymbol{v}_w\right) - t\right) \cdot \boldsymbol{v}_w - \eta\lambda\boldsymbol{v}_d \tag{21}$$