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 to learn high quality word vectors 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 syntactic and semantic word similarities.

1 Introduction

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. 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. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train N-grams on virtually all available data (trillions of words [3]).

However, the simple techniques are at their limits in many tasks. For example, the amount of relevant in-domain data for automatic speech recognition is limited - the performance is usually dominated by the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques.

With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. Probably the most successful concept is to use distributed representations of words [10]. For example, neural network based language models significantly outperform N-gram models [1, 27, 17].

1.1 Goals of the Paper

The main goal of this paper is to introduce techniques that can be used for learning high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary. As far as we know, none of the previously proposed architectures has been successfully trained on more

than a few hundred of millions of words, with a modest dimensionality of the word vectors between 50 - 100.

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** [20]. 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 [13, 14].

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* [20].

In this paper, we try to maximize accuracy of these vector operations by developing new model architectures that preserve the linear regularities among words. We design a new comprehensive test set for measuring both syntactic and semantic regularities[1], and show that many such regularities can be learned with high accuracy. Moreover, we discuss how training time and accuracy depends on the dimensionality of the word vectors and on the amount of the training data.

1.2 Previous Work

Representation of words as continuous vectors has a long history [10, 26, 8]. A very popular model architecture for estimating neural network language model (NNLM) 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.

Another interesting architecture of NNLM was presented in [13, 14], where the word vectors are first learned using neural network with a single hidden layer. The word vectors are then used to train the NNLM. Thus, the word vectors are learned even without constructing the full NNLM. In this work, we directly extend this architecture, and focus just on the first step where the word vectors are learned using a simple model.

It was later shown that the word vectors can be used to significantly improve and simplify many NLP applications [4, 5, 29]. Estimation of the word vectors itself was performed using different model architectures and trained on various corpora [4, 29, 23, 19, 9], and some of the resulting word vectors were made available for future research and comparison[2]. However, as far as we know, these architectures were significantly more computationally expensive for training than the one proposed in [13], with the exception of certain version of log-bilinear model where diagonal weight matrices are used [23].

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 distributed representations of words learned by neural networks, as it was previously shown that they perform significantly better than LSA for preserving linear regularities among words [20, 31]; LDA moreover becomes computationally very expensive on large data sets.

Similar to [18], 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.

[1] The test set is available at `www.fit.vutbr.cz/~imikolov/rnnlm/word-test.v1.txt`

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

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 [26].

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 [25, 23, 18], or avoiding normalized models completely by using models that are not normalized during training [4, 9]. 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 [16]. 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 [15, 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 $H \times H$.

2.3 Parallel Training of Neural Networks

To train models on huge data sets, we have implemented several models on top of a large-scale distributed framework called DistBelief [6], including the feedforward NNLM and the new models proposed in this paper. The framework allows us to run multiple replicas of the same model in parallel, and each replica synchronizes its gradient updates through a centralized server that keeps all the parameters. For this parallel training, we use mini-batch asynchronous gradient descent with an adaptive learning rate procedure called Adagrad [7]. Under this framework, it is common to use one hundred or more model replicas, each using many CPU cores at different machines in a data center.

3 New Log-linear Models

In this section, we propose two new model architectures for learning distributed representations of words that try to minimize computational complexity. 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.

The new architectures directly follow those proposed in our earlier work [13, 14], where it was found that neural network language model can be successfully trained in two steps: first, continuous word vectors are learned using simple model, and then the N-gram NNLM is trained on top of these distributed representations of words. While there has been later substantial amount of work that focuses on learning word vectors, we consider the approach proposed in [13] to be the simplest one. Note that related models have been proposed also much earlier [26, 8].

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, as unlike standard bag-of-words model, it uses continuous distributed representation of the context. The model architecture is shown at Figure 1. Note that the weight matrix between the input and the projection layer is shared for all word positions in the same way as in the NNLM.

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

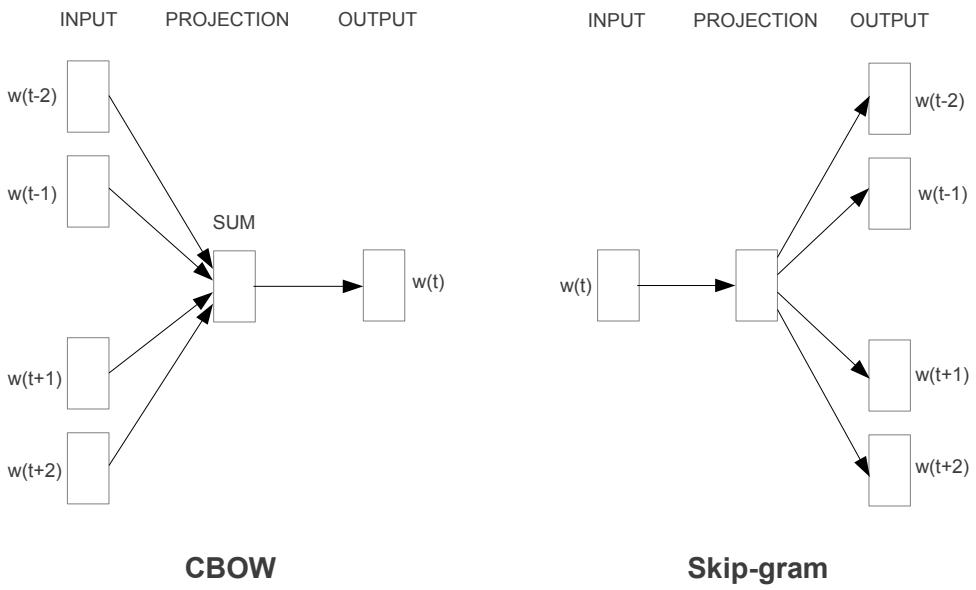


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

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* [20]. 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 (we discard the input question words during this search). 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.

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 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 model architecture 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 currently popular to train word vectors on relatively large amounts of data, but with insufficient size

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 words 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 architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

(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 4, 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

First we compare different model architectures for deriving the word vectors using the same training data and using the same dimensionality of 640 of the word vectors. In the further experiments, we use full set of questions in the new Semantic-Syntactic Word Relationship test set, i.e. unrestricted to the 30k vocabulary. We also include results on a test set introduced in [20] that focuses on syntactic similarity between words[3].

The training data consists of several LDC corpora and is described in detail in [18] (320M words, 82K vocabulary). We used these data to provide a comparison to a previously trained recurrent neural network language model that took about 8 weeks to train on a single CPU. We trained a feedforward NNLM with the same number of 640 hidden units using the DistBelief parallel training [6], using a history of 8 previous words (thus, the NNLM has more parameters than the RNNLM, as the projection layer has size 640×8).

In Table 3, it can be seen that the word vectors from the RNN (as used in [20]) perform well mostly on the syntactic questions. The NNLM vectors perform significantly better than the RNN - this is not surprising, as the word vectors in the RNNLM are directly connected to a non-linear hidden layer. The CBOW architecture works better than the NNLM on the syntactic tasks, and about the same on the semantic one. Finally, the Skip-gram architecture works slightly worse on the syntactic task than the CBOW model (but still better than the NNLM), and much better on the semantic part of the test than all the other models.

Next, we evaluated our models trained using one CPU only and compared the results against publicly available word vectors. The comparison is given in Table 4. The CBOW model was trained on subset

[3]We thank Geoff Zweig for providing us the test set.

Table 4: *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	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
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih 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	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Table 5: *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	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
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

of the Google News data in about a day, while training time for the Skip-gram model was about three days.

For experiments reported further, we used just one training epoch (again, we decrease the learning rate linearly so that it approaches zero at the end of training). Training a model on twice as much data using one epoch gives comparable or better results than iterating over the same data for three epochs, as is shown in Table 5, and provides additional small speedup.

4.4 Large Scale Parallel Training of Models

As mentioned earlier, we have implemented various models in a distributed framework called DistBelief. Below we report the results of several models trained on the Google News 6B data set, with mini-batch asynchronous gradient descent and the adaptive learning rate procedure called Adagrad [7]. We used 50 to 100 model replicas during the training. The number of CPU cores is an

Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Table 7: *Comparison and combination of models on the Microsoft Sentence Completion Challenge.*

Architecture	Accuracy [%]
4-gram [32]	39
Average LSA similarity [32]	49
Log-bilinear model [24]	54.8
RNNLMs [19]	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	**58.9**

estimate since the data center machines are shared with other production tasks, and the usage can fluctuate quite a bit. Note that due to the overhead of the distributed framework, the CPU usage of the CBOW model and the Skip-gram model are much closer to each other than their single-machine implementations. The result are reported in Table 6.

4.5 Microsoft Research Sentence Completion Challenge

The Microsoft Sentence Completion Challenge has been recently introduced as a task for advancing language modeling and other NLP techniques [32]. This task consists of 1040 sentences, where one word is missing in each sentence and the goal is to select word that is the most coherent with the rest of the sentence, given a list of five reasonable choices. Performance of several techniques has been already reported on this set, including N-gram models, LSA-based model [32], log-bilinear model [24] and a combination of recurrent neural networks that currently holds the state of the art performance of 55.4% accuracy on this benchmark [19].

We have explored the performance of Skip-gram architecture on this task. First, we train the 640-dimensional model on 50M words provided in [32]. Then, we compute score of each sentence in the test set by using the unknown word at the input, and predict all surrounding words in a sentence. The final sentence score is then the sum of these individual predictions. Using the sentence scores, we choose the most likely sentence.

A short summary of some previous results together with the new results is presented in Table 7. While the Skip-gram model itself does not perform on this task better than LSA similarity, the scores from this model are complementary to scores obtained with RNNLMs, and a weighted combination leads to a new state of the art result 58.9% accuracy (59.2% on the development part of the set and 58.7% on the test part of the set).

5 Examples of the Learned Relationships

Table 8 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, *Paris* - *France* + *Italy* = *Rome*. As it can be seen, accuracy is quite good, although there is clearly a lot of room for further improvements (note that using our accuracy metric that

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram 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

assumes exact match, the results in Table 8 would score only about 60%). 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 to form the relationship vector (we average the individual vectors together), 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 the quality of vector representations of words derived by various models on a collection of syntactic and semantic language tasks. We observed 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, it is possible to compute very accurate high dimensional word vectors from a much larger data set. Using the DistBelief distributed framework, it should be possible to train the CBOW and Skip-gram models even on corpora with one trillion words, for basically unlimited size of the vocabulary. That is several orders of magnitude larger than the best previously published results for similar models.

An interesting task where the word vectors have recently been shown to significantly outperform the previous state of the art is the SemEval-2012 Task 2 [11]. The publicly available RNN vectors were used together with other techniques to achieve over 50% increase in Spearman's rank correlation over the previous best result [31]. The neural network based word vectors were previously applied to many other NLP tasks, for example sentiment analysis [12] and paraphrase detection [28]. It can be expected that these applications can benefit from the model architectures described in this paper.

Our ongoing work shows that the word vectors can be successfully applied to automatic extension of facts in Knowledge Bases, and also for verification of correctness of existing facts. Results from machine translation experiments also look very promising. In the future, it would be also interesting to compare our techniques to Latent Relational Analysis [30] and others. We believe that our comprehensive test set will help the research community to improve the existing techniques for estimating the word vectors. We also expect that high quality word vectors will become an important building block for future NLP applications.

7 Follow-Up Work

After the initial version of this paper was written, we published single-machine multi-threaded C++ code for computing the word vectors, using both the continuous bag-of-words and skip-gram architectures[4]. The training speed is significantly higher than reported earlier in this paper, i.e. it is in the order of billions of words per hour for typical hyperparameter choices. We also published more than 1.4 million vectors that represent named entities, trained on more than 100 billion words. Some of our follow-up work will be published in an upcoming NIPS 2013 paper [21].

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. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M.A. Ranzato, A. Senior, P. Tucker, K. Yang, A. Y. Ng., Large Scale Distributed Deep Networks, NIPS, 2012.

[7] J.C. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 2011.

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

[9] 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.

[10] G.E. Hinton, J.L. McClelland, D.E. Rumelhart. Distributed representations. In: Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations, MIT Press, 1986.

[11] D.A. Jurgens, S.M. Mohammad, P.D. Turney, K.J. Holyoak. Semeval-2012 task 2: Measuring degrees of relational similarity. In: Proceedings of the 6th International Workshop on Semantic Evaluation (SemEval 2012), 2012.

[12] A.L. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In Proceedings of ACL, 2011.

[13] T. Mikolov. Language Modeling for Speech Recognition in Czech, Masters thesis, Brno University of Technology, 2007.

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

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

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

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

[4] The code is available at `https://code.google.com/p/word2vec/`

[18] 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.

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

[20] T. Mikolov, W.T. Yih, G. Zweig. Linguistic Regularities in Continuous Space Word Representations. NAACL HLT 2013.

[21] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. Accepted to NIPS 2013.

[22] A. Mnih, G. Hinton. Three new graphical models for statistical language modelling. ICML, 2007.

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

[24] A. Mnih, Y.W. Teh. A fast and simple algorithm for training neural probabilistic language models. ICML, 2012.

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

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

[27] H. Schwenk. Continuous space language models. Computer Speech and Language, vol. 21, 2007.

[28] R. Socher, E.H. Huang, J. Pennington, A.Y. Ng, and C.D. Manning. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In NIPS, 2011.

[29] J. Turian, L. Ratinov, Y. Bengio. Word Representations: A Simple and General Method for Semi-Supervised Learning. In: Proc. Association for Computational Linguistics, 2010.

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

[31] A. Zhila, W.T. Yih, C. Meek, G. Zweig, T. Mikolov. Combining Heterogeneous Models for Measuring Relational Similarity. NAACL HLT 2013.

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

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

Abstract

Recent methods for learning vector space representations of words have succeeded in capturing fine-grained semantic and syntactic regularities using vector arithmetic, but the origin of these regularities has remained opaque. We analyze and make explicit the model properties needed for such regularities to emerge in word vectors. The result is a new global log-bilinear regression model that combines the advantages of the two major model families in the literature: global matrix factorization and local context window methods. Our model efficiently leverages statistical information by training only on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus. The model produces a vector space with meaningful substructure, as evidenced by its performance of 75% on a recent word analogy task. It also outperforms related models on similarity tasks and named entity recognition.

1 Introduction

Semantic vector space models of language represent each word with a real-valued vector. These vectors can be used as features in a variety of applications, such as information retrieval (Manning et al., 2008), document classification (Sebastiani, 2002), question answering (Tellex et al., 2003), named entity recognition (Turian et al., 2010), and parsing (Socher et al., 2013).

Most word vector methods rely on the distance or angle between pairs of word vectors as the primary method for evaluating the intrinsic quality of such a set of word representations. Recently, Mikolov et al. (2013c) introduced a new evaluation scheme based on word analogies that probes the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference. For example, the analogy "king is to queen as man is to woman" should be encoded in the vector space by the vector equation $king - queen = man - woman$. This evaluation scheme favors models that produce dimensions of meaning, thereby capturing the multi-clustering idea of distributed representations (Bengio, 2009).

The two main model families for learning word vectors are: 1) global matrix factorization methods, such as latent semantic analysis (LSA) (Deerwester et al., 1990) and 2) local context window methods, such as the skip-gram model of Mikolov et al. (2013c). Currently, both families suffer significant drawbacks. While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts.

In this work, we analyze the model properties necessary to produce linear directions of meaning and argue that global log-bilinear regression models are appropriate for doing so. We propose a specific weighted least squares model that trains on global word-word co-occurrence counts and thus makes efficient use of statistics. The model produces a word vector space with meaningful substructure, as evidenced by its state-of-the-art performance of 75% accuracy on the word analogy dataset. We also demonstrate that our methods outperform other current methods on several word similarity tasks, and also on a common named entity recognition (NER) benchmark.

We provide the source code for the model as well as trained word vectors at `http://nlp.stanford.edu/projects/glove/`.

2 Related Work

Matrix Factorization Methods. Matrix factorization methods for generating low-dimensional word representations have roots stretching as far back as LSA. These methods utilize low-rank approximations to decompose large matrices that capture statistical information about a corpus. The particular type of information captured by such matrices varies by application. In LSA, the matrices are of "term-document" type, i.e., the rows correspond to words or terms, and the columns correspond to different documents in the corpus. In contrast, the Hyperspace Analogue to Language (HAL) (Lund and Burgess, 1996), for example, utilizes matrices of "term-term" type, i.e., the rows and columns correspond to words and the entries correspond to the number of times a given word occurs in the context of another given word.

A main problem with HAL and related methods is that the most frequent words contribute a disproportionate amount to the similarity measure: the number of times two words co-occur with *the* or *and*, for example, will have a large effect on their similarity despite conveying relatively little about their semantic relatedness. A number of techniques exist that addresses this shortcoming of HAL, such as the COALS method (Rohde et al., 2006), in which the co-occurrence matrix is first transformed by an entropy- or correlation-based normalization. An advantage of this type of transformation is that the raw co-occurrence counts, which for a reasonably sized corpus might span 8 or 9 orders of magnitude, are compressed so as to be distributed more evenly in a smaller interval. A variety of newer models also pursue this approach, including a study (Bullinaria and Levy, 2007) that indicates that positive pointwise mutual information (PPMI) is a good transformation. More recently, a square root type transformation in the form of Hellinger PCA (HPCA) (Lebret and Collobert, 2014) has been suggested as an effective way of learning word representations.

Shallow Window-Based Methods. Another approach is to learn word representations that aid in making predictions within local context windows. For example, Bengio et al. (2003) introduced a model that learns word vector representations as part of a simple neural network architecture for language modeling. Collobert and Weston (2008) decoupled the word vector training from the downstream training objectives, which paved the way for Collobert et al. (2011) to use the full context of a word for learning the word representations, rather than just the preceding context as is the case with language models.

Recently, the importance of the full neural network structure for learning useful word representations has been called into question. The skip-gram and continuous bag-of-words (CBOW) models of Mikolov et al. (2013a) propose a simple single-layer architecture based on the inner product between two word vectors. Mnih and Kavukcuoglu (2013) also proposed closely-related vector log-bilinear models, vLBL and ivLBL, and Levy et al. (2014) proposed explicit word embeddings based on a PPMI metric.

In the skip-gram and ivLBL models, the objective is to predict a word's context given the word itself, whereas the objective in the CBOW and vLBL models is to predict a word given its context. Through evaluation on a word analogy task, these models demonstrated the capacity to learn linguistic patterns as linear relationships between the word vectors.

Unlike the matrix factorization methods, the shallow window-based methods suffer from the disadvantage that they do not operate directly on the co-occurrence statistics of the corpus. Instead, these models scan context windows across the entire corpus, which fails to take advantage of the vast amount of repetition in the data.

3 The GloVe Model

The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations, and although many such methods now exist, the question still remains as to how meaning is generated from these statistics, and how the resulting word vectors might represent that meaning. In this section, we shed some light on this question. We use our insights to construct a new model for word representation which we call GloVe, for Global Vectors, because the global corpus statistics are captured directly by the model.

First we establish some notation. Let the matrix of word-word co-occurrence counts be denoted by X, whose entries X_{ij} tabulate the number of times word j occurs in the context of word i. Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i. Finally, let $P_{ij} = P(j|i) = X_{ij}/X_i$ be the probability that word j appear in the

Table 1: Co-occurrence probabilities for target words *ice* and *steam* with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like *water* and *fashion* cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

context of word i.

We begin with a simple example that showcases how certain aspects of meaning can be extracted directly from co-occurrence probabilities. Consider two words i and j that exhibit a particular aspect of interest; for concreteness, suppose we are interested in the concept of thermodynamic phase, for which we might take $i = ice$ and $j = steam$. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k. For words k related to ice but not steam, say $k = solid$, we expect the ratio P_{ik}/P_{jk} will be large. Similarly, for words k related to steam but not ice, say $k = gas$, the ratio should be small. For words k like *water* or *fashion*, that are either related to both ice and steam, or to neither, the ratio should be close to one. Table 1 shows these probabilities and their ratios for a large corpus, and the numbers confirm these expectations. Compared to the raw probabilities, the ratio is better able to distinguish relevant words (*solid* and *gas*) from irrelevant words (*water* and *fashion*) and it is also better able to discriminate between the two relevant words.

The above argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves. Noting that the ratio P_{ik}/P_{jk} depends on three words i, j, and k, the most general model takes the form,

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{1}$$

where $w \in \mathbb{R}^d$ are word vectors and $\tilde{w} \in \mathbb{R}^d$ are separate context word vectors whose role will be discussed in Section 4.2. In this equation, the right-hand side is extracted from the corpus, and F may depend on some as-of-yet unspecified parameters. The number of possibilities for F is vast, but by enforcing a few desiderata we can select a unique choice. First, we would like F to encode the information present the ratio P_{ik}/P_{jk} in the word vector space. Since vector spaces are inherently linear structures, the most natural way to do this is with vector differences. With this aim, we can restrict our consideration to those functions F that depend only on the difference of the two target words, modifying Eqn. (1) to,

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}. \tag{2}$$

Next, we note that the arguments of F in Eqn. (2) are vectors while the right-hand side is a scalar. While F could be taken to be a complicated function parameterized by, e.g., a neural network, doing so would obfuscate the linear structure we are trying to capture. To avoid this issue, we can first take the dot product of the arguments,

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}}, \tag{3}$$

which prevents F from mixing the vector dimensions in undesirable ways. Next, note that for word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary and that we are free to exchange the two roles. To do so consistently, we must not only exchange $w \leftrightarrow \tilde{w}$ but also $X \leftrightarrow X^T$. Our final model should be invariant under this relabeling, but Eqn. (3) is not. However, the symmetry can be restored in two steps. First, we require that F be a homomorphism between the groups $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$, i.e.,

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}, \tag{4}$$

which, by Eqn. (3), is solved by,

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}. \tag{5}$$

The solution to Eqn. (4) is $F = \exp$, or,

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i). \tag{6}$$

Next, we note that Eqn. (6) would exhibit the exchange symmetry if not for the $\log(X_i)$ on the right-hand side. However, this term is independent of k so it can be absorbed into a bias b_i for w_i. Finally, adding an additional bias $\tilde{b}_k$ for $\tilde{w}_k$ restores the symmetry,

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik}) \,. \tag{7}$$

Eqn. (7) is a drastic simplification over Eqn. (1), but it is actually ill-defined since the logarithm diverges whenever its argument is zero. One resolution to this issue is to include an additive shift in the logarithm, $\log(X_{ik}) \rightarrow \log(1 + X_{ik})$, which maintains the sparsity of X while avoiding the divergences. The idea of factorizing the log of the co-occurrence matrix is closely related to LSA and we will use the resulting model as a baseline in our experiments. A main drawback to this model is that it weighs all co-occurrences equally, even those that happen rarely or never. Such rare co-occurrences are noisy and carry less information than the more frequent ones — yet even just the zero entries account for 75–95% of the data in X, depending on the vocabulary size and corpus.

We propose a new weighted least squares regression model that addresses these problems. Casting Eqn. (7) as a least squares problem and introducing a weighting function $f(X_{ij})$ into the cost function gives us the model

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \,, \tag{8}$$

where V is the size of the vocabulary. The weighting function should obey the following properties:

1. $f(0) = 0$. If f is viewed as a continuous function, it should vanish as $x \rightarrow 0$ fast enough that the $\lim_{x \rightarrow 0} f(x) \log^2 x$ is finite.

2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.

3. $f(x)$ should be relatively small for large values of x, so that frequent co-occurrences are not overweighted.

Of course a large number of functions satisfy these properties, but one class of functions that we found to work well can be parameterized as,

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \,. \tag{9}$$

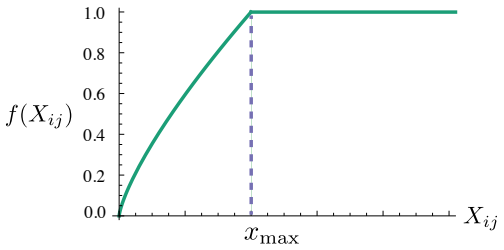


Figure 1: Weighting function f with $\alpha = 3/4$.

The performance of the model depends weakly on the cutoff, which we fix to $x_{\max} = 100$ for all our experiments. We found that $\alpha = 3/4$ gives a modest improvement over a linear version with $\alpha = 1$. Although we offer only empirical motivation for choosing the value 3/4, it is interesting that a similar fractional power scaling was found to give the best performance in (Mikolov et al., 2013a).

3.1 Relationship to Other Models

Because all unsupervised methods for learning word vectors are ultimately based on the occurrence statistics of a corpus, there should be commonalities between the models. Nevertheless, certain models remain somewhat opaque in this regard, particularly the recent window-based methods like skip-gram and ivLBL. Therefore, in this subsection we show how these models are related to our proposed model, as defined in Eqn. (8).

The starting point for the skip-gram or ivLBL methods is a model Q_{ij} for the probability that word j appears in the context of word i. For concreteness, let us assume that Q_{ij} is a softmax,

$$Q_{ij} = \frac{\exp(w_i^T \tilde{w}_j)}{\sum_{k=1}^{V} \exp(w_i^T \tilde{w}_k)} \,. \tag{10}$$

Most of the details of these models are irrelevant for our purposes, aside from the the fact that they attempt to maximize the log probability as a context window scans over the corpus. Training proceeds in an on-line, stochastic fashion, but the implied global objective function can be written as,

$$J = -\sum_{\substack{i \in \text{corpus} \\ j \in \text{context}(i)}} \log Q_{ij} \,. \tag{11}$$

Evaluating the normalization factor of the softmax for each term in this sum is costly. To allow for efficient training, the skip-gram and ivLBL models introduce approximations to Q_{ij}. However, the sum in Eqn. (11) can be evaluated much

more efficiently if we first group together those terms that have the same values for i and j,

$$J = -\sum_{i=1}^{V}\sum_{j=1}^{V} X_{ij} \log Q_{ij} \,, \tag{12}$$

where we have used the fact that the number of like terms is given by the co-occurrence matrix X.

Recalling our notation for $X_i = \sum_k X_{ik}$ and $P_{ij} = X_{ij}/X_i$, we can rewrite J as,

$$J = -\sum_{i=1}^{V} X_i \sum_{j=1}^{V} P_{ij} \log Q_{ij} = \sum_{i=1}^{V} X_i H(P_i, Q_i) \,, \tag{13}$$

where $H(P_i, Q_i)$ is the cross entropy of the distributions P_i and Q_i, which we define in analogy to X_i. As a weighted sum of cross-entropy error, this objective bears some formal resemblance to the weighted least squares objective of Eqn. (8). In fact, it is possible to optimize Eqn. (13) directly as opposed to the on-line training methods used in the skip-gram and ivLBL models. One could interpret this objective as a "global skip-gram" model, and it might be interesting to investigate further. On the other hand, Eqn. (13) exhibits a number of undesirable properties that ought to be addressed before adopting it as a model for learning word vectors.

To begin, cross entropy error is just one among many possible distance measures between probability distributions, and it has the unfortunate property that distributions with long tails are often modeled poorly with too much weight given to the unlikely events. Furthermore, for the measure to be bounded it requires that the model distribution Q be properly normalized. This presents a computational bottleneck owing to the sum over the whole vocabulary in Eqn. (10), and it would be desirable to consider a different distance measure that did not require this property of Q. A natural choice would be a least squares objective in which normalization factors in Q and P are discarded,

$$\hat{J} = \sum_{i,j} X_i \left(\hat{P}_{ij} - \hat{Q}_{ij}\right)^2 \tag{14}$$

where $\hat{P}_{ij} = X_{ij}$ and $\hat{Q}_{ij} = \exp(w_i^T \tilde{w}_j)$ are the unnormalized distributions. At this stage another problem emerges, namely that X_{ij} often takes very large values, which can complicate the optimization. An effective remedy is to minimize the squared error of the logarithms of $\hat{P}$ and $\hat{Q}$ instead,

$$\begin{aligned} \hat{J} &= \sum_{i,j} X_i \left(\log \hat{P}_{ij} - \log \hat{Q}_{ij}\right)^2 \\ &= \sum_{i,j} X_i \left(w_i^T \tilde{w}_j - \log X_{ij}\right)^2 . \end{aligned} \tag{15}$$

Finally, we observe that while the weighting factor X_i is preordained by the on-line training method inherent to the skip-gram and ivLBL models, it is by no means guaranteed to be optimal. In fact, Mikolov et al. (2013a) observe that performance can be increased by filtering the data so as to reduce the effective value of the weighting factor for frequent words. With this in mind, we introduce a more general weighting function, which we are free to take to depend on the context word as well. The result is,

$$\hat{J} = \sum_{i,j} f(X_{ij}) \left(w_i^T \tilde{w}_j - \log X_{ij}\right)^2 , \tag{16}$$

which is equivalent[1] to the cost function of Eqn. (8), which we derived previously.

3.2 Complexity of the model

As can be seen from Eqn. (8) and the explicit form of the weighting function $f(X)$, the computational complexity of the model depends on the number of nonzero elements in the matrix X. As this number is always less than the total number of entries of the matrix, the model scales no worse than $O(|V|^2)$. At first glance this might seem like a substantial improvement over the shallow window-based approaches, which scale with the corpus size, $|C|$. However, typical vocabularies have hundreds of thousands of words, so that $|V|^2$ can be in the hundreds of billions, which is actually much larger than most corpora. For this reason it is important to determine whether a tighter bound can be placed on the number of nonzero elements of X.

In order to make any concrete statements about the number of nonzero elements in X, it is necessary to make some assumptions about the distribution of word co-occurrences. In particular, we will assume that the number of co-occurrences of word i with word j, X_{ij}, can be modeled as a power-law function of the frequency rank of that word pair, r_{ij}:

$$X_{ij} = \frac{k}{(r_{ij})^\alpha} \,. \tag{17}$$

[1] We could also include bias terms in Eqn. (16).

The total number of words in the corpus is proportional to the sum over all elements of the co-occurrence matrix X,

$$|C| \sim \sum_{ij} X_{ij} = \sum_{r=1}^{|X|} \frac{k}{r^\alpha} = k H_{|X|,\alpha}\,, \qquad (18)$$

where we have rewritten the last sum in terms of the generalized harmonic number $H_{n,m}$. The upper limit of the sum, $|X|$, is the maximum frequency rank, which coincides with the number of nonzero elements in the matrix X. This number is also equal to the maximum value of r in Eqn. (17) such that $X_{ij} \geq 1$, i.e., $|X| = k^{1/\alpha}$. Therefore we can write Eqn. (18) as,

$$|C| \sim |X|^\alpha \, H_{|X|,\alpha}\,. \qquad (19)$$

We are interested in how $|X|$ is related to $|C|$ when both numbers are large; therefore we are free to expand the right hand side of the equation for large $|X|$. For this purpose we use the expansion of generalized harmonic numbers (Apostol, 1976),

$$H_{x,s} = \frac{x^{1-s}}{1-s} + \zeta(s) + O(x^{-s}) \quad \text{if} \quad s > 0,\, s \neq 1\,, \qquad (20)$$

giving,

$$|C| \sim \frac{|X|}{1-\alpha} + \zeta(\alpha)\,|X|^\alpha + O(1)\,, \qquad (21)$$

where $\zeta(s)$ is the Riemann zeta function. In the limit that X is large, only one of the two terms on the right hand side of Eqn. (21) will be relevant, and which term that is depends on whether $\alpha > 1$,

$$|X| = \begin{cases} O(|C|) & \text{if } \alpha < 1, \\ O(|C|^{1/\alpha}) & \text{if } \alpha > 1. \end{cases} \qquad (22)$$

For the corpora studied in this article, we observe that X_{ij} is well-modeled by Eqn. (17) with $\alpha = 1.25$. In this case we have that $|X| = O(|C|^{0.8})$. Therefore we conclude that the complexity of the model is much better than the worst case $O(V^2)$, and in fact it does somewhat better than the on-line window-based methods which scale like $O(|C|)$.

4 Experiments

4.1 Evaluation methods

We conduct experiments on the word analogy task of Mikolov et al. (2013a), a variety of word similarity tasks, as described in (Luong et al., 2013), and on the CoNLL-2003 shared benchmark dataset for NER (Tjong Kim Sang and De Meulder, 2003).

Table 2: Results on the word analogy task, given as percent accuracy. Underlined scores are best within groups of similarly-sized models; bold scores are best overall. HPCA vectors are publicly available[2]; (i)vLBL results are from (Mnih et al., 2013); skip-gram (SG) and CBOW results are from (Mikolov et al., 2013a,b); we trained SG† and CBOW† using the `word2vec` tool[3]. See text for details and a description of the SVD models.

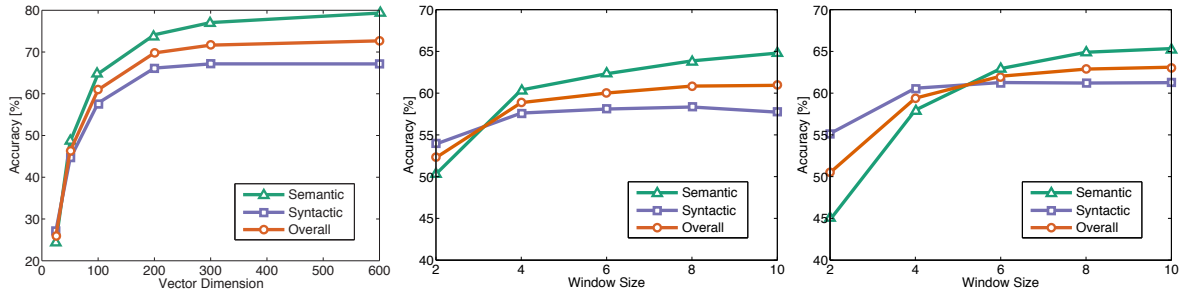
Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

Word analogies. The word analogy task consists of questions like, "*a* is to *b* as *c* is to __?" The dataset contains 19,544 such questions, divided into a semantic subset and a syntactic subset. The semantic questions are typically analogies about people or places, like "Athens is to Greece as Berlin is to __?". The syntactic questions are typically analogies about verb tenses or forms of adjectives, for example "dance is to dancing as fly is to __?". To correctly answer the question, the model should uniquely identify the missing term, with only an exact correspondence counted as a correct match. We answer the question "*a* is to *b* as *c* is to __?" by finding the word d whose representation w_d is closest to $w_b - w_a + w_c$ according to the cosine similarity.[4]

[2] http://lebret.ch/words/

[3] http://code.google.com/p/word2vec/

[4] Levy et al. (2014) introduce a multiplicative analogy evaluation, 3CosMul, and report an accuracy of 68.24% on



(a) Symmetric context (b) Symmetric context (c) Asymmetric context

Figure 2: Accuracy on the analogy task as function of vector size and window size/type. All models are trained on the 6 billion token corpus. In (a), the window size is 10. In (b) and (c), the vector size is 100.

Word similarity. While the analogy task is our primary focus since it tests for interesting vector space substructures, we also evaluate our model on a variety of word similarity tasks in Table 3. These include WordSim-353 (Finkelstein et al., 2001), MC (Miller and Charles, 1991), RG (Rubenstein and Goodenough, 1965), SCWS (Huang et al., 2012), and RW (Luong et al., 2013).

Named entity recognition. The CoNLL-2003 English benchmark dataset for NER is a collection of documents from Reuters newswire articles, annotated with four entity types: person, location, organization, and miscellaneous. We train models on CoNLL-03 training data on test on three datasets: 1) ConLL-03 testing data, 2) ACE Phase 2 (2001-02) and ACE-2003 data, and 3) MUC7 Formal Run test set. We adopt the BIO2 annotation standard, as well as all the preprocessing steps described in (Wang and Manning, 2013). We use a comprehensive set of discrete features that comes with the standard distribution of the Stanford NER model (Finkel et al., 2005). A total of 437,905 discrete features were generated for the CoNLL-2003 training dataset. In addition, 50-dimensional vectors for each word of a five-word context are added and used as continuous features. With these features as input, we trained a conditional random field (CRF) with exactly the same setup as the CRF_{join} model of (Wang and Manning, 2013).

4.2 Corpora and training details

We trained our model on five corpora of varying sizes: a 2010 Wikipedia dump with 1 billion tokens; a 2014 Wikipedia dump with 1.6 billion tokens; Gigaword 5 which has 4.3 billion tokens; the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens; and on 42 billion tokens of web data, from Common Crawl[5]. We tokenize and lowercase each corpus with the Stanford tokenizer, build a vocabulary of the 400,000 most frequent words[6], and then construct a matrix of co-occurrence counts X. In constructing X, we must choose how large the context window should be and whether to distinguish left context from right context. We explore the effect of these choices below. In all cases we use a decreasing weighting function, so that word pairs that are d words apart contribute $1/d$ to the total count. This is one way to account for the fact that very distant word pairs are expected to contain less relevant information about the words' relationship to one another.

For all our experiments, we set $x_{\text{max}} = 100$, $\alpha = 3/4$, and train the model using AdaGrad (Duchi et al., 2011), stochastically sampling nonzero elements from X, with initial learning rate of 0.05. We run 50 iterations for vectors smaller than 300 dimensions, and 100 iterations otherwise (see Section 4.6 for more details about the convergence rate). Unless otherwise noted, we use a context of ten words to the left and ten words to the right.

The model generates two sets of word vectors, W and $\tilde{W}$. When X is symmetric, W and $\tilde{W}$ are equivalent and differ only as a result of their random initializations; the two sets of vectors should perform equivalently. On the other hand, there is evidence that for certain types of neural networks, training multiple instances of the network and then combining the results can help reduce overfitting and noise and generally improve results (Ciresan et al., 2012). With this in mind, we choose to use

the analogy task. This number is evaluated on a subset of the dataset so it is not included in Table 2. 3CosMul performed worse than cosine similarity in almost all of our experiments.

[5]To demonstrate the scalability of the model, we also trained it on a much larger sixth corpus, containing 840 billion tokens of web data, but in this case we did not lowercase the vocabulary, so the results are not directly comparable.

[6]For the model trained on Common Crawl data, we use a larger vocabulary of about 2 million words.

the sum $W + \tilde{W}$ as our word vectors. Doing so typically gives a small boost in performance, with the biggest increase in the semantic analogy task.

We compare with the published results of a variety of state-of-the-art models, as well as with our own results produced using the `word2vec` tool and with several baselines using SVDs. With `word2vec`, we train the skip-gram (SG†) and continuous bag-of-words (CBOW†) models on the 6 billion token corpus (Wikipedia 2014 + Gigaword 5) with a vocabulary of the top 400,000 most frequent words and a context window size of 10. We used 10 negative samples, which we show in Section 4.6 to be a good choice for this corpus.

For the SVD baselines, we generate a truncated matrix X_{trunc} which retains the information of how frequently each word occurs with only the top 10,000 most frequent words. This step is typical of many matrix-factorization-based methods as the extra columns can contribute a disproportionate number of zero entries and the methods are otherwise computationally expensive.

The singular vectors of this matrix constitute the baseline "SVD". We also evaluate two related baselines: "SVD-S" in which we take the SVD of $\sqrt{X_{\text{trunc}}}$, and "SVD-L" in which we take the SVD of $\log(1+X_{\text{trunc}})$. Both methods help compress the otherwise large range of values in X.[7]

4.3 Results

We present results on the word analogy task in Table 2. The GloVe model performs significantly better than the other baselines, often with smaller vector sizes and smaller corpora. Our results using the `word2vec` tool are somewhat better than most of the previously published results. This is due to a number of factors, including our choice to use negative sampling (which typically works better than the hierarchical softmax), the number of negative samples, and the choice of the corpus.

We demonstrate that the model can easily be trained on a large 42 billion token corpus, with a substantial corresponding performance boost. We note that increasing the corpus size does not guarantee improved results for other models, as can be seen by the decreased performance of the SVD-L model on this larger corpus. The fact that this basic SVD model does not scale well to large corpora lends further evidence to the necessity of the type of weighting scheme proposed in our model.

Table 3: Spearman rank correlation on word similarity tasks. All vectors are 300-dimensional. The CBOW* vectors are from the `word2vec` website and differ in that they contain phrase vectors.

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

Table 3 shows results on five different word similarity datasets. A similarity score is obtained from the word vectors by first normalizing each feature across the vocabulary and then calculating the cosine similarity. We compute Spearman's rank correlation coefficient between this score and the human judgments. CBOW* denotes the vectors available on the `word2vec` website that are trained with word and phrase vectors on 100B words of news data. GloVe outperforms it while using a corpus less than half the size.

Table 4 shows results on the NER task with the CRF-based model. The L-BFGS training terminates when no improvement has been achieved on the dev set for 25 iterations. Otherwise all configurations are identical to those used by Wang and Manning (2013). The model labeled *Discrete* is the baseline using a comprehensive set of discrete features that comes with the standard distribution of the Stanford NER model, but with no word vector features. In addition to the HPCA and SVD models discussed previously, we also compare to the models of Huang et al. (2012) (HSMN) and Collobert and Weston (2008) (CW). We trained the CBOW model using the `word2vec` tool[8]. The GloVe model outperforms all other methods on all evaluation metrics, except for the CoNLL test set, on which the HPCA method does slightly better. We conclude that the GloVe vectors are useful in downstream NLP tasks, as was first

[7] We also investigated several other weighting schemes for transforming X; what we report here performed best. Many weighting schemes like PPMI destroy the sparsity of X and therefore cannot feasibly be used with large vocabularies. With smaller vocabularies, these information-theoretic transformations do indeed work well on word similarity measures, but they perform very poorly on the word analogy task.

[8] We use the same parameters as above, except in this case we found 5 negative samples to work slightly better than 10.

Table 4: F1 score on NER task with 50d vectors. *Discrete* is the baseline without word vectors. We use publicly-available vectors for HPCA, HSMN, and CW. See text for details.

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

shown for neural vectors in (Turian et al., 2010).

4.4 Model Analysis: Vector Length and Context Size

In Fig. 2, we show the results of experiments that vary vector length and context window. A context window that extends to the left and right of a target word will be called symmetric, and one which extends only to the left will be called asymmetric. In (a), we observe diminishing returns for vectors larger than about 200 dimensions. In (b) and (c), we examine the effect of varying the window size for symmetric and asymmetric context windows. Performance is better on the syntactic subtask for small and asymmetric context windows, which aligns with the intuition that syntactic information is mostly drawn from the immediate context and can depend strongly on word order. Semantic information, on the other hand, is more frequently non-local, and more of it is captured with larger window sizes.

4.5 Model Analysis: Corpus Size

In Fig. 3, we show performance on the word analogy task for 300-dimensional vectors trained on different corpora. On the syntactic subtask, there is a monotonic increase in performance as the corpus size increases. This is to be expected since larger corpora typically produce better statistics. Interestingly, the same trend is not true for the semantic subtask, where the models trained on the smaller Wikipedia corpora do better than those trained on the larger Gigaword corpus. This is likely due to the large number of city- and country-based analogies in the analogy dataset and the fact that Wikipedia has fairly comprehensive articles for most such locations. Moreover, Wikipedia's

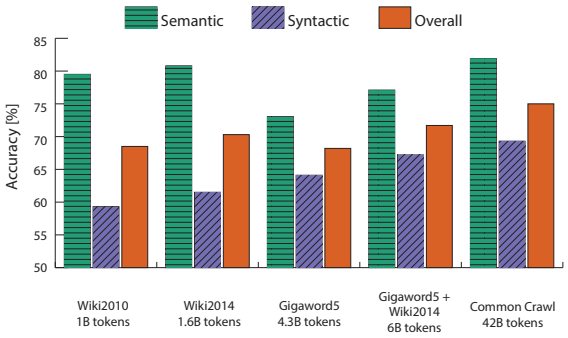


Figure 3: Accuracy on the analogy task for 300-dimensional vectors trained on different corpora.

entries are updated to assimilate new knowledge, whereas Gigaword is a fixed news repository with outdated and possibly incorrect information.

