Improving Paragraph2Vec

Seokho Hong
seokho@stanford.edu

Abstract

Paragraph vectors were proposed as a powerful unsupervised method of learning representations of arbitrary lengths of text. Although paragraph vectors had the advantage of being versatile, being unsupervised and unconstrained by lengths of text, the concept has not been further developed since its first publication. We propose two extensions upon the initial formulation of the paragraph vector, and test its performance on two separate semantic-based tasks. Although the results are limited by the fact that our attempt to reproduce the original paragraph vectors was not successful, we can still show that the extended models outperform the original paragraph vectors.

1 Introduction

The Paragraph Vector (Le & Mikolov 2014) was proposed with the objective of unsupervised semantic learning of arbitrary lengths of texts. While the PV-DM and PV-DBOW models proposed perform very well, their formulations leave room for improvement. In particular, the performance differences between PV-DM and PV-DBOW suggest that improvements to PV-DM could yield be significant. Also, the paper finds that PV-DM and PV-DBOW vectors concatenated achieve the best performance. Even if different training methods do not necessarily produce better vectors, concatenating them with the original paragraph vectors may offer better results than the PV-DM and PV-DBOW concatenated alone.

The main mathematical limitation of the PV-DM and PV-DBOW models is that they do not allow the paragraph vector to interact in complex, non-linear ways with the word vectors. While this design is not entirely unjustified, since different sections of text do not radically change the distribution and sequence of English words, there is nonetheless a limitation on the extent to which a paragraph vector can assist in the word-prediction task the paragraph vector uses.

We propose two different formulations of training unsupervised paragraph vectors, which we will call the hidden layer model and the tensor model. While both models offer improvements upon the original paragraph models, we cannot make definitive conclusions because we were unable to replicate the same level of performance reported in (Le & Mikolov 2014).

2 Background

Recent works have proposed various deep methods for learning the semantics of sentences. Recursive neural networks with various substructures (Socher et al. 2013), (Tai et al. 2015) and convolutional neural networks (Kalchbrenner et al. 2014), offer excellent performance that approximately matches or exceeds that of paragraph vectors across different tasks. These models, however, tend to be far more complex and deeper than the paragraph vector method and therefore take significantly longer to train. They also have other limitations that could potentially limit their range of applications. Recursive models work only for sentences and need a parser, which is not

only an extra requirement, but also a source of errors given that parsers are imperfect. Convolutional neural networks as suggested in (Kalchbrenner et al. 2014) do not need a parser, but remain untested for longer pieces of text. The max-pooling layer the paper proposes appears as though it will be less effective as the text gets much longer.

The paragraph vector has the advantage of simplicity and versatility. Perhaps it is a bit too simple. The models suggested here were inspired by the recent gains from the increasingly complex models. It does not appear unreasonable to train modestly more complex models along the lines of the paragraph vector framework if it will result in performance gains.

3 Approach

Paragraph Vector Framework

The paragraph vector framework is the general approach to training unsupervised paragraph vectors, and is common to the models discussed here as well as the original PV-DM. The framework is a word prediction task. Given a set of words $w_1, w_2, \cdots, w_n$, the model trains by predicting one of the words' vector, w_j given the other $n-1$ words' vectors. The model also takes a paragraph vector p_i where i identifies which body of text $w_1, w_2, \cdots, w_n$ come from.

The original authors proposed both hierarchical softmax and negative sampling as a replacement for the traditional but expensive softmax for the objective function, but we will use only negative sampling here. The cost function is therefore:

$$\log(\sigma(r \cdot v_{w_j})) - \sum_i \log(\sigma(r \cdot v_{w_i})) \tag{1}$$

Where r is the predicted vector, w_i is the vector of a random word, for which there are k random words. For the models trained here, k is fixed at 10.

The training is done via backpropagation through structure using Adagrad.

In training, both the word vectors and paragraph vectors are initialized randomly with values in the range of -0.01 to 0.01.

Hidden Layer Model

The original PV-DM model has the following equation for r:

$$r = \sigma(W \cdot [c; p_i]) \tag{2}$$
$$\tag{3}$$

where c is the concatenation of the input word vectors for the prediction task.

