A Lazy Random Walk Operator for Matrix Factorization Word Embeddings

ELE Independent Work '18

Gene Li – gxli@princeton.edu

October 7, 2018

Abstract

This paper explores the curious phenomenon: a simple preprocessing step, applied to word co-occurrences before computing PMI embeddings, drastically improves the ability of such embeddings to solve analogy tasks.

1 Introduction

Word embeddings are vector representations of words that are encode relationships between words. Two well-known properties of word embeddings are cosine similarity and linear algebraic structure. Cosine similarity, is the fact that word embeddings for similar words have large inner product. Linear algebraic structure is the fact that word embeddings are able to solve analogies, i.e.:

$$v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} = v_{\text{queen}} \tag{1}$$

Word embeddings are generated by many different methods, however, the key intuition is that words that appear near each other many times in a natural language corpus should be related. A simple way to build word embeddings that can solve word similarity and analogy solving tasks is to build a co-occurrence matrix $X \in \mathbb{R}^{n \times n}$, where n represents the number of words in a vocabulary and each element X_{ij} is a record of the number of times word w_i and word w_j occur near each other in the natural language corpus (i.e. within 5 words of each other). A nonlinearity $\phi(X)$ is applied to the co-occurrence matrix, followed by a rank-d singular value decomposition (SVD): $X = U\Sigma V^T$, where $U, V \in \mathbb{R}^{n \times d}$ for some $d \ll n$. A popular ϕ that works well in practice is the so-called pointwise mutual information (PMI) operator, an element-wise operation defined by:

$$\text{PMI}_{ij} = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)} \tag{2}$$

The word vectors are then defined as the row vectors of U.

We propose a lazy random walk operator on the matrix X as a preprocessing step before computing matrix factorization embeddings. Section 3 describes the operator. Empirical results are presented in Section 4. We find that this operator improves the ability of PMI word embeddings to solve analogy tasks. In Section 5, several possible theoretical explanations are proposed.

2 Related Work

Matrix factorization methods for word embeddings have had a long history. Many reweightings of the word co-occurrence matrix have been proposed, including PMI, Logarithm, etc. Word embeddings can be generated with a term-document approach where the columns represents "documents" that the words occur in, or term-term approaches where both the rows and columns index words. In general, these matrices can be asymmetric, however in our case we are only concerned with symmetric word co-occurrence matrices (term-term).

Another matrix factorization approach that has been highly successful in solving analogy tasks is the GloVe method ([1]), which solves a weighted matrix factorization problem of the form:

$$J = \sum_{i,j} f(X_{ij})(\log X_{ij} - \langle v_i, v_j \rangle - s_i - s_j)^2 \tag{3}$$

where v_k are the word vectors and s_k are scalar bias terms associated with each word. Here, $f(x)$ is some empirically tuned reweighting function (i.e. $f(x) = x^{3/4}$), used to limit the impact of very commonly co-occurring pairs. This objective function is derived from the observation that *ratios of co-occurrence probabilities* should be used to encode meaning. GloVe is generally considered state of the art.

The other highly successful approach is word2vec ([2]), which trains a neural network on a text corpus to maximize the likelihood of predicting a window of context words from the center word (SGNS). The "word vectors" derived are essentially an internal matrix of the neural network.

There are some theoretical explanations for why these various embedding methods are able to solve analogies. Levy and Goldberg '14 relate word2vec with PMI - stating that the "skip-gram with negative sampling" (SGNS) variant of word2vec is implicitly factorizing a shifted positive pointwise mutual information matrix (SPPMI), namely:

$$\text{SPPMI}_k(w_i, w_j) = \max(\text{PMI}_{ij} - \log k, 0). \tag{4}$$

Here k has the interpretation of the number of negative samples (i.e. randomly drawn pairs of words) to compare against the word pair that actually occurs in text, used in the logistic regression objective function of SGNS.

