Homework 3: Adventures with word embeddings (100 points)

Kathleen McKeown, Fall 2019
COMS W4705: Natural Language Processing

Due 11/13/19 at 11:59pm

The aim of this assignment is to furnish you with a solid practical and theoretical understanding of the inner workings of word embeddings. In the first part of the assignment, you will explore the effects of the various hyperparameters word embedding algorithms may be tuned with. In the second part, you will examine the connection between the algorithms themselves.

General instructions

Please post all clarification questions about this homework on the class Piazza under the "hw3" folder. You may post your question privately to the instructors if you wish. If your question includes code or a partial solution, please post it privately to the **instructors only.**

This is an individual assignment. Although you may discuss the questions with other students, you should not discuss the answers with other students. All code and written answers must be entirely your own.

Late policy: You may use late days on this assignment. However, you MUST include at the top of your submission how many late days you are using. If you don't tell us or have used all your late days, 10% per late day will be deducted from the homework grade. You may choose to save your late days for harder assignments later in the class.

You should budget at least a day just for your full set of experiments to run successfully. This likely means several days' worth of runtime in the debugging phase. We strongly recommend that you start coding as soon as possible, but at minimum a week in advance of the deadline.

1 Parameter search (50 points)

This section of the assignment will involve a hands-on exploration of the effects of various hyperparameters on the information learned by different word embedding models, loosely following the methodology of Levy et al. [2015]. In overview, you will be training a battery of models across a range of algorithms and hyperparameters, while evaluating the information they learn via three different tasks. The hyperparameters you will be exploring are:

- Context window size (2, 5, 10)
- Dimension (100, 300, 1000)
- Number of negative samples (1, 5, 15) (applicable to SGNS only)

The models you will be testing them with are

- word2vec skip-gram with negative sampling
- SVD on the positive PMI matrix

This comes out to $27 + 9 = 36$ unique settings of parameters and model total. You will evaluate each of these using the provided script `evaluate.py` and report and analyze results in your writeup.

Writeup

Your submission for this part of the assignment will consist of (1) a table containing the results of your parameter search; and (2) a written analysis of your results. Each row of the table should contain numerical results for one choice of algorithm and parameter setting. The columns should include algorithm, context window, dimension, number of negative samples (indicate '-' for SVD), correlation on WordSim353, accuracy for at least three BATS categories (you pick the ones you think are most interesting from among the 40 low-level categories, the 4 high-level categories, or the total score), and accuracy on the MSR paraphrase corpus.

The table should look something like this:

Algorithm	Win.	Dim.	N. s.	WordSim	BATS 1	BATS 2	BATS 3	MSR
word2vec	2	100	5	47.05	0.01	0.02	0.03	36
SVD	2	100	-	12.34	0.04	0.05	0.06	63
⋮		⋮		⋮		⋮		⋮

Table 1: An example results table.

Your writeup should at least address the following prompts, but you are also encouraged to include other interesting observations or hypotheses you have made.

1. Does larger dimensionality always equate to better performance? In which categories and for which models? Why do you think this is?
2. Does better performance on one task mean better performance on the others? Provide a hypothesis as to why or why not.
3. Was performance roughly similar across all analogy categories? If different, how did it vary? Why do you think you observed this variation? Perform a brief error analysis and compare errors across the BATS categories you selected for your table.

You do not have to answer these in order, but you should explicitly indicate where you answer each of them in your writeup.

1.1 Training corpus

You will train your embedding models on the Brown corpus. The corpus is provided in `data/brown.txt`. The file is formatted with one sentence per line, tokens space-separated. You may experiment with different methods of preprocessing, but it is recommended that you at least lowercase because some of the evaluation tasks only use lowercased words.

1.2 Evaluation corpora

You will evaluate the embedding models you have trained on the following three tasks and corpora (two intrinsic and one extrinsic):

- Word similarity - WordSim353 [Finkelstein et al., 2002]
- Analogy prediction - the Bigger Analogy Test Set [Gladkova et al., 2016]
- Paraphrase detection - the MSR paraphrase corpus [Dolan et al., 2004]

We provide a script, `evaluate.py`, which runs your trained vectors against these benchmarks. It takes vectors in word2vec `.bin` format; in the format saved by gensim's KeyedVectors; or as a `.txt` file with one vector per line: space separated, with the word first, then each of the values in the vector. (See the documentation in the code for an example.)

