Learning Composition Models for Phrase Embeddings

Mo Yu
Machine Intelligence
& Translation Lab
Harbin Institute of Technology
Harbin, China
gflfof@gmail.com

Mark Dredze
Human Language Technology Center of Excellence
Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD, 21218
mdredze@cs.jhu.edu

Abstract

Lexical embeddings can serve as useful representations for words for a variety of NLP tasks, but learning embeddings for phrases can be challenging. While separate embeddings are learned for each word, this is infeasible for every phrase. We construct phrase embeddings by learning how to compose word embeddings using features that capture phrase structure and context. We propose efficient unsupervised and task-specific learning objectives that scale our model to large datasets. We demonstrate improvements on both language modeling and several phrase semantic similarity tasks with various phrase lengths. We make the implementation of our model and the datasets available for general use.

1 Introduction

Word embeddings learned by neural language models (Bengio et al., 2003; Collobert and Weston, 2008; Mikolov et al., 2013b) have been successfully applied to a range of tasks, including syntax (Collobert and Weston, 2008; Turian et al., 2010; Collobert, 2011) and semantics (Huang et al., 2012; Socher et al., 2013b; Hermann et al., 2014). However, phrases are critical for capturing lexical meaning for many tasks. For example, Collobert and Weston (2008) showed that word embeddings yielded state-of-the-art systems on word-oriented tasks (POS, NER) but performance on phrase oriented tasks, such as SRL, lags behind.

We propose a new method for compositional semantics that *learns* to compose word embeddings into phrases. In contrast to a common approach to phrase embeddings that uses pre-defined composition operators (Mitchell and Lapata, 2008), e.g., component-wise sum/multiplication, we learn composition functions that rely on phrase structure and context. Other work on learning compositions relies on matrices/tensors as transformations (Socher et al., 2011; Socher et al., 2013a; Hermann and Blunsom, 2013; Baroni and Zamparelli, 2010; Socher et al., 2012; Grefenstette et al., 2013). However, this work suffers from two primary disadvantages. First, these methods have high computational complexity for dense embeddings: $O(d^2)$ or $O(d^3)$ for composing every two components with d dimensions. The high computational complexity restricts these methods to use very low-dimensional embeddings (25 or 50). While low-dimensional embeddings perform well for syntax (Socher et al., 2013a) and sentiment (Socher et al., 2013b) tasks, they do poorly on semantic tasks. Second, because of the complexity, they use supervised training with small task-specific datasets. An exception is the unsupervised objective of recursive auto-encoders (Socher et al., 2011). Yet this work cannot utilize contextual features of phrases and still poses scaling challenges.

In this work we propose a novel compositional transformation called the Feature-rich Compositional Transformation (FCT) model. FCT produces phrases from their word components. In contrast to previous work, our approach to phrase composition can efficiently utilize high dimensional embeddings (e.g. $d = 200$) with an unsupervised objective, both of which are critical to doing well on semantics tasks. Our composition function is parameter-

Transactions of the Association for Computational Linguistics, vol. 3, pp. 227–242, 2015. Action Editor: Joakim Nivre.
Submission batch: 2/2015; Revision batch 4/2015; Published 5/2015.

ized to allow the inclusion of features based on the phrase structure and contextual information, including positional indicators of the word components. The phrase composition is a weighted summation of embeddings of component words, where the summation weights are defined by the features, which allows for fast composition.

We discuss a range of training settings for FCT. For tasks with labeled data, we utilize task-specific training. We begin with embeddings trained on raw text and then learn compositional phrase parameters as well as fine-tune the embeddings for the specific task's objective. For tasks with unlabeled data (e.g. most semantic tasks) we can train on a large corpus of unlabeled data. For tasks with both labeled and unlabeled data, we consider a joint training scheme. Our model's efficiency ensures we can incorporate large amounts of unlabeled data, which helps mitigate over-fitting and increases vocabulary coverage.

We begin with a presentation of FCT (§2), including our proposed features for the model. We then present three training settings (§3) that cover language modeling (unsupervised), task-specific training (supervised), and joint (semi-supervised) settings. The remainder of the paper is devoted to evaluation of each of these settings.

2 Feature-rich Compositional Transformations from Words to Phrases

We learn transformations for composing phrase embeddings from the component words based on extracted features from a phrase, where we assume that the phrase boundaries are given. The resulting phrase embedding is based on a per-dimension weighted average of the component phrases. Consider the example of base noun phrases (NP), a common phrase type which we want to compose. Base NPs often have flat structures – all words modify the head noun – which means that our transformation should favor the head noun in the composed phrase embedding. For each of the N words w_i in phrase p we construct the embedding:

$$e_p = \sum_i^N \lambda_i \odot e_{w_i} \quad (1)$$

where e_{w_i} is the embedding for word i; and $\odot$ refers to point-wise product. λ_i is a weight vector that is constructed based on the features of p and the model parameters:

$$\lambda_{ij} = \sum_k \alpha_{jk} f_k(w_i, p) + b_{ij} \quad (2)$$

where $f_k(w_i, p)$ is a feature function that considers word w_i in phrase p and b_{ij} is a bias term. This model is fast to train since it has only linear transformations: the only operations are vector summation and inner product. Therefore, we learn the model parameters α together with the embeddings. We call this the Feature-rich Compositional Transformation (FCT) model.

Consider some example phrases and associated features. The phrase "the museum" should have an embedding nearly identical to "museum" since "the" has minimal impact the phrase's meaning. This can be captured through part-of-speech (POS) tags, where a tag of DT on "the" will lead to $\lambda_i \approx \vec{0}$, removing its impact on the phrase embedding. In some cases, words will have specific behaviors. In the phrase "historic museum", the word "historic" should impact the phrase embedding to be closer to "landmark". To capture this behavior we add smoothed lexical features, where smoothing reduces data sparsity effects. These features can be based on word clusters, themselves induced from pre-trained word embeddings.

Our feature templates are shown in Table 1. Phrase boundaries, tags and heads are identified using existing parsers or from Annotated Gigaword (Napoles et al., 2012) as described in Section 5. In Eq. (1), we do not limit phrase structure though the features in Table 1 tend to assume a flat structure. However, with additional features the model could handle longer phrases with hierarchical structures, and adding these features does not change our model or training objectives. Following the semantic tasks used for evaluation we experimented with base NPs (including both bigram NPs and longer ones). We leave explorations of features for complex structures to future work.

