Word Embeddings for Fine-Grained Sentiment Analysis

Dylan Bacon , Rajan Dalal , Mohan Rao Divate Kodandarama
Madan Raj Hari , Raghavan Vellore Muneeswaran

Department of Computer Sciences
University of Wisconsin-Madison

{dbacon, rdalal, divatekodand, mhari2, velloremunee}@wisc.edu

Abstract

In this paper, we define and explore the known notion of a word embedding - a way of assigning values to words in a vocabulary via value vectors. We explore different algorithms that word vectors can be used with and machine learning methods to apply them to. These include applying word embeddings to non-neural network learning models, and making modifications to a known sentiment analyzer that already utilizes word embeddings. We find that applying a proper word vector algorithm to sentiment trainers increases accuracy by a noticeable degree for many models. We attempt to integrate the end-result of sentiment analysis into embeddings by suggesting a new approach of vectorization - Cosine Loss Word Embeddings (CLWE), that takes into consideration antonyms.

1 Introduction

Sentiment analysis is a fast-growing field in machine learning, both in terms of techniques utilized in it and its importance to business and societal applications. It involves the ability to predict a human-derived emotion from a piece of text. The usages of such technology are far-ranging and have been applied in areas from stock markets (such as by Bloomberg L.P. [2]) to politics. Opinion mining of social media can provide the zeitgeist of a certain theme. Indeed, as shown by Pak and Paroubek [9], Twitter can be a very valuable source of information to mine for a variety of different topics. As social media is positioning to be a mover and shaker of social change, being able to scan through such a large body of words in a timely and efficient manner will be valuable. This is especially true when the volume of work to consume is greater than any reasonable number of humans can parse through and analyze.

There are different dimensions to a possible sentiment analyzer. Much of the foundation work in the field can be seen in the survey done by Pang and Lee [10]. This work includes both single and multi-document sentiment analyses as well as different approaches on how to obtain a sentiment, and on what granularity of sentiment to detect. Many traditional approaches only use a two-sentiment approach, so in essence they could be considered a binary classifier. Human opinion is often on a spectrum more meaningfully diverse than just two options, however. Simply defining documents as positive or negative is usually insufficient. At the very least, there should be included a neutral third option. To that end the field of fine-grained sentiment analysis has emerged. It seeks to classify opinions on meaningful quantities of text at a range greater than just two options.

In this paper, we seek to implement new methodologies to improve the accuracy of sentiment analysis systems. We focus on the concept of word embeddings, which we will define shortly, and apply those to both non-neural network based learning approaches and neural network based ideas. We also try to build upon existing vectorization schemes and see the efficacy of these updated vectors.

1.1 Word Vectors

Word vectors are a comparatively new and novel technique to reason about phrases longer than a single word. This approach to encoding originated with Milokov et al. [8] in work done at Google, culminating in the word2vec model. At its core, a word vector is a way to assign vector values to each word in a phrase. This can be done in various ways, such as assigning to each word in the vocabulary a vector where each position in the vector relates to a given trait that the word implies or has about a subject, or by calculating its relevance to other words in the corpus. We, following word2vec, use the latter approach. In this paper, we use the terms 'word vectors' and 'word embeddings' interchangeably.

1.2 Related Works

There have been previous attempts to implement word embeddings into commonly-used machine learning models and for sentiment analysis in particular. Another group, whose work we attempt to build upon in this paper, is the Stanford Natural Language Processing group. In particular, Socher et al. [14] built a sentiment treebank system that builds on prior work done in the field. With this system, they were able to create a tree-based sentiment analysis structure on a large body of work.

Outside of the Stanford project, sentiment analysis has also been examined by Zirn et al. [16] through the use of Markov chains. Additionally, Fink et al. [5] explored opinion mining, both fine and coarsely grained, in the realm of politics as well.

2 Antonym-enhancing cosine loss

Word vectors can perform surprisingly well in representing semantic and syntactic relationships between words. For example, word vectors learned from current state-of-the-art approaches (e.g. [8]) can answer syntactic questions such as - "What is the word that is similar to 'big' in the same sense as 'smaller' is to 'small'?". However, such vector representations only capture similarity of words as per their place in a corpus. In particular, they fail to capture the compositional semantics present in text. Consider the following two sentences:

The movie was excellent.

The movie was terrible.

