Research Paper Summary/Notes

Distributed Representations of Words and Phrases and their Compositionality, Google.

Laxman Singh Tomar

October 2019

Research Paper Summary/Notes

Distributed Representations of Words and Phrases and their Compositionality, Google.

Laxman Singh Tomar

October 2019

Contents

Dedication

I would like to dedicate this work to teachers whom I've never met in real but their valuable contribution have made this work possible.

Acknowledgement

(I encountered several resources which helped me in getting through this work)

I would like to express my sincere gratitude to:

- Tomas Mikolov et. al.
- Chris Mccormick
- Jay Alammar

Abstract

This paper introduces extensions to improve quality of word vectors and training speed to the Skipgram Model which is an efficient method for learning high-quality word vectors capturing large number of semantic and syntactic relationships.

Keywords: Frequent Words; Negative Sampling; Phrase Skip-gram

Chapter 1

Introduction

Distributed representation of Words in vector space aka word vectors help learning algorithms in various NLP tasks ranging from Speech Recognition to Machine Translation due to their ability of grouping similar words together.

Recently, T. Mikolov[1] introduced Skipgram Model which while being good at learning high quality word vectors wasn't making use of dense matrix multiplications. Due to which, training could be done on over billions of words in a single day. Although at the same time, you have millions of weights in model to update which required large number of training examples in order to avoid overfitting. So, we've addressed these issues:

1. Subsampling frequent words to decrease the number of training examples; hence speed improves($\sim 2 - 10 \times times$).

2. Modifying the optimization objective i.e. Hierarchical Softmax with a technique called "Negative Sampling", which causes each training sample to update only a small percentage of the model's weights.

3. Identifying phrases (e.g. "Canada Air" vs. "Canada" and "Air") and treating them as single tokens.

4. An interpretation of word vector addition.

1.1 Motivation

Proposed architecture of Skip-gram Model had to tune large amount of weights for which it required large number of training examples leading to computation overhead.

1.2 Objectives

Introduce techniques that can be used to improve the quality of the word vectors and training speed.

Chapter 2

Methods

2.1 Skipgram Model

The training objective of this model is to find word vectors that are useful for predicting the surrounding words in a sentence or a document. Given a sequence of words $w_1, w_2, \ldots, w_T$, the Skip-gram model aims to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

where T is the size of training corpus, w_t are the words and c is the size of the training context. Larger c results in more training examples and thus can lead to a higher accuracy at the expense of increased training time. The probability $p(w_O|w_I)$ is represented with a softmax function:

$$p(w_0|w_I) = \frac{\exp({v'_{wo}}^T v_{wI})}{\sum_{w=1}^{W}\exp({v'_w}^T v_{wI})}$$

where v_w and v_w are the "input" and "output" vector representations of w, and W is the number of words in the vocabulary. The issue is cost of computing $\Delta \log p(w_0|w_I)$ is roughly proportional to W.

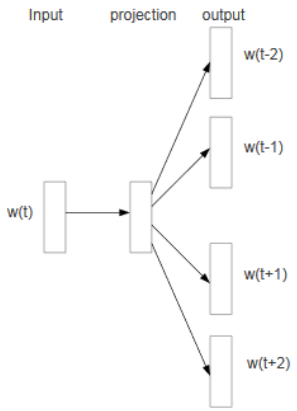

Figure 1: The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words.

2.1.1 Hierarchical Softmax

Instead of evaluating W output nodes of a neural network to get the probability distribution, where W is the size of the target dictionary, only need to evaluate about $\log_2(W)$ nodes.

The idea is to represent the output layer as a binary tree with W leaves and, for each node, explicitly represents the relative probabilities of its child nodes. Then the probability $p(w_O|w_I)$ can be defined by the product of probabilities of a path down the tree from the root. The root here is the first word in the sequence. The individual probabilities are outputs of a sigmoid, scaled by $+1$ or -1 if the current word w's probability matches that of its child.

It can reduce Q by using hierarchical softmax and Huffman binary tree (for storing vocabulary).