FCT has two sets of parameters: one is the feature weights $(\alpha, \mathbf{b})$, the other is word embeddings $(\mathbf{e_w})$. We could directly use the word embeddings learned by neural language models. However, our experiments show that those word embeddings are often not suited for FCT. Therefore we propose to

Simple Features		Compound Features
POS tags	$t(w_{i-1}), t(w_i), t(w_{i+1})$	$< t(w_k), t(w_{k+1}) > k \in \{i-1, i\}$
Word clusters	$c(w_{i-1}), c(w_i), c(w_{i+1})$	$< c(w_k), c(w_{k+1}) > k \in \{i-1, i\}$
	w_{i-1}, w_i, w_{i+1} if w_i is function word	
Head word	$I[i = h]$	$< t(w_k), I[i = h] > k \in \{i-1, i, i+1\}$
		$< c(w_k), I[i = h] > k \in \{i-1, i, i+1\}$
Distance from head	$\text{Dis}(h - i)$	$< t(w_k)$,$\text{Dis}(h - i) > k \in \{i-1, i, i+1\}$
		$< c(w_k)$,$\text{Dis}(h - i) > k \in \{i-1, i, i+1\}$
Head tag/cluster	$t(w_h), c(w_h)$ if $i \neq h$	$< t(w_h), t(w_i) >, < c(w_h), c(w_i) >$ if $i \neq h$

Table 1: Feature templates for word w_i in phrase p. $t(w)$: POS tag; $c(w)$: word cluster (when w is a function word, i.e. a preposition word or conjunction word, there is no need to have smoothed version of the word features based on clusters. Therefore we directly use the word forms as features as shown in line 3 of the table); h: position of head word of the phrase p; $\text{Dis}(i - j)$: distance between w_i and w_j (distance in tokens). $< f_1, f_2 >$ refers to the conjunction (i.e. Cartesian product) between two feature templates f_1 and f_2.

learn both the feature weights and the word embeddings with objectives in Section 3. Moreover, experiments show that starting with the baseline word embeddings leads to better learning results comparing to random initializations. Therefore in the rest of the paper, if not specifically mentioned, we always initialize the embeddings of FCT with baseline word embeddings learned by Mikolov et al. (2013b).

3 Training Objectives

The speed and flexibility of FCT enables a range of training settings. We consider standard unsupervised training (language modeling), task-specific training and joint objectives.

3.1 Language Modeling

For unsupervised training on large scale raw texts (language modeling) we train FCT so that phrase embeddings – as composed in Section 2 – predict contextual words, an extension of the skip-gram objective (Mikolov et al., 2013b) to phrases. For each phrase $p_i = (w_{i_1}, ..., w_{i_n}) \in \mathcal{P}, w_{i_j} \in V$, where $\mathcal{P}$ is the set of all phrases and V is the word vocabulary. Here i is the index of a phrase in set $\mathcal{P}$ and i_j is the absolute index of the jth component word of p_i in the sentence. For predicting the c words to the left and right the skip-gram objective becomes:

$$\max_{\alpha, \mathbf{b}, \mathbf{e_w}, \mathbf{e'_w}} \frac{1}{|\mathcal{P}|} \sum_{i=1}^{|\mathcal{P}|} \left(\sum_{0<j\leq c} \log P\left(e'_{w_{i_1-j}} | e_{p_i}\right) + \sum_{0<j\leq c} \log P\left(e'_{w_{i_n+j}} | e_{p_i}\right) \right),$$

$$\text{where } P(e'_w | e_{p_i}) = \frac{\exp\left({e'_w}^T e_{p_i}\right)}{\sum_{w' \in V} \exp\left({e'_{w'}}^T e_{p_i}\right)}, \quad (3)$$

where $\alpha, \mathbf{b}, \mathbf{e_w}$ are parameters (the word embeddings $\mathbf{e_w}$ become parameters when fine-tuning is enabled) of FCT model defined in Section 2. As is common practice, when predicting the context words we use a second set of embeddings $\mathbf{e'_w}$ called *output embeddings* (Mikolov et al., 2013b). During training FCT parameters ($\alpha, \mathbf{b}$) and word embeddings ($\mathbf{e_w}$ and $\mathbf{e'_w}$) are updated via back-propagation. e_{p_i} is the phrase embedding defined in Eq. (1). w_{i_1-j} is the j-th word before phrase p_i and w_{i_n+j} is the j-th word after p_i. We can use negative sampling based Noise Contrastive Estimation (NCE) or hierarchical softmax training (HS) in (Mikolov et al., 2013b) to deal with the large output space. We refer to this objective as the **language modeling** (LM) objective.

3.2 Task-specific Training

When we have a task for which we want to learn embeddings, we can utilize task-specific training of the model parameters. Consider the case where we wish to use phrase embeddings produced by FCT in a classification task, where the goal is to determine whether a phrase p_s is semantically similar to a candidate phrase (or word) p_i. For a phrase p_s and a set of candidate phrases $\{p_i, y_i\}_1^N$, $y_i = 1$ indicates semantic similarity of p_s and p_i and $y_i = 0$ otherwise,

we use a classification objective:

$$\max_{\alpha,\mathbf{b},\mathbf{e_w}} \sum_{p_s} \sum_{i=1}^{N} y_i \log P(y_i = 1|p_s, p_i)$$
$$= \max_{\alpha,\mathbf{b},\mathbf{e_w}} \sum_{p_s} \sum_{i=1}^{N} y_i \log \frac{\exp\left({e_{p_s}}^T e_{p_i}\right)}{\sum_j^N \exp\left({e_{p_s}}^T e_{p_j}\right)}. \quad (4)$$

where e_p is the phrase embedding from Eq. (1). When a candidate phrase p_i is a single word, a lexical embedding can be used directly to derive e_{p_i}. When $N = 1$ for each p_s, i.e., we are working on binary classification problems, the objective will reduce to logistic loss and a bias b will be added. For very large sets, e.g., the whole vocabulary, we use NCE to approximate the objective. We call Eq. (4) the **task-specific** (TASK-SPEC) objective.

In addition to updating only the FCT parameters, we can update the embeddings themselves to improve the task-specific objective. We use the fine-tuning strategy (Collobert and Weston, 2008; Socher et al., 2013a) for learning task-specific word embeddings, first training FCT and the embeddings with the LM objective and then fine-tuning the word embeddings using labeled data for the target task. We refer to this process as "**fine-tuning word emb**" in the experiment session. Note that fine tuning can be also applied to baseline word embeddings trained with the TASK-SPEC objective or the LM objective above.

3.3 Joint Training

While labeled data is the most helpful for training FCT for a task, relying on labeled data alone will yield limited improvements: labeled data has low coverage of the vocabulary, which can lead to over-fitting when we update FCT model parameters Eq. (4) and fine-tune word embeddings. In particular, the effects of fine-tuning word embeddings are usually limited in NLP applications. In contrast to other applications, like vision, where a single input can cover most or all of the model parameters, word embeddings are unique to each word, so a word will have its embedding updated only when the word appears in a training instance. As a result, only words that appear in the labeled data will benefit from fine-tuning and, by changing only part of the embedding space, the performance may be worse overall.

Language modeling provides a method to update all embeddings based on a large unlabeled corpus. Therefore, we combine the language modeling object (Eq. (3)) and the task-specific object (Eq. (4)) to yield a joint objective. When a word's embedding is changed in a task-specific way, it will impact the rest of the embedding space through the LM objective. Thus, all words can benefit from the task-specific training.

We call this the **joint** objective and call the resulted model FCT-Joint (FCT-J for short), since it updates the embeddings with both the LM and TASK-SPEC objectives.

In addition to jointly training both objectives, we can create a pipeline. First, we train FCT with the LM objective. We then fine-tune all the parameters with the TASK-SPEC objective. We call this FCT-Pipeline (FCT-P for short).