The first sentence has a strong positive connotation, while the second sentence has a strong negative connotation. Current word vector representations represent antonyms such as 'excellent' and 'terrible' as very similar and place them close in the vector space. We hypothesize that a vectorization that takes into account these antonyms would be immensely beneficial for sentiment recognition systems relying on word vector representations.

To address this, we propose a novel word vector representation called the Cosine Loss Word Embeddings (CLWE), which tries to learn orthogonal vector representations for antonyms. More formally, the architecture of learning CLWE is similar to Continuous Skip-gram Model with negative sampling ([8]), but it additionally adds a cosine loss term for every pair of antonyms seen during the training phase. Continuous Skip-gram model tries to maximize the classification of a word based on another word in the same sentence. Given a sequence of training words $w_1, w_2, w_3, ..., w_T$, the objective of the Skip-gram model is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c<=j<=c, j\neq 0} log\, p(w_{t+j}|w_t) \tag{1}$$

where c is the size of the training context. The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{exp(v_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} exp(v_{w_O}^T v_{w_I})} \tag{2}$$

where v_{w_O} and v_{w_I} are input and output vector representations of w, and W is the number of words in the vocabulary. This formulation is impractical because of large cost of computing the gradient of the softmax. Hence an alternate approach called the Negative Sampling is often used. Negative Sampling Objective:

$$log\,\sigma(v_{w_O}^T v_{w_I}) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)}[log\,\sigma(v_{w_i}^T v_{w_I})] \tag{3}$$

In CLWE, in addition to maximizing the negative sampling objective, we try to minimize the cosine similarity between the word vectors corresponding to antonyms.

$$L = \text{CosineLoss}(u, v) = \frac{u^T v}{||u||\,||v||} \tag{4}$$

For each pair of antonyms seen during the training phase, we add the above additional loss term. This results in an additional gradient term, described below for pairs of antonyms seen during the training phase.

$$\frac{\partial L}{\partial u_i} = \frac{v_i \sum u_i^2 - u_i\, u^T v}{(\sum u_i^2)^{\frac{3}{2}} \sqrt{\sum v_i^2}} \tag{5}$$

Where u and v are word vector representation for a pair of antonyms. In addition, a hyper-parameter β is use to tune the learning rate of the cosine loss term. Our experiments indicate that the values of β in the range 10^{-5} to 10^{-4} are useful. Table 1 shows that this approach learns orthogonal word vector representations for antonyms. Further, Table 2 shows the nearest neighbors of a given word, with respect to cosine similarity measure. It is seen that the nearest neighbor set of a word in CLWE does not contain any of its antonyms.

Table 1: Cosine similarity between 300 dimension word vectors corresponding to common antonyms

Antonym Pairs	Word2vec skip-gram	CLWE
bad : good	0.17013	0.00151
ended : begin	0.10618	0.00115
short : long	0.24898	2.1166e-05
demonstrating : disprove	0.17387	0.04159
forgot : remember	0.17388	0.00542
Rejecting : accept	0.20799	0.00237
early : late	0.19553	0.01330
fall : ascent	0.14891	0.00211
contradict : affirm	0.14889	0.044182
Sells : buy	0.14847	0.06270

We used the a portion (40 million words) of a 1 billion word language model benchmark dataset [1] for learning all our word embeddings. In particular, we trained the standard word2vec skip-gram model with negative sampling and a CLWE on this dataset. Antonym pair information was mined using the WordNet®[4] lexical database. We evaluate the quality of each of the word vector representations by using them in several sentiment recognition models.

Table 2: Nearest neighbor of the word *good* in Embedding space

Word2vec Skip-gram model		CLWE	
Nearest Neighbour	Cosine Similarity	Nearest Neighbour	Cosine Similarity
bad	0.717059	better	0.945991
lousy	0.666702	best	0.93051
decent	0.652319	well	0.78963
nice	0.646020	improving	0.56774
mediocre	0.582150	ameliorate	0.529369

2.1 Results on non-neural net models

Using a CLWE-based word vector (or an anti-parallel transform) on SVM and Logistic Regression as described in section 3 did not provide statistically different results (p-value > 0.4 on t-test for SVM 5-labeling, similar for others). We hypothesize that this might be due to the relatively low number of antonym pairs affected by the CLWE tranform. Regardless, the word vectors carried more information as pertaining to the sentiment analysis, and the performance did not decrease. Further research in embedding sentiment information in the vector space is required, but a CLWE with additional vocabulary might be a good direction to follow.