2.1.2 Negative Sampling

This is a simplification of something called Noise Contrastive Estimation[2], the idea being that you should be able to distinguish positive examples from negative examples using Logistic Regression (a.k.a. binary classification) rather than picking out the correct class from the entire vocabulary. Thus avoiding hierarchical softmax entirely. This is done by sampling k times from a "noise distribution" over the

vocabulary and training the model to pick out the correct word (k was $\sim 2-20$, with larger k for smaller datasets). NCE additionally uses the numerical probabilities of the noise distribution for some statistical guarantees, but the simpler negative sampling approach turned out to be sufficient.

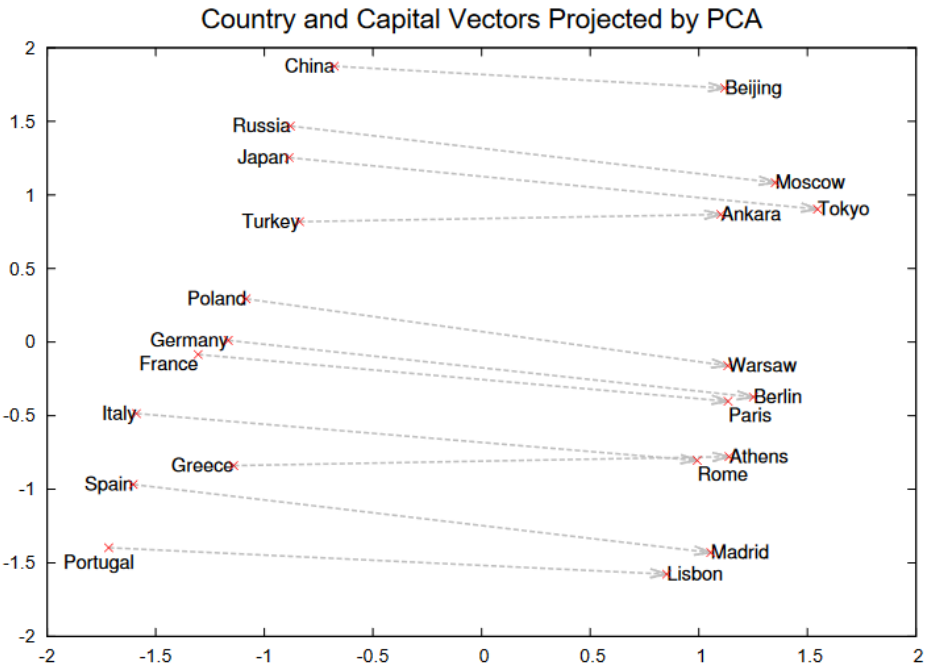

Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

For instance, suppose you had your entire training corpus as a list of words, and you chose your 5 negative samples by picking randomly from the list. In this case, the probability for picking a particular word would be equal to the number of times that word appears in the corpus, divided by the total number of words in the corpus as shown:

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n}(f(w_j))}$$

One interesting point that we found out was that the 3/4th power of the Unigram distribution for negative samples, came out empirically better than other distribu-

tions that we tried:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4} \right)}$$

Damn! If you plug in some sample values, you'll find that, compared to the simpler equation, this one has the tendency to increase the probability for less frequent words and decrease the probability for more frequent words. See Table 1.

2.1.3 Subsampling of Frequent Words

There are some issues associated with frequent words like "the":

1. "the" appears in the context of almost every other word.

2. Frequent words like "the" practically don't add any value even after training millions of examples due to their vector representation unlikely to get changed.

To address this issue we introduced a unique subsampling scheme where each word w_i in the training set is discarded with probability computed by the formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where t is a threshold ($\sim 1e-5$) and f is the word frequency. We chose this subsampling formula because it aggressively subsamples words whose frequency is greater than t while preserving the ranking of the frequencies. It can be understood this way, that this term becomes negative for frequencies rarer than t. It throws away common words and preserves rare words.