3.4 Applications to Other Phrase Composition Models

While our focus is the training of FCT, we note that the above training objectives can be applied to other composition models as well. As an example, consider a recursive neural network (RNN) (Socher et al., 2011; Socher et al., 2013a), which recursively computes phrase embeddings based on the binary sub-tree associated with the phrase with matrix transformations. For the bigram phrases considered in the evaluation tasks, suppose we are given phrase $p = (w_1, w_2)$. The model then computes the phrase embedding e_p as:

$$e_p = \sigma\left(W \cdot [e_{w_1} : e_{w_2}]\right), \quad (5)$$

where $[e_{w_1} : e_{w_2}]$ is the concatenation of two embedding vectors. W is a matrix of parameters to be learned, which can be further refined according to the labels of the children. Back-propagation can be used to update the parameter matrix W and the word embeddings during training. It is possible to train the RNN parameters W with our TASK-SPEC or LM objective: given syntactic trees, we can use RNN (instead of FCT) to compute phrase embeddings e_p, which can be used to compute the objective, and then have W updated via back-propagation. The experiments below show results for this method, which we call RNN, with TASK-SPEC training. However,

while we can train RNNs using small amounts of labeled data, it is impractical to scale it to large corpora (i.e. LM training). In contrast, FCT easily scales to large corpora.

Remark (comparison between FCT and RNN): Besides efficiency, our FCT is also expressive. A common approach to composition, a weighted sum of the embeddings (which we include in our experiments as `Weighted SUM`), is a special case of FCT with no non-lexical features, and a special case of RNN if we restrict the W matrix of RNN to be diagonal. Therefore, RNN and FCT can be viewed as two different ways of improving the expressive strength of `Weighted SUM`. The RNNs increase expressiveness by making the transformation a full matrix (more complex but less efficient), which does not introduce any interaction between one word and its contexts.[1] On the other hand, FCT can make the transformation for one word depend on its context words by extracting relevant features, while keeping the model linear.

As supported by the experimental results, our method for increasing expressiveness is more effective, because the contextual information is critical for phrase compositions. By comparison, the matrix transformations in RNNs may be unnecessarily complicated and are not significantly more helpful in modeling the target tasks and make the models more likely to over-fit.

4 Parameter Estimation

Training of FCT can be easily accomplished by stochastic gradient descent (SGD). While SGD is fast, training with the LM or joint objectives requires the learning algorithm to scale to large corpora, which can be slow even for SGD.

Asynchronous SGD for FCT: We use the distributed asynchronous SGD-based algorithm from Mikolov et al. (2013b). The shared embeddings are updated by each thread based on training data within the thread independently. With word embeddings, the collision rate is low since it is unlikely that different threads will update the same word at the same time. However, adding training of FCT to this setup introduces a problem; the shared feature weights α in the phrase composition models have a much higher collision rate. To prevent conflicts, we modify asynchronous SGD so that only a single thread updates both α and lexical embeddings simultaneously, while the remaining threads only update the lexical embeddings. When training with the LM objective, only a single (arbitrarily chosen) thread can update FCT feature weights; all other threads treat them as fixed during back-propagation. While this reduces the data available for training FCT parameters to only that of a single thread, the small number of parameters α means that even a single thread's data is sufficient for learning them.

We take a similar approach for updating the task-specific (TASK-SPEC) part of the joint objective during FCT-Joint training. We choose a single thread to optimize the TASK-SPEC objective while all other threads optimize the LM objective. This means that αs are updated using the task-specific thread. Restricting updates for both sets of parameters to a single thread does not slow training since gradient computation is very fast for the embeddings and αs.

For joint training, we can tradeoff between the two objectives (TASK-SPEC and LM) by setting a weight for each objective (e.g. c_1 and c_2.) However, under the multi-threaded setting we cannot do this explicitly since the number of threads assigned to each part of the objective influences how the terms are weighted. Suppose that we assign n_1 threads to TASK-SPEC and n_2 to LM. Since each thread takes a similar amount of time, the actual weights will be roughly $c_1 = {c_1}' * n_1$ and $c_2 = {c_2}' * n_2$. Therefore, we first fix the numbers of threads and then tune ${c_1}'$ and ${c_2}'$. In all of our experiments that use distributed training, we use 12 threads.

Training Details: Unless otherwise indicated we use 200-dimensional embeddings, which achieved a good balance between accuracy and efficiency. We use L2 regularization on the weights $\boldsymbol{\alpha}$ in FCT as well as for the matrices W of RNN baselines in Section 6. In all experiments, the learning rates, numbers of iterations and the weights of L2 regularizers are tuned on development data.

We experiment with both negative sampling based NCE training (Mikolov et al., 2013b) for training

[1] As will be discussed in the related work session, there do exist some more expressive extensions of RNN, which can exploit the interaction between a word and its contexts.

PPDB XXL	Total Pairs	Training Pairs	Vocab Size	NYT	Phrases	Words	Vocab Size
Train	120,552	24,300	5,954	Train	84,149,192	518,103,942	518,235
Dev	644	500	-	Dev	30,000	-	-
Test	645	500	-	Test	30,000	-	-

Table 2: Statistics of NYT and PPDB data. "Training pairs" are pairs of bigram phrase and word used in experiments.

word2vec embeddings, the LM objective, and the TASK-SPEC objective; as well as use hierarchical softmax training (HS) for language modeling experiments. We use a window size c=5, the default of word2vec. We remove types that occur less than 5 times (default setting of word2vec). The vocabulary is the same for all evaluations. For NCE training we sample 15 words as negative samples for each training instance according to their frequencies in raw texts. Following Mikolov et al. (2013b) if w has frequency $u(w)$ we set the sampling probability of w to $p(w) \propto u(w)^{3/4}$. For HS training we build a Hoffman tree based on word frequency.

Pre-trained Word Embeddings For methods that require pre-trained lexical embeddings (FCT with pre-training, SUM (Section 5), and the FCT and RNN models in Section 6) we always use embeddings[2] trained with the skip-gram model of word2vec. The embeddings are trained with NCE estimation using the same settings described above.

5 Experiments: Language Modeling

We begin with experiments on FCT for language modeling tasks (Section 3.1). The resultant embeddings can then be used for pre-training in task-specific settings (Section 6).

Data We use the 1994-97 subset from the New York Times (NYT) portion of Gigaword v5.0 (Parker et al., 2011). Sentences are tokenized using OpenNLP.[3] We removed words with frequencies less than 5, yielding a vocabulary of 518,235 word forms and 515,301,382 tokens for training word embeddings.

This dataset is used for both training baseline word embeddings and evaluating our models trained with the LM objective. When evaluating the LM task we consider bigram NPs in isolation (see the "Phrases" column in Table 2). For FCT features that require syntactic information, we extract the NYT portion of Annotated Gigaword (Napoles et al., 2012), which uses the Stanford parser's annotations. We use all bigram noun phrases (obtained from the annotated data) as the input phrases for Eq. (3). A subset from January 1998 of NYT data is withheld for evaluation.

Baselines We include two baselines. The first is to use each component word to predict the context of the phrase with the skip gram model (Mikolov et al., 2013a) and then average the scores to get the probability (denoted as word2vec). The second is to use SUM of the skip-gram embeddings to predict the scores. Training the FCT models with pre-trained word embeddings requires running the skip-gram model on NYT data for 2 iterations: one for word2vec training and one for learning FCT. Therefore, we also run the word2vec model for two epochs to provide embeddings for the baselines.