The hidden layer model we propose simply adds another layer between r and the two input vectors.

$$z = \sigma(W \cdot [c; p_i]) \tag{4}$$
$$r = \sigma(U \cdot z) \tag{5}$$
$$\tag{6}$$

While simple, it allows the components of p_i and c to interact in more complex ways. The original equation for r allows only a single non-linear transformation of a linear function of p_i and c.

Tensor Model

To allow even more interaction between the components of p_i and c, we propose the following:

$$r = \sigma(c \cdot T \cdot p_i + W \cdot [c; p_i]) \tag{7}$$
$$\tag{8}$$

where T is a tensor.

3.1 Training

Given a collection of text that can be divided into n documents, or paragraphs, each paragraph is assigned a vector. Training involves sliding the window of context words across each word of each paragraph, for every paragraph. At each data point, the input is the window of context words, and the paragraph vector corresponding to the origin of the words, and the target output is a particular word within or adjacent to the context words (depending on the hyper parameters). The target output is obviously not given as input. At either end of the paragraph, a special "NULL" word is applied as needed to fill the necessary window space. The "NULL" word is trained like every other word.

Backpropagation is used to train both the paragraph vectors and the word vectors simultaneously.

3.2 Testing

Testing involves running gradient descent to train new paragraph vectors for each new paragraph. At test time, all parameters of the model are frozen, including the word vectors, and the backpropagation is only applied to the paragraph vectors.

4 Experiments

We tested the two new models on two fairly standard tasks, one on which other models have been benchmarked: sentiment analysis and semantic textual similarity.

4.1 Sentiment Analysis - Stanford Sentiment Treebank

Each phrase in the treebank was treated as a separate paragraph, and training and testing was done according to the dataset's specifications. In an attempt to reproduce the results from (Le and Mikolov, 2014), we tried to match the hyperparameters as closely as possible.

4.1.1 Hyperparameters for PV-DM

Word vector dimensions: 100; paragraph vector dimension: 400; context window size: 7 words, predict 8th; trained using Adagrad with 0.01 learning rate, minibatches of size 300, L2 regularization of 1e-4. Trained for approximately 10 hours.

4.1.2 Hyperparameters for Hidden Layer Model

Word vector dimensions: 100; paragraph vector dimension: 400; context window size: 7 words, predict 8th; trained using Adagrad with 0.01 learning rate, minibatches of size 300, L2 regularization of 1e-4. Trained for approximately 16 hours.

4.1.3 Hyperparameters for Tensor Model

Word vector dimensions: 100; paragraph vector dimension: 200; context window size: 7 words, predict 8th; trained using Adagrad with 0.005 learning rate, minibatches of size 300, L2 regularization of 1e-3. Trained for approximately 48 hours.

The tensor model training becomes very expensive with large word or paragraph vectors so the paragraph vector dimensionality was reduced.

Final Classification

After pre-training the paragraph vectors (both for training and testing), a random forest classifier was used to predict the final sentiment label for each paragraph. On the fine-grained task, each paragraph was classified in categories from 1 to 5, where the original sentiment is scaled up from 0.0 to 1.0. On the binary task, each paragraph rated in categories 2 and 4 were used. We used the random forest classifier from the sci-kit learn package, with 100 trees as the only modification from the default hyperparameters.

Table 1: SST Results

Method	Fine-Grained	Binary
PV-DM	41.6	81.2
PV-Hidden	42.7	82.2
PV-Tensor	44.1	84.3
PV-DM (Le and Mikolov, 2014)	48.7	87.8

5 Attempted Optimization

We made many attempts to fully reproduce the results from (Le and Mikolov, 2014). The only major difference is that we reduced the word vector dimensionality to 100 from 400, but this reduction was necessary to allow the model to train in a reasonable amount of time. Training a model with 200 dimensions did improve the results, but not enough to suggest that the reduction from 400 is responsible for the disparity in results.

Training duration was determined by convergence on the development set (10% randomly sampled from the training set). If there was no improvement over 10 epochs, or passes through the entire training set, then the training was halted.

Dropping out for the Hidden Layer model ($p = 0.5$ on W) and Tensor model ($p = 0.2$ on T) was used.

