Learning about Word Vector Representations and Deep Learning through Implementing Word2vec

David Jurgens
School of Information
University of Michigan
jurgens@umich.edu

Abstract

Word vector representations are an essential part of an NLP curriculum. Here, we describe a homework that has students implement a popular method for learning word vectors, word2vec. Students implement the core parts of the method, including text preprocessing, negative sampling, and gradient descent. Starter code provides guidance and handles basic operations, which allows students to focus on the conceptually challenging aspects. After generating their vectors, students evaluate them using qualitative and quantitative tests.

1 Introduction

NLP curricula typically include content on word semantics, how semantics can be learned computationally through word vectors, and what are the vectors' uses. This document describes an assignment for having students implement word2vec (Mikolov et al., 2013a,b), a popular method that relies on a single-layer neural network. This homework is designed to introduce students to word vectors and simple neural networks by having them implement the network from scratch, without the use of deep-learning libraries. The assignment is appropriate for upper-division undergraduates or graduate students who are familiar with python programming, have some experience with the `numpy` library (Harris et al., 2020), and have been exposed to concepts around gradients and neural networks. Through implementing major portions of the word2vec software and using the learned vectors, students will gain a deeper understanding of how networks are trained, how to learn word vectors, and their uses in downstream tasks.

2 Design and Learning Goals

This homework is designed to take place just before the middle stretch of the class, after lexical semantics and machine learning concepts have been introduced. The content is designed at the level of an NLP student who (1) has some technical background and at least one advanced course in statistics and (2) will implement or adapt new NLP methods. This level is deeper than what is needed for a purely Applied NLP setting but too shallow for a more Machine Learning focused NLP class, which would likely benefit from additional derivations and proofs around the gradient descent to solidify understanding. The homework has typically been assigned over a three to four week period; many students complete the homework in the course of a week, but the longer time frame enables students with less background or programming experience to work through the steps. The material prepares students for advanced NLP concepts around deep learning and pre-trained language models, as well as provides intuition for what steps modern deep learning libraries perform.

The homework has three broad learning goals. First, the training portion of the homework helps deepen students' understanding of machine learning concepts, gradient descent, and develop complex NLP software. Central to this design is having students turn the equations in the homework and formal descriptions of word2vec into software operations. This step helps students understand how to ground equations found in some papers into the more-familiar language of programming, while also building a more intuition for how gradient descent and backpropagation work in practice.

Second, the process of software development aids students in developing larger NLP software methods that involve end-to-end development. This goal includes seeing how different algorithmic software designs work and are implemented. The speed of training requires that students be moderately efficient in how they implement their software. For example, the use of for loops instead of vectorized `numpy` operations will lead to a significant slow down in performance. In class instruction and

Proceedings of the Fifth Workshop on Teaching NLP, pages 108–111

tutorials detail how to write the relevant efficient numerical operations, which help guide students to identify where and how to selectively optimize. However, slow code will still finish correctly allowing students to debug for their initial implementations for correctness. This need for performant code creates opportunities for students to practice their performance optimizing skills.

Third, the lexical semantics portion of the homework exposes students to the uses and limitations of word vectors. Through training the vectors, students understand how statistical regularities in co-occurrence can be used to learn meaning. Qualitative and quantitative evaluations show students what their model has learned (e.g., using vector analogies) and introduce them to concepts of polysemy, fostering a larger discussion on what can be captured in a vector representation.

3 Homework Description

The homework has students implement two core aspects of the word2vec algorithm using `numpy` for the numeric portions, and then evaluate with two downstream tasks. The first aspect has students perform the commonly-used text preprocessing steps that turn a raw text corpus into self-supervised training examples. This step includes removing low-frequency tokens and subsampling tokens based on their frequency. The second aspect focuses on the core training procedure, including (i) negative sampling for generating negative examples of context words, (ii) performing gradient descent to update the two word vector matrices, and (iii) computing the negative log-likelihood. These tasks are broken into eight discrete steps that guide students in how to do each aspect. The assignment document includes links to more in-depth descriptions of the method including the extensive description of Rong (2014) and the recent chapter of Jurafsky and Martin (2021, ch. 6) to help students understand the math behind the training procedure.

