Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day and one CPU to derive high quality 300-dimensional vectors for one million vocabulary from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring various types of word similarities. We intend to publish this test set to be used by the research community.

1 Introduction

Many current NLP systems and techniques treat words as atomic units. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. However, with recent progress in machine learning and the increase of computational power, the simple techniques are at their limits in many tasks. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train these models on virtually all available data (trillions of words [3]).

In recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. For example, neural network language models trained on a few hundred million words significantly outperform N-gram models trained on the same data [13]. For many tasks, the amount of relevant in-domain data is actually limited - speech recognition performance is usually dominated by the size of high quality transcribed speech data (millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, we are in a situation where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques.

In this paper, we focus on the increasingly popular technique of representing words in a continuous vector space, which may overcome data sparsity problems that arise when words are treated as atomic units. A good motivation for considering continuous vector representations of words might be statistical language modeling, because similar words can share parameters, which leads to more robust estimation of the parameters during training. Compared to N-gram models that treat each

word as an atomic unit, this approach gives significantly better results [1]. Language models based on these techniques are currently the state of the art [11].

1.1 Previous work

Representation of words as continuous vectors has a long history [17, 6]. A very popular model architecture for estimating word vectors was proposed in [1], where a feedforward neural network with a linear projection layer and a non-linear hidden layer was used to learn jointly the word vector representation and a statistical language model. This work has been followed by many others - it was shown later that the word vectors can be used to significantly improve and simplify many NLP applications [5]. Estimation of the word vectors itself was performed using different model architectures and trained on various corpora [4, 18, 15, 13, 7], and some of the resulting word vectors were made available for future research and comparison[1].

1.2 Goals of the Paper

We propose two new architectures for estimating continuous word vectors. We use recently proposed techniques for measuring the quality of the resulting vector representations, with the expectation that not only will similar words tend to be close to each other, but that words can have **multiple degrees of similarity** [14]. This has been observed earlier in the context of inflectional languages - for example, nouns can have multiple word endings, and if we search for similar words in a subspace of the original vector space, it is possible to find words that have similar endings [8].

Somewhat surprisingly, it was found that similarity of word representations goes beyond simple syntactic regularities. Using a word offset technique where simple algebraic operations are performed on the word vectors, it was shown for example that *vector("King") - vector("Man") + vector("Woman")* results in a vector that is closest to the vector representation of the word *Queen* [14].

In this paper, we try to maximize accuracy of these vector operations by designing new model architectures that preserve the linear regularities among words. We design a new comprehensive test set for measuring both syntactic and semantic regularities, and show that many such regularities can be learned with high accuracy. Moreover, we discuss hyper-parameter selection, so that high-quality word vectors can be estimated efficiently even from data sets with billions of words, and with millions of words in the vocabulary.

2 Model Architectures

Many different types of models were proposed for estimating continuous representations of words, including the well-known Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). In this paper, we focus on neural network architectures, as it was previously shown that they perform significantly better than LSA for preserving linear regularities among words [14]; LDA moreover becomes computationally very expensive on large data sets.

Similar to Mikolov et al. [12], to compare different model architectures we define first the computational complexity of a model as the number of parameters that need to be accessed to fully train the model. Next, we will try to maximize the accuracy, while minimizing the computational complexity.

For all the following models, the training complexity is proportional to

$$O = E \times T \times Q, \tag{1}$$

where E is number of the training epochs, T is the number of the words in the training set and Q is defined further for each model architecture. Common choice is $E = 3 - 50$ and T up to one billion. All models are trained using stochastic gradient descent and backpropagation [17].

[1] `http://ronan.collobert.com/senna/`
`http://metaoptimize.com/projects/wordreprs/`
`http://www.fit.vutbr.cz/~imikolov/rnnlm/`
`http://ai.stanford.edu/~ehhuang/`

2.1 Feedforward Neural Net Language Model (NNLM)

The probabilistic feedforward neural network language model has been proposed in [1]. It consists of input, projection, hidden and output layers. At the input layer, N previous words are encoded using 1-of-V coding, where V is size of the vocabulary. The input layer is then projected to a projection layer P that has dimensionality $N \times D$, using a shared projection matrix. As only N inputs are active at any given time, composition of the projection layer is a relatively cheap operation.