2.2 Learning Phrases

Many phrases have a meaning that is not a simple composition of the meanings of its individual words. To learn vector representation for phrases, we first find words that appear frequently together, and infrequently in other contexts. Say, "Boston Globe" (a newspaper) has a much different meaning than the individual words "Boston" and "Globe". So it makes sense to treat "Boston Globe", wherever it occurs in the text,

as a single word with its own word vector representation.

Although it's beyond scope of this paper, but we can form reasonable phrases based on unigram and bigram counts using:

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)}$$

This equation determines which word combinations to turn into phrases. It counts the number of times each combination of two words appears in the training text often relative to the number of individual occurrences. It also eliminates phrases made of infrequent words in order to avoid making phrases out of common words like "and the" or "this is". Finally, this ratio is multiplied by the total number of words in the training text. Presumably, this has the effect of making the threshold value more independent of the training set size. See Table 1.

Table 1: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples).The goal is to compute the fourth phrase using the first three.Our best model achieved an accuracy of 72% on this dataset.

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Candiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company Executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

2.3 Additive Compositionality

We demonstrated that the word and phrase vectors learned by the Skip-gram model tend to show a linear structure that makes it possible to perform precise analogical

reasoning using simple vector arithmetic. Interestingly, we found that the Skip-gram vectors exhibit another kind of linear structure that makes it possible to meaningfully combine words by an element-wise addition of their vector representations.The question is, how should one interpret the sum of two word vectors? Empirically we get results like:

$$vector(\text{"}French\text{"}) + vector(\text{"}actress\text{"}) \approx vector(\text{"}JulietteBinoche\text{"})$$

But why? A word's vector represents the distribution of the word's context, and since these are log probabilities (by virtue of the objective function's structure), adding vectors is like multiplying probabilities. So this acts like an AND function. It can be put it this way, "if "Volga River" appears frequently in the same sentence together with the words "Russian" and "river", the sum of these two word vectors will result in such a feature vector that is close to the vector of "Volga River"." See Table 2.

Table 2: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

Czech + currency	**Vietnam + capital**	**German + airlines**	**Russian + river**
koruna	Hanoi	airline Lufthansa	Moscow
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River
Polish zolty	Vietnam	flag carrier Lufthansa	upriver
CTK	Vietnamese	Lufthansa	Russia

Chapter 3

Results

3.1 Empirical Results

In this section we evaluate the Hierarchical Softmax (HS), Noise Contrastive Estimation, Negative Sampling, and subsampling of the training words. We used the analogical reasoning task. It consists of analogies such as "Germany" : "Berlin" :: "France" : ?, which are solved by finding a vector x such that $vector(x)$ is closest to $vector("Berlin") - vector("Germany") + vector("France")$ according to the cosine distance.

This specific example is considered to have been answered correctly if x is "Paris". The task has two broad categories: the syntactic analogies (such as "quick": "quickly" :: "slow" : "slowly") and the semantic analogies, such as the country to capital city relationship.

For training the Skip-gram models, we have used an internal Google dataset with one billion words consisting of various news articles. We discarded from the vocabulary all words that occurred less than 5 times in the training data. The performance of various Skip-gram models on the word analogy test set is reported in Table 3. The table shows that Negative Sampling outperforms the Hierarchical Softmax on the analogical reasoning task, and has even slightly better performance than the Noise Contrastive Estimation. The subsampling of the frequent words improves the

training speed several times and makes the word representations significantly more accurate.

Table 3: Accuracy of various Skip-gram 300-dimensional models on the analogical reasoning task as defined in last paper. NEG-k stands for Negative Sampling with k negative samples for each positive sample; NCE stands for Noise Contrastive Estimation and HS-Huffman stands for the Hierarchical Softmax with the frequency-based Huffman codes.

