Unsupervised Learning of Word-Sequence Representations from Scratch via Convolutional Tensor Decomposition

Furong Huang FURONGH@UCI.EDU
Animashree Anandkumar A.ANANDKUMAR@UCI.EDU
Electrical Engineering and Computer Science Dept.
University of California, Irvine
Irvine, USA 92697, USA

Editor:

Abstract

Text embeddings have played a key role in obtaining state-of-the-art results in natural language processing. Word2Vec and its variants have successfully mapped words with similar syntactic or semantic meanings to nearby vectors. However, extracting universal embeddings of longer word-sequences remains a challenging task. We employ the convolutional dictionary model for unsupervised learning of embeddings for variable length word-sequences. We propose a two-phase ConvDic+DeconvDec framework that first learns dictionary elements (i.e., phrase templates), and then employs them for decoding the activations. The estimated activations are then used as embeddings for downstream tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity estimation. We propose a convolutional tensor decomposition algorithm for learning the phrase templates. It is shown to be more accurate, and much more efficient than the popular alternating minimization in dictionary learning literature. Our word-sequence embeddings achieve state-of-the-art performance in sentiment classification, semantic textual similarity estimation, and paraphrase detection over eight datasets from various domains, without requiring pre-training or additional features.

1. Introduction

We have recently witnessed the tremendous success of word embeddings or word vector representations in natural language processing. This involves mapping words to vector representations such that words which share similar semantic or syntactic meanings are close to one another in the vector space Bengio et al. (2006); Collobert and Weston (2008); Collobert et al. (2011); Mikolov et al. (2013); Pennington et al. (2014). Word embeddings have attained state-of-the-art performance in tasks such as part-of-speech (POS) tagging, chunking, named entity recognition (NER), and semantic role labeling. Despite this impressive performance, word embeddings do not suffice for more advanced tasks which require context-aware information or word orders, e.g. paraphrase detection, sentiment analysis, plagiarism detection, information retrieval and machine translation. Therefore, extracting word-sequence vector representations is crucial for expanding the realm of automated text understanding.

Previous works on word-sequence embeddings are based on a variety of mechanisms. A popular method is to learn the composition operators in sequences Mitchell and Lapata

(2010); Yu and Dredze (2015). The complexity of the compositionality varies widely: from simple operations such as addition Mitchell and Lapata (2010); Yu and Dredze (2015) to complicated recursive neural networks Socher et al. (2011, 2013); Belanger and Kakade (2015), convolutional neural networks Kalchbrenner et al. (2014b,b), long short-term memory (LSTM) recurrent neural networks Tai et al. (2015), or combinations of these architectures Wieting et al. (2015). All these methods produce sentence representations that depend on a supervised task, and the class labels are back-propagated to update the composition weights Kalchbrenner et al. (2014a).

Since the above methods rely heavily on the downstream task and the domain of the training samples, they can hardly be used as universal embeddings across domains, and require intensive pre-training and hyper-parameter tuning. The state-of-the-art unsupervised framework is Skip-thought Kiros et al. (2015), based on an objective function that abstracts the skip-gram model to the sentence level, and encodes a sentence to predict the sentences around it. However, the skip-thought model requires a large corpus of contiguous text, such as the book corpus with more than 74 million sentences. Can we instead efficiently learn sentence embeddings using small amounts of samples without supervision/labels or annotated features(such as parse trees)? Also, can the sentence embeddings be context-aware, can handle variable lengths, and is not limited to specific domains?

We propose an unsupervised ConvDic+DeconvDec framework that satisfies all the above constraints. It is composed of two phases, a *comprehension phase* which summarizes template phrases using *convolutional dictionary* elements, followed by a *feature-extraction phase* which extracts activations using *deconvolutional decoding*. We propose a novel learning algorithm for the comprehension phase based on convolutional tensor decomposition, as described in Section 1.1. Note that in the *comprehension phase*, phrase templates are learned over fixed length small patches (patch length is equal to phrase template length), whereas entire word-sequence is decoded to get the final word-sequence embedding in the *feature-extraction phase*.

