From Paraphrase Database to Compositional Paraphrase Model and Back

John Wieting* **Mohit Bansal†** **Kevin Gimpel†** **Karen Livescu†**
*University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA
wieting2@illinois.edu
†Toyota Technological Institute at Chicago, Chicago, IL, 60637, USA
{mbansal,kgimpel,klivescu}@ttic.edu

Abstract

The Paraphrase Database (PPDB; Ganitkevitch et al., 2013) is an extensive semantic resource, consisting of a list of phrase pairs with (heuristic) confidence estimates. However, it is still unclear how it can best be used, due to the heuristic nature of the confidences and its necessarily incomplete coverage. We propose models to leverage the phrase pairs from the PPDB to build parametric paraphrase models that score paraphrase pairs more accurately than the PPDB's internal scores while simultaneously improving its coverage. They allow for learning phrase embeddings as well as improved word embeddings. Moreover, we introduce two new, manually annotated datasets to evaluate short-phrase paraphrasing models. Using our paraphrase model trained using PPDB, we achieve state-of-the-art results on standard word and bigram similarity tasks and beat strong baselines on our new short phrase paraphrase tasks.[1]

1 Introduction

Paraphrase detection[2] is the task of analyzing two segments of text and determining if they have the same meaning despite differences in structure and wording. It is useful for a variety of NLP tasks like question answering (Rinaldi et al., 2003; Fader et al., 2013), semantic parsing (Berant and Liang, 2014), textual entailment (Bosma and Callison-Burch, 2007), and machine translation (Marton et al., 2009).

One component of many such systems is a paraphrase table containing pairs of text snippets, usually automatically generated, that have the same meaning. The most recent work in this area is the Paraphrase Database (PPDB; Ganitkevitch et al., 2013), a collection of confidence-rated paraphrases created using the pivoting technique of Bannard and Callison-Burch (2005) over large parallel corpora. The PPDB is a massive resource, containing 220 million paraphrase pairs. It captures many short paraphrases that would be difficult to obtain using any other resource. For example, the pair {*we must do our utmost*, *we must make every effort*} has little lexical overlap but is present in PPDB. The PPDB has recently been used for monolingual alignment (Yao et al., 2013), for predicting sentence similarity (Bjerva et al., 2014), and to improve the coverage of FrameNet (Rastogi and Van Durme, 2014).

Though already effective for multiple NLP tasks, we note some drawbacks of PPDB. The first is lack of coverage: to use the PPDB to compare two phrases, both must be in the database. The second is that PPDB is a nonparametric paraphrase model; the number of parameters (phrase pairs) grows with the size of the dataset used to build it. In practice, it can become unwieldy to work with as the size of the database increases. A third concern is that the confidence estimates in PPDB are a heuristic combination of features, and their quality is unclear.

We address these issues in this work by introducing ways to use PPDB to construct parametric paraphrase models. First we show that initial skip-gram word vectors (Mikolov et al., 2013a) can be fine-tuned for the paraphrase task by training on word pairs from PPDB. We call them **PARAGRAM** word vectors. We find additive composition of PARAGRAM vectors to be a simple but effective way to

[1] We release our datasets, code, and trained models at http://web.engr.illinois.edu/~wieting2/.

[2] See Androutsopoulos and Malakasiotis (2010) for a survey on approaches for detecting paraphrases.

Transactions of the Association for Computational Linguistics, vol. 3, pp. 345–358, 2015. Action Editor: Hal Daumeé III.
Submission batch: 1/2015; Revision batch 4/2015; Published 6/2015.

embed phrases for short-phrase paraphrase tasks. We find improved performance by training a recursive neural network (RNN; Socher et al., 2010) directly on phrase pairs from PPDB.

We show that our resulting word and phrase representations are effective on a wide variety of tasks, including two new datasets that we introduce. The first, Annotated-PPDB, contains pairs from PPDB that were scored by human annotators. It can be used to evaluate paraphrase models for short phrases. We use it to show that the phrase embeddings produced by our methods are significantly more indicative of paraphrasability than the original heuristic scoring used by Ganitkevitch et al. (2013). Thus we use the power of PPDB to improve its contents.

Our second dataset, ML-Paraphrase, is a re-annotation of the bigram similarity corpus from Mitchell and Lapata (2010). The task was originally developed to measure semantic similarity of bigrams, but some annotations are not congruent with the functional similarity central to paraphrase relationships. Our re-annotation can be used to assess paraphrasing capability of bigram compositional models.

In summary, we make the following contributions:

Provide new PARAGRAM word vectors, learned using PPDB, that achieve state-of-the-art performance on the SimLex-999 lexical similarity task (Hill et al., 2014b) and lead to improved performance in sentiment analysis.

Provide ways to use PPDB to embed phrases. We compare additive and RNN composition of PARAGRAM vectors. Both can improve PPDB by re-ranking the paraphrases in PPDB to improve correlations with human judgments. They can be used as concise parameterizations of PPDB, thereby vastly increasing its coverage. We also perform a qualitative analysis of the differences between additive and RNN composition.

Introduce two new datasets. The first contains PPDB phrase pairs and evaluates how well models can measure the quality of short paraphrases. The second is a new annotation of the bigram similarity task in Mitchell and Lapata (2010) that makes it suitable for evaluating bigram paraphrases.

We release the new datasets, complete with annotation instructions and raw annotations, as well as our code and the trained models.[3]

2 Related Work

There is a vast literature on representing words as vectors. The intuition of most methods to create these vectors (or embeddings) is that similar words have similar contexts (Firth, 1957). Earlier models made use of latent semantic analysis (LSA) (Deerwester et al., 1990). Recently, more sophisticated neural models, work originating with (Bengio et al., 2003), have been gaining popularity (Mikolov et al., 2013a; Pennington et al., 2014). These embeddings are now being used in new ways as they are being tailored to specific downstream tasks (Bansal et al., 2014).