2.2 Results on neural net models

Followed in section 4.3.

3 Word Embeddings in non-neural net models

On a more general basis, we wish to test the performance of word-embeddings as opposed to traditional bag-of-words models. Various papers have shown the effectiveness of traditional models. Models like Support Vector Machines (SVMs) [15] and Logistic Regression [6] are known to be some of the best non-neural net models for classifying text. For our analysis, we used an SVM with a linear kernel and $C = 1.0$, and logistic regression on a SAG solver [13]. The data was parsed from the Stanford Treebank dataset [14], and involved a labelling of sentences with sentiment labels from 1 through 5. For a 3-labeling system, the 5 labels were converted as:

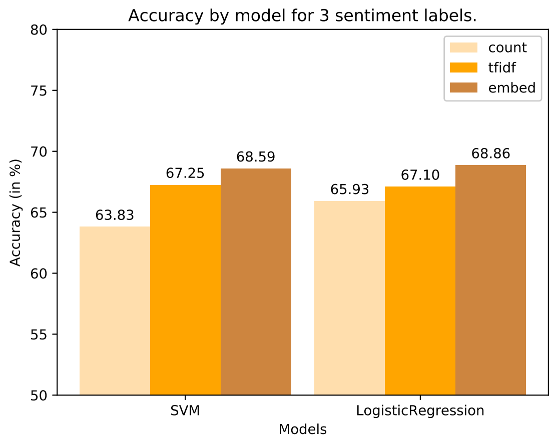
$$(1, 2) \to 0, (3) \to 1, (4, 5) \to 2$$

We considered three different sub-models for each model-set. First, we considered a One-Hot encoding with uni, bi, and tri-grams. This is usually the most basic encoding of words done in text classification. For our second model, we used a TF-IDF encoding [12] for uni, bi, and trigrams. TF-IDF for a word represents its relative importance in the sentence controlling for the ubiquity of the word in the entire corpus. This is calculated by using the inverse proportion of the frequency of the particular word in a particular sentence to the percentage of sentences the word appears in.

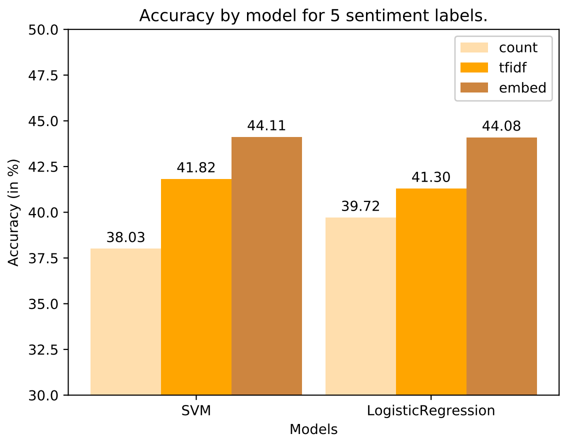
Lastly, we considered the word embedding sub-model. We used the GLoVe word vectoring [11] trained on Common Crawl [`https://commoncrawl.org/`] as implemented by spaCy [7]. This resulted in a vector-dimension of 300 for each word. We analyzed further sub-models such as weighing each word-vector by the TF-IDF of the word. These analyses follow below in section 3.2. In the graphs below, ‘count’ refers to the One-Hot encoding, ‘tfidf’ the TF-IDF encoding, and ‘embed’ our best word-embedding model.

3.1 Logistic Regression and SVM

Figures 1a and 1b give the observed accuracies of the various models. All experiments were conducted on a 5-fold stratified cross validation of the dataset. There were a total of about $11,800$ sentences in the dataset.

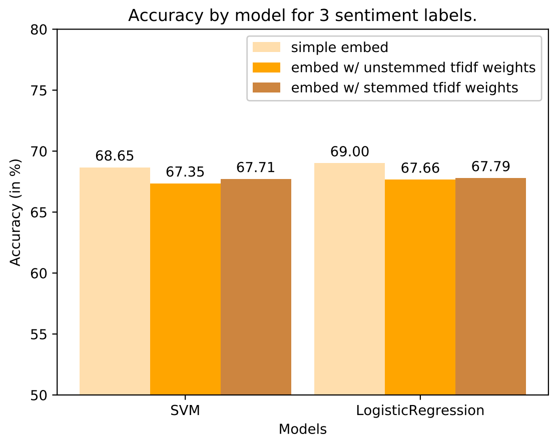


(a) Model performances on 3 sentiment labels

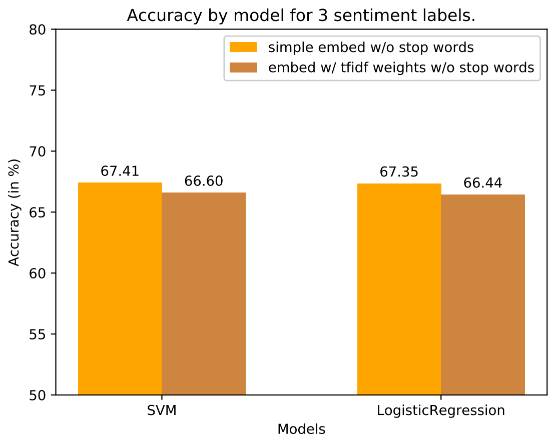


(b) Model performances on 5 sentiment labels

Figure 1: Performance by SVM and Logistic Regression



(a) Effect of TF-IDF weighing



(b) Effect of stop words elimination. TF-IDF on unstemmed words.

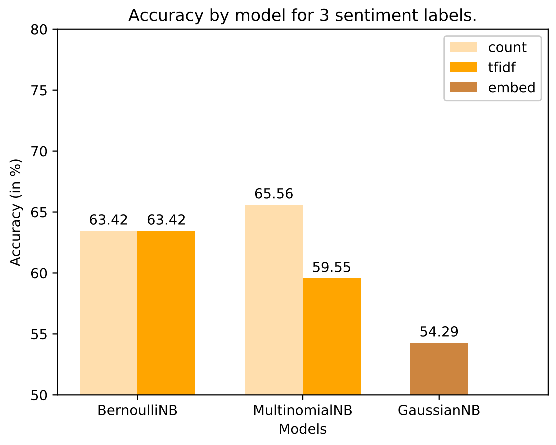
Figure 2: Fine-Tuning

The impact of word embeddings is significant. Looking at the SVM-tfidf vs SVM-embed, the latter is significantly statistically different (with $p < 0.05$ on a t-test for SVM, similar for logistic regression). The difference in the 5-labelling experiment is even more staggering (with $p < 0.001$ on a t-test, for SVM, similar for logistic regression). Clearly, the more labels there are, the greater a resolution word-embeddings provide. Note that these measurements fall in line with those observed in other papers [14].

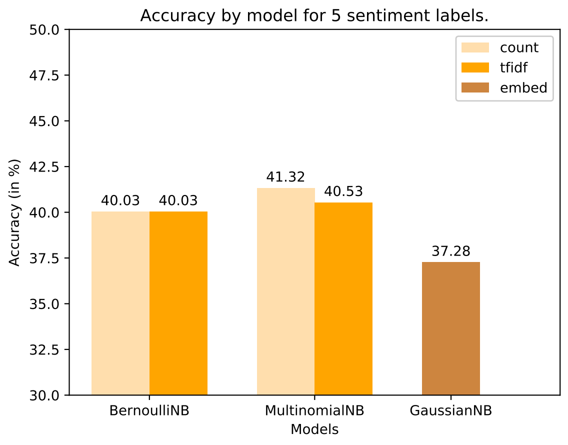
3.2 Fine-Tuning

For the base 'embed' model, we used a mean of word vectors for all the words in a sentence and used this mean vector as the feature space. We also tried different variations on this sub-model. We calculated the weighted mean of the word vectors as weighted by the TF-IDF score for that word. Figure 2a shows the results for the 3-label experiment. The 5-label result was similar. We tried two different TF-IDF schemes - the first on the raw word, the second on the stemmed version of the word.

It turned out that the simple mean outperforms both the TF-IDF weighted methods. We hypothesized that the main cause of this is improper pruning of noisy words. We tried to alleviate this by



(a) Model performances on 3 sentiment labels



(b) Model performances on 5 sentiment labels

Figure 3: Performance by various Naive-Bayes models

looking at stop words (Figure 2b : 3 label results; the 5 label results were similar). Stop words are words common in the lexicon (e.g. 'the') that often carry little-to-no emotional meaning. Removing stop words in non-word vector approaches is often a positive method to improve the performance of the model (e.g. [3]).

Surprisingly, elimination of stop words *reduced* the performance of the models - both mean and TF-IDF weighted (see Figure 2a for comparison). This might imply that some, if not all, stop words carry subtle emotional meaning and cannot be simply ruled out. Clearly, further research is required on this.

3.3 Naive-Bayes

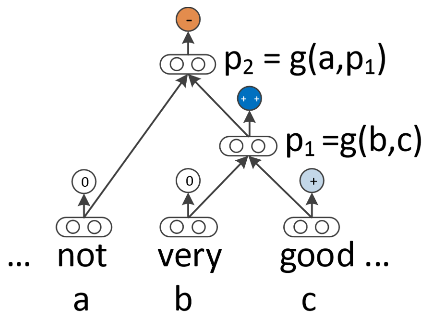
Finally, for completeness, we tested word-embeddings on Naive-Bayes (NB) (Figure 3).

BernoulliNB looked at the binary occurrence of words (hence why count and tfidf have identical scores). To account for the nature of word vectors, we used a GaussianNB for the word embeddings. As could have been predicted, a Multinomial bag-of-words far outperformed the GaussianNB. Naive Bayes methodologies cannot be used to apply probability-based priors on the vector-space of word embeddings. Despite comparable performance of bag-of-words NB with SVM and Logistic Regression (see Figure 1 for comparison), NB cannot be improved by word embeddings, due to the nature of the NB model.

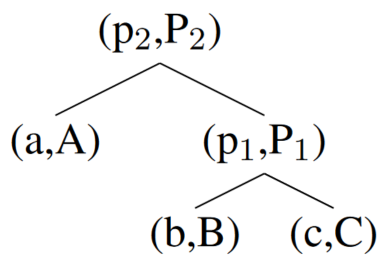
4 Word Embeddings in recursive neural net models

In this section we discuss the Recursive Neural Tensor Network(RNTN) as proposed in [14]. This model uses d- dimensional word vectors in training. The word vectors are initialized by randomly sampling values from Uniform distribution: $U(-r, r)$. Word vectors for each word are combined and considered as a word embedding matrix $L \in R^{d \times |V|}$ where $|V|$ is the vocabulary size. Though the word vectors are initialized randomly they are also used as parameters and are used in training along with compositionality models.

When a sentence is given to the composition model, it is converted to a binary tree where leaves correspond to the words and are represented by vectors. Recursive models compute parent vectors in a bottom up fashion. Different models use different functions g, for getting the parent vectors. In turn, these parent vectors are used for calculating their parents (see Figure 4).



(a) Parent vectors computed in bottom up fashion.



(b) Tree representing the vector and matrix at each node.

Figure 4: Tree structure in RNTN. From [14]

Posterior Probability is computed for classification into five classes.

$$y^a = \text{softmax}(W_s a)$$

Where $W_s \in \mathbb{R}^{5\times d}$ is the sentiment classification matrix. This is repeated for words b and c and the difference between these models lies in the hidden layer.

4.1 RNN: Recursive Neural Network

This is the simplest models of all recursive neural models. In this, first the parent for which the children are already computed are identified and the value of parent vector is computed. RNN uses the following for computing the parent vectors.

$$p_1 = f\left(W\begin{bmatrix} b \\ c \end{bmatrix}\right), p_s = f\left(W\begin{bmatrix} a \\ p_1 \end{bmatrix}\right) \tag{6}$$

where $f = tanh$ (from [14]). The parent vectors are given to the softmax function and labels are determined for the parent nodes.

4.2 RNTN: Recursive Neural Tensor Network

Recursive Neural Tensor Network was proposed to solve the problem of the increased number of parameters and dependency on vocabulary size. RNTN proposes a single composition function with fixed number of parameters. The idea proposed was to use the same tensor function for all nodes in the parse tree. RNTN uses a tensor layer and a standard layer for computing the parent vector.

