Learning Distributed Representations of Phrases

Konstantin Lopyrev
klopyrev@stanford.edu

December 12, 2014

Abstract

Recent work in Natural Language Processing has focused on learning distributed representations of words, phrases, sentences, paragraphs and even whole documents. In such representations, text is represented using multi-dimensional vectors and similarity between pieces of text can be measured using similarity between such vectors. In this project I focus my attention on learning representations of phrases - sequences of two or more words that can function as a single unit in a sentence.

1 Introduction

Representing arbitrary text as multidimensional vectors $w \in \mathbb{R}^k$ has many uses. Such representations can be used directly for computing similarity between text by, for example, taking the cosine similarity between the two vectors. They can also be fed into other algorithms as compact representations that have many nice properties. Finally, combined with techniques like locality sensitive hashing, distributed representations can be used for information retrieval. For this project I focus my attention on computing distributed representations for phrases. I learn such representations from free-form text and evaluate them on 2 text similarity test sets.

My method consists of two phases. In the first phase I use one of two algorithms with published source code to learn representations for words and some common phrases. In the second phase, I learn how to combine the representations of the words in a phrase into a representation for the whole phrase, allowing me to compute representations for all possible phrases.

2 Data Set Description

I train my algorithms using two data sets. The first data set is the English Gigaword New York Times Newswire Service corpus that is available for download from the Stanford Linguistics department. This data set contains news articles spanning the years 1994 to 2006, totaling 1.2 billion words. The second data set that I use is a large phrase dictionary. I collect phrases from 2 different sources:

- I download the list of Wikipedia article titles. This list contains 9.7 million phrases, including book and movie titles, names of places, names of high-level concepts, and so on.
- I scrape all the phrases from thesaurus.com and from urbandictionary.com. Theusarus.com and urbandictionary.com contain 97 thousand and 650 thousand phrases, respectively, including idiomatic phrases, commonly used noun and verb phrases, and so on.

I combine these two data sets into a single training set as follows:

1. I apply the Stanford tokenizer on the news corpus to strip out all markup, to lowercase all letters, and to apply several PTB3 token transforms not described here for conciseness.
2. I add and remove newline characters so that each paragraph ends up on a separate line.
3. I use the phrase dictionary to identify phrases in the text that I then merge into a single token using underscores. For example, the words *abrupt change* become a single token *abrupt_change*.
4. I shuffle all the lines so that they are ordered randomly.

The resulting training set has a vocabulary that contains 587 thousand words and 545 thousand phrases, counting only vocabulary entries that occur at least 5 times.

3 Models

3.1 Phase 1: Learning representations for words and dictionary phrases

During the first phase of learning I apply one of two algorithms for learning vector representations for words and dictionary phrases from the training set. Both of these algorithms have source code available for download online. These algorithms learn representations for tokens using the other tokens that appear in context, following a principle in linguistics best stated by the linguist John Firth: "You shall know a word by the company it keeps."

3.1.1 Glove: Global Vectors for Word Representation

The Glove model [5] learns from token co-occurrences. To train the Glove model I first preprocess the training set by computing the number of times each token appears in the context of each other token. Specifically, I pick a window size W and then compute the number of times each pair of tokens occurs in the same paragraph with at most $W-1$ other tokens between them. Each co-occurence is weighed by the inverse of the distance between the tokens. The result is a matrix $X \in {\mathbb{R}_+}^{V \times V}$ where V is the number of distinct tokens.

The Glove model represents each token i using two multi-dimensional vectors $w_i, \tilde{w}_i \in \mathbb{R}^n$, as well as two bias terms $b_i, \tilde{b}_j \in \mathbb{R}$. The objective for Glove is

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

where

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

and x_{max} and α are parameters that control how much weight is given to different cooccurrences. The objective is optimized using multiple iterations of stochastic gradient descent over the cooccurrence matrix.