We employ our sentence embeddings in the tasks of sentiment classification, semantic textual similarity estimation, and paraphrase detection over eight datasets from various domains. These are challenging tasks since they require a contextual understanding of text relationships rather than bags of words. We learn the embeddings from scratch without using any auxiliary information. While previous works use information such as parse trees, Wordnet or pre-train on a much larger corpus, we train from scratch on small amounts of text and obtain competitive results, which are close or even better than the state-of-the-art.

This is due to the combination of efficient modeling and learning approaches in our work. The convolutional model incorporates word orders and phrase representations, and our tensor decomposition algorithm can efficiently learn an set of parameters (phrase templates) for the convolutional model. We describe our framework in detail below.

1.1 Convolutional Dictionary Model and Tensor Decomposition Algorithm

Word embeddings focus on mapping words to fixed length vector representations and ignores the order of the words. To model the word-sequence or the word order, we consider a convolutional dictionary learning model which posits that the observed word-sequence is generated from a superposition of *phrase templates* (a.k.a dictionary elements) $\{f_1^*, \dots f_L^*\}$

activated at various locations in the word-sequence. Therefore, the convolutional model $x = \sum_{i\in[L]} f_i^* * w_i^*$ is natural, where activations are represented by activation maps $\{w_1^*, \dots w_L^*\}$. Now training the convolutional dictionary model is the problem of joint prediction of a good set of *phrase templates* $f_1^*, \dots f_L^*$ and *activation maps* $w_1^*, \dots w_L^*$ given the observed word-sequence x. And the activation maps $w_1^*, \dots w_L^*$ are further used as the word-sequences embedding for word-sequences x as it contains the discriminative features that distinguish different word-sequences.

To be precise, the convolutional dictionary model learning solves the following optimization problem

$$\min_{f_i, w_i : \|f_i\|=1} \|x - \sum_{i\in[L]} f_i * w_i\|^2. \tag{1}$$

A popular heuristic for solving (1) is based on alternate minimization (AM) Bristow and Lucey (2014), where the phrase embeddings f_i are optimized, while keeping the activations w_i fixed, and vice versa. Each alternating update can be solved efficiently since it is linear in each of the variables. However, there are two main drawbacks: computational inefficiency and sub-optimality. AM requires a pass over all the samples in each iteration and is therefore computationally expensive in the large sample setting. Moreover, due to the non-convexity of the objective function as in (1), obtaining the global optimum of (1) is NP-hard in general. AM has no local or global convergence guarantees even in usual dictionary learning setting (multiplicative model). This problem is severely amplified in the convolutional setting due to additional symmetries. Due to shift invariance of the convolutional operator, shifting a phrase embedding f_i by some amount, and applying a corresponding negative shift on the activation w_i leaves the objective in (1) unchanged. Thus, solving (1) is fundamentally *ill-posed* and has a large number of equivalent solutions.

To solve the computational inefficiency and sub-optimality problem, we propose a convolutional tensor decomposition method Huang and Anandkumar (2015). Our convolutional tensor decomposition method employs the inverse method of moments, and decompose a data cumulant (empirically computed from aggregate statistics or data moments) as phrase embeddings and shifted versions of phrase embeddings. The entire process requires one pass of data to compute the cumulant whereas AM requires data passes in each iteration. The reason why our tensor decomposition framework avoids multiple passes of the data samples is that we only estimate the phrase embeddings f_i in the learning step. Moreover, the algorithm is carefully implemented and algorithmically optimized that it requires only simple operations such as Fast Fourier Transforms (FFT) and matrix multiplications. These operations have a high degree of parallelism: for estimating L phrase embeddings, each of length n, we require $O(\log n + \log L)$ time and $O(L^2 n^3)$ processors. Our convolutional tensor decomposition yields optimization problems (in each iteration) that can be solved in closed form and it converges much faster compared to AM Huang and Anandkumar (2015).