Phrase representations can be created from word vectors using compositional models. Simple but effective compositional models were studied by Mitchell and Lapata (2008; 2010) and Blacoe and Lapata (2012). They compared a variety of binary operations on word vectors and found that simple point-wise multiplication of explicit vector representations performed very well. Other works like Zanzotto et al. (2010) and Baroni and Zamparelli (2010) also explored composition using models based on operations of vectors and matrices.

More recent work has shown that the extremely efficient neural embeddings of Mikolov et al. (2013a) also do well on compositional tasks simply by adding the word vectors (Mikolov et al., 2013b). Hashimoto et al. (2014) introduced an alternative word embedding and compositional model based on predicate-argument structures that does well on two simple composition tasks, including the one introduced by Mitchell and Lapata (2010).

An alternative approach to composition, used by Socher et al. (2011), is to train a recursive neural network (RNN) whose structure is defined by a binarized parse tree. In particular, they trained their RNN as an unsupervised autoencoder. The RNN captures the latent structure of composition. Recent work has shown that this model struggles in tasks involving compositionality (Blacoe and Lapata, 2012; Hashimoto et al., 2014).[4] However, we found suc-

[3] http://web.engr.illinois.edu/~wieting2/

[4] We also replicated this approach and found training to be time-consuming even using low-dimensional word vectors.

cess using RNNs in a *supervised* setting, similar to Socher et al. (2014), who used RNNs to learn representations for image descriptions. The objective function we used in this work was motivated by their multimodal objective function for learning joint image-sentence representations.

Lastly, the PPDB has been used along with other resources to learn word embeddings for several tasks, including semantic similarity, language modeling, predicting human judgments, and classification (Yu and Dredze, 2014; Faruqui et al., 2015). Concurrently with our work, it has also been used to construct paraphrase models for short phrases (Yu and Dredze, 2015).

3 New Paraphrase Datasets

We created two novel datasets: (1) Annotated-PPDB, a subset of phrase pairs from PPDB which are annotated according to how strongly they represent a paraphrase relationship, and (2) ML-Paraphrase, a re-annotation of the bigram similarity dataset from Mitchell and Lapata (2010), again annotated for strength of paraphrase relationship.

3.1 Annotated-PPDB

Our motivation for creating Annotated-PPDB was to establish a way to evaluate compositional paraphrase models on *short phrases*. Most existing paraphrase tasks focus on words, like SimLex-999 (Hill et al., 2014b), or entire sentences, such as the Microsoft Research Paraphrase Corpus (Dolan et al., 2004; Quirk et al., 2004). To our knowledge, there are no datasets that focus on the paraphrasability of short phrases. Thus, we created Annotated-PPDB so that researchers can focus on local compositional phenomena and measure the performance of models directly—avoiding the need to do so indirectly in a sentence-level task. Models that have strong performance on Annotated-PPDB can be used to provide more accurate confidence scores for the paraphrases in the PPDB as well as reduce the need for large paraphrase tables altogether.

Annotated-PPDB was created in a multi-step process (outlined below) involving various automatic filtering steps followed by crowdsourced human annotation. One of the aims for our dataset was to collect a variety of paraphrase types—we wanted to include pairs that were non-trivial to recognize as well as those with a range of similarity and length. We focused on phrase pairs with limited lexical overlap to avoid including those with only trivial differences.

We started with candidate phrases extracted from the first 10M pairs in the XXL version of the PPDB and then executed the following steps.[5]

Filter phrases for quality: Only those phrases whose tokens were in our vocabulary were retained.[6] Next, all duplicate paraphrase pairs were removed; in PPDB, these are distinct pairs that contain the same two phrases with the order swapped.

Filter by lexical overlap: Next, we calculated the *word overlap score* in each phrase pair and then retained only those pairs that had a score of less than 0.5. By *word overlap score*, we mean the fraction of tokens in the smaller of the phrases with Levenshtein distance ≤ 1 to a token in the larger of the phrases. This was done to exclude less interesting phrase pairs like $\langle$*my dad had*, *my father had*$\rangle$ or $\langle$*ballistic missiles*, *of ballistic missiles*$\rangle$ that only differ in a synonym or the addition of a single word.

Select range of paraphrasabilities: To balance our dataset with both clear paraphrases and erroneous pairs in PPDB, we sampled 5,000 examples from ten chunks of the first 10M initial phrase pairs where a chunk is defined as 1M phrase pairs.

Select range of phrase lengths: We then selected 1,500 phrases from each 5000-example sample that encompassed a wide range of phrase lengths. To do this, we first binned the phrase pairs by their *effective size*. Let n_1 be the number of tokens of length greater than one character in the first phrase and n_2 the same for the second phrase. Then the *effective size* is defined as $\max(n_1, n_2)$. The bins contained pairs of *effective size* of 3, 4, and 5 or more, and 500 pairs were selected from each bin. This gave us a total of 15,000 phrase pairs.

Prune to 3,000: 3,000 phrase pairs were then selected randomly from the 15,000 remaining pairs to

[5]Note that the confidence scores for phrase pairs in PPDB are based on a weighted combination of features with weights determined heuristically. The confidence scores were used to place the phrase pairs into their respective sets (S, M, L, XL, XXL, etc.), where each larger set subsumes all smaller ones.

[6]Throughout, our vocabulary is defined as the most common 100K word types in English Wikipedia, following tokenization and lowercasing (see §5).

form an initial dataset, Annotated-PPDB-3K. The phrases were selected so that every phrase in the dataset was unique.

Annotate with Mechanical Turk: The dataset was then rated on a scale from 1-5 using Amazon Mechanical Turk, where a score of 5 denoted phrases that are equivalent in a large number of contexts, 3 meant that the phrases had some overlap in meaning, and 1 indicated that the phrases were dissimilar or contradictory in some way (e.g., *can not adopt* and *is able to accept*).

We only permitted workers whose location was in the United States and who had done at least 1,000 HITS with a 99% acceptance rate. Each example was labeled by 5 annotators and their scores were averaged to produce the final rating. Table 1 shows some statistics of the data. Overall, the annotated data had a mean deviation (MD)[7] of 0.80. Table 1 shows that overall, workers found the phrases to be of high quality, as more than two-thirds of the pairs had an average score of at least 3. Also from the Table, we can see that workers had stronger agreement on very low and very high quality pairs and were less certain in the middle of the range.