Note: many of the low-level category accuracies on BATS will be zero. This is expected behavior. You will want to focus on analyzing performance on the categories that display interesting nonzero scores.

1.3 Implementation details

We recommend that you train your word2vec models using the `gensim` package. You are welcome to code up your own implementation (e.g. in PyTorch) if you like, but if you choose to do this, you are responsible for ensuring that your implementation is correct.

You should implement the SVD method yourself. The matrix you will factorize here is the *positive pointwise mutual information* matrix, or the matrix M whose entries are

$$M_{ij} = \max\{0, PMI(w_i, w_j)\},$$

where $PMI(a, b) = \log \frac{p(a, b)}{p(a)p(b)}$. Using the truncated components of the singular value decomposition

$$M = U\Sigma V^T \approx U_k\Sigma_k V_k^T = WC^T,$$

where U_k, Σ_k and V_k are the versions of the respective matrices truncated to the top k singular values, and W and C are the matrices whose rows are the word and context embeddings respectively, you will compute the symmetric decomposition

$$W = U_k\Sigma_k^{1/2}$$

$$C = V_k\Sigma_k^{1/2}.$$

You may use existing SVD solvers, e.g. numpy or scikit-learn's SVD, but you will need to implement co-occurrence collection and postprocessing on your own.

We recommend using a VM with at least 16GB of memory to ensure that you have space to save all your models at the same time.

Hint: You won't be able to store the entire co-occurrence matrix in memory as a dense (e.g. numpy) array. Instead, you'll need to use scipy's lil_matrix format while counting and performing other operations, then convert to coo_matrix before saving.

Hint: You'll probably want to compute the co-occurrence matrix just once for each context window size and save it.

1.4 Bonus (5 points)

For up to five points of extra credit, you may also do an additional evaluation and analysis. You can evaluate the above parameter settings on GloVe (in this case we recommend using the implementation of GloVe available from its website); or you may load a single pretrained BERT model (see Hugging Face's Transformers library) and evaluate it on the same tasks. If you choose to use BERT, you will need to implement the evaluation yourself, as the provided evaluation script only runs on static models.

Alternatively, you may implement a novel modification to word2vec, SVD-PPMI or GloVe and evaluate it for up to five points of extra credit. **Hint:** SVD will probably be easiest.

You can receive at most five points of extra credit total.

2 Fun with objective functions (50 points)

In this portion of the assignment we will uncover the relationship between the objective function of word2vec and matrix factorization methods.

We will relate skip-gram with negative sampling to matrix factorization by first reducing its global objective to a local one under the assumption that individual word-context pairs are independent, then analyzing the optimal solution for each pair.

You should show all work in this section, and simplify expressions as much as possible.

Hint: You may find it useful to consult the provided reference when doing problem 2, but problem 3 can be solved much more neatly than they do in the paper.

Notation and definitions

We follow the notation of Levy and Goldberg [2014] with minor modifications.

Setting. We are given the following:

- A word vocabulary V_w and context vocabulary V_c.
- An input corpus D, consisting of a sequence of pairs (w, c) where $w \in V_w$ and $c \in V_c$, with $|D| = N$.
- Some number k of negative samples.

Each word w is associated with a word vector $\vec{w}$, and each context word c with a context vector $\vec{c}$. We will denote the number of times a word w appears in D by N_w, the number of times a context c appears by N_c, and the number of times any particular pair (w, c) appears by $N_{w,c}$.

Recall that the objective for skipgram with negative sampling is to maximize the log likelihood of its observed corpus, modeled as follows:

$$\begin{aligned}
L &= \sum_{(w,c)\in D} \log p(w, c) \\
&= \sum_{w\in V_w} \sum_{c\in V_c} N_{w,c} \log p(w, c) \\
&\approx \sum_{w\in V_w} \sum_{c\in V_c} N_{w,c} \Big(\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c' \sim P_n(c)} [\log \sigma(-\vec{w} \cdot \vec{c}')] \Big).
\end{aligned}$$

1) **(Preliminaries.)** The sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

i) Find an expression for $\sigma(-x)$ in terms of $\sigma(x)$.