5.1 Results

We evaluate the perplexity of language models that include lexical embeddings and our composed phrase embeddings from FCT using the LM objective. We use the perplexity computation method of Mikolov et al. (2013a) suitable for skip-gram models. The FCT models are trained by the HS strategy, which can output the exact probability efficiently and was shown by Yu and Dredze (2014) to obtain better performance on language modeling. Since in Section 6.1 we use FCT models trained by NCE, we also include the results of models trained by NCE. Note that scores obtained from a model trained with HS or NCE are not comparable. While the model trained by HS is efficient to evaluate perplexities, NCE training requires summation over all words in the vocabulary in the denominator of the softmax to compute perplexity, an impracticality for large vocabulary. Therefore, we report NCE loss with a fixed set of samples for NCE trained models.

[2]We use "input embeddings" learned by word2vec.

[3]https://opennlp.apache.org/

Model	Perplexity (HS training)			NCE loss (NCE training)		
	Subset Train	**Dev**	**Test**	**Subset Train**	**Dev**	**Test**
SUM (2 epochs)	7.620	7.577	7.500	2.312	2.226	2.061
word2vec (2 epochs)	7.103	7.074	7.052	2.274	2.195	2.025
FCT (random init, 2 epochs)	6.753	6.628	6.713	1.879	1.722	1.659
FCT (with pre-training, 1 epochs)	6.641	6.540	6.552	1.816	1.691	1.620

Table 3: Language model perplexity and NCE loss on a subset of train, dev, and test NYT data.

Model	$\|\lambda_1\| \gg \|\lambda_2\|$		$\|\lambda_1\| \approx \|\lambda_2\|$		$\|\lambda_1\| \ll \|\lambda_2\|$	
	biological diversity	**north-eastern part**	**dead body**	**medicinal products**	**new trial**	**an extension**
FCT	sensitivity natural abilities species	northeastern sprawling preserve area	remains grave skeleton	drugs uses chemicals	proceeding cross-examination defendant	signed terminated temporary full
SUM	destruction racial genetic cultural	portion result integral chunk	unconscious dying flesh	marijuana packaging substances	new judge courtroom	an renewal another signing

Table 4: Differences in the nearest neighbors from the two phrase embedding models.

Table 3 shows results for the NYT training data (subset of the full training data containing 30,000 phrases with their contexts from July 1994), development and test data. Language models with FCT performed much better than the SUM and word2vec baselines, under both NCE and HS training. Note that FCT with pre-training makes a single pass over the whole NYT corpus and then a pass over only the bigram NPs, and the random initialization model makes a pass over the bigrams twice. This is less data compared to two passes over the full data (baselines), which indicates that FCT better captures the context distributions of phrases.

Qualitative Analysis Table 4 shows words and their most similar phrases (nearest neighbors) computed by FCT and SUM. We show three types of phrases: one where the two words in a phrase contribute equally to the phrase embedding, where the first word dominates the second in the phrase embedding, and vice versa. We measure the effect of each word by computing the total magnitude of the $\boldsymbol{\lambda}$ vector for each word in the phrase. For example, for the phrase "an extension", the embedding for the second word dominates the resulting phrase embedding ($\|\boldsymbol{\lambda}_1\| \ll \|\boldsymbol{\lambda}_2\|$) as learned by FCT. The table highlights the differences between the methods by showing the most relevant phrases *not* selected as most relevant by the other method. It is clear that words selected using FCT are more semantically related than those of the baseline.

6 Experiments: Task-specific Training: Phrase Similarity

Data We consider several phrase similarity datasets for evaluating task-specific training. Table 5 summarizes these datasets and shows examples of inputs and outputs for each task.

PPDB The Paraphrase Database (PPDB)[4] (Ganitkevitch et al., 2013) contains tens of millions of automatically extracted paraphrase pairs, including words and phrases. We extract all paraphrases containing a bigram noun phrase and a noun word from PPDB. Since articles usually have little contributions to the phrase meaning, we removed the easy cases of all pairs in which the phrase is composed of an article and a noun.Next, we removed duplicate pairs: if <A,B> occurred in PPDB, we removed relations of <B,A>. PPDB is organized into 6 parts, ranging from S (small) to XXXL. Division into these sets is based on an automatically derived accuracy metric. We extracted paraphrases from the XXL set. The most accurate (i.e. first) 1,000 pairs are used for evaluation and divided into a dev set (500 pairs) and test set (500 pairs); the remaining pairs were used for training. Our PPDB task is an extension of measuring PPDB semantic similarity between words (Yu

[4] http://www.cis.upenn.edu/~ccb/ppdb/

Data Set	Input	Output
(1) PPDB	medicinal products	drugs
(2) SemEval2013	<small spot, flect>	True
	<male kangaroo, point>	False
(3) Turney2012	monosyllabic word	**monosyllable**, hyalinization, fund, gittern, killer
(4) PPDB (ngram)	contribution of the european union	eu contribution

Table 5: Examples of phrase similarity tasks. (1) PPDB is a ranking task, in which an input bigram and a output noun are given, and the goal is to rank the output word over other words in the vocabulary. (2) SemEval2013 is a binary classification task: determine whether an input pair of a bigram and a word form a paraphrase (True) or not (False). (3) Turney2012 is a multi-class classification task: determine the word most similar to the input phrase (in bold) from the five output candidates. For the 10-choice task, the goal is to select the most similar pair between the combination of one bigram phrase, i.e., the input phrase or the swapped input ("*word monosyllabic*" for this example), and the five output candidates. The correct answer in this case should still be the pair of original input phrase and the original correct output candidate (in bold). (4) PPDB (ngram) is similar to PPDB, but in which both inputs and outputs becomes noun phrases with arbitrary lengths.

and Dredze, 2014) to that between phrases. Data details appear in Table 2.

Phrase Similarity Datasets We use a variety of human annotated datasets to evaluate phrase semantic similarity: the SemEval2013 shared task (Korkontzelos et al., 2013), and the noun-modifier problem (Turney2012) in Turney (2012). Both tasks provide evaluation data and training data. **SemEval2013 Task 5(a)** is a classification task to determine if a word phrase pair are semantically similar. **Turney2012** is a task to select the closest matching candidate word for a given phrase from candidate words. The original task contained seven candidates, two of which are component words of the input phrase (seven-choice task). Followup work has since removed the components words from the candidates (five-choice task). Turney (2012) also propose a 10-choice task based on this same dataset. In this task, the input bigram noun phrase will have its component words swapped. Then all the pairs of swapped phrase and a candidate word will be treated as a negative example. Therefore, each input phrase will correspond to 10 test examples where only one of them is the positive one.

Longer Phrases: PPDB (ngram-to-ngram) To show the generality of our approach we evaluate our method on phrases longer than bigrams. We extract arbitrary length noun phrase pairs from PPDB. We only include phrase pairs that differ by more than one word; otherwise the task would reduce to evaluating unigram similarity. Similar to the bigram-to-unigram task, we used the XXL set and removed duplicate pairs. We used the most accurate pairs for development (2,821 pairs) and test (2,920 pairs); the remaining 148,838 pairs were used for training.