In the second part of the homework, students evaluate the learned vectors in two downstream tasks. The first task has students load these vectors using the Gensim package (Rehurek and Sojka, 2010) and perform vector arithmetic operations to find word-pair analogies and examine the nearest-neighbors of words; this qualitative evaluation exposes students to what is or is not learned by the model. The second task is quantitative evaluation that has students generate word-pair similarity scores for the subset of the SimLex-999 (Hill et al., 2015) present in their training corpus, which is uploaded to Kaggle InClass[1] to see how their vectors compare with others; this leaderboard helps students identify a bug in their code (via a low-scoring submission) and occasionally prompts students to think about how to improve/extend their code to attain a higher score.

Potential Extensions The word2vec method has been extended in numerous ways in NLP to improve its vectors (e.g., Ling et al., 2015; Yu and Dredze, 2014; Tissier et al., 2017). This assignment includes descriptions of other possible extensions that students can explore, such as implementing dropout, adding learning rate decay, or making use of external knowledge during training. Typically, a single extension to word2vec is included as a part of the homework to help ground the concept in code but without increasing the difficulty of the assignment. Students who are interested in deepening their understanding can use these as starting points to see how to develop their own NLP methods as a part of a course project.

This assignment also provides multiple possibilities for examining the latent biases learned in word vectors. Prior work has established that pretrained vectors often encode gender and racial biases based on the corpora they are trained on (e.g., Caliskan et al., 2017; Manzini et al., 2019). In a future extension, this assignment could be adapted to use Wikipedia biographies as a base corpus and have students identify how occupations become more associated with gendered words during training (Garg et al., 2018). Once this bias is discovered, students can discuss various methods for mitigating it (e.g., Bolukbasi et al., 2016; Zhao et al., 2017) and how their method might be adapted to avoid other forms of bias. This extension can help students critically think about what is and is not being captured in pretrained vectors and models.

4 Reflection on Student Experiences

Student experiences on this homework have been very positive, with multiple students expressing a strong sense of satisfaction at completing the homework and being able to understand the algorithm and software backing word2vec. Several students reported feeling like completing this assignment was a great confidence boost and that they were

[1] https://www.kaggle.com/c/about/inclass

now more confident in their ability to understand NLP papers and connect algorithms, equations, and code. The majority of student difficulties happen in two sources. First, the vast majority of bugs happen when implementing the gradient descent and calculating the negative log-likelihood (NLL). While only a few lines of code in total, this step requires translating the loss function for word2vec (in the negative sampling case) into `numpy` code. This translation task appeared daunting at first for many students, though they found creating the eventual solution rewarding for being able to ground similar equations in NLP papers. Two key components for mitigating early frustration were (1) including built-in periodic reports of the NLL, which help students quickly spot whether there are numeric errors that lead to infinity or `NaN` values and (2) adding early-stopping and printing nearest neighbors of instructor-provided words (e.g., "January") which should be thematically coherent after only a few minutes of training. These components help students quickly identify the presence of a bug in the gradient descent.

The second student difficulty comes from the text preprocessing steps. The removal of low-frequency words and frequency-based subsampling steps require students to have a solid distinction of type versus token in practice in order to subsample tokens (versus types). I suspect that because many of these routine preprocessing steps are done for the student by common libraries (e.g., the `CountVectorizer` of Scikit Learn (Pedregosa et al., 2011)), these steps feel unfamiliar. Common errors in this theme were subsampling types or producing a sequence of word types (rather than tokens) to use for training.

References

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 2016. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS)*.

Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186.

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou. 2018. Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16):E3635–E3644.

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. 2020. Array programming with numpy. *Nature*, 585(7825):357–362.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.

Dan Jurafsky and James H. Martin. 2021. *Speech & Language Processing*, 3rd edition. Prentice Hall.

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

Thomas Manzini, Lim Yao Chong, Alan W Black, and Yulia Tsvetkov. 2019. Black is to criminal as caucasian is to police: Detecting and removing multiclass bias in word embeddings. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 615–621.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

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. *the Journal of machine Learning research*, 12:2825–2830.

Radim Rehurek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 workshop on new challenges for NLP frameworks*. Citeseer.

Xin Rong. 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2017. Dict2vec: Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 545–550.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 2017. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2979–2989.