Solution:

$$\sigma(-x) = \frac{1}{1 + e^{-(-x)}} = \frac{1}{1 + e^{x}}$$

$$
\begin{aligned}
&= \frac{e^{-x}}{e^{-x}+1} \\
&= \frac{e^{-x}+1}{e^{-x}+1} - \frac{1}{e^{-x}+1} \\
&= 1 - \sigma(x).
\end{aligned}
$$

ii) Calculate the derivative $\frac{d}{dx}\sigma(x)$.

Solution:

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -1 * (1+e^{-x})^{-2} * \frac{d}{dx}(1+e^{-x}) \\
&= -(1+e^{-x})^{-2} * (e^{-x} * -1) \\
&= \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \frac{1}{e^{x}+1} \\
&= \sigma(x)\sigma(-x).
\end{aligned}
$$

iii) Calculate the derivative $\frac{d}{dx}\log(\sigma(x))$. Simplify to an expression in terms of σ.

Solution:

$$
\begin{aligned}
\frac{d}{dx}\log\sigma(x) &= \frac{1}{\sigma(x)}\left(\frac{d}{dx}\sigma(x)\right) \\
&= \frac{1}{\sigma(x)}\sigma(x)\sigma(-x) \\
&= \sigma(-x).
\end{aligned}
$$

2) **(A simplified global objective.)** In general, $P_n(c)$ can be any appropriate "noise distribution" over the context vocabulary. Here, for simplicity's sake, we will consider the empirical unigram distribution: $P_n(c) = \frac{N_c}{N}$.

i) Recall that the expectation of a random variable X drawn from a discrete distribution over possible outcomes $\{x_1, x_2, ..., x_n\}$ is $\mathbb{E}[X] = \sum_{i=1}^{n} p(x_i)x_i$. Using the above definition of $P_n(c)$, write out the inner expectation in the SGNS loss function as a sum.

Solution:

$$\begin{aligned}\mathbb{E}[\log\sigma(-\vec{w}\cdot\vec{c}')] &= \sum_{c'\in V_c} p(c')(\log\sigma(-\vec{w}\cdot\vec{c}'))\\ &= \sum_{c'\in V_c}\frac{N_{c'}}{N}(\log\sigma(-\vec{w}\cdot\vec{c}')).\end{aligned}$$

ii) Reduce this into one double sum. Your final expression should look like

$$L = \sum_{w\in V_w}\sum_{c\in V_c}(\text{something here}).$$

Hint: What is $\sum_{c\in V_c} N_{w,c}$?

Solution:

$$\begin{aligned}L &= \sum_{w\in V_w}\sum_{c\in V_c} N_{w,c}\log\sigma(\vec{w}\cdot\vec{c}) + k\sum_{c'\in V_c}\frac{N_{c'}}{N}(\log\sigma(-\vec{w}\cdot\vec{c}'))\\ &= \sum_{w\in V_w}\sum_{c\in V_c} N_{w,c}\log\sigma(\vec{w}\cdot\vec{c}) + k\sum_{w\in V_w}\sum_{c\in V_c} N_{w,c}\sum_{c'\in V_c}\frac{N_{c'}}{N}(\log\sigma(-\vec{w}\cdot\vec{c}'))\\ &= \sum_{w\in V_w}\sum_{c\in V_c} N_{w,c}\log\sigma(\vec{w}\cdot\vec{c}) + k\sum_{w\in V_w} N_w\sum_{c'\in V_c}\frac{N_{c'}}{N}(\log\sigma(-\vec{w}\cdot\vec{c}')).\end{aligned}$$

Since we no longer have a c in the second term, we are free to rename our index c' to c:

$$\begin{aligned}L &= \sum_{w\in V_w}\sum_{c\in V_c} N_{w,c}\log\sigma(\vec{w}\cdot\vec{c}) + k\sum_{w\in V_w} N_w\sum_{c\in V_c}\frac{N_c}{N}(\log\sigma(-\vec{w}\cdot\vec{c}))\\ &= \sum_{w\in V_w}\sum_{c\in V_c} N_{w,c}\log\sigma(\vec{w}\cdot\vec{c}) + kN_w\frac{N_c}{N}(\log\sigma(-\vec{w}\cdot\vec{c})).\end{aligned}$$

3) **(Optimizing at the local level.)** If we allow our vectors sufficient dimensionality, we can assume the values of each $\vec{w}\cdot\vec{c}$ are independent. (Think about why this is the case.)