The NNLM architecture becomes complex for computation between the projection and the hidden layer, as values in the projection layer are dense. For a common choice of $N = 10$, the size of the projection layer (P) might be 500 to 2000, while the hidden layer size H is typically 500 to 1000 units. Moreover, the hidden layer is used to compute probability distribution over all the words in the vocabulary, resulting in an output layer with dimensionality V. Thus, the computational complexity per each training example is

$$Q = N \times D + N \times D \times H + H \times V, \tag{2}$$

where the dominating term is $H \times V$. However, several practical solutions were proposed for avoiding it; either using hierarchical versions of the softmax [16, 15, 12], or avoiding normalized models completely by using models that are not normalized during training [4, 7]. With binary tree representations of the vocabulary, the number of output units that need to be evaluated can go down to around $log_2(V)$. Thus, most of the complexity is caused by the term $N \times D \times H$.

In our models, we use hierarchical softmax where the vocabulary is represented as a Huffman binary tree. This follows previous observations that the frequency of words works well for obtaining classes in neural net language models [10]. Huffman trees assign short binary codes to frequent words, and this further reduces the number of output units that need to be evaluated: while balanced binary tree would require $log_2(V)$ outputs to be evaluated, the Huffman tree based hierarchical softmax requires only about $log_2(Unigram\_perplexity(V))$. For example when the vocabulary size is one million words, this results in about two times speedup in evaluation. While this is not crucial speedup for neural network LMs as the computational bottleneck is in the $N \times D \times H$ term, we will later propose architectures that do not have hidden layers and thus depend heavily on the efficiency of the softmax normalization.

2.2 Recurrent Neural Net Language Model (RNNLM)

Recurrent neural network based language model has been proposed to overcome certain limitations of the feedforward NNLM, such as the need to specify the context length (the order of the model N), and because theoretically RNNs can efficiently represent more complex patterns than the shallow neural networks [9, 2]. The RNN model does not have a projection layer; only input, hidden and output layer. What is special for this type of model is the recurrent matrix that connects hidden layer to itself, using time-delayed connections. This allows the recurrent model to form some kind of short term memory, as information from the past can be represented by the hidden layer state that gets updated based on the current input and the state of the hidden layer in the previous time step.

The complexity per training example of the RNN model is

$$Q = H \times H + H \times V, \tag{3}$$

where the word representations D have the same dimensionality as the hidden layer H. Again, the term $H \times V$ can be efficiently reduced to $H \times log_2(V)$ by using hierarchical softmax. Most of the complexity then comes from $H \times H$.

3 New Log-linear Models

In this section, we propose two new model architectures that try to minimize the computational complexity, and thus allow high-dimensional word vectors to be estimated efficiently from large amounts of data. The main observation from the previous section was that most of the complexity is caused by the non-linear hidden layer in the model. While this is what makes neural networks so attractive, we decided to explore simpler models that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently.

3.1 Continuous Bag-of-Words Model

The first proposed architecture is similar to the feedforward NNLM, where the non-linear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words in the history does not influence the projection. Furthermore, we also use words from the future; we have obtained the best performance on the task introduced in the next section by building a log-linear classifier with four future and four history words at the input, where the training criterion is to correctly classify the current (middle) word. Training complexity is then

$$Q = N \times D + D \times log_2(V). \tag{4}$$

We denote this model further as CBOW.

3.2 Continuous Skip-gram Model

The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. We found that increasing the range improves quality of the resulting word vectors, but it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those words in our training examples.

The training complexity of this architecture is proportional to

$$Q = C \times (D + D \times log_2(V)), \tag{5}$$

where C is the maximum distance of the words. Thus, if we choose $C = 5$, for each training word we will select randomly a number R in range $< 1; C >$, and then use R words from history and R words from the future of the current word as correct labels. This will require us to do $R \times 2$ word classifications, with the current word as input, and each of the $R + R$ words as output. In the following experiments, we use $C = 10$.

4 Results

To compare the quality of different versions of word vectors, previous papers typically use a table showing example words and their most similar words, and understand them intuitively. Although it is easy to show that word *France* is similar to *Italy* and perhaps some other countries, it is much more challenging when subjecting those vectors in a more complex similarity task, as follows. We follow previous observation that there can be many different types of similarities between words, for example, word *big* is similar to *bigger* in the same sense that *small* is similar to *smaller*. Example of another type of relationship can be word pairs *big* - *biggest* and *small* - *smallest* [14]. We further denote two pairs of words with the same relationship as a question, as we can ask: "What is the word that is similar to *small* in the same sense as *biggest* is similar to *big*?"