As before, we rely on negative sampling to efficiently compute the objective during training. For each source/target n-gram pair, we sample negative noun phrases as outputs. Both the target phrase and the negative phrases are transformed to their phrase embeddings with the current parameters. We then compute inner products between embedding of the source phrase and these output embeddings, and update the parameters according to the NCE objective. We use the same feature templates as in Table 1.

Notice that the XXL set contains several subsets (e.g., M, L ,XL) ranked by accuracy. In the experiments we also investigate their performance on dev data. Unless otherwise specified, the full set is selected (performs best on dev set) for training.

Baselines We compare to the common and effective point-wise addition (SUM) method (Mitchell and Lapata, 2010).[5] We additionally include Weighted SUM, which learns overall dimension specific weights from task-specific training, the equivalent of FCT with α_{jk}=0 and b_{ij} learned from data. Furthermore, we compare to dataset specific

[5]Mitchell and Lapata (2010) also show success with point-wise product (MULTI) for VSMs. However, MULTI is ill-suited to word embeddings and gave poor results in all our experiments. Mikolov et al. (2013b) show that sum of embeddings is related to product of context distributions because of the logarithmic computation in the output layer.

baselines: we re-implemented the recursive neural network model (RNN) (Socher et al., 2013a) and the Dual VSM algorithm in Turney (2012)[6] so that they can be trained on our dataset. We also include results for fine-tuning word embeddings in SUM and Weighted SUM with TASK-SPEC objectives, which demonstrate improvements over the corresponding methods without fine-tuning. As before, word embeddings are pre-trained with word2vec.

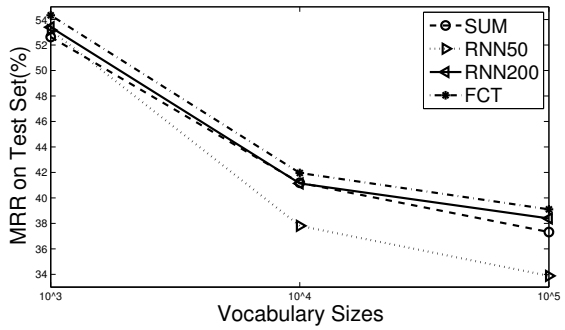
RNNs serve as another way to model the compositionally of bigrams. We run an RNN on bigrams and associated sub-trees, the same setting FCT uses, and are trained on our TASK-SPEC objectives with the technique described in Section 3.4. As in Socher et al. (2013a), we refine the matrix W in Eq. (5) according to the POS tags of the component words.[7] For example, for a bigram NP like new/ADJ trial/NN, we use a matrix W_{ADJ-NN} to transform the two word embeddings to the phrase embedding. In the experiments we have 60 different matrices in total for bigram NPs. The number is larger than that in Socher et al. (2013a) due to incorrect tags in automatic parses.

Since the RNN model has time complexity $O(n^2)$, we compare RNNs with different sized embeddings. The first one uses embeddings with 50 dimensions, which has the same size as the embeddings used in Socher et al. (2013a), and has similar complexity to our model with 200 dimension embeddings. The second model uses the same 200 dimension embeddings as our model but is significantly more computationally expensive.

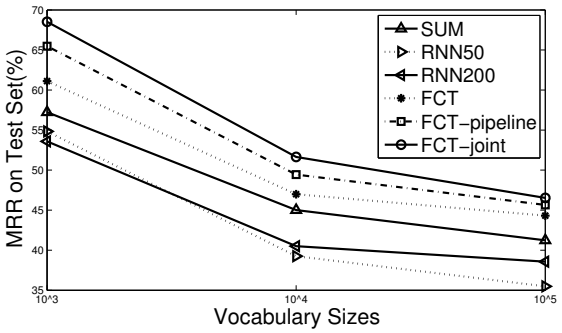
For all models, we normalize the embeddings so that the L-2 norm equals 1, which is important in measuring semantic similarity via inner product.

6.1 Results: Bigram Phrases

PPDB Our first task is to measure phrase similarity on PPDB. Training uses the TASK-SPEC objective (Eq. (4) with NCE training) where data are phrase-word pairs $< A,B >$. The goal is to select B from a set of candidates given A, where pair similarity is measured using inner product. We use candidate sets of size 1k/10k/100k from the most frequent N words in NYT and report mean reciprocal rank (MRR).



(a) MRR of models with fixed word embeddings



(b) MRR of models with fine-tuning

Figure 1: Performance on PPDB task (test set).

We report results with the baseline methods (SUM, Weighted SUM, RNN). For FCT we report training with the TASK-SPEC objective, the joint-objective (FCT-J) and the pipeline approach (FCT-P). To ensure that the TASK-SPEC objective has a stronger influence in FCT-Joint, we weighted each training instance of LM by 0.01, which is equivalent to setting the learning rate of the LM objective equal to $\eta/100$ and that of the TASK-SPEC objective as η. Training makes the same number of passes with the same learning rate as training with the TASK-SPEC objective only. For each method we report results with and without fine-tuning the word embeddings on the labeled data. We run FCT on the PPDB training data for 5 epochs with learning rate $\eta = 0.05$, which are both selected from development set.

Fig. 1 shows the overall MRR results on differ-

[6] We did not include results for a holistic model as in Turney (2012), since most of the phrases (especially for those in PPDB) in our experiments are common phrases, making the vocabulary too large to train. One solution would be to only train holistic embeddings for phrases in the test data, but examination of a test set before training is not a realistic assumption.

[7] We do not compare the performance between using a single matrix and several matrices since, as discussed in Socher et al. (2013a), Ws refined with POS tags work much better than using a single W. That also supports the argument in this paper, that it is important to determine the transformation with more features.

Model	Objective	Fine-tuning Word Emb	MRR @ 10k
SUM	-	-	41.19
SUM	TASK-SPEC	Y	45.01
WSum	TASK-SPEC	Y	45.43
RNN 50	TASK-SPEC	N	37.81
RNN 50	TASK-SPEC	Y	39.25
RNN 200	TASK-SPEC	N	41.13
RNN 200	TASK-SPEC	Y	40.50
FCT	TASK-SPEC	N	41.96
FCT	TASK-SPEC	Y	46.99
FCT	LM	Y	42.63
FCT-P	TASK-SPEC+LM	Y	49.44
FCT-J	TASK-SPEC+LM	joint	**51.65**

Table 6: Performance on the PPDB task (test data).

ent candidate vocabulary sizes (1k, 10k and 100k), and Table 6 highlights the results on the vocabulary using the top 10k words. Overall, FCT with TASK-SPEC training improves over all the baseline methods in each setting. Fine-tuning word embeddings improves all methods except RNN (d=200). We note that the RNN performs poorly, possibly because it uses a complex transformation from word embedding to phrase embeddings, making the learned transformation difficult to generalize well to new phrases and words when the task-specific labeled data is small. As a result, there is no guarantee of comparability between new pairs of phrases and word embeddings. The phrase embeddings may end up in a different part of the subspace from the word embeddings.