2. Word-Sequence Modeling and Formulation

Our ConvDic+DeconvDec framework focuses on a convolutional dictionary model to summarize phrase templates, and then decode word-sequence signals to obtain the word-sequence

embeddings. The first question is how to encode the word sequence into a signal, to be input to the convolutional model and we discuss that below.

2.1 From raw text to signals

Word encoding: A word is represented as a *one-hot encoding vector*, i.e. with vector $e_i \in \mathbb{R}^d$ whose i^{th} entry is 1 and other entries are 0, where i is the index of the word in the dictionary. Alternatively, one could use the word2vec embeddings instead of one-hot encodings. We then stack the one-hot encoding vectors of each sentence together to form a *encoding matrix*. The stacking order conforms the word-sequence order.

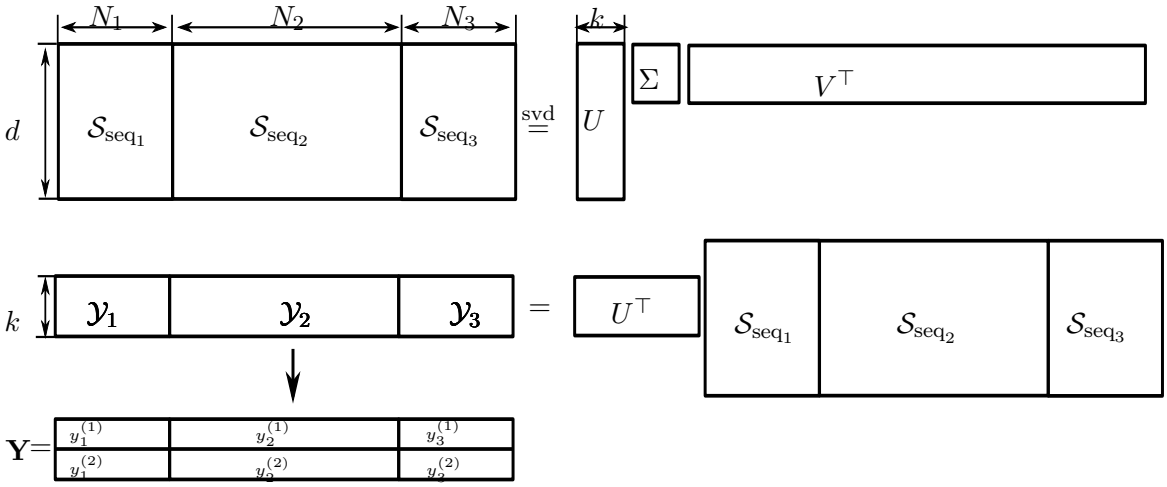


Figure 1: Principal component projection to obtain $[\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_M] = U^\top \mathbf{S} = U^\top [\mathcal{S}_{\text{seq}_1}, \mathcal{S}_{\text{seq}_2}, \ldots, \mathcal{S}_{\text{seq}_M}]$ using $\mathcal{S}$. Note that U is the top k left eigenvectors of $\mathbf{S}$.

To be precise, let us consider sentenc with N words. The *encoding matrix* of this word-sequence $\mathcal{S}_{\text{seq}}$ is $\mathcal{S}_{\text{seq}} := [s_{\text{word}_1}, s_{\text{word}_2}, \ldots, s_{\text{word}_N}] \in \mathbb{R}^{d \times N}$.

Principal components: Now that we have encoded words in each sentence, we want to find a compact representation of them in terms of a dictionary model. However, the encoding matrices are too sparse to fit a convolutional model in the word space. Instead, we perform dimensionality reduction through PCA and carry out dictionary modeling in the projected space.

Concretely, we stack the encoding matrices side by side as $\mathbf{S} := [\mathcal{S}_{\text{seq}_1}, \mathcal{S}_{\text{seq}_2}, \ldots, \mathcal{S}_{\text{seq}_M}] \in \mathbb{R}^{d \times (\sum_{i=1}^M N_i)}$, assuming there are M number of sentences in the collection of varying lengths N_1, N_2 and so on. Let $U \in \mathbb{R}^{d \times k}$ denote the top k left eigenvectors of $\mathbf{S}$. We consider $\mathcal{Y}_i := U^\top \mathcal{S}_{\text{seq}_1} \in \mathbb{R}^{k \times N_i}$, for each sentence i. We treat the rows of $\mathcal{Y}_i$ independently in parallel and fit convolutional model to each row. Denote j^{th} row of $\mathcal{Y}_i$ as $y_i^{(j)}$, and thus $\mathcal{Y}_i = \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_i^{(k)} \end{bmatrix}$.

