Exploring Semantic and Syntactic Relationships between Word Vectors

Tyler Wengert
UCCS
twengert@uccs.edu

Abstract

As Natural Language Processing (NLP) has advanced and moved toward neural network implementations, we needed a better way to represent words that allowed us to use them efficiently with neural networks and capture more information about the relationships between words. With those two qualities in mind, the word vector has emerged as the preferred format for a large majority of NLP tasks.

1 Introduction

One of the first, and simplest, ways that researchers first derived word vectors was by creating an adjacency matrix, whose dimensions were V x V, where V is the number of words in the vocabulary, and the entry in a cell is the number of times that word V[i] has occurred after word V[j]. These matrices tended to be very sparse, since a large majority of words will not ever co-occur in a corpus. More recent approaches frequently take the neural network approach, with the vector being generated based on the nodes in the system, which produces dense vectors that do not have any filler 0's for words with which it does not co-occur. Since 2013, many researchers have extended or modified the approach originally laid out by (Mikolov et al., 2013). Pennington et al. (2014) laid out their own framework for computing dense word vectors, called GloVe, a year later. These two styles of word vector continue to dominate the NLP space today.

An extension of word vectors, document vectors are a single vector that represents an entire series of words. They are also frequently referred to as sentence vectors or paragraph vectors, depending on the size of the documents in a particular application. There have been several ways of creating document vectors, including summing its word vectors, taking a weighted average of its word vectors, and even concatenating vectors as in (Dai et al., 2015a). Řehůřek and Sojka (2010) created a module called doc2vec that computes document vectors from word2vec word vectors that is widely used as a part of the GenSim Python package.

2 Related Works

Undoubtedly, (Mikolov et al., 2013) and (Pennington et al., 2014) are the two foundational methods for generating word vectors as we use them today. They have enjoyed continued widespread use and serve as the basis for many cutting edge NLP programs. Rong (2014) provides a breakdown of the word2vec model for researchers looking to generate their own word vectors that is quite comprehensive and explains the formula and methodology of the origianl word2vec implementation in an easily readable manner. There are some very notable extensions and improvements on these methodologies, such as (Bojanowski et al., 2017), who represents words as a bag of characters and computes character level embeddings. They showed that these embeddings allow the model to work with words that it was not trained on, and outperform other cutting edge methods. Turney (2006b) propose ways to measure relational and attributional similarity among word vectors for a variety of embedding types.

3 Implementation

This program was built using Python 3.7, this time in the bulkier Spyder Anaconda environment, due to the large files needed for the word2vec vectors. The program uses the GenSim package to load in the word vectors from a format file provided by Google, and also makes use of NumPy for mathematics involving the vectors. The following sections describe, respectively, the way in which the program made a prediction to complete an analogy, and the way in which it scored sentence similarity.

3.1 Analogies

The prediction function was passed 3 parameters, a list of positive words, a list of negative words, and an integer value result which determined how many predictions were returned. Following from (Řehůřek and Sojka, 2010), words in the positive list were given a weight of 1, while those in the negative list were weighted -1. A word vector $\mathrm{v_i}$ was multiplied by its weight $\mathrm{w_i}$ and then from these new weighted vectors an average was taken, and the resulting vector was the prediction.

$$\frac{\sum_{i=0}^{size(positive[]+negative[])-1}(v_{\mathrm{i}} * w_{\mathrm{i}})}{size(positive[] + negative[]) - 1}$$

When the program was run on the Google analogy set, assuming that the first word is A, the second B, the third C, and the fourth D, words A and C were passed to predict() in the positive field, and word B in the negative field. Word D was not passed to the predict function, but was kept to compare to the output prediction.

3.2 Sentence Comparison

For this task, the program needed a way to generate sentence vectors as well as a way to score them. Firstly, to generate the vectors, the program implemented a simplified version of the methodology used by (Schmidt, 2019). Stop-words ('the', 'and', 'a', etc.) were given a weight of .5, while other words were given a weight of 2, then the mean of the weighted word vectors was computed to come to the final sentence vector.

$$\frac{\sum_{i=0}^{size(sentence[])-1}(v_{\mathrm{i}} * w_{\mathrm{i}})}{size(sentence[]) - 1}$$

Once the sentence vectors were obtained, they needed to be compared. To do this, the cosine similarity was computed between the two target sentence vectors, $\mathrm{s_i}$ and $\mathrm{s_j}$.

$$cos(x) = \frac{s_{\mathrm{i}} \cdot s_{\mathrm{j}}}{||s_{\mathrm{i}}|| * ||s_{\mathrm{j}}||}$$

The result was normalized to 1, so it was simply multiplied by 5 to fit the scale of the examples.

4 Results

4.1 Analogies

There were a total of 19,545 analogy examples in the provided file. The computer was only given a point for a correct answer if it predicted the exact word that was expected. Words having the same base but the incorrect tense, or that used a plural instead of a singular, were counted as incorrect. Overall, the program correctly completed 6,211 of the analogy phrases, or about 31.7%. The program performed best in the plurality category, where it correctly completed 972 out of 1332 (72.9%) total analogy phrases. It performed worst on the city/state examples where it completed 515 out of 2468 (20.86%) total analogy phrases.

4.2 Sentence Comparison

There were a total of 750 sentence pairs with a human generated score to test the program on. After the computer came up with its score, it was compared to the human score and considered correct if it was within a margin of ±0.2 The program scored the sentence pairs correctly 299 out of 750 times, or 39.8%.

5 Conclusions

Between the two tasks, the computer had a harder time with the analogy completion. This is likely due to the fact that the sentence comparator uses simpler logic to compare the document vectors. It is also likely that since the program for the analogies had to average 3 vectors instead of just two, there was less likely to be error. Among the different analogy types, the algorithm seemed to capture syntactic data like the relationship between singular and plural forms or between different tenses of the word better than semantic relationships such as a country and its capital or a city and its state. The semantic relationships certainly constitute a deeper level of understanding to capture, and so are therefore difficult for the computer. During the sentence task, even though the algorithm only matched correctly matched the human score 39% of the time, many of the incorrect responses were within a small margin (variance of 0.3 - 0.9) of the human given response. This demonstrates that the program is capturing a great deal of information from the sentence vectors.

6 Further Work and Improvements

The weighting algorithm used in the sentence task was a simple one, and many cutting edge techniques have found that they can improve results drastically through the use of more in depth weighting techniques. Words that are not stop words could be broken down into further categories and weighted more or less depending on importance to the sentence. Checks could be made to identify words at the start of the sentence, named entities, and important verbs that could each receive a unique weight based on its class. This would result in sentence vectors that represent more of the most vital information inside while filtering out less important words. For the analogies task, a cosine similarity check was considered as the prediction method, but an averaging algorithm was used instead. It is possible the the use of cosine similarity, or the combination of cosine similarity and weighted averaging could provide better results.

7 Acknowledgements

The following are papers that were referenced, but not cited specifically in this paper. For analogy prediction: (Turney, 2006a). For document vectors: (Dai et al., 2015b)

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5:135–146.

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

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

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

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

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

Craig W Schmidt. 2019. Improving a tf-idf weighted document vector embedding. arXiv preprint arXiv:1902.09875.

Peter D Turney. 2006a. Expressing implicit semantic relations without supervision. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, pages 313–320. Association for Computational Linguistics.

Peter D Turney. 2006b. Similarity of semantic relations. Computational Linguistics, 32(3):379–416.