3.1.2 Word2Vec

The word2vec model [3] works on the training set directly and learns to predict a token given the token's context. Similar to the Glove model each token i is represented using two multi-dimensional vectors $w_i, \tilde{w}_i \in \mathbb{R}^n$, except that no biases are used. For each paragraph P of tokens $p_1, \ldots, p_T$ the objective for the model, called the continuous bag-of-words objective [2], is

$$J = \frac{1}{T} \sum_{i=1}^{T} \log p(p_t | p_{t-c}, \ldots, p_{t-1}, p_{t+1}, \ldots p_{t+c})$$

where c is the size of the window around a given token and is picked randomly to be in the range $1, \ldots, W$. The word2vec program has two different ways of modeling the probability $\log p(p_t | p_{t-c}, \ldots, p_{t-1}, p_{t+1}, \ldots p_{t+c})$, both of which are approximations to the following softmax model

$$p(p_t | p_{t-c}, \ldots, p_{t-1}, p_{t+1}, \ldots p_{t+c}) = \frac{\exp\left(\tilde{w}_{p_t}^T \sum_{-c \le j \le c, j \neq 0} w_{p_{t+j}} \right)}{\sum_i \exp\left(\tilde{w}_i^T \sum_{-c \le j \le c, j \neq 0} w_{p_{t+j}} \right)}$$

The two different ways of approximating the above model are called hierarchichal softmax and negative sample, and are not described here for conciseness. I find that negative sampling has better performance and I use it throughout. The word2vec objective is optimized using multiple iterations of stochastic gradient descent over the text.

3.2 Phase 2: Extending to all phrases

The output of phase 1 is a learned representation for words as well as phrases that appear both in the training text as well as in the phrase dictionary. In order to learn representations for phrases that are not in the dictionary I use a neural network. The neural network is trained using all phrases of length 2 up to a fixed length. For most of my experiments I used the fixed length of 12. Each training example is a single phrase. The input layer for the phrase is the concatentation of the vectors for each of the words of the phrase, where 0s are added to the end for phrases that have length less than

12. The tanh function is used as the activation function for the hidden layers. The output layer is a linear output layer and the length is the same as the length of the vector representation of the whole phrase. Each of the hidden and output units has a bias. The network is trained so that the output layer for a phrase matches the vector for the phrase.

If y_i is the output of the neural network for a given phrase, w_i is the vector representation for the phrase, w_c is the vector representation for some other random phrase, and f_i is the frequency of the phrase in the text, the loss for the neural network for a single example is defined using a ranking loss [1]:

$$J_i = (\log f_i) \max\left\{0, 0.3 - \frac{y_i^T w_i}{||y_i||_2||w_i||_2} + \frac{y_i^T w_c}{||y_i||_2||w_c||_2}\right\}$$

The gradient for a single example with respect to the output layer is defined as

$$\nabla_{y_i} J_i = 1\{J_i > 0\}(\log f_i)\left[-\frac{1}{||y_i||_2||w_i||_2}w_i + \frac{y_i^T w_i}{||y_i||_2^3||w_i||_2}y_i + \frac{1}{||y_i||_2||w_c||_2}w_c - \frac{y_i^T w_c}{||y_i||_2^3||w_c||_2}y_i\right]$$

The gradients with respect to each of the weights in each of the weight matrices can be computed from the above using standard backpropagation formulas.

Parallelizing the training of the neural network is somewhat tricky. The approach I ended up with is to use batch gradient descent. I split up the training phrases into batches of size 2048 and I compute the gradient for each batch using 16 threads running on 8 CPUs with hyperthreading. Then, I do a gradient update using the gradient for all the phrases in the batch.

Given the model in order to represent a new phrase in the same vector space as all the words and dictionary phrases I simply build the input layer of the neural network as described above using the vectors for each of the words in the phrase, and pass that input through the neural network. The output is then used as the representation of the phrase.

4 Experiments

I evaluate my models on 2 phrase similarity test sets. Both of these test sets contain examples where 2 pieces of text are given along with a human rating of the similarity of the 2 pieces of text. The quality of the models is measured by computing the correlation of the output of the models with the human ratings. I evaluate Glove as well as Word2Vec, and I look at how well my neural network combines the word representations into phrase representations compared to simple averaging of the word representations. During the training of the neural network I evaluate the neural network on the SemEval 2014 task 3 training data, described below, and I save the weights that give the best results on that data. I train the neural network for several hours, and reset the neural network with random parameters every 20 iterations. Although, usually the neural network starts to get good performance in several minutes after only 1 or 2 iterations.

4.1 Mitchell-Lapata 2010

I evaluate my models on a test set described in [4] by Mitchell et al. This test set contains responses from 162 human participants where each participant is asked to compare 2 pairs of words at a time and give their similarity. There are 3 types of questions: verb-object questions, adjective-noun questions and compound noun questions. For example "*knowledge use - influence exercise*" is a verb-object question, "*national government - cold air*" is an adjective-noun question and "*training programme - research contract*" is a compound noun question. Evaluation on this test set is done by computing Spearman's rank correlation coefficient or Spearman's ρ between each of the participants responses and the model outputs, and then averaging all the ρ's over all participants.

I present my results in the table below. My models are compared to the best results given in [4].

	Mitchell-Lapata Spearman's ρ		
	Adjective-Noun	Noun-Noun	Verb-Object
Mitchell-Lapata 2010 Best	**.46**	.49	.41
Glove + Vector Average	.39	.41	.41
Glove + Neural Network	.37	.31	.37
Word2Vec + Vector Average	.40	**.50**	**.46**
Word2Vec + Neural Network	.44	**.50**	.44

4.2 SemEval 2014 Task 3

The SemEval 2014 Task 3 data consists of several distinct sets of data. There are 3 categories of questions: Paragraph-to-Sentence, Sentence-to-Phrase and Phrase-to-Word. For each category there are questions where the model needs to compare a larger piece of text to a smaller piece of text. For example, a question from the Phrase-to-Word set is "*Whatever the mind of man can conceive and believe - dreams*". For each category, there are 3 sets of data: a training set consisting of 500 questions, a test set consisting of 500 questions, and a trial set consisting of 34-38 questions. Each question has a human rating, and model quality is evaluated by computing Pearson's product-moment correlation coefficient, or Pearson's ρ between the model responses and all the human ratings for a given test set. I use the Phrase-to-Word training set for selecting the best weights during the training of the neural network. I use the Phrase-to-Word test set for selecting the best hyperparameters, described in a section below. Finally, I state my results on all of the trial sets, where I pass the text through the neural network several times recursively for text larger than a phrase.

My results are presented below. Along with the performance of my models, I also state the best results for each subtask by any model from the SemEval 2014 competion, including submissions made after the end of the competion. I also state the results of the model from the competition with the best overall performance - Meerkat Mafia - submitted after the end of the competition.

	SemEval 2014 Task 3 Pearson's ρ					
	Paragraph-to-Sentence		Sentence-to-Phrase		Phrase-to-Word	
	Training	Trial	Training	Trial	Training	Trial
SemEval 2014 Best		**.845**		**.777**		.457
SemEval 2014 Meerkat Mafia		.794		.704		.457
Glove + Vector Average		.655		.266		.268
Glove + Neural Network		.315		.263	.383	.464
Word2Vec + Vector Average		.743		.630		.505
Word2Vec + Neural Network		.733		.609	.438	**.597**

4.3 Tuning the Neural Network

I ran a number of experiments to tune the neural network. Different sets of hyperparameters were compared using the performance of the neural network on the SemEval 2014 Task 3 Phrase-to-Word test data. I find that using a loss function that is the Euclidean distance between the output layer and the phrase vector gives somewhat worse performance. I find that using cosine similarity in the ranking loss, as opposed to dot product gives slightly better results. The bigger benefit is that when cosine similarity is used the loss and the margin has more meaning. I also find that using a logit activation function for the hidden layers as opposed to a tanh activation function gives much worse performance. Additionally, the models turn out to be somewhat sensitive to the exact margin used in the ranking loss. I find that a margin of 0.3 gives best performance.