Somewhat surprisingly, these questions can be answered by performing simple algebraic operations with the vector representation of words. To find a word that is similar to *small* in the same sense as *biggest* is similar to *big*, we can simply compute vector $X = vector(\text{"}biggest\text{"}) - vector(\text{"}big\text{"}) + vector(\text{"}small\text{"})$. Then, we search in the vector space for the word closest to X measured by cosine distance, and use it as the answer to the question. When the word vectors are well trained, it is possible to find the correct answer (word $smallest$) using this method.

Finally, we found that when we train high dimensional word vectors on a large amount of data, the resulting vectors can be used to answer very subtle semantic relationships between words, such as a city and the country it belongs to, e.g. France is to Paris as Germany is to Berlin. Word vectors with such semantic relationships could be used to improve many existing NLP applications, such as machine translation, information retrieval and question answering systems, and may enable other future applications yet to be invented. We intend to publish a set of high quality word vectors trained on large amounts of data later.

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

4.1 Task Description

To measure quality of the word vectors, we define a comprehensive test set that contains five types of semantic questions, and nine types of syntactic questions. Two examples from each category are shown in Table 1. Overall, there are 8869 semantic and 10675 syntactic questions. The questions in each category were created in two steps: first, a list of similar word pairs was created manually. Then, a large list of questions is formed by connecting two word pairs. For example, we made a list of 68 large American cities and the states they belong to, and formed about 2.5K questions by picking two word pairs at random. We have included in our test set only single token words, thus multi-word entities are not present (such as *New York*). We plan to publish the test set soon, so that other researchers can use it in their experiments.

We evaluate the overall accuracy for all question types, and for each question type separately (semantic, syntactic). Question is assumed to be correctly answered only if the closest word to the vector computed using the above method is exactly the same as the correct word in the question; synonyms are thus counted as mistakes. This also means that reaching 100% accuracy is likely to be impossible, as the current models do not have any input information about word morphology. However, we believe that usefulness of the word vectors for certain applications should be positively correlated with this accuracy metric. Further progress can be achieved by incorporating information about structure of words, especially for the syntactic questions.

4.2 Maximization of Accuracy

We have used a Google News corpus for training the word vectors. This corpus contains about 6B tokens. We have restricted the vocabulary size to 1 million most frequent words. Clearly, we are facing time constrained optimization problem, as it can be expected that both using more data and higher dimensional word vectors will improve the accuracy. To estimate the best choice of hyper-parameters for obtaining as good as possible results quickly, we have first evaluated models trained on subsets of the training data, with vocabulary restricted to the most frequent 30k words. The results using the CBOW architecture with different choice of word vector dimensionality and increasing amount of the training data are shown in Table 2.

It can be seen that after some point, adding more dimensions or adding more training data provides diminishing improvements. So, we have to increase both vector dimensionality and the amount of the training data together. While this observation might seem trivial, it must be noted that it is

Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Table 3: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words [Million]	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mhih NNLM	50	37M	1.8	9.1	5.8
Mhih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW log-linear model	300	783M	15.5	53.1	36.1
Skip-gram log-linear model	300	783M	**50.0**	55.9	**53.3**

currently popular to train word vectors on relatively large amounts of data, but with insufficient size (such as 50 - 100). Given Equation 4, increasing amount of training data twice results in about the same increase of computational complexity as increasing vector size twice.

For the experiments reported in Tables 2 and 3, we used three training epochs with stochastic gradient descent and backpropagation. We chose starting learning rate 0.025 and decreased it linearly, so that it approaches zero at the end of the last training epoch.

4.3 Comparison of Model Architectures

To evaluate our best models and to compare against publicly available word vectors, we have used full set of questions, i.e. unrestricted to the 30k vocabulary. We also included word vectors derived from a feedforward NNLM similar to [1], trained on a larger Google News corpus using about one hundred machine cores in parallel over a period of about two weeks. The comparison is given in Table 3. The CBOW model was trained on subset of the Google News data in about a day, while training time for the skip-gram log-linear model was about three days.

For experiments reported further, we use just one training epoch (again, we decrease learning rate linearly so that it approaches zero at the end of training), as we have more training data than we can efficiently process. Training model on twice more data using one epoch gives comparable or

Table 4: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

Model	Vector Dimensionality	Training words [Million]	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
Our NNLM	100	6B	34.2	64.5	50.8	14
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Table 5: *Examples of the word pair relationships, using the best word vectors from Table 3 (skip-gram log-linear model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

better results than iterating over the same data for three epochs, as is shown in Table 4, and provides additional speedup.