5.1 Semantic Textual Similarity - SemEval 2014 Task 1

While (Le and Mikolov, 2014) did not benchmark their PV-DM on this task, many other papers benchmark their neural networks on this dataset, suggesting that the dataset is a reliable one.

The dataset here is the SICK dataset, 10000 pairs of English sentences, each labeled with a semantic similarity score from 1 to 5. There is a Train, Test, and Trial division of the dataset, and each were used respectively for training, testing, and validating the models.

5.1.1 Hyperparameters for PV-DM

Word vector dimensions: 200; paragraph vector dimension: 400; context window size: 9 words, predict 10th; trained using Adagrad with 0.01 learning rate, minibatches of size 300, L2 regularization of 1e-4. Trained for approximately 6 hours.

5.1.2 Hyperparameters for Hidden Layer Model

Word vector dimensions: 200; paragraph vector dimension: 400; context window size: 9 words, predict 10th; trained using Adagrad with 0.01 learning rate, minibatches of size 300, L2 regularization of 1e-4. Trained for approximately 10 hours.

5.1.3 Hyperparameters for Tensor Model

Word vector dimensions: 200; paragraph vector dimension: 200; context window size: 9 words, predict 10th; trained using Adagrad with 0.005 learning rate, minibatches of size 300, L2 regularization of 1e-3. Trained for approximately 26 hours.

Final Classification

After pre-training the paragraph vectors (both for training and testing), a random forest regressor was used to predict the final sentiment label for each paragraph. We used the random forest regressor from the sci-kit learn package, with 100 trees as the only modification from the default hyperparameters.

Table 2: Semantic Textual Similarity Results

Method	MSE
Mean Vectors (Tai et al, 2015)	0.455
LSTM (Tai et al, 2015)	0.281
PV-DM	0.392
PV-Hidden	0.388
PV-Tensor	0.365

The Mean vectors baseline in (Tai et al, 2015) computes a semantic relatedness score from the average of the word vectors of the words in the sentence. While the LSTM is not the focus of the paper, it is one of the most effective models for the task, and puts the paragraph vector models in perspective, at least our implementations.

6 Attempted Optimization

Since this example used the same implementation as the one used for the Stanford Sentiment Treebank, no doubt there are some flaws holding back the scores here.

The SICK dataset is smaller than the previous one, which allowed slightly larger models to be trained in a reasonable time.

Training duration was determined by convergence on the development set, as specified in the dataset. If there was no improvement over 10 epochs, or passes through the entire training set, then the training was halted.

Dropping out for the Hidden Layer model ($p = 0.5$ on W) and Tensor model ($p = 0.2$ on T) was used.

7 Conclusions

Despite the subpar results on the PV-DM implementation, it is safe to say that additional hidden layer and the tensor models improve upon the original PV-DM formulation. The hidden layer model

probably is not worth the additional training time required for the slight performance gains. The tensor model is much better, but it takes even longer to converge.

The performance of the original PV-DM showed that modeling the distribution and ordering of a paragraph's words, is an effective method of modeling the semantics of a sentence or any piece of text. The performance of the hidden layer and tensor models shows that there is room for improvement by allowing the paragraph vector to learn a more complex function for how the paragraph influences the distribution of words.

Regarding direct improvement of the model, perhaps a deeper, more complex model would further improve the performance of paragraph vectors, but at that point it would no longer be the fairly simple model that trains quickly compared to other deep models.

For improving performance on particular tasks such as sentiment analysis, models that train directly supervised representations of text are outperforming the unsupervised paragraph vector. It intuitively seems obvious that deep methods that optimize paragraph representations for a particular task will outperform a shallow classifier on an unsupervised paragraph representation. It is also not possible to fine tune the unsupervised paragraph vectors on a specific task unless all the paragraphs are available during training time, which does not prove generalization. Thus improving the paragraph vector is probably not the best method for attempting to improve upon state of the art performance on separate tasks.

References

[1] Kalchbrenner, Nel, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. *ACL14*

[2] Le, Quoc V and Tomas Mikolov. 2014. Dis- tributed representations of sentences and doc- uments.*arXiv preprint arXiv*

[3] Tai, Kai Sheng, Richard Socher, Christopher D. Manning 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks uments.*arXiv preprint arXiv*

[4] Socher, Richard, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank uments.*EMNLP 2013*