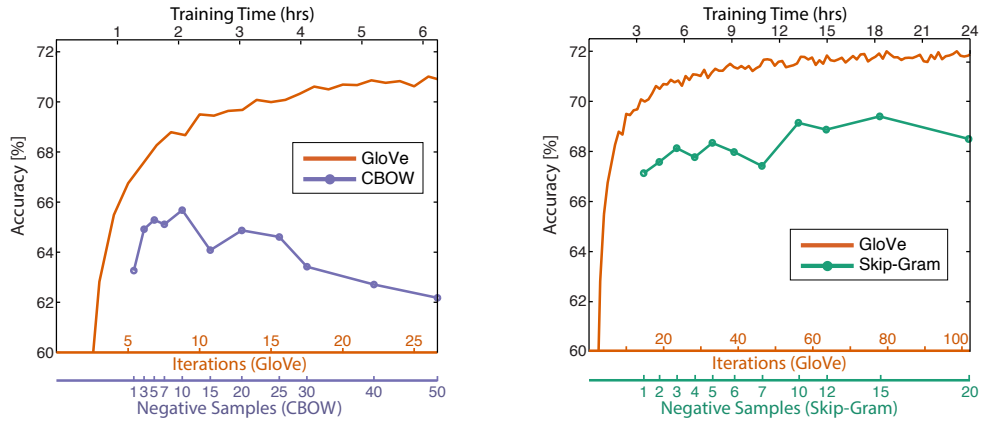
4.6 Model Analysis: Run-time

The total run-time is split between populating X and training the model. The former depends on many factors, including window size, vocabulary size, and corpus size. Though we did not do so, this step could easily be parallelized across multiple machines (see, e.g., Lebret and Collobert (2014) for some benchmarks). Using a single thread of a dual 2.1GHz Intel Xeon E5-2658 machine, populating X with a 10 word symmetric context window, a 400,000 word vocabulary, and a 6 billion token corpus takes about 85 minutes. Given X, the time it takes to train the model depends on the vector size and the number of iterations. For 300-dimensional vectors with the above settings (and using all 32 cores of the above machine), a single iteration takes 14 minutes. See Fig. 4 for a plot of the learning curve.

4.7 Model Analysis: Comparison with `word2vec`

A rigorous quantitative comparison of GloVe with `word2vec` is complicated by the existence of many parameters that have a strong effect on performance. We control for the main sources of variation that we identified in Sections 4.4 and 4.5 by setting the vector length, context window size, corpus, and vocabulary size to the configuration mentioned in the previous subsection.

The most important remaining variable to control for is training time. For GloVe, the relevant parameter is the number of training iterations. For `word2vec`, the obvious choice would be the number of training epochs. Unfortunately, the code is currently designed for only a single epoch:



(a) GloVe vs CBOW

(b) GloVe vs Skip-Gram

Figure 4: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus (Wikipedia 2014 + Gigaword 5) with the same 400,000 word vocabulary, and use a symmetric context window of size 10.

it specifies a learning schedule specific to a single pass through the data, making a modification for multiple passes a non-trivial task. Another choice is to vary the number of negative samples. Adding negative samples effectively increases the number of training words seen by the model, so in some ways it is analogous to extra epochs.

We set any unspecified parameters to their default values, assuming that they are close to optimal, though we acknowledge that this simplification should be relaxed in a more thorough analysis.

In Fig. 4, we plot the overall performance on the analogy task as a function of training time. The two x-axes at the bottom indicate the corresponding number of training iterations for GloVe and negative samples for `word2vec`. We note that `word2vec`'s performance actually decreases if the number of negative samples increases beyond about 10. Presumably this is because the negative sampling method does not approximate the target probability distribution well.[9]

For the same corpus, vocabulary, window size, and training time, GloVe consistently outperforms `word2vec`. It achieves better results faster, and also obtains the best results irrespective of speed.

5 Conclusion

Recently, considerable attention has been focused on the question of whether distributional word representations are best learned from count-based methods or from prediction-based methods. Currently, prediction-based models garner substantial support; for example, Baroni et al. (2014) argue that these models perform better across a range of tasks. In this work we argue that the two classes of methods are not dramatically different at a fundamental level since they both probe the underlying co-occurrence statistics of the corpus, but the efficiency with which the count-based methods capture global statistics can be advantageous. We construct a model that utilizes this main benefit of count data while simultaneously capturing the meaningful linear substructures prevalent in recent log-bilinear prediction-based methods like `word2vec`. The result, GloVe, is a new global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks.

[9] In contrast, noise-contrastive estimation is an approximation which improves with more negative samples. In Table 1 of (Mnih et al., 2013), accuracy on the analogy task is a non-decreasing function of the number of negative samples.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. Stanford University gratefully acknowledges the support of the Defense Threat Reduction Agency (DTRA) under Air Force Research Laboratory (AFRL) contract no. FA8650-10-C-7020 and the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Filtering of Text (DEFT) Program under AFRL contract no. FA8750-13-2-0040. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DTRA, AFRL, DEFT, or the US government.

References

Tom M. Apostol. 1976. *Introduction to Analytic Number Theory*. Introduction to Analytic Number Theory.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL*.

Yoshua Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *JMLR*, 3:1137–1155.

John A. Bullinaria and Joseph P. Levy. 2007. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526.

Dan C. Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. 2012. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, pages 2852–2860.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of ICML*, pages 160–167.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *JMLR*, 12:2493–2537.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12.

Lev Finkelstein, Evgenly Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM.

Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2012. Improving Word Representations via Global Context and Multiple Word Prototypes. In *ACL*.

Rémi Lebret and Ronan Collobert. 2014. Word embeddings through Hellinger PCA. In *EACL*.

Omer Levy, Yoav Goldberg, and Israel Ramat-Gan. 2014. Linguistic regularities in sparse and explicit word representations. *CoNLL-2014*.

Kevin Lund and Curt Burgess. 1996. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instrumentation, and Computers*, 28:203–208.

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

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshop Papers*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.

Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. 2013c. Linguistic regularities in continuous space word representations. In *HLT-NAACL*.

George A. Miller and Walter G. Charles. 1991. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*.

Douglas L. T. Rohde, Laura M. Gonnerman, and David C. Plaut. 2006. An improved model of semantic similarity based on lexical co-occurence. *Communications of the ACM*, 8:627–633.

Herbert Rubenstein and John B. Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing With Compositional Vector Grammars. In *ACL*.

Stefanie Tellex, Boris Katz, Jimmy Lin, Aaron Fernandes, and Gregory Marton. 2003. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the SIGIR Conference on Research and Development in Informaion Retrieval*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *CoNLL-2003*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394.

Mengqiu Wang and Christopher D. Manning. 2013. Effect of non-linear deep architecture in sequence labeling. In *Proceedings of the 6th International Joint Conference on Natural Language Processing (IJCNLP)*.