Prune to 1,260: To create our final dataset, Annotated-PPDB, we selected 1,260 phrase pairs from the 3,000 annotations. We did this by first binning the phrases into 3 categories: those with scores in the interval $[1, 2.5)$, those with scores in the interval $[2.5, 3.5]$, and those with scores in the interval $(3.5, 5]$. We took the 420 phrase pairs with the lowest MD in each bin, as these have the most agreement about their label, to form Annotated-PPDB.

These 1,260 examples were then randomly split into a development set of 260 examples and a test set of 1,000 examples. The development set had an MD of 0.61 and the test set had an MD of 0.60, indicating the final dataset had pairs of higher agreement than the initial 3,000.

3.2 ML-Paraphrase

Our second newly-annotated dataset, ML-Paraphrase, is based on the bigram similarity task originally introduced by Mitchell and Lapata (2010); we refer to the original annotations as the ML dataset.

Score Range	MD	% of Data
$[1, 2)$	0.66	8.1
$[2, 3)$	1.05	20.0
$[3, 4)$	0.93	34.9
$[4, 5]$	0.59	36.9

Table 1: An analysis of Annotated-PPDB-3K extracted from PPDB. The statistics shown are for the splits of the data according to the average score by workers. MD denotes mean deviation and % of Data refers to the percentage of our dataset that fell into each range.

The ML dataset consists of human similarity ratings for three types of bigrams: adjective-noun (JN), noun-noun (NN), and verb-noun (VN). Through manual inspection, we found that the annotations were not consistent with the notion of similarity central to paraphrase tasks. For instance, *television set* and *television programme* were the highest rated phrases in the NN section (based on average annotator score). Similarly, one of the highest ranked JN pairs was *older man* and *elderly woman*. This indicates that the annotations reflect topical similarity in addition to capturing functional or definitional similarity.

Therefore, we had the data re-annotated by two authors of this paper who are native English speakers.[8] The bigrams were labeled on a scale from 1-5 where 5 denotes phrases that are equivalent in a large number of contexts, 3 indicates the phrases are roughly equivalent in a narrow set of contexts, and 1 means the phrases are not at all equivalent in any context. Following annotation, we collapsed the rating scale by merging 4s and 5s together and 1s and 2s together.

Data	IA ρ	IA κ	ML comp. ρ	ML Human ρ
JN	0.87	0.79	0.56	0.52
NN	0.64	0.58	0.38	0.49
VN	0.73	0.73	0.55	0.55

Table 2: Inter-annotator agreement of ML-Paraphrase and comparison with ML dataset. Columns 2 and 3 show the inter-annotator agreement between the two annotators measured with Spearman ρ and Cohen's κ. Column 4 shows the ρ between ML-Paraphrase and all of the ML dataset. The last column is the average human ρ on the ML dataset.

[7] MD is similar to standard deviation, but uses absolute value instead of squared value and thus is both more intuitive and less sensitive to outliers.

[8] We tried using Mechanical Turk here, but due to such short phrases, with few having the paraphrase relationship, workers did not perform well on the task.

Statistics for the data are shown in Table 2. We show inter-annotator Spearman ρ and Cohen's κ in columns 2 and 3, indicating substantial agreement on the JN and VN portions but only moderate agreement on NN. In fact, when evaluating our NN annotations against those from the original ML data (column 4), we find ρ to be 0.38, well below the average human correlation of 0.49 (final column) reported by Mitchell and Lapata and also surpassed by pointwise multiplication (Mitchell and Lapata, 2010). This suggests that the original NN portion, more so than the others, favored a notion of similarity more related to association than paraphrase.

4 Paraphrase Models

We now present parametric paraphrase models and discuss training. Our goal is to embed phrases into a low-dimensional space such that cosine similarity in the space corresponds to the strength of the paraphrase relationship between phrases.

We use a recursive neural network (RNN) similar to that used by Socher et al. (2014). We first use a constituent parser to obtain a binarized parse of a phrase. For phrase p, we compute its vector $g(p)$ through recursive computation on the parse. That is, if phrase p is the yield of a parent node in a parse tree, and phrases c_1 and c_2 are the yields of its two child nodes, we define $g(p)$ recursively as follows:

$$g(p) = f(W[g(c_1); g(c_2)] + b)$$

where f is an element-wise activation function (tanh), $[g(c_1); g(c_2)] \in \mathbb{R}^{2n}$ is the concatenation of the child vectors, $W \in \mathbb{R}^{n \times 2n}$ is the composition matrix, $b \in \mathbb{R}^n$ is the offset, and n is the dimensionality of the word embeddings. If node p has no children (i.e., it is a single token), we define $g(p) = W_w^{(p)}$, where W_w is the word embedding matrix in which particular word vectors are indexed using superscripts. The trainable parameters of the model are W, b, and W_w.

4.1 Objective Functions

We now present objective functions for training on pairs extracted from PPDB. The training data consists of (possibly noisy) pairs taken directly from the original PPDB. In subsequent sections, we discuss how we extract training pairs for particular tasks.

We assume our training data consists of a set X of phrase pairs $\langle x_1, x_2 \rangle$, where x_1 and x_2 are assumed to be paraphrases. To learn the model parameters (W, b, W_w), we minimize our objective function over the data using AdaGrad (Duchi et al., 2011) with mini-batches. The objective function follows:

$$\begin{aligned} \min_{W,b,W_w} \frac{1}{|X|} \Bigg(& \sum_{\langle x_1, x_2 \rangle \in X} \\ & \max(0, \delta - g(x_1) \cdot g(x_2) + g(x_1) \cdot g(t_1)) \\ & + \max(0, \delta - g(x_1) \cdot g(x_2) + g(x_2) \cdot g(t_2)) \Bigg) \\ & + \lambda_W (\|W\|^2 + \|b\|^2) + \lambda_{W_w} \|W_{w_{initial}} - W_w\|^2 \end{aligned} \tag{1}$$

where λ_W and λ_{W_w} are regularization parameters, $W_{w_{initial}}$ is the initial word embedding matrix, δ is the **margin** (set to 1 in all of our experiments), and t_1 and t_2 are carefully-selected **negative examples** taken from a mini-batch during optimization.

The intuition for this objective is that we want the two phrases to be more similar to each other $(g(x_1) \cdot g(x_2))$ than either is to their respective negative examples t_1 and t_2, by a margin of at least δ.

Selecting Negative Examples To select t_1 and t_2 in Eq. 1, we simply chose the most similar phrase in the mini-batch (other than those in the given phrase pair). E.g., for choosing t_1 for a given $\langle x_1, x_2 \rangle$:

$$t_1 = \operatorname*{argmax}_{t:\langle t, \cdot \rangle \in X_b \setminus \{\langle x_1, x_2 \rangle\}} g(x_1) \cdot g(t)$$

where $X_b \subseteq X$ is the current mini-batch. That is, we want to choose a negative example t_i that is similar to x_i according to the current model parameters. The downside of this approach is that we may occasionally choose a phrase t_i that is actually a true paraphrase of x_i. We also tried a strategy in which we selected the least similar phrase that would trigger an update (i.e., $g(t_i) \cdot g(x_i) > g(x_1) \cdot g(x_2) - \delta$), but we found the simpler strategy above to work better and used it for all experiments reported below.

Discussion The objective in Eq. 1 is similar to one used by Socher et al. (2014), but with several differences. Their objective compared text and projected images. They also did not update the underlying word embeddings; we do so here, and in a way such

that they are penalized from deviating from their initialization. Also for a given $\langle x_1, x_2 \rangle$, they do not select a single t_1 and t_2 as we do, but use the entire training set, which can be very expensive with a large training dataset.

We also experimented with a simpler objective that sought to directly minimize the squared L2-norm between $g(x_1)$ and $g(x_2)$ in each pair, along with the same regularization terms as in Eq. 1. One problem with this objective function is that the global minimum is 0 and is achieved simply by driving the parameters to 0. We obtained much better results using the objective in Eq. 1.

Training Word Paraphrase Models To train just word vectors on word paraphrase pairs (again from PPDB), we used the same objective function as above, but simply dropped the composition terms. This gave us an objective that bears some similarity to the skip-gram objective with negative sampling in `word2vec` (Mikolov et al., 2013a). Both seek to maximize the dot products of certain word pairs while minimizing the dot products of others. This objective function is:

$$\min_{W_w} \frac{1}{|X|} \Bigg(\sum_{\langle x_1, x_2 \rangle \in X} \max(0, \delta - W_w^{(x_1)} \cdot W_w^{(x_2)} + W_w^{(x_1)} \cdot W_w^{(t_1)}) + \max(0, \delta - W_w^{(x_1)} \cdot W_w^{(x_2)} + W_w^{(x_2)} \cdot W_w^{(t_2)}) \Bigg) + \lambda_{W_w} \left\| W_{w_{initial}} - W_w \right\|^2 \quad (2)$$

It is like Eq. 1 except with word vectors replacing the RNN composition function and with the regularization terms on the W and b removed.

We further found we could improve this model by incorporating constraints. From our training pairs, for a given word w, we assembled all other words that were paired with it in PPDB and all of their lemmas. These were then used as constraints during the pairing process: a word t could only be paired with w if it was not in its list of assembled words.

5 Experiments – Word Paraphrasing

We first present experiments on learning lexical paraphrasability. We train on word pairs from PPDB and evaluate on the SimLex-999 dataset (Hill et al., 2014b), achieving the best results reported to date.

5.1 Training Procedure

To learn word vectors that reflect paraphrasability, we optimized Eq. 2. There are many tunable hyperparameters with this objective, so to make training tractable we fixed the initial learning rates for the word embeddings to 0.5 and the margin δ to 1. Then we did a coarse grid search over a parameter space for λ_{W_w} and the mini-batch size. We considered λ_{W_w} values in $\{10^{-2}, 10^{-3}, ..., 10^{-7}, 0\}$ and mini-batch sizes in $\{$100, 250, 500, 1000$\}$. We trained for 20 epochs for each set of hyperparameters using AdaGrad (Duchi et al., 2011).

For all experiments, we initialized our word vectors with skip-gram vectors trained using `word2vec` (Mikolov et al., 2013a). The vectors were trained on English Wikipedia (tokenized and lowercased, yielding 1.8B tokens).[9] We used a window size of 5 and a minimum count cut-off of 60, producing vectors for approximately 270K word types. We retained vectors for only the 100K most frequent words, averaging the rest to obtain a single vector for unknown words. We will refer to this set of the 100K most frequent words as our **vocabulary**.

5.2 Extracting Training Data

For training, we extracted word pairs from the lexical XL section of PPDB. We used the XL data for all experiments, including those for phrases. We used XL instead of XXL because XL has better quality overall while still being large enough so that we could be selective in choosing training pairs. There are a total of 548,085 pairs. We removed 174,766 that either contained numerical digits or words not in our vocabulary. We then removed 260,425 redundant pairs, leaving us with a final training set of 112,894 word pairs.

5.3 Tuning and Evaluation

Hyperparameters were tuned using the wordsim-353 (WS353) dataset (Finkelstein et al., 2001), specifically its similarity (WS-S) and relatedness (WS-R) partitions (Agirre et al., 2009). In particular, we tuned to maximize 2×WS-S correlation minus the WS-R correlation. The idea was to reward vectors with high similarity and relatively low relatedness, in order to target the paraphrase relationship.

[9] We used the December 2, 2013 snapshot.

Model	n	SL999 ρ
skip-gram	25	0.21
skip-gram	1000	0.38
PARAGRAM $_{\text{WS}}$	25	0.56*
+ constraints	25	**0.58***
Hill et al. (2014b)	200	0.446
Hill et al. (2014a)	-	0.52
inter-annotator agreement	N/A	0.67

Table 3: Results on the SimLex-999 (SL999) word similarity task obtained by performing hyperparameter tuning based on $2\times$WS-S $-$WS-R and treating SL999 as a held-out test set. n is word vector dimensionality. A $*$ indicates statistical significance ($p < 0.05$) over the 1000-dimensional skip-gram vectors.

After tuning, we evaluated the best hyperparameters on the SimLex-999 (SL999) dataset (Hill et al., 2014b). We chose SL999 as our primary test set as it most closely evaluates the paraphrase relationship. Even though WS-S is a close approximation to this relationship, it does not include pairs that are merely associated and assigned low scores, which SL999 does (see discussion in Hill et al., 2014b).

Note that for all experiments we used cosine similarity as our similarity metric and evaluated the statistical significance of dependent correlations using the one-tailed method of (Steiger, 1980).

5.4 Results

Table 3 shows results on SL999 when improving the initial word vectors by training on word pairs from PPDB, both with and without constraints. The "PARAGRAM $_{\text{WS}}$" rows show results when tuning to maximize $2\times$WS-S $-$ WS-R. We also show results for strong skip-gram baselines and the best results from the literature, including the state-of-the-art results from Hill et al. (2014a) as well as the inter-annotator agreement from Hill et al. (2014b).[10]

The table illustrates that, by training on PPDB, we can surpass the previous best correlations on SL999 by 4-6% absolute, achieving the best results reported to date. We also find that we can train low-dimensional word vectors that exceed the performance of much larger vectors. This is very useful as using large vectors can increase both time and memory consumption in NLP applications.

To generate word vectors to use for downstream applications, we chose hyperparameters so as to maximize performance on SL999.[11] These word vectors, which we refer to as **PARAGRAM** vectors, had a ρ of 0.57 on SL999. We use them as initial word vectors for the remainder of the paper.

5.5 Sentiment Analysis

As an extrinsic evaluation of our PARAGRAM word vectors, we used them in a convolutional neural network (CNN) for sentiment analysis. We used the simple CNN from Kim (2014) and the binary sentence-level sentiment analysis task from Socher et al. (2013). We used the standard data splits, removing examples with a neutral rating. We trained on all constituents in the training set while only using full sentences from development and test, giving us train/development/test sizes of 67,349/872/1,821.

The CNN uses m-gram filters, each of which is an $m \times n$ vector. The CNN computes the inner product between an m-gram filter and each m-gram in an example, retaining the maximum match (so-called "max-pooling"). The score of the match is a single dimension in a feature vector for the example, which is then associated with a weight in a linear classifier used to predict positive or negative sentiment.

While Kim (2014) used m-gram filters of several lengths, we only used unigram filters. We also fixed the word vectors during learning (called "static" by Kim). After learning, the unigram filters correspond to locations in the fixed word vector space. The learned classifier weights represent how strongly each location corresponds to positive or negative sentiment. We expect this static CNN to be more effective if the word vector space separates positive and negative sentiment.

In our experiments, we compared baseline skip-gram embeddings to our PARAGRAM vectors. We used AdaGrad learning rate of 0.1, mini-batches of size 10, and a dropout rate of 0.5. We used 200 unigram filters and rectified linear units as the activation (applied to the filter output + filter bias). We trained for 30 epochs, predicting labels on the development set after each set of 3,000 examples. We recorded the highest development accuracy and used those parameters to predict labels on the test set.

Results are shown in Table 4. We see improve-

[10] Hill et al. (2014a) did not report the dimensionality of the vectors that led to their state-of-the-art results.

[11] We did not use constraints during training.

word vectors	n	accuracy (%)
skip-gram	25	77.0
skip-gram	50	79.6
PARAGRAM	25	**80.9**

Table 4: Test set accuracies when comparing embeddings in a static CNN on the binary sentiment analysis task from Socher et al. (2013).

ments over the baselines when using PARAGRAM vectors, even exceeding the performance of higher-dimensional skip-gram vectors.

6 Experiments – Compositional Paraphrasing

In this section, we describe experiments on a variety of compositional phrase-based paraphrasing tasks. We start with the simplest case of bigrams, and then proceed to short phrases. For all tasks, we again train on appropriate data from PPDB and test on various evaluation datasets, including our two novel datasets (Annotated-PPDB and ML-Paraphrase).

6.1 Training Procedure

We trained our models by optimizing Eq. 1 using AdaGrad (Duchi et al., 2011). We fixed the initial learning rates to 0.5 for the word embeddings and 0.05 for the composition parameters, and the margin to 1. Then we did a coarse grid search over a parameter space for λ_{W_w}, λ_W, and mini-batch size.

For λ_{W_w}, our search space again consisted of $\{10^{-2}, 10^{-3}, ..., 10^{-7}, 0\}$, for λ_W it was $\{10^{-1}, 10^{-2}, 10^{-3}, 0\}$, and we explored batch sizes of {100, 250, 500, 1000, 2000}. When initializing with PARAGRAM vectors, the search space for λ_{W_w} was shifted upwards to be $\{10, 1, 10^{-1}, 10^{-3}, ..., 10^{-6}\}$ to reflect our increased confidence in the initial vectors. We trained only for 5 epochs for each set of parameters. For baselines, we used the same initial skip-gram vectors as in Section 5.

6.2 Evaluation and Baselines

For all experiments, we again used cosine similarity as our similarity metric and evaluated the statistical significance using the method of (Steiger, 1980).

A baseline used in all compositional experiments is vector addition of skip-gram (or PARAGRAM) word vectors. Unlike explicit word vectors, where point-wise multiplication acts as a conjunction of features and performs well on composition tasks (Mitchell and Lapata, 2008), using addition with skip-gram vectors (Mikolov et al., 2013b) gives better performance than multiplication.

6.3 Bigram Paraphrasability

To evaluate our ability to paraphrase bigrams, we consider the original bigram similarity task from Mitchell and Lapata (2010) as well as our newly-annotated version of it: ML-Paraphrase.

Extracting Training Data Training data for these tasks was extracted from the XL portion of PPDB. The bigram similarity task from Mitchell and Lapata (2010) contains three types of bigrams: adjective-noun (JN), noun-noun (NN), and verb-noun (VN). We aimed to collect pairs from PPDB that mirrored these three types of bigrams.

We found parsing to be unreliable on such short segments of text, so we used a POS tagger (Manning et al., 2014) to tag the tokens in each phrase. We then used the word alignments in PPDB to extract bigrams for training. For JN and NN, we extracted pairs containing aligned, adjacent tokens in the two phrases with the appropriate part-of-speech tag. Thus we extracted pairs like ⟨*easy job*, *simple task*⟩ for the JN section and ⟨*town meeting*, *town council*⟩ for the NN section. We used a different strategy for extracting training data for the VN subset: we took aligned VN tokens and took the closest noun after the verb. This was done to approximate the direct object that would have been ideally extracted with a dependency parse. An example from this section is ⟨*achieve goal*, *achieve aim*⟩.

We removed phrase pairs that (1) contained words not in our vocabulary, (2) were redundant with others, (3) contained brackets, or (4) had Levenshtein distance ≤ 1. The final criterion helps to ensure that we train on phrase pairs with non-trivial differences. The final training data consisted of 133,997 JN pairs, 62,640 VN pairs and 35,601 NN pairs.

Baselines In addition to RNN models, we report baselines that use vector addition as the composition function, both with our skip-gram embeddings and PARAGRAM embeddings from Section 5.

We also compare to several results from prior work. When doing so, we took their *best* correla-

Model			Mitchell and Lapata (2010) Bigrams				ML-Paraphrase			
word vectors	n	comp.	JN	NN	VN	Avg	JN	NN	VN	Avg
skip-gram	25	+	0.36	0.44	0.36	0.39	0.32	0.35	0.42	0.36
PARAGRAM	25	+	0.44*	0.34	0.48*	0.42	0.50*	0.29	**0.58***‡	0.46
PARAGRAM	25	RNN	**0.51***†	0.40†	**0.50***‡	**0.47**	**0.57***‡	**0.44**†	0.55*	**0.52**
Hashimoto et al. (2014)			0.49	0.45	0.46	**0.47**	0.38	0.39	0.45	0.41
Mitchell and Lapata (2010)			0.46	**0.49**	0.38	0.44	-	-	-	-
Human			-	-	-	-	0.87	0.64	0.73	0.75

Table 5: Results on the test section of the bigram similarity task of Mitchell and Lapata (2010) and our newly annotated version (ML-Paraphrase). (n) shows the word vector dimensionality and ("comp.") shows the composition function used: "+" is vector addition and "RNN" is the recursive neural network. The * indicates statistically significant ($p < 0.05$) over the skip-gram model, † statistically significant over the {PARAGRAM, +} model, and ‡ statistically significant over Hashimoto et al. (2014).

tions for each data subset. That is, the JN and NN results from Mitchell and Lapata (2010) use their multiplicative model and the VN results use their dilation model. From Hashimoto et al. (2014) we used their PAS-CLBLM Add_{l} and PAS-CLBLM Add_{nl} models. We note that their vector dimensionalities are larger than ours, using $n = 2000$ and 50 respectively.

Results Results are shown in Table 5. We report results on the test portion of the original Mitchell and Lapata (2010) dataset (ML) as well as the entirety of our newly-annotated dataset (ML-Paraphrase). RNN results on ML were tuned on the respective development sections and RNN results on ML-Paraphrase were tuned on the entire ML dataset.

Our RNN model outperforms results from the literature on most sections in both datasets and its average correlations are among the highest.[12] The one subset of the data that posed difficulty was the NN section of the ML dataset. We suspect this is due to the reasons discussed in Section 3.2; for our ML-Paraphrase dataset, by contrast, we do see gains on the NN section.

We also outperform the strong baseline of adding 1000-dimensional skip-gram embeddings, a model with 40 times the number of parameters, on our ML-Paraphrase dataset. This baseline had correlations of 0.45, 0.43, and 0.47 on the JN, NN, and VN partitions, with an average of 0.45—below the average ρ of the RNN (0.52) and even the {PARAGRAM, +} model (0.46).

[12]The results obtained here differ from those reported in Hashimoto et al. (2014) as we scored their vectors with a newer Python implementation of Spearman ρ that handles ties (Hashimoto, P.C.).

Interestingly, the type of vectors used to initialize the RNN has a significant effect on performance. If we initialize using the 25-dimensional skip-gram vectors, the average ρ on ML-Paraphrase drops to 0.43, below even the {PARAGRAM, +} model.

6.4 Phrase Paraphrasability

In this section we show that by training a model based on filtered phrase pairs in PPDB, we can actually distinguish between quality paraphrases and poor paraphrases in PPDB better than the original heuristic scoring scheme from Ganitkevitch et al. (2013).

Extracting Training Data As before, training data was extracted from the XL section of PPDB. Similar to the procedure to create our Annotated-PPDB dataset, phrases were filtered such that only those with a *word overlap score* of less than 0.5 were kept. We also removed redundant phrases and phrases that contained tokens not in our vocabulary. The phrases were then binned according to their *effective size* and 20,000 examples were selected from bins of *effective sizes* of 3, 4, and more than 5, creating a training set of 60,000 examples. Care was taken to ensure that none of our training pairs was also present in our development and test sets.

Baselines We compare our models with strong lexical baselines. The first, *strict word overlap*, is the percentage of words in the smaller phrase that are also in the larger phrase. We also include a version where the words are lemmatized prior to the calculation.

We also train a support vector regression model (epsilon-SVR) (Chang and Lin, 2011) on the 33 features that are included for each phrase pair in PPDB.

We scaled the features such that each lies in the interval $[-1, 1]$ and tuned the parameters using 5-fold cross validation on our dev set.[13] We then trained on the entire dev set after finding the best performing C and ϵ combination and evaluated on the test set of Annotated-PPDB.

Model			
word vectors	n	comp.	Annotated-PPDB
skip-gram	25	+	0.20
PARAGRAM	25	+	0.32*
PARAGRAM	25	RNN	**0.40***†‡
Ganitkevitch et al. (2013)			0.25
word overlap (strict)			0.26
word overlap (lemmatized)			0.20
PPDB+SVR			0.33

Table 6: Spearman correlation on Annotated-PPDB. The * indicates statistically significant ($p < 0.05$) over the skip-gram model, the † indicates statistically significant over the {PARAGRAM, +} model, and the ‡ indicates statistically significant over PPDB+SVR.

Results We evaluated on our Annotated-PPDB dataset described in §3.1. Table 6 shows the Spearman correlations on the 1000-example test set. RNN models were tuned on the development set of 260 examples. All other methods had no hyperparameters and therefore required no tuning.

We note that the confidence estimates from Ganitkevitch et al. (2013) reach a ρ of 0.25 on the test set, similar to the results of strict overlap. While 25-dimensional skip-gram embeddings only reach 0.20, we can improve this to 0.32 by fine-tuning them using PPDB (thereby obtaining our PARAGRAM vectors). By using the PARAGRAM vectors to initialize the RNN, we reach a correlation of 0.40, which is better than the PPDB confidence estimates by 15% absolute.

We again consider addition of 1000-dimensional skip-gram embeddings as a baseline, and they continue to perform strongly ($\rho = 0.37$). The RNN initialized with PARAGRAM vectors does reach a higher ρ (0.40), but the difference is not statistically significant ($p = 0.16$). Thus we can achieve similarly-strong results with far fewer parameters.

This task also illustrates the importance of initializing our RNN model with appropriate word embeddings. An RNN initialized with skip-gram vectors has a modest ρ of 0.22, well below the ρ of the RNN initialized with PARAGRAM vectors. Clearly, initialization is important when optimizing non-convex objectives like ours, but it is noteworthy that our best results came from first improving the word vectors and then learning the composition model, rather than jointly learning both from scratch.

[13] We tuned both parameters over $\{2^{-10}, 2^{-9}, ..., 2^{10}\}$.

7 Qualitative Analysis

Score Range	+	RNN
$[1, 2)$	2.35	2.08
$[2, 3)$	1.56	1.38
$[3, 4)$	0.87	0.85
$[4, 5]$	0.43	0.47

Table 7: Average absolute error of addition and RNN models on different ranges of gold scores.

We performed a qualitative analysis to uncover sources of error and determine differences between adding PARAGRAM vectors and using an RNN initialized with them. To do so, we took the output of both systems on Annotated-PPDB and mapped their cosine similarities to the interval $[1, 5]$. We then computed their absolute error as compared to the gold ratings.

Table 7 shows how the average of these absolute errors changes with the magnitude of the gold ratings. The RNN performs better (has lower average absolute error) for less similar pairs. Vector addition only does better on the most similar pairs. This is presumably because the most positive pairs have high word overlap and so can be represented effectively with a simpler model.

To further investigate the differences between these models, we removed those pairs with gold scores in $[2, 4]$, in order to focus on pairs with extreme scores. We identified two factors that distinguished the performance between the two models: length ratio and the amount of lexical overlap. We did not find evidence that non-compositional phrases, such as idioms, were a source of error as these were not found in ML-Paraphrase and only appear rarely in Annotated-PPDB.

We define length ratio as simply the number of tokens in the smaller phrase divided by the number of tokens in the larger phrase. Overlap ratio is the number of *equivalent tokens* in the phrase pair di-

Index	Phrase 1	Phrase 2	Length Ratio	Overlap Ratio	Gold	RNN	+
1	scheduled to be held in	that will take place in	1.0	0.4	4.6	2.9	**4.4**
2	according to the paper ,	the newspaper reported that	0.8	0.5	4.6	2.8	**4.1**
3	at no cost to	without charge to	0.75	1.0	4.8	3.1	**4.6**
4	's surname	family name of	0.67	1.0	4.4	2.8	**4.1**
5	could have an impact on	may influence	0.4	0.5	4.6	**4.2**	3.2
6	to participate actively	to play an active role	0.6	0.67	5.0	**4.8**	4.0
7	earliest opportunity	early as possible	0.67	0.0	4.4	**4.3**	2.9
8	does not exceed	is no more than	0.75	0.0	5.0	**4.8**	3.5

Table 8: Illustrative phrase pairs from Annotated-PPDB with gold similarity > 4. The last three columns show the gold similarity score, the similarity score of the RNN model, and the similarity score of vector addition. We note that addition performs better when the pairs have high length ratio (rows 1–2) or overlap ratio (rows 3–4) while the RNN does better when those values are low (rows 5–6 and 7–8 respectively). Boldface indicates smaller error compared to gold scores.

vided by the number of tokens in the smaller of the two phrases. *Equivalent tokens* are defined as tokens that are either exact matches or are paired up in the lexical portion of PPDB used to train the PARAGRAM vectors.

Table 9 shows how the performance of the models changes under different values of length ratio and overlap ratio.[14] The values in this table are the percentage changes in absolute error when using the RNN over the PARAGRAM vector addition model. So negative values indicate superior performance by the RNN.

Length Ratio	$[0, 0.6]$	$(0.6, 0.8]$	$(0.8, 1]$
Positive Examples	-22.4	10.0	35.5
Negative Examples	-9.9	-11.1	-12.2
Both	-13.0	-6.4	-2.0
Overlap Ratio	$[0,\frac{1}{3}]$	$(\frac{1}{3},\frac{2}{3}]$	$(\frac{2}{3},1]$
Positive Examples	-4.5	7.0	19.4
Negative Examples	-11.3	-7.5	-15.0
Both	-10.6	-5.3	0.0

Table 9: Comparison of the addition and RNN model on phrase pairs of different overlap and length ratios. The values in the table are the percent change in absolute error from the addition model to the RNN model. Negative examples are defined as pairs from Annotated-PPDB whose gold score is less than 2 and positive examples are those with scores greater than 4. "Both" refers to both negative and positive examples.

A few trends emerge from this table. One is that as the length ratio increases (i.e., the phrase pairs are closer in length), addition surpasses the RNN for positive examples. For negative examples, the trend is reversed. The same trend appears for overlap ratio. Examples from Annotated-PPDB illustrating these trends on positive examples are shown in Table 8.

When considering both positive and negative examples ("Both"), we see that the RNN excels on the most difficult examples (large differences in phrase length and less lexical overlap). For easier examples, the two fare similarly overall (-2.0 to 0.0% change), but the RNN does much better on negative examples. This aligns with the intuition that addition should perform well when two paraphrastic phrases have high lexical overlap and similar length. But when they are not paraphrases, simple addition is misled and the RNN's learned composition function better captures the relationship. This may suggest new architectures for modeling compositionality differently depending on differences in length and amount of overlap.

8 Conclusion

We have shown how to leverage PPDB to learn state-of-the-art word embeddings and compositional models for paraphrase tasks. Since PPDB was created automatically from parallel corpora, our models are also built automatically. Only small amounts of annotated data are used to tune hyperparameters.

We also introduced two new datasets to evaluate compositional models of short paraphrases[15], filling a gap in the NLP community, as currently there are no datasets created for this purpose. Successful models on these datasets can then be used to extend the coverage of, or provide an alternative to, PPDB.

[14]The bin delimiters were chosen to be uniform over the range of output values of the length ratio ([0.4,1] with one outlier data point removed) and overlap ratio ([0,1]).

[15]http://web.engr.illinois.edu/~wieting2/

There remains a great deal of work to be done in developing new composition models, whether with new network architectures or distance functions. In this work, we based our composition function on constituent parse trees, but this may not be the best approach—especially for short phrases. Dependency syntax may be a better alternative (Socher et al., 2014). Besides improving composition, another direction to explore is how to use models for short phrases in sentence-level paraphrase recognition and other downstream tasks.

Acknowledgements

We thank the editor and the anonymous reviewers as well as Juri Ganitkevitch, Dan Roth, Weiran Wang, and Kazuma Hashimoto for their valuable comments and technical assistance. We also thank Chris Callison-Burch, Dipanjan Das, Kuzman Ganchev, Ellie Pavlick, Slav Petrov, Owen Rambow, David Sontag, Oscar Täckström, Kapil Thadani, Lyle Ungar, Benjamin Van Durme, and Mo Yu for helpful conversations.

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics.

Ion Androutsopoulos and Prodromos Malakasiotis. 2010. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research*, pages 135–187.

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 597–604. Association for Computational Linguistics.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1193. Association for Computational Linguistics.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of ACL*.

Johannes Bjerva, Johan Bos, Rob van der Goot, and Malvina Nissim. 2014. The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. *SemEval 2014*, page 642.

William Blacoe and Mirella Lapata. 2012. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 546–556, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wauter Bosma and Chris Callison-Burch. 2007. Paraphrase substitution for recognizing textual entailment. In *Proceedings of the 7th International Conference on Cross-Language Evaluation Forum: Evaluation of Multilingual and Multi-modal Information Retrieval*, CLEF'06, pages 502–509, Berlin, Heidelberg. Springer-Verlag.

Chih-Chung Chang and Chih-Jen Lin. 2011. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27.

Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407.

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of Coling 2004*, pages 350–356, Geneva, Switzerland, Aug 23–Aug 27. COLING.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1608–1618, Sofia, Bulgaria, August. Association for Computational Linguistics.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615.

Lev Finkelstein, Evgeniy 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.

J.R. Firth. 1957. *A Synopsis of Linguistic Theory, 1930-1955*.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. Ppdb: The paraphrase database. In *HLT-NAACL*, pages 758–764. The Association for Computational Linguistics.

Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly learning word representations and composition functions using predicate-argument structures. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, October. Association for Computational Linguistics.

Felix Hill, KyungHyun Cho, Sebastien Jean, Coline Devin, and Yoshua Bengio. 2014a. Not all neural embeddings are born equal. *arXiv preprint arXiv:1410.0718*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2014b. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *CoRR*, abs/1408.3456.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Yuval Marton, Chris Callison-Burch, and Philip Resnik. 2009. Improved statistical machine translation using monolingually-derived paraphrases. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 381–390, Singapore, August. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *ACL*, pages 236–244. Citeseer.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1439.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12.

Chris Quirk, Chris Brockett, and William Dolan. 2004. Monolingual machine translation for paraphrase generation. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 142–149, Barcelona, Spain, July. Association for Computational Linguistics.

Pushpendre Rastogi and Benjamin Van Durme. 2014. Augmenting FrameNet via PPDB. In *Proceedings of the Second Workshop on EVENTS: Definition, Detection, Coreference, and Representation*, pages 1–5, Baltimore, Maryland, USA, June. Association for Computational Linguistics.

Fabio Rinaldi, James Dowdall, Kaarel Kaljurand, Michael Hess, and Diego Mollá. 2003. Exploiting paraphrases in a question answering system. In *Proceedings of the Second International Workshop on Paraphrasing*, pages 25–32, Sapporo, Japan, July. Association for Computational Linguistics.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9.

Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October. Association for Computational Linguistics.

Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. 2014.

Grounded compositional semantics for finding and describing images with sentences. *TACL*, 2:207–218.

James H Steiger. 1980. Tests for comparing elements of a correlation matrix. *Psychological Bulletin*, 87(2):245.

Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. 2013. Semi-markov phrase-based monolingual alignment. In *EMNLP*, pages 590–600.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 545–550, Baltimore, Maryland, June. Association for Computational Linguistics.

Mo Yu and Mark Dredze. 2015. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3:227–242.

Fabio Massimo Zanzotto, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh Manandhar. 2010. Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1263–1271. Association for Computational Linguistics.