In order to maximize the correct prediction at each node the error is minimized by using the following equation in training the dataset (from [14]).

$$E(\theta) = \sum_i \sum_j t^i_j log y^i_j + \lambda||\theta||^2 \tag{7}$$

4.3 RNTN-CLWE

Our approach to training the model lay in using a constant word vector during the training as opposed to the models described above. We plugged in the already trained word embeddings from CLWE (section 2) into the RNTN models and trained it for sentiment classification. In equation7 for error calculation, $\theta = (V, W, Ws, L)$ containing vectors is updated in each batch of the training. We kept the word vectors matrix L as a constant during the training. We then trained on CLWE vectors and normal word vectors. Table 3 lists our observations. In it, RNTN-canonical refers to the native RNTN trained on randomly initialized vectors. Root accuracy refers to the accuracy of the word in the root node of the tree.

Table 3: Performance of RNTN-L constant and RNTN-canonical over CLWE and non-CLWE vectors

vector set	number of word-vectors	number of epochs	RNTN-L constant	RNTN-canonical
non-CLWE	4,000	50	accuracy: 0.687617 root accuracy: 0.233032	accuracy: 0.791671 root accuracy: 0.4
non-CLWE	100,000	50	accuracy: 0.699257 root accuracy: 0.271493	accuracy: 0.791671 root accuracy: 0.4
CLWE	100,000	50	accuracy: 0.701102 root accuracy: 0.271946	accuracy: 0.791671 root accuracy: 0.4

As in section 2.1, the addition of CLWE did not degrade the results but were quite similar to the non-CLWE vector performance. As before, we hypothesize that this may be because of the low number of antonym-pairs affected by the vectorization.

5 Conclusion

In this paper, we briefly noted the importance of sentiment analysis and previous approaches in the field to model it. We applied our own techniques to improve the accuracy of several popular models like SVM and logistic regression, and found that we were able to train fine grained sentiment analysis with deeper accuracy than conventional models.

We developed a novel vectorization, Cosine Loss Word Embedding (CLWE) to enhance methods that used other types of word embeddings. CLWE attempted to incorporate end-result opinion values of words by reconfiguring the feature space for antonym pairs. We found that CLWE did not necessarily improve popular models, including RNN-based ones. But the embeddings themselves better represented the similarity of antonyms in regards to sentiment analysis. We feel that with the benefits gained with non-neural network approaches and CLWE, there is room to explore for better vectorization, and therefore, for further improvement of sentiment-based analysis.

References

[1] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., and Koehn, P. One billion word benchmark for measuring progress in statistical language modeling. *CoRR abs/1312.3005* (2013).

[2] Cui, X., Lam, D., and Verma, A. Embedded value in bloomberg news and social sentiment data. Tech. rep., Bloomberg L.P., 2016.

[3] El-Khair, I. A. Effects of stop words elimination for arabic information retrieval: a comparative study. *International Journal of Computing & Information Sciences 4*, 3 (2006), 119–133.

[4] Fellbaum, C. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[5] Fink, C. R., Chou, D. S., Kopecky, J. J., and Llorens, A. J. Coarse- and fine-grained sentiment analysis of social media text. *Johns hopkins apl technical digest 30*, 1 (2011), 22–30.

[6] Hamdan, H., Bellot, P., and Bechet, F. Lsislif: Crf and logistic regression for opinion target extraction and sentiment polarity analysis. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)* (2015), pp. 753–758.

[7] Honnibal, M., and Montani, I. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear* (2017).

[8] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. Efficient estimation of word representations in vector space, 2013.

[9] Pak, A., and Paroubek, P. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc* (2010), vol. 10, pp. 1320–1326.

[10] Pang, B., Lee, L., et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval 2*, 1–2 (2008), 1–135.

[11] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1532–1543.

[12] Ramos, J., et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning* (2003), vol. 242, pp. 133–142.

[13] Schmidt, M., Le Roux, N., and Bach, F. Minimizing Finite Sums with the Stochastic Average Gradient. *Mathematical Programming B 162*, 1-2 (2017), 83–112. Revision from January 2015 submission. Major changes: updated literature follow and discussion of subsequent work, additional Lemma showing the validity of one of the formulas, somewhat simplified presentation of Lyapunov bound, included code needed for checking proofs rather than the polynomials generated by the code, added error regions to the numerical experiments.

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

[15] Wang, S., and Manning, C. D. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2* (Stroudsburg, PA, USA, 2012), ACL '12, Association for Computational Linguistics, pp. 90–94.

[16] Zirn, C., Niepert, M., Stuckenschmidt, H., and Strube, M. Fine-grained sentiment analysis with structural features. In *Proceedings of 5th International Joint Conference on Natural Language Processing* (2011), pp. 336–344.