Recently, Arora et al. ([3]) proposed a generative model for text generation, which relies on a latent "discourse vector" c_t that does random walk in $\mathbb{R}^d$. Words are generated according to probability:

$$P(w|c_t) \propto \exp(\langle c_t, v_w \rangle). \tag{5}$$

Their key assumption is the isotropic nature of word embeddings $v_w \in \mathbb{R}^d$, which allows for concentration of the partition function

$$Z_c = \sum_w \exp(\langle v_w, c \rangle) \tag{6}$$

to be close to a constant Z independent of the discourse vector c. Thus they are able to derive in their Theorem 2.2 the following expressions for co-occurrence and marginal probabilities:

$$\begin{aligned} \log p(w_i, w_j) &= \frac{\|v_i + v_j\|_2^2}{2d} - 2\log Z \\ \log p(w_i) &= \frac{\|v_i\|_2^2}{2d} - \log Z \end{aligned} \tag{7}$$

Based on this derivation, they proposed the **Squared Norm** algorithm, which is solving the following objective function:

$$\min_{\{v_k\}, C} \sum_{i,j} X_{ij} (\log X_{ij} - \|v_i + v_j\|_2^2 - C)^2 \tag{8}$$

Using this model, they give theoretical justification for non-linear models of PMI, word2vec, and GloVe, and explain why these models are able to solve analogies (they call this property of word vectors the "linear algebraic structure"). Their model explains the property

$$\text{PMI}_{ij} \approx \langle v_i, v_j \rangle. \tag{9}$$

3 Lazy Random Walk

3.1 Vanilla Algorithm

In general, the PMI embedding method does not perform as well as GloVe and word2vec, for reasons not well understood. We can improve the performance of PMI embeddings through a simple scheme which has the interpretation of a "lazy random walk".

Let X be the matrix of co-occurrences of word pairs. WLOG, we can renormalize X by dividing by $\sum_{i,j} X_{ij}$. Thus, each entry X_{ij} has the interpretation of a probability of w_i, w_j co-occurring.

Next, we can factorize $X = DR$, where D is a diagonal matrix with $D_{ij} = \sum_j X_{ij}$. R has the well-known interpretation of the transition matrix in a random walk on a Markov Chain. R is row-normalized, i.e. $\sum_j R_{ij} = 1$, and each entry can be viewed as $R_{ij} = \hat{p}(w_j | w_i)$.

Then, we can produce a new empirical co-occurrence probability matrix:

$$X' = D \sum_{k=1}^{\infty} \alpha_k R^k. \tag{10}$$

If $\sum a_k = 1, a_k \geq 0$, we have the following interpretation: for each word, the next word can be generated as follows: with probability a_k, pick a step-length of k. Then generate the next word by moving k steps on the Markov Chain that is represented by the empirical co-occurrence counts.

We find that we can improve the quality of the PMI embedding by using X' as the input instead of X. Experimentally, we find that only considering a weighted average between the first term and the second term in the equation gives us good performance boosts. The vanilla algorithm (with two terms) is summarized in Algorithm 1.

Algorithm 1 Lazy Random Walk Embedding

- Input: co-occurrence matrix X. weighting hyperparameter α.
- Compute R, D, such $X = DR$.
- Let $X' = \alpha DR + (1-\alpha)DR^2$.
- Apply rank-d SVD to: $PMI(X') = U\Sigma V^T$.
- Let rows of U be word vectors v_w.

3.2 Properties of the Lazy Random Walk Matrix

For each term, we can write an explicit update rule. For simplicity, we only consider the two-term case (where our X' is a weighted average between DR and DR^2.

$$\begin{aligned} X'_{ij} &= \alpha(DR)_{ij} + (1-\alpha)(DR^2)_{ij} \\ &= \alpha\hat{p}(w_j|w_i)\hat{p}(w_i) + (1-\alpha)\left(\sum_k \hat{p}(w_j|w_k)\hat{p}(w_k|w_i)\right)\hat{p}(w_i) \end{aligned} \tag{11}$$

Here we can see that X' is a weighted average between the original X and some new matrix that encodes 2-step random walk information.

In addition, one can also see that the marginals of X' are the same as the marginals of X, which is perhaps a property of theoretical interest. This is easy to see from the above equation if we sum over i or j and compare to the original marginal $\hat{p}(w_j)$, $\hat{p}(w_i)$ respectively.

3.3 Modifications

A computation issue arises: it becomes very difficult to store powers of matrices, because powers densify matrices that are originally sparse. In our case, it is easy to see that X' is a completely dense matrix. If X' is viewed as the adjacency matrix of some weighted graph, one can note that all words/nodes are at most distance 2 from each other due to connecting

words such as "the" and "and", which have co-occurrences with every other word in the vocabulary. Thus $X'_{ij} > 0$ for all i, j.

There are several ways to get around this:

1. If our matrix is too big to power and store in memory, we can truncate the original matrix so that only the top k words are represented, where k is a reasonable number that we can square and store a dense $k \times k$ matrix. In doing so, we are discarding a lot of information in the words that occur less frequently. However, it should be noted that for matrix factorization embeddings such as PMI, this is already being done in practice when we set thresholds on word count for us to include them in our data (i.e. unigram count must be > 1000 in our corpus.)

2. We can choose to apply the random walk operator to a sub-matrix of the original co-occurrence matrix. Intuitively, it seems that more common words do not need this random walk adjustment, so we would like to apply it to words that occur less frequently and do not have many other observed co-occurrences with other words. We can select a sub-matrix of size $k \times k$ that we apply this operator to, either deterministically or probabilistically. For example:

   (a) Select only the words with the smallest unigram counts (i.e. bottom 20%).

   (b) Select words if distribution $p(w_i, w_j)$ for fixed w_i (corresponding to a row of matrix) is sparse.

   (c) Select words if the distribution $p(w_i, w_j)$ for fixed w_i is far from the uniform distribution in Kullback-Leibler divergence, or a similar metric.

4 Empirical Findings

4.1 Methods

Our code can be found at: `https://github.com/GXLI97/lazyrandomwalk`.

Our corpus is processed in the same way as [3]. We download the latest version of English Wikipedia (August 2018) and preprocess it in the standard way (removing punctuation, tokenization, and sentence splitting), resulting in a dataset with 4.5 billion tokens. Words that appear less than 1000 times are ignored, resulting in vocabulary size of 79660 unique words. Note that our dataset is slightly larger than that of [3] because we are using a later version of English Wikipedia.

The co-occurrences are computed using unweighted windows of size 10 around the focus word, using GloVe software with a memory limit of 128 GB. This results in a matrix with 934,455,743 nonzero co-occurrence entries. Thus, the sparsity is $\approx 15\%$.

Our lazy random walk embeddings are computed on the co-occurrence matrix, using a randomized SVD algorithm in scikit-learn ([4]). With this dataset size, we can store a dense matrix and do not need to apply any of the modifications as in Section 3.3, and can calculate the full lazy random walk on a university research computing cluster.

For comparison, we train a GloVe model on the same co-occurrence matrix using the default out-of-the-box settings. The only change we make is to reduce the eta learning rate from the default of 0.05 to 0.01, as we were running into numerical stability issues.

We test on the analogy datasets using GloVe evaluation code, which is a linear algebraic query used by [1] and [3]. Vectors are normalized to unit norm and analogies "a:b::c:?" are solved by:

$$\underset{d}{\operatorname{argmin}} \|v_a - v_b - v_c + v_d\|_2^2. \tag{12}$$

Our vocabulary allows us to solve a maximum of 94.9% of the total testbench - 18539 of 19544 questions. In other words, 94.9% of the analogy questions have all 4 words in our processed vocabulary. For subsequent results, the accuracies reported are based on the "in-vocab" analogy questions (of which there are 18539).

4.2 Results

First, we empirically verify that this lazy random walk operator indeed improves the performance of PMI embeddings on analogy datasets. In Figure 1, we can see the performance boost one one gets when using the lazy random walk PMI embedding. Note that $\alpha = 1$ denotes the baseline PMI embedding without the lazy random walk preprocessing, and as we decrease α, we see an improvement in performance. The best performance is achieved at $\alpha = 0.8$ with overall accuracy of 62.7%, which is an improvement of over 13 percentage points from baseline $\alpha = 1$ overall accuracy of 49.4%. For reference, GloVe performance on this co-occurrence matrix is 71.2%.

Next we examine the affect of embedding dimension d on performance for baseline PMI, lazy random walk PMI, and GloVe embeddings. Here, we fix $\alpha = 0.8$ for the lazy random walk PMI embeddings. Our semantic, syntactic, and overall results are plotted in 2. Here we can see that the lazy random walk PMI embeddings consistently beat baseline PMI, but fail to achieve GloVe performance for any of the dimensions.

Lastly, we plot the spectrum of singular values in the PMI embeddings for the baseline PMI vs the lazy random walk PMI. The singular values are normalized by dividing each singular value by the sum of the $d = 500$ singular values. This can be seen in Fig. 3. From this, we can see that the more mass is allotted to the first few singular values in the lazy random matrix PMI case.

5 Possible Theoretical Justifications

5.1 Graph Powering

The lazy random walk is loosely motivated by recent work on graph powering in the sparse regime of the stochastic block model (SBM), which is an unweighted graph with community structure [5]. The main idea is to use powers of the adjacency matrix to solve community detection down to the information-theoretic threshold. If we consider the two-community

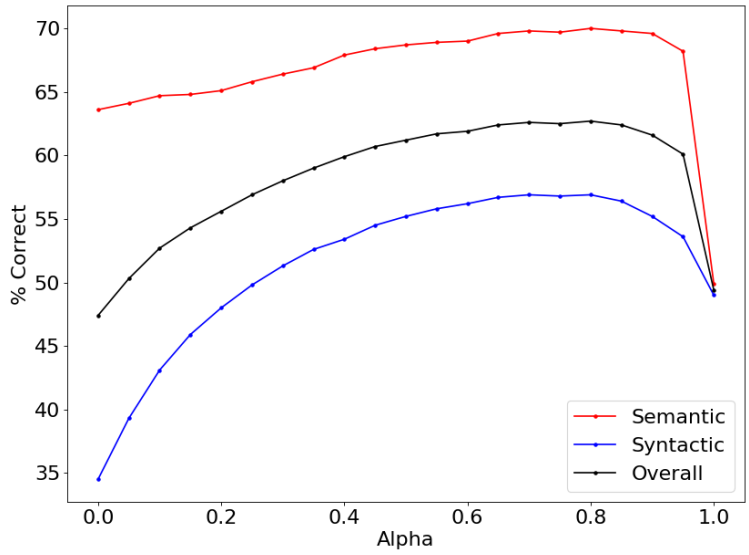
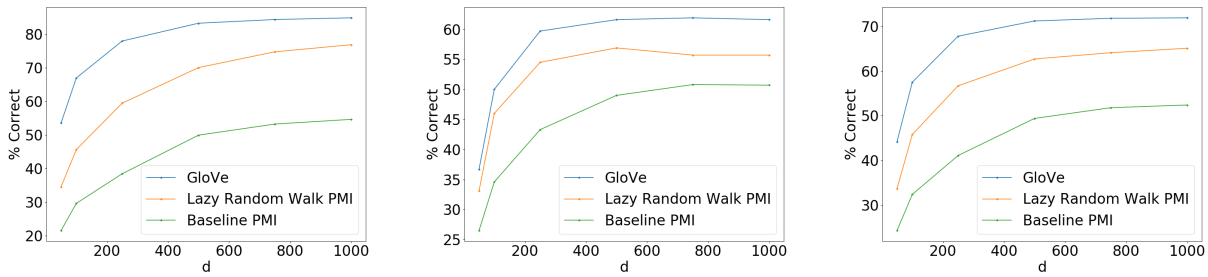


Figure 1: Semantic, syntactic, and overall accuracy of the lazy random walk PMI Embeddings for various α. Maximum accuracy is obtained for $\alpha = 0.8$. Overall accuracy is a weighted average of semantic and syntactic because there are different number of analogy questions for semantic and syntactic.



Semantic Syntactic Overall

Figure 2: Accuracies for various d.

SBM, we can use spectral methods to classify each node into community 1 or 2 (up to permutation). Graph powering does the following: edges are added into the adjacency matrix corresponding to walks on the graph of length $1, 2, ..k$. More precisely, we consider the matrix:

$$\mathbb{1}\{(I + A)^k > 0\}. \tag{13}$$

and apply spectral methods to this matrix to recover the communities.

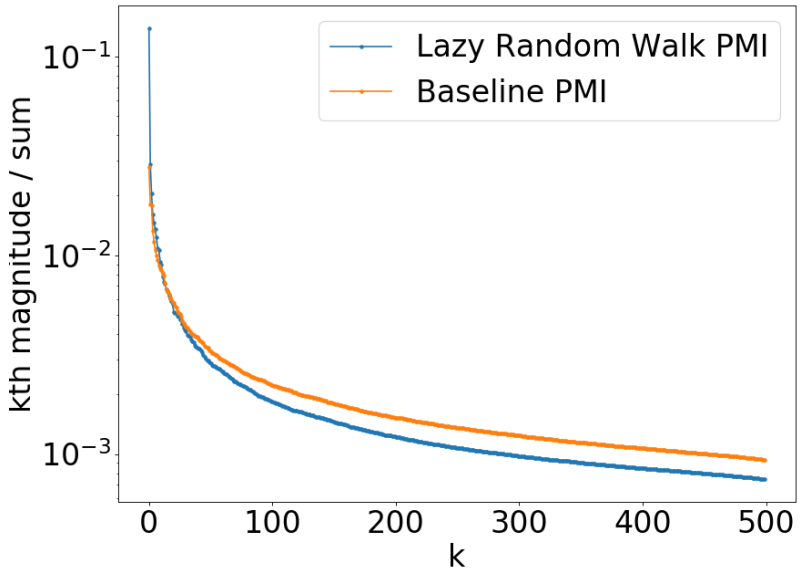


Figure 3: Comparison of singular value spectrum for the baseline PMI vs the lazy random walk PMI. Both use 500-dimensional vectors. The lazy random walk PMI is calculated with $\alpha = 0.8$. Y axis is log scale.

In the same way, we can view the word co-occurrence matrix as an adjacency matrix for some weighted graph G. With the lazy random walk operator, we are updating and adding edges to our graph G. Generally speaking, this has a chance of working because sometimes words may be "related" even if the co-occurrence counts between them are small - for example "walk" and "walker" may not have high co-occurrence, but encode similar meanings. Thus, we can try to fill in the missing co-occurrences by adding edges.

Naively powering the co-occurrence matrix itself does not work. It is hard to control the truncation mechanism for weighted graphs (the $\mathbb{1}$ in the above eq.), because in this case, a lot of the information about co-occurrences is lost due to the powering adding in edges between words that do not relate, solely because "the" or "and" is connected to all words. Another issue is that in order to recover linear algebraic structure, we have to apply very specific kernels such as the PMI operator - simply taking the SVD of the original matrix X to create word embeddings does not allow you to solve analogies. The matrix X^2 does not have a good interpretation and it is difficult to come up with a suitable kernel to recover linear algebraic structure from X^2.

The lazy random walk operator resolves these two difficulties. In the random walk matrix, the importance of the filler words "the" and "and" is much discounted. Since each row is normalized to 1, the corresponding probabilities are much more spread out for filler words and they do not contribute much to the summation $\sum_k p(w_j|w_k)p(w_k|w_i)$. Values in the powered matrix are more likely to encode strong semantic relationship. Only words w_k that

have large values in both $p(w_j|w_k)$ and $p(w_k|w_i)$ will contribute much to the sum, and it is easy to see that filler words do not satisfy this. Secondly, the resulting matrix X' still has a valid interpretation as some empirical co-occurrence matrix - see the discussion in section 3. In this way, we do not have to engineer a new kernel and can instead use pre-existing methods such as GloVe and PMI out of the box.

5.2 Sentence Inverse Frequency as a Re-weighting

Another possible avenue for theoretical exploration comes from an interpretation of Sentence Inverse Frequency (SIF) model ([6]). SIF is a modification to the original RAND-WALK model that adds smoothing terms to account for the fact that words like "the" appear regardless of the context. The new model is as follows:

$$p(w|c_s) = \alpha p(w) + (1-\alpha)\frac{\exp(\langle c_s v_w \rangle)}{Z_{c_s}} \tag{14}$$

(Here we only consider the first modification of the original paper: adding smoothing terms corresponding to unigram probabilities. The second modification of the common discourse vector is not included.)

We can interpret this as follows. At each time, a weighted coin is flipped, and a new word is generated with probability α according to its unigram distribution and probability $1-\alpha$ according to the original RAND-WALK model. Instead of considering discourse vectors c_s, let us substitute another word vector:

$$p(w_j|w_i) = \alpha p(w_j) + (1-\alpha)\frac{\exp(\langle v_i, v_j \rangle)}{Z_{v_i}} \tag{15}$$

Let us denote the original RAND-WALK model as generating the probabilities:

$$p_0(w_j|w_i) = \frac{\exp(\langle v_i, v_j \rangle)}{Z_{v_i}} \tag{16}$$

Then multiplying by $p(w_i)$ on both sides we arrive at:

$$p(w_i, w_j) = \alpha p(w_i)p(w_j) + (1-\alpha)p_0(w_i, w_j) \tag{17}$$

From this, we can see that SIF is equivalent to a re-weighting towards the matrix $M = pp^T$, where p is the unigram marginals. M_{ij} corresponds to the co-occurrence of w_i and w_j as if they were independent.

Thus, this idea of re-weighting towards with matrix is already implicitly present in successful word embedding modification steps!

6 Future Directions

A few possible future directions are suggested below.

1. **The correct definition of graph powering for weighted matrices.** This lazy random walk operator is probably not the correct way to interpret graph powering for weighted matrices. So maybe we can consider how to interpret the graph powering algorithm for the stochastic block model to apply to a different parametric model.

   Specifically, let us assume a PMI model:

   $$p(w_1, w_2) \propto \exp(\|v_{w_1} + v_{w_2}\|_2^2) \tag{18}$$

   as is done in [3]. We observe finite samples of co-occurrence pairs drawn from this distribution to build an empirical co-occurrence matrix. It's not clear whether we get nice concentration for all co-occurrence pairs, especially those of "rare" words. Using techniques from random matrix theory, we can try to understand the statistics of the co-occurrence matrix. Then we can try to apply a graph powering definition to robustly recover the vectors v_{w_i}. See [5] for recommendations on how to extend graph powering for weighted matrices.

2. **Empirical observations**. Not much is known about how this operator affects the vectors that are learned. For example we can try to compare inner products for "clusters" of words, and see how those inner products between vectors differ for GloVe, word2vec, PMI, and our (modified) PMI.

   An interpretation of powering as a way to get "noisy contexts" could be valid. Maybe we can try to process words for smaller contexts, i.e. window sizes of 2, 4, 8, etc. around the central word - to see how an operator behaves along this dimension.

   Yet another idea that could be explored experimentally is the notion of isotropy. Are better-performing word vectors more isotropic than worse-performing word vectors (performance is not well defined here, say for some downstream task). Isotropy is an assumption used in the proof of [3], however its not known why such an assumption makes sense as a property of word embeddings. Some work has been done by [7] which suggests that simple post-processing of word vectors to enforce first and second order notions of isotropy improves performance of word vectors on nearly all downstream tasks. An intuition can be given as follows: Even though we would like to group similar words together to have high inner product, maybe we don't want an inner product that is too high (≈ 1). This could be because we still want to differentiate fine details between different words. Decreasing the inner products between words, even those that are similar in meaning, could allow us to do this.

3. **Engineering Tricks.** Lastly, there are numerous engineering modifications that can be applied to improve the quality of PMI embeddings.

7 Conclusion

In this paper, we describe a simple preprocessing step that improves the effectiveness of PMI word embeddings on analogy testbenches. This modification does not attain GloVe perfor-

mance, but further techniques can be used to improve the effectiveness. Several theoretical justifications are proposed, but refinement of these arguments is left to future work.

References

[1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[3] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Rand-walk: A latent variable model approach to word embeddings. *arXiv preprint arXiv:1502.03520*, 2015.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Emmanuel Abbe, Enric Boix, Peter Ralli, and Colin Sandon. Graph powering and spectral robustness. *arXiv preprint arXiv:1809.04818*, 2018.

[6] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2016.

[7] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. *arXiv preprint arXiv:1702.01417*, 2017.