Comparing to SUM and Weighted SUM, FCT is capable of using features providing critical contextual information, which is the source of FCT's improvement. Additionally, since the RNNs also used POS tags and parsing information yet achieved lower scores than FCT, our results show that FCT more effectively uses these features. To better show this advantage, we train FCT models with only POS tag features, which achieve 46.37/41.20 on MRR@10k with/without fine-tuning word embeddings, still better than RNNs. See Section 6.3 for a full ablation study of features in Table 1.

Semi-supervised Results: Table 6 also highlighted the improvement from semi-supervised learning. First, the fully unsupervised method (LM) improves over SUM, showing that improvements in language modeling carry over to semantic similarity tasks. This correlation between the LM objective and the target task ensures the success of semi-supervised training. As a result, both semi-supervised methods, FCT-J and FCT-P improves over the supervised methods; and FCT-J achieves the best results of all methods, including FCT-P. This demonstrates the effectiveness of including large amounts of unlabeled data while learning with a TASK-SPEC objective. We believe that by adding the LM objective, we can propagate the semantic information of embeddings to the words that do not appear in the labeled data (see the differences between vocabulary sizes in Table 2).

The improvement of FCT-J over FCT-P also indicates that the joint training strategy can be more effective than the traditional pipeline-based pre-training. As discussed in Section 3.3, the pipeline method, although commonly used in deep learning literatures, does not suit NLP applications well because of the sparsity in word embeddings. Therefore, our results suggest an alternative solution to a wide range of NLP problems where labeled data has low coverage of the vocabulary. For future work, we will further investigate the idea of joint training on more tasks and compare with the pipeline method.

Results on SemEval2013 and Turney2012 We evaluate the same methods on SemEval2013 and the Turney2012 5- and 10-choice tasks, which both provide training and test splits. The same baselines in the PPDB experiments, as well as the Dual Space method of Turney (2012) and the recursive auto-encoder (RAE) from Socher et al. (2011) are used for comparison. Since the tasks did not provide any development data, we used cross-validation (5 folds) for tuning the parameters, and finally set the training epochs to be 20 and $\eta = 0.01$. For joint training, the weight of the LM objective is weighted by 0.005 (i.e. with a learning rate equal to 0.005η) since the training sets for these two tasks are much smaller. For convenience, we also include results for Dual Space as reported in Turney (2012), though they are not comparable here since Turney (2012) used a much larger training set.

Table 7 shows similar trends as PPDB. One difference here is that RNNs do better with 200 dimen-

Model	Objective	Fine-tuning Word Emb	SemEval2013 Test	Turney2012		
				Acc (5)	Acc (10)	MRR @ 10k
SUM	-	-	65.46	39.58	19.79	12.00
SUM	TASK-SPEC	Y	67.93	48.15	24.07	14.32
Weighted Sum	TASK-SPEC	Y	69.51	52.55	26.16	**14.74**
RNN (d=50)	TASK-SPEC	N	67.20	39.64	25.35	1.39
RNN (d=50)	TASK-SPEC	Y	70.36	41.96	27.20	1.46
RNN (d=200)	TASK-SPEC	N	71.50	40.95	27.20	3.89
RNN (d=200)	TASK-SPEC	Y	**72.22**	42.84	29.98	4.03
Dual Space[1]	-	-	52.47	27.55	16.36	2.22
Dual Space[2]	-	-	-	58.3	41.5	-
RAE	auto-encoder	-	51.75	22.99	14.81	0.16
FCT	TASK-SPEC	N	68.84	41.90	33.80	8.50
FCT	TASK-SPEC	Y	70.36	52.31	38.66	13.19
FCT	LM	-	67.22	42.59	27.55	14.07
FCT-P	TASK-SPEC+LM	Y	70.64	53.09	**39.12**	14.17
FCT-J	TASK-SPEC+LM	joint	70.65	**53.31**	**39.12**	14.25

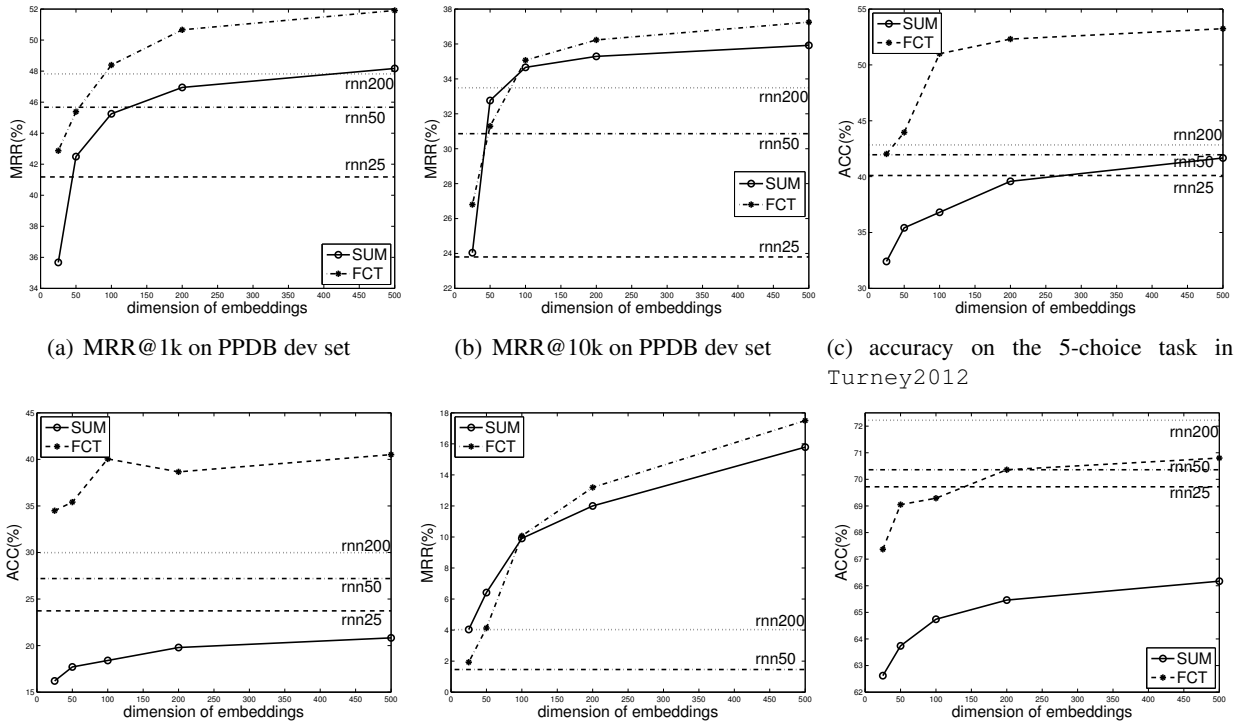
Table 7: Performance on SemEval2013 and Turney2012 semantic similarity tasks. Dual Space[1]: Our reimplementation of the method in (Turney, 2012). Dual Space[2]: The result reported in Turney (2012). RAE is the recursive auto-encoder in (Socher et al., 2011), which is trained with the reconstruction-based objective of auto-encoder.

sional embeddings on SemEval2013, though at a dimensionality with similar computational complexity to FCT ($d = 50$), FCT improves. Additionally, on the 10-choice task of Turney2012, both the FCT and the RNN models, either with or without fine-tuning word embeddings, significantly outperform SUM, showing that both models capture the word order information. Fine tuning gives smaller gains on RNNs likely because the limited number of training examples is insufficient for the complex RNN model. The LM objective leads to improvements on all three tasks, while RAE does not perform significantly better than random guessing. These results are perhaps attributable to the lack of assumptions in the objective about the relations between word embeddings and phrase embeddings, making the learned phrase embeddings not comparable to word embeddings.