Method	**Time[min]**	**Syntactic[%] Semantic[%]**	**Total Accuracy[%]**
NEG-5	38	63 54	59
NEG-15	97	63 58	61
HS-Huffman	41	53 40	47
NCE-5	38	60 45	53
The following results use $10^{-5} subsampling$			
NEG-5	14	61 58	60
NEG-15	36	61 61	61
HS-Huffman	21	52 59	55

3.2 Phrase Skipgram Results

Using same Google News data here as well, we first constructed the phrase based training corpus and then we trained several Skip-gram models using different hyper-parameters. As before, we used vector dimensionality 300 and context size 5. This setting already achieves good performance on the phrase dataset, and allowed us to quickly compare the Negative Sampling and the Hierarchical Softmax, both with and without subsampling of the frequent tokens.

Results show that $k = 5$ gets us a respectable accuracy while $k = 15$ provides a much better performance. In some cases, when we downsampled frequent words it results into much better accuracy as compared to Hierarchical softmax trained without subsampling. What it implies is that subsampling can result in faster training and can also improve accuracy.

In phrase-analogy task, when data(6B words) was increased(33B words) we achieved 72% while former resulted into 66%; using hierarchical softmax, dimensionality of 1000, and entire sentence for the context.

With the previous results, it seems that the best representations of phrases are

learned by a model with the hierarchical softmax and subsampling.

Table 4: Accuracies of the Skip-gram models on the phrase analogy dataset. The models were trained on approximately one billion words from the news dataset.

Method	Dimensionality	No subsampling [%]	$10^{-5} subsampling$[%]
NEG-5	300	24	27
NEG-15	300	27	42
HS-Huffman	300	19	47

3.3 Comparison to Published Word Representations

Skip-gram model trained on a large corpus visibly outperforms all the other models prior to it in the quality of the learned representations. This can be attributed in part to the fact that this model has been trained on about 30 billion words, which is about two to three orders of magnitude more data than the typical size used in the prior work. Interestingly, although the training set is much larger, the training time of the Skip-gram model is just a fraction of the time complexity required by the previous model architectures.

Chapter 4

Discussion

4.1 Discussion

This work has several key contributions:

1. We showed how to train word vectors and phrase vectors with the Skip-gram model and demonstrate that these representations exhibit linear structure that makes precise analogical reasoning possible.

2. Due to efficient architectures, we managed to train models on large amounts of data. It lead to better word vectors especially for infrequent words.

3. Subsampling of the frequent words results in both faster training and significantly better representations of uncommon words.

4. Negative sampling algorithm, is an extremely simple training method that learns accurate representations especially for frequent words.

5. Word vectors can be somewhat meaningfully combined using just simple vector addition. Another approach for learning phrase vectors is to simply represent the phrases with a single token. Combination of these two approaches gives a powerful yet simple way how to represent longer pieces of text, while having minimal computational complexity.

4.2 Conclusions

The choice of the training algorithm and the hyper-parameter selection is a task specific decision, as we found that different problems have different optimal hyperparameter configurations. In our experiments, the most crucial decisions that affecting the performance are:

1. Choice of the model architecture

2. Size of the vectors

3. Subsampling rate

4. Size of the training window

Our work can thus be seen as complementary to the existing approach that attempts to represent phrases using recursive matrix-vector operations.

List of Figures

List of Tables

Bibliography

[1] Tomas Mikolov, G. C., Kai Chen & Dean, J. Efficient estimation of word representations in vector space. *ICLR* (2013).

[2] Gutmann, M. U. & arinen., A. H. Noise-contrastive estimation of unnormalized statistical mod-els, with applications to natural image statistics. *The Journal of Machine Learning Research, 13:307–361.* (2012).

Appendix A

Google's Word2Vec Implementation in C

`https://code.google.com/archive/p/word2vec/`

You can also find here some pre-trained models that they have provided. Note that it's possible to load these pre-trained models into gensim if you want to work with them in Python.

Appendix B

Presentation by T. Mikolov

On December 9th, 2013 at NIPS 2013 Tomas Mikolov from Google gave a presentation on Word2Vec. I think this is mainly a re-hash of the content in the two papers(there is a follow-up paper other than this one). Seeing it presented differently may help you pull out some additional insights, though.

`https://docs.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit`