Exploring Optimizations to Paragraph Vectors

Gabriele Fisher
Stanford University
Department of Computer Science
gsfisher@stanford.edu

Maya Israni
Stanford University
Department of Computer Science
mayai@stanford.edu

Zoe Robert
Stanford University
Department of Computer Science
zrobert7@stanford.edu

Abstract

This paper replicates the results of Dai, Olah, and Le's paper "Document Embedding with Paragraph Vectors" and compares the performance of three unsupervised document modeling algorithms [1]. We built and compared the results of Paragraph Vector, Latent Dirichlet Allocation, and traditional Word2Vec models on Wikipedia browsing. We then built three extensions to the original Paragraph Vector model, finding that combinations of paragraph structures assist in optimizing Paragraph Vector training.

1 Introduction

The optimization of similarity detection between documents–and those implications for vector representations of text–serve as an open question in the field of Natural Language Processing. Improving vector representations is a popular goal among contemporary NLP papers, and recent contributions to this field frequently utilize Paragraph Vectors. Well-known vector generation models like Word2Vec now compete with the likes of Paragraph Vectors.

First introduced by Le and Mikolov in 2014, a paragraph vector is an unsupervised framework that can learn a variable length of text. In this model, the paragraph vector is concatenated with numerous word vectors from a paragraph to predict the following word. The model uses stochastic gradient descent and back propagation to train the word vectors and Paragraph Vectors. The word vectors are shared; the Paragraph Vectors are unique [2].

2 Background/Related Work

Paragraph Vectors were first introduced by Le and Mikolov's 2014 paper, "Distributed Representations of Sentences and Documents." The paper outlines several variations of the Paragraph Vector representations: Distributed Bag of Words version of Paragraph Vector (PV-DBOW), Distributed Memory version of Paragraph Vector (PV-DM), and a model that is a concatenation of the two [2]. When trained on a fairly large IMBD dataset these vectors obtain a lowest error rate of 3.82%. In comparison, Le and Mikolov achieved a 10.25% error rate for vector averaging, 8.10% for bag-of-words, 7.28% for bag-of-bigrams, and 5.67% for weighted bag-of-bigrams.

In 2015, Dai, Olah, and Le published "Document Embeddings With Paragraph Vectors," in which they compared Paragraph Vectors to a number of other document representations including LDA, Bag of Words, and averaged word vectors on two semantic analysis tasks: Wikipedia and arXiv article browsing. They supported the 2014 paper by showing that paragraph vectors outperformed

every other representation with a high accuracy of 93% on a full Wikipedia corpus data set and extended the usability of paragraph vectors as effective in local and non-local browsing of large corpora.

Since then, a number of additional academic papers have been produced that study variations on the paragraph vector model. Grzegorczyk and Kurdziel's 2016 paper, "Binary Paragraph Vectors," extended Paragraph Vectors by introducing a sigmoid nonlinearity before the softmax that predicts words in documents [3]. Their introduced Binary Paragraph Vectors were generated by simple neural networks that learned short binary codes for fast information retrieval. Grzegorczyk and Kurdziel found that binary paragraph vectors outperform autoencoder-based binary codes, despite using fewer bits.

Palangi, Deng et al.'s 2015 paper, "Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval," found that the use of bidirectional-LSTM-RNNs demonstrated an improvement of 5.2% over Paragraph Vector usage in information retrieval. Given modifications to and contemporary interest in Paragraph Vector representations in NLP, and the improvement of PV within NLP-related fields from document clustering to information retrieval, we examine possible optimizations for PV.

3 Approach

For our research, we chose to replicate the results above from Dai, Olah, and Le's 2015 paper "Document Embeddings With Paragraph Vectors." Beyond establishing comparative models with Latent Dirichlet allocation (LDA) and Word2Vec with CBOW, we sought to replicate the accuracy of Paragraph Vectors and to extend a baseline PV in an attempt to optimize the accuracy of PV in topic clustering.

We chose several variations of paragraph vectors mentioned both in Dai, Olah, Le's paper and in Le, Miklov's paper. These include three singular paragraph vector models (DMC, DMM, DBOW) and two concatenated models that combine the singular representations. DMC and DMM represent the Distributed Memory Paragraph Vector model with concatenated or averaged paragraph vectors and word vectors, respectively. Dai, Olah, Le suggest that DMC outperforms DMM and DBOW as a singular PV model. They also found that while the singular Distributed Memory Paragraph Vector model works well for most tasks, in combination with the Distributed Bag of Words Paragraph Vector model (DBOW) it performs more consistently across many tasks. We sought to test these predictions and experiment with these optimizations to the basic PV model.

As an additional optimization, we experimented with the training data size and concatenation of multiple models. We split the training data into two chunks, trained each of these chunks with the doc2vec model, and concatenated the two subsequent vectors whilst testing. This 'chunk2vec' model was thoroughly experimental, and we therefore did not have predictions for this model's performance.

We chose to use the gensim python library to conduct our model training.

4 Experiments

4.1 Dataset

During the development stage, we used a small Wikipedia subset of about 171 MB representing about 8% of all of Wikipedia. For final testing, we moved to a larger subset of Wikipedia of about 3.6 GB representing about a quarter of all of Wikipedia. The datasets were scraped of casing and punctuation and then formatted using the gensim WikiCorpus class.

4.2 Evaluation Metric

The goal of our experiments was to compare the performance of various article vector representations. Ideally, two similar articles should have similar vector representations. To test this, we used a triplet system as our evaluation metric. We used the publicly available hand selected triplets from

Dai, Olah, Le. Similarity between documents was measured using cosine similarity for doc2vec, word2vec and Hellinger distancce for LDA. The formula to determine accuracy is as follows:

$$\text{accuracy} = \frac{\text{\# triplets where articles 1 and 2 are more similar than articles 2 and 3}}{\text{total \# triplets}}$$

4.3 Baseline Models

Our baseline models included word2vec with CBOW, LDA, and Paragraph Vectors as implemented by Le and Mikolov. Mirroring the results of Le and Mikolov's 2014 paper, our PV outperforms Word2Vec and LDA on our evaluation, as shown in Figure 1. Our PV implementation, also referred to as doc2vec, did not perform as well as Le and Mikolov's implementation; while our maximum accuracy was 80% for our PV implementation, Le and Mikolov achieved a max accuracy of 93% in their 2015 PV topic clustering paper.

Our degraded performance can be potentially explained by our space restraints on our GPU. Given limited memory, we could not feasibly store multiple models, their relevant files, and the entirety of the Wikipedia corpus on our machine at once. Le and Mikolov used the entirety of the Wikipedia corpus to train their model, while we only used a fourth of the Wikipedia corpus. Therefore, Le and Mikolov's model may have captured more nuances of text, given that it ingested more training data.

Space constraints also affected the size of the vectors we could save as the requisite Docarrays needed for gensim model construction. While we used vectors of dimension 400 to represent Wikipedia articles, as done in Le and Mikolov's 2014 PV paper, Dai, Olah, and Le's 2015 paper used paragraph vectors of dimension 10000. These larger vector representations of articles could contribute to even more precise representations of features in documents.

Given a GPU with more ample memory to store multiple models at once for comparative purposes, we believe we could have made further strides towards reaching Le and Mikolov's reported accuracy in their 2015 paper.

Representation	Accuracy
LDA	0.715116
word2vec	0.773256
doc2vec	0.796512

Figure 1: Baseline Accuracies on Big Wikipedia Subset

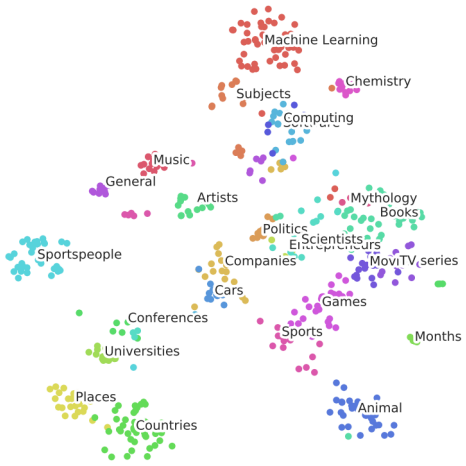

Figure 2: Averaged Word Vecs (word2vec) model results on Wikipedia dataset

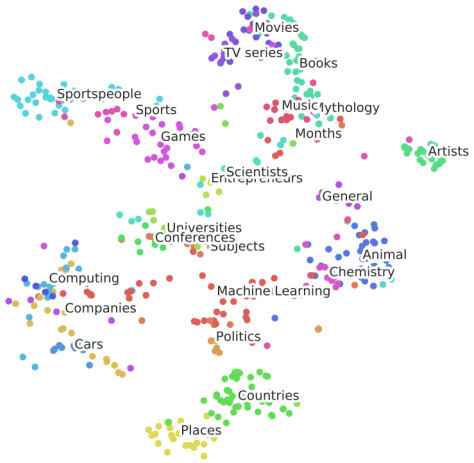

Figure 3: LDA model results on Wikipedia dataset

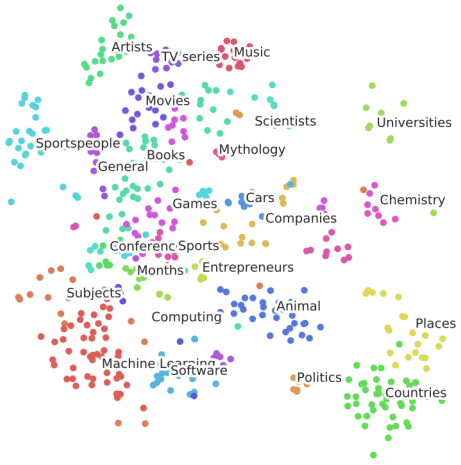

Figure 4: Basic doc2vec model results on Wikipedia dataset

4.4 Doc2Vec Concatenation Variations

In the original paragraph vector paper by Le and Mikolov, the authors suggest several variations of paragraph vector representations, including a concatenation of multiple trained models. We built several improvements on top of the most basic paragraph vector implementation. These models were run with a minimum word count of 2, a feature size of 100, and a window size of 5 to 10. The variations are as follows: instead of summing context word vectors we use the mean (DMM), instead of summing/ averaging we concatenate context word vectors (DMC), ignore context words and force the model to predict words randomly sampled from the paragraph (DBOW), and concatenations of each DM model with DBOW.

doc2vec Variation	Accuracy
DMM	0.761628
DMC	0.796512
DBOW	0.825581
DBOW+DMM	0.808140
DBOW+DMC	0.813953

Figure 5: Performance on the Wikipedia subset

Of these, we selected one singular and one concatenated model to represent in similarity scatter plots. It is clear from the scatter plots that the concatenated model DBOW+DMC produced more distinct and spread paragraph vector representations for each topic. Averaging the context word vectors in DMM allows for information loss, likely causing the drop in performance.

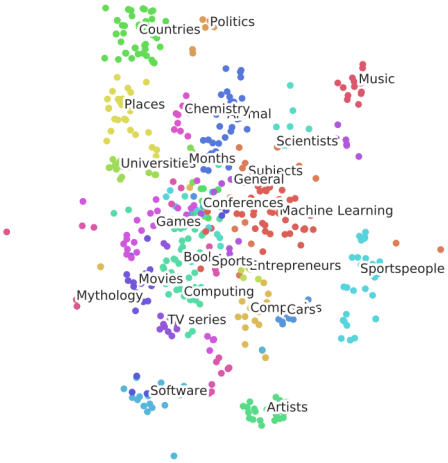

Figure 6: DMM model results on Wikipedia datase

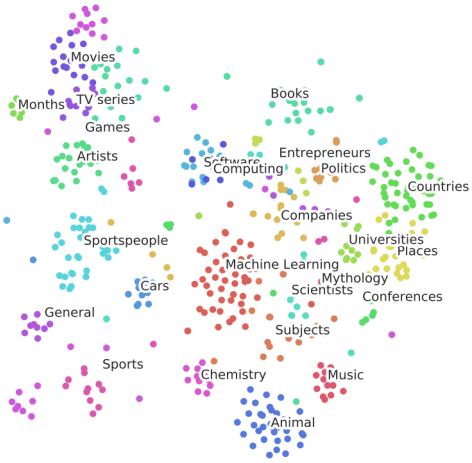

Figure 7: DBOW+DMC model results on Wikipedia datase

4.5 Doc2Vec Concatenated-Chunks Variation

As an additional variation, we split the original Wikipedia dataset into two equal-sized text files. We trained each of these two chunks with our Doc2Vec model and concatenated the two models when evaluating. This concatenated-chunk design achieved a 0.819767 accuracy.

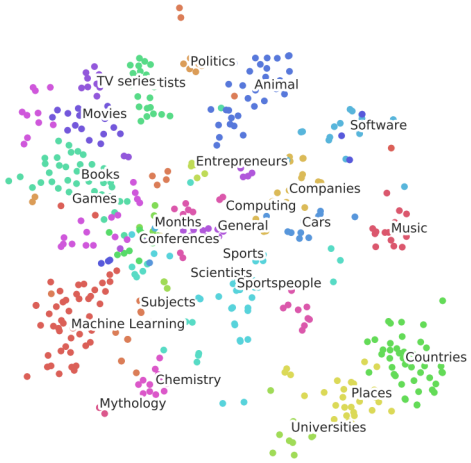

Figure 8: Concatenated chunks doc2vec model results on Wikipedia datase

4.7 Analysis

Dai, Olah, and Les paper trained all models on the entire English Wikipedia database. The paper achieved 0.849, 0.82, and 0.93 on word2vec, LDA, and doc2vec models, respectively. In comparison, our models trained on one-fourth of the English Wikipedia database and corresponded to Dai, Olah, and Les relative rankings of these accuracies.

Our paragraph vector model and all subsequent optimizations outperformed the baseline models we implemented. As seen in its graph representation, the word2vec model more tightly clusters vectors than doc2vec. This is because word2vec focuses on individual word appearances and limited window contexts. Doc2vec, however, incorporates n-gram-like order-sensitive collections of words in addition to broader co-occurrences in the corpus. The clusters in the graph for the doc2vec DBOW model are more spread out and reflect the best accuracy achieved.

Dai, Olah, Le suggest that the Distributed Memory model (DM) performs well because it remembers the topic of a paragraph while learning the current context and that concatenation consistently outperforms averaging. Our results support this finding. However, they also found that DM is consistently better than DBOW, while we saw opposite results.

In the graphs above, it is worth noting that word2vec produces clusters tighter than those produced by doc2vec. This could imply that word2vec produces vectors strongly influenced by the presence of extremely similar–or even the same–language. doc2vec, on the other hand, produces clusters that overlap with each other while maintaining relationships with closely related clusters (for example, examine the close proximity between the Computing, Machine Learning, and Software clusters). This implies that doc2vec can capture context more effectively than word2vec.

We believe that DBOW's accuracy readings of 83% in the task of topic clustering, the best results we achieved and results superior to those produced by Le and Mikolov's baseline model of doc2vec, can be potientially explained by DBOW's sampling of random words over windows of text in a document. Similarities between commonly used subsets of words in an article–as opposed to precise n-gram patterns–allows for similar vector representations among two articles that pertain to similar topics and consequently share vocabularies and word co-occurences within a window. This explains why clustering in graphs using DBOW, such as Figure 7, is tight like that in word2vec, but still allows for significant overlaps between topic clusters that touch upon similar content (for example, the overlap between computing and software).

5 Conclusion

Our results aligned with those of the paper we replicated. Paragraph Vectors outperformed in measuring semantic similarity of Wikipedia articles in comparison to the LDA and Word2Vec models. Our optimizations of Paragraph Vectors, particularly the PV-DBOW model, outperformed the traditional Paragraph Model. This improved accuracy suggests opportunities for further optimizations.

If given additional time and resources, we would experiment with testing and iterations of our hyperparameter values, including the minimum frequency count of words, number of features, and number of topics on our models. We would also test our optimized doc2vec models on the entire Wikipedia corpus and implement additional variations. Our team also strove to build an LSTM-Seq encoder for Paragraph Vectors in an attempt to further train PV representations, though this involved implementation proved unfeasible given the difficulty of projecting unsupervised model-building onto a set of methods typically used for supervised learning and time constraints for this project. If allotted more time for this project, we would continue developing our LSTM-Seq encoder. Progress towards a LSTM-Seq remains on our GitHub repository.

This paper ultimately reinforces the Paragraph Vector model as a promising unsupervised learning algorithm for semantic analysis of larger bodies of text.

Acknowledgments

This research was supported by Stanford University's Natural Language Processing with Deep Learning course, CS224N. We thank our mentor, Arun Chaganty, who provided insight and expertise that greatly assisted the project..

References

[1] Dai, A.M., Olah, C. & Le, Q.V. (2015) Document Embedding with Paragraph Vectors.

[2] Le, Q., Mikolov, T. (2014) Distributed Representations of Sentences and Documents. In *International Conference on Machine Learning*.

[3] Grzegorczyk, Karol; Kurdziel, Marcin. "Binary Paragraph Vectors." November 2017. Under review as a conference paper at *ICLR 2017*. arXiv:1611.01116.

[4] Palangi, Hamid; Deng, Li; Shen, Yelong; Gao, Jianfeng; He, Xiaodong; Chen, Jianshu; Song, Xinying; Ward, Rabab. "Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval." February 2015. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. arXiv:1502.06922.

[5] Metz, Luke. "Visualizing with T-SNE." Indico. Web. 25 Aug. 2015. https://indico.io/blog/visualizing-with-t-sne/

Contributions

Gabbi: data collection and scraping of training set; research into implementation of LSTM on PVs; t-SNE visualizations of model topic clustering.
Maya: data scraping and cleaning of Wikipedia articles test set; doc2vec chunks optimization implementation; research into implementation of binary paragraph vectors; training/tuning word2vec models
Zoe: word2vec averaged word vector implementation; LDA evaluation script; doc2vec modifications with DBOW, DMM, and DMC;