6.2 Dimensionality and Complexity

A benefit of FCT is that it is computationally efficient, allowing it to easily scale to embeddings of 200 dimensions. By contrast, RNN models typically use smaller sized embeddings ($d = 25$ proved best in Socher et al., 2013a) and cannot scale up to large datasets when larger dimensionality embeddings are used. For example, when training on the PPDB data, the FCT with $d = 200$ processes 2.33 instances per ms, while the RNN with the same dimensionality processes 0.31 instance/ms. Training an RNN with $d = 50$ is of comparable speed to FCT with $d = 200$. Figure 2 (a-b) shows the MRR on PPDB for 1k and 10k candidate sets for both the SUM baseline and FCT with a TASK-SPEC objective and full features, as compared to RNNs with different sized embeddings. Both FCT and RNN use fine-tuned embeddings. With a small number of embedding dimensions, RNNs achieve better results. However, FCT can scale to much higher dimensionality embeddings, which easily surpasses the results of RNNs. This is especially important when learning a large number of embeddings: the 25-dimensional space may not be sufficient to capture the semantic diversity, as evidenced by the poor performance of RNNs with lower dimensionality.

Similar trends observed on the PPDB data also appear on the tasks of Turney2012 and SemEval2013. Figure 2 (c-f) shows the performances on these two tasks. On the Turney2012 task, the FCT even outperforms the RNN model using embeddings with the same dimensionality. One possible reason is due to overfitting of the more complex RNN models on these small training sets. Figure 2(d) shows that the performances of FCT on the 10-choice task are less affected by the dimensions of embeddings. That is because the composition models can well handle the word order information,



(a) MRR@1k on PPDB dev set

(b) MRR@10k on PPDB dev set

(c) accuracy on the 5-choice task in `Turney2012`

(d) accuracy on the 10-choice task in `Turney2012`

(e) MRR@10k on `Turney2012`

(f) accuracy on the `SemEval2013`

Figure 2: Effects of embedding dimension on the semantic similarity tasks. The notations "`RNN`$< d >$" in the figures stand for the `RNN` models trained with d-dimensional embeddings.

which is critical to solving the 10-choice task, without relying on too much semantic information from word embeddings themselves. Figure 2(e) shows that when the dimensionality of embeddings is lower than 100, both `FCT` and `RNN` do worse than the baseline. This is likely because in the case of low dimensionality, updating embeddings is likely to change the whole structure of embeddings of training words, making both the fine-tuned word embeddings and the learned phrase embeddings incomparable to the other words. The performance of `RNN` with 25-dimension embeddings is too low so it is omitted.

6.3 Experiments on Longer Phrases

So far our experiments have focused on bigram phrases. We now show that `FCT` improves for longer n-gram phrases (Table 8). Without fine-tuning, `FCT` performs significantly better than the other models, showing that the model can better capture the context and annotation information related to phrase semantics with the help of rich features. With different amounts of training data, we found that `WSum` and `FCT` both perform better when trained on the PPDB-L set, a more accurate subset of XXL with 24,279 phrase pairs. This can be viewed as a low resource setting, where there is limited data for fine-tuning word embeddings.

Model	Train Set	Fine-tuning Word Emb	MRR @10k	MRR @ 100k
`SUM`	-	N	46.53	16.62
`WSum`	L	N	51.10	18.92
`FCT`	L	N	**68.91**	**29.04**
`SUM`	XXL	Y	74.30	29.14
`WSum`	XXL	Y	75.37	31.13
`FCT`	XXL	Y	**79.68**	**36.00**

Table 8: Results on `PPDB ngram-to-ngram` task.

With fine-tuning of word embeddings, `FCT` still significantly beats the other models. All three methods get their best results on the full XXL set, likely because it contains more phrase pairs to alleviate over fitting caused by fine-tuning word embeddings. Notice that fine-tuning greatly helps all the methods, including `SUM`, indicating that this `ngram-to-ngram` task is still largely dominated

Feature Set	MRR @ 10k
FCT	**79.68**
-clus	76.82
-POS	77.67
-Compound	79.40
-Head	77.50
-Distance	78.86
WSum	75.37
SUM	74.30

Table 9: Ablation study on dev set of the PPDB ngram-to-ngram task (MRR @ 10k).

by the quality of single word semantics. Therefore, we expect larger gains from FCT on tasks where single word embeddings are less important, such as relation extraction (long distance dependencies) and question understanding (intentions are largely dependent on interrogatives).

Finally, we demonstrate the efficacy of different features in FCT (Table 1) with an ablation study (Table 9). Word cluster features contribute most, because the point-wise product between word embedding and its context word cluster representation is actually an approximation of the word-word interaction, which is believed important for phrase compositions. Head features, though few, also make a big difference, reflecting the importance of syntactic information. Compound features do not have much of an impact, possibly because the simpler features capture enough information.

7 Related Work

Compositional semantic models aim to build distributional representations of a phrase from its component word representations. A traditional approach for composition is to form a point-wise combination of single word representations with compositional operators either pre-defined (e.g. element-wise sum/multiplication) or learned from data (Le and Mikolov, 2014). However, these approaches ignore the inner structure of phrases, e.g. the order of words in a phrase and its syntactic tree, and the point-wise operations are usually less expressive. One solution is to apply a matrix transformation (possibly followed by a non-linear transformation) to the concatenation of component word representations (Zanzotto et al., 2010). For longer phrases, matrix multiplication can be applied recursively according to the associated syntactic trees (Socher et al., 2010). However, because the input of the model is the concatenation of word representations, matrix transformations cannot capture interactions between a word and its contexts, or between component words.

There are three ways to restore these interactions: The first is to use word-specific/tensor transformations to force the interactions between component words in a phrase. In these methods, word-specific transformations, which are usually matrices, are learned for a subset of words according to their syntactic properties (e.g. POS tags) (Baroni and Zamparelli, 2010; Socher et al., 2012; Grefenstette et al., 2013; Erk, 2013). Composition between a word in this subset and another word becomes the multiplication between the matrix associated with one word and the embedding of the other, producing a new embedding for the phrase. Using one tensor (not word-specific) to compose two embedding vectors (has not been tested on phrase similarity tasks) (Bordes et al., 2014; Socher et al., 2013b) is a special case of this approach, where a "word-specific transformation matrix" is derived by multiplying the tensor and the word embedding. Additionally, word-specific matrices can only capture the interaction between a word and one of its context words; others have considered extensions to multiple words (Grefenstette et al., 2013; Dinu and Baroni, 2014). The primary drawback of these approaches is the high computational complexity, limiting their usefulness for semantics (Section 6.2.)

A second approach draws on the concept of contextualization (Erk and Padó, 2008; Dinu and Lapata, 2010; Thater et al., 2011), which sums embeddings of multiple words in a linear combination. For example, Cheung and Penn (2013) apply contextualization to word compositions in a generative event extraction model. However, this is an indirect way to capture interactions (the transformations are still unaware of interactions between components), and thus has not been a popular choice for composition.

The third approach is to refine word-independent compositional transformations with annotation features. FCT falls under this approach. The primary advantage is that composition can rely on richer linguistic features from the context. While the em-

beddings of component words still cannot interact, they can interact with other information (i.e. features) of their context words, and even the global features. Recent research has created novel features based on combining word embeddings and contextual information (Nguyen and Grishman, 2014; Roth and Woodsend, 2014; Kiros et al., 2014; Yu et al., 2014; Yu et al., 2015). Yu et al. (2015) further proposed converting the contextual features into a hidden layer called feature embeddings, which is similar to the α matrix in this paper. Examples of applications to phrase semantics include Socher et al. (2013a) and Hermann and Blunsom (2013), who enhanced RNNs by refining the transformation matrices with phrase types and CCG super tags. However, these models are only able to use limited information (usually one property for each compositional transformation), whereas FCT exploits multiple features.

Finally, our work is related to recent work on low-rank tensor approximations. When we use the phrase embedding e_p in Eq. (1) to predict a label y, the score of y given phrase p will be $s(y,p) = U_y^T e_p = \sum_i^N U_y^T(\lambda_i \odot e_{w_i})$ in log-linear models, where U_y is the parameter vector for y. This is equivalent to using a parameter tensor T to evaluate the score with $s'(y,p) = \sum_i^N T \times_1 y \times_2 f(w_i,p) \times e_{w_i}$, while forcing the tensor to have a low-rank form as $T \approx \mathbf{U} \otimes \alpha \otimes \mathbf{e_w}$. Here $\times_k$ indicates tensor multiplication of the kth view, and $\otimes$ indicates matrix outer product (Kolda and Bader, 2009). From this point of view, our work is closely related to the discriminative training methods for low-rank tensors in NLP (Cao and Khudanpur, 2014; Lei et al., 2014), while it can handle more complex ngram-to-ngram tasks, where the label y also has its embedding composed from basic word embeddings. Therefore our model can capture the above work as special cases. Moreover, we have a different method of decomposing the inputs, which results in views of lexical parts and non-lexical features. As we show in this paper, this input decomposition allows us to benefit from pre-trained word embeddings and feature weights.

8 Conclusion

We have presented FCT, a new composition model for deriving phrase embeddings from word embeddings. Compared to existing phrase composition models, FCT is very efficient and can utilize high dimensional word embeddings, which are crucial for semantic similarity tasks. We have demonstrated how FCT can be utilized in a language modeling setting, as well as tuned with task-specific data. Fine-tuning embeddings on task-specific data can further improve FCT, but combining both LM and TASK-SPEC objectives yields the best results. We have demonstrated improvements on both language modeling and several semantic similarity tasks. Our implementation and datasets are publicly available.[8]

While our results demonstrate improvements for longer phrases, we still only focus on flat phrase structures. In future work we plan to FCT with the idea of recursively building representations. This would allow the utilization of hierarchical structure while restricting compositions to a small number of components.

Acknowledgments

We thank Matthew R. Gormley for his input and anonymous reviewers for their comments. Mo Yu is supported by the China Scholarship Council and by NSFC 61173073.

References

Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1183–1193.

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

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2014. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259.

Yuan Cao and Sanjeev Khudanpur. 2014. Online learning in tensor space. In *Association for Computational Linguistics (ACL)*, pages 666–675.

Jackie Chi Kit Cheung and Gerald Penn. 2013. Probabilistic domain modelling with contextualized distributional semantic vectors. In *Association for Computational Linguistics (ACL)*, pages 392–401.

[8] https://github.com/Gorov/FCT_PhraseSim_TACL

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning (ICML)*, pages 160–167.

Ronan Collobert. 2011. Deep learning for efficient discriminative parsing. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 224–232.

Georgiana Dinu and Marco Baroni. 2014. How to make words with vectors: Phrase generation in distributional semantics. In *Association for Computational Linguistics (ACL)*, pages 624–633.

Georgiana Dinu and Mirella Lapata. 2010. Measuring distributional similarity in context. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1162–1172.

Katrin Erk and Sebastian Padó. 2008. A structured vector space model for word meaning in context. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 897–906.

Katrin Erk. 2013. Towards a semantics for distributional representations. In *International Conference on Computational Semantics (IWCS 2013)*, pages 95–106.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. Ppdb: The paraphrase database. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 758–764.

Edward Grefenstette, Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh, and Marco Baroni. 2013. Multi-step regression learning for compositional distributional semantics. *arXiv:1301.6939*.

Karl Moritz Hermann and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 894–904.

Karl Moritz Hermann, Dipanjan Das, Jason Weston, and Kuzman Ganchev. 2014. Semantic frame identification with distributed word representations. In *Association for Computational Linguistics (ACL)*, pages 1448–1458.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Association for Computational Linguistics (ACL)*, pages 873–882.

Ryan Kiros, Richard Zemel, and Ruslan R Salakhutdinov. 2014. A multiplicative model for learning distributed text-based attribute representations. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2348–2356.

Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

Ioannis Korkontzelos, Torsten Zesch, Fabio Massimo Zanzotto, and Chris Biemann. 2013. Semeval-2013 task 5: Evaluating phrasal semantics. In *Joint Conference on Lexical and Computational Semantics (* SEM)*, pages 39–47.

Quoc V Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Association for Computational Linguistics (ACL)*, pages 1381–1391.

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 Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Association for Computational Linguistics (ACL)*, pages 236–244.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. 2012. Annotated gigaword. In *ACL Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, pages 95–100.

Thien Huu Nguyen and Ralph Grishman. 2014. Employing word representations and regularization for domain adaptation of relation extraction. In *Association for Computational Linguistics (ACL)*, pages 68–74.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition, june. *Linguistic Data Consortium, LDC2011T07*.

Michael Roth and Kristian Woodsend. 2014. Composition of word representations improves semantic role labelling. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 407–413.

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

Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 151–161.

Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1201–1211.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013a. Parsing with compositional vector grammars. In *Association for Computational Linguistics (ACL)*, pages 455–465.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642.

Stefan Thater, Hagen Fürstenau, and Manfred Pinkal. 2011. Word meaning in context: A simple and effective vector model. In *International Joint Conference on Natural Language Processing (IJCNLP)*, pages 1134–1143.

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

Peter D Turney. 2012. Domain and function: A dual-space model of semantic relations and compositions. *Journal of Artificial Intelligence Research (JAIR)*, 44:533–585.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Association for Computational Linguistics (ACL)*, pages 545–550.

Mo Yu, Matthew Gormley, and Mark Dredze. 2014. Factor-based compositional embedding models. In *NIPS Workshop on Learning Semantics*.

Mo Yu, Matthew R. Gormley, and Mark Dredze. 2015. Combining word embeddings and feature embeddings for fine-grained relation extraction. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Fabio Massimo Zanzotto, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh Manandhar. 2010. Estimating linear models for compositional distributional semantics. In *International Conference on Computational Linguistics (COLING)*, pages 1263–1271.