5 Examples of the Learned Relationships

Table 5 shows words that follow various relationships. We follow the approach described above: the relationship is defined by subtracting two word vectors, and the result is added to another word. Thus for example, *France - Paris + Italy = Rome*. As it can be seen, accuracy is quite good, although there is clearly a lot of room for further improvements. We believe that word vectors trained on even larger data sets with larger dimensionality will perform significantly better, and will enable the development of new innovative applications. Another way to improve accuracy is to provide more than one example of the relationship. By using ten examples instead of one, we have observed improvement of accuracy of our best models by about 10% absolutely on the semantic-syntactic test.

It is also possible to apply the vector operations to solve different tasks. For example, we have observed good accuracy for selecting out-of-the-list words, by computing average vector for a list of words, and finding the most distant word vector. This is a popular type of problems in certain human intelligence tests. Clearly, there is still a lot of discoveries to be made using these techniques.

6 Conclusion

In this paper we studied quality of vector representations of words derived by various models. We found that it is possible to train high quality word vectors using very simple model architectures, compared to the popular neural network models (both feedforward and recurrent). Because of the much lower computational complexity, we hope that it will be possible to compute very accurate high dimensional word vectors from a much larger data set. We intend to publish a set of high quality word vectors in the future.

In the future, it would be interesting to compare our techniques to Latent Relational Analysis [19] and others. Also, application to certain previously proposed problems such as sentence completion [20] seems appealing. We believe that our comprehensive test set will help the research community to improve the existing techniques for estimating word vectors. We also hope that high quality word vectors will become an important building block for future NLP applications.

References

[1] Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155, 2003.

[2] Y. Bengio, Y. LeCun. Scaling learning algorithms towards AI. In: Large-Scale Kernel Machines, MIT Press, 2007.

[3] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Language Learning, 2007.

[4] R. Collobert and J. Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In International Conference on Machine Learning, ICML, 2008.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research, 12:2493-2537, 2011.

[6] J. Elman. Finding Structure in Time. Cognitive Science, 14, 179-211, 1990.

[7] Eric H. Huang, R. Socher, C. D. Manning and Andrew Y. Ng. Improving Word Representations via Global Context and Multiple Word Prototypes. In: Proc. Association for Computational Linguistics, 2012.

[8] T. Mikolov, J. Kopecký, L. Burget, O. Glembek and J. Černocký: Neural network based language models for higly inflective languages, In: Proc. ICASSP 2009.

[9] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, S. Khudanpur: Recurrent neural network based language model, In: Proceedings of Interspeech, 2010.

[10] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, S. Khudanpur: Extensions of recurrent neural network language model, In: Proceedings of ICASSP 2011.

[11] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, J. Černocký. Empirical Evaluation and Combination of Advanced Language Modeling Techniques, In: Proceedings of Interspeech, 2011.

[12] T. Mikolov, A. Deoras, D. Povey, L. Burget, J. Černocký. Strategies for Training Large Scale Neural Network Language Models, In: Proc. Automatic Speech Recognition and Understanding, 2011.

[13] T. Mikolov. Statistical Language Models based on Neural Networks. PhD thesis, Brno University of Technology, 2012.

[14] T. Mikolov, G. Zweig. Linguistic Regularities in Continuous Space Word Representations. In: submitted to NAACL HLT 2012.

[15] A. Mnih, G. Hinton. A Scalable Hierarchical Distributed Language Model. Advances in Neural Information Processing Systems 21, MIT Press, 2009.

[16] F. Morin, Y. Bengio: Hierarchical Probabilistic Neural Network Language Model. AISTATS, 2005.

[17] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by back-propagating errors. Nature, 323:533.536, 1986.

[18] Joseph Turian, Lev-Arie Ratinov, Yoshua Bengio. Word Representations: A Simple and General Method for Semi-Supervised Learning. In: Proc. Association for Computational Linguistics, 2010.

[19] P. D. Turney. Measuring Semantic Similarity by Latent Relational Analysis. In: Proc. International Joint Conference on Artificial Intelligence, 2005.

[20] G. Zweig, C.J.C. Burges. The Microsoft Research Sentence Completion Challenge, Microsoft Research Technical Report MSR-TR-2011-129, 2011.