Under this assumption, we can optimize the global objective by individually optimizing the local objective for each (w, c):

$$l = N_{w,c}\log\sigma(\vec{w}\cdot\vec{c}) + k\cdot N_w\frac{N_c}{N}\cdot\log\sigma(-\vec{w}\cdot\vec{c}).$$

Note that this can be viewed as a function of a single scalar variable $x = \vec{w}\cdot\vec{c}$.

i) Find the derivative of l with respect to x.

Solution:

$$l = N_{w,c} \log \sigma(x) + k \cdot N_w \cdot \frac{N_c}{N} \cdot \log \sigma(-x).$$

Applying the derivatives we calculated in problem 1, we have

$$\frac{dl}{dx} = N_{w,c} \cdot \sigma(-x) + k \cdot N_w \cdot \frac{N_c}{N} \cdot \sigma(x).$$

ii) Set the derivative to zero and solve for x.

Solution:

$$\begin{aligned} 0 &= N_{w,c} \cdot \sigma(-x) - k \cdot N_w \cdot \frac{N_c}{N} \cdot \sigma(x) \\ &= N_{w,c}(1 - \sigma(x)) - kN_w \frac{N_c}{N} \sigma(x) \\ &= N_{w,c} - kN_w \frac{N_c}{N} \sigma(x) - N_{w,c}\sigma(x). \end{aligned}$$

Then

$$\begin{aligned} (kN_w \frac{N_c}{N} + N_{w,c})\sigma(x) &= N_{w,c} \\ \sigma(x) = \frac{1}{1+e^{-x}} &= \frac{N_{w,c}}{N_{w,c} + kN_w N_c/N} \\ 1 + e^{-x} = \frac{N_{w,c} + kN_w N_c/N}{N_{w,c}} &= 1 + \frac{kN_w N_c/N}{N_{w,c}} \\ e^{-x} &= \frac{kN_w N_c}{N_{w,c} N} \\ -x &= \log\left(\frac{kN_w N_c}{N_{w,c} N}\right) \\ x &= \log\left(\frac{N_{w,c} N}{kN_w N_c}\right). \end{aligned}$$

iii) Now substitute $\vec{w} \cdot \vec{c}$ back in for x. What is the optimal $\vec{w} \cdot \vec{c}$ in terms of $PMI(w, c)$?

Solution:

$$\begin{aligned} \vec{w} \cdot \vec{c} = \log\left(\frac{N_{w,c} N}{kN_w N_c}\right) &= \log\left(\frac{N_{w,c} N}{N_w N_c}\right) - \log k \\ &= PMI(w, c) - \log k. \end{aligned}$$

3 Submission instructions

Your writeup for section 1 and your solutions for section 2 should be submitted as a **single pdf file** on Gradescope. We prefer that this pdf be named `<your uni>.pdf`.

You should zip your solution code for section 1 (and README, if you have one) in a single folder named `<your uni>-hw3`, and submit the zip on CourseWorks. The data and helper scripts we provided you do not need to be included in this zip.

4 Academic integrity

Copying or paraphrasing someone's work (code included), or permitting your own work to be copied or paraphrased, even if only in part, is not allowed, and will result in an automatic grade of 0 for the entire assignment or exam in which the copying or paraphrasing was done. Your grade should reflect your own work. If you believe you are going to have trouble completing an assignment, please talk to the instructor or a TA in advance of the due date.

References

Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 350–356, Geneva, Switzerland, aug 23–aug 27 2004. COLING. URL https://www.aclweb.org/anthology/C04-1051.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. *ACM Transactions on information systems*, 20(1):116–131, 2002.

Anna Gladkova, Aleksandr Drozd, and Satoshi Matsuoka. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. In *Proceedings of the NAACL Student Research Workshop*, pages 8–15, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-2002.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2177–2185. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf.

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. doi: 10.1162/tacl_a_00134.