I tried a few different sizes for the hidden layers, settling on using hidden layers of size 1000. I tried training network with different numbers of hidden layers. The performance of the neural network model on the test data for different sizes of the hidden layers is given below.

	Hidden Layers			
	0	1	2	3
SemEval 2014 Task 3 Phrase-to-Word Test Pearson's ρ	.401	.415	.396	.400

Finally, I tried training the models on a larger training text. I used the entire Gigaword corpus consisting of 3B words. I find that the performance on the Phrase-to-Word trial data improves slightly from .597 to .609.

5 Discussion

I find that these 2 test sets are more useful for measuring the quality of the models, compared to several other test sets. Another one of the test sets that I tried consisted of questions such as "*Baltimore* is to *Baltimore Sun* as *Boston* is to what?" where the answer is "*Boston Globe*." In [3] Mikolov et al. use this test set for evaluating their phrase representations. The issue with using that test set is that evaluation is done using exact matching. Thus, the answer is correct only if the first result returned for the query matches. I find that this comparison does not account for inexact matches that are still correct: for example, if "*The Boston Globe*" is returned instead of "*Boston Glove*" or if "*Apple Corporation*" is returned instead of "*Apple*".

Overall, my system is able to get state-of-the-art results on the SemEval 2014 Task 3 Phrase-to-Word data. Word2Vec seems to give significantly better performance than Glove, which is somewhat unexpected given that the authors of the Glove paper show that Glove outperforms Word2Vec [5]. For the best model, most of the good performance comes from Word2Vec, a fairly robust tool for learning general purpose vector representations for words as well as some phrases. Using a neural network, however, does give significantly better results on the SemEval data.

By spot checking a few phrases and looking at the closest vocabulary entries to those phrases I find that qualitatively the neural network does a better job of combining the word vectors into a phrase vector than just simple vector averaging. With vector averaging the closest vocabulary entries end up being words and phrases that are close to one of the words in the phrase, but not so much to the phrase as a whole. Below, I give 2 examples of phrases and interesting vocabulary entries that are closest to those phrases when the phrase vector is computed using the neural network:

- ***Visit a Russian city**: Moscow, City of St. Petersburg, Lenin Hills, Izmailovsky Park*
- ***Flying through the sky**: in the sky, hot-air balloons, vapor trail, curvature of the Earth*

One major limitation of my method is that the neural network is trained using phrases that are *idiomatic* phrases. As a result I'm training the neural network on data that is distributed differently than the data that I'm testing it on. However, it is not clear how to generate phrase vectors for more standard phrases.

6 Future

One possible way to continue the work is to simplify the loss function. After switching to using cosine similarity, as opposed to dot product, the value of the loss becomes more meaningful, and the similarity of the output and the target phrase vector becomes fixed to be in the range $[-1, 1]$. As such, sampling a random other phrase is no longer necessary. Preliminary investigation into the following loss function:

$$J_i = (\log f_i) \left[-\frac{y_i^T w_i}{||y_i||_2 ||w_i||_2} \right]$$

shows that it works as well or better than the ranking loss.

Another area worth investigating is how to generate phrase vectors from text for more standard phrases and not *idiomatic* phrases. Finally, my approach currently does not take into account the structure of the phrase. It would be interesting to see if the same approach can be used to learn a recursive neural network where the word vectors are combined using the parse tree for the phrase.

7 Conclusion

I find that I'm able to train models that can be used for measuring similarity between short pieces of text, with the primary focus of my work being phrases. My models get state-of-the-art performance on a well-known test set.

8 Acknowledgements

I'd like to thank Richard Socher for advice on this project.

References

[1] Minh-Thang Luong, Richard Socher, and Christopher D. Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, 2013.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[4] Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. cognitive science. *Cognitive Science*, 2010.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. 2014.