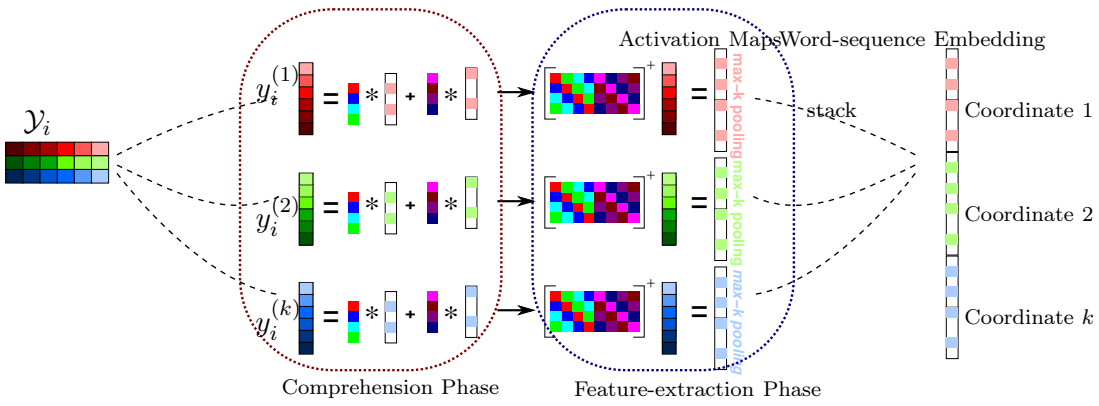
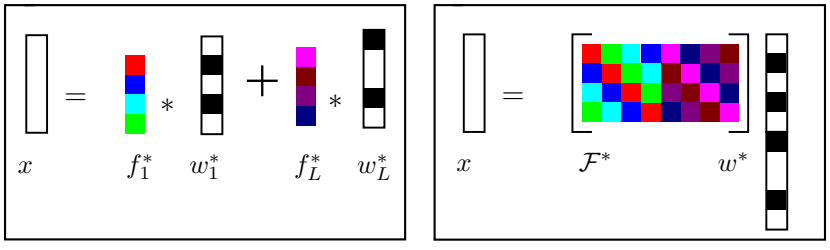


Figure 2: Overview of our ConvDic+DeconvDec framework for the i^{th} word-sequence over k coordinates. The Comprehension Phase learns phrase templates using tensor decomposition algorithm. The Feature-extraction Phase decodes activation maps using deconvolutional decoding algorithm. The activation maps are max-k pooled and stacked as the final word-sequence embedding.

Each $y_i^{(j)}$ is generated through a convolutional dictionary model over phrase templates and activation maps. Our goal in the learning phase is to learn template phrases for the collection of $[y_i^{(j)}]$ over all word-sequences $\forall i \in [M]$ across all parallel directions $\forall j \in [k]$. We will state the learning problem formally in the next section. Since all the coordinates are independent and the phrase templates are learned in parallel over all the coordinates, we drop the index j to denote a coordinate of the i^{th} word sequence $y_i^{(j)}$. In the following subsection, a patch from $y_i^{(j)}$ will be denoted as x.

2.2 Comprehension Phase – Learning Phrase Templates



(a) Convolutional model **(b)** Reformulated model

Figure 3: Convolutional tensor decomposition for learning convolutional ICA models Huang and Anandkumar (2015).(a) The convolutional generative model with template phrases. (b) Reformulated multiplicative model where $\mathcal{F}^*$ is column-stacked circulant matrix.

A word sequence is composed of superposition of overlapping patches, therefore we are interested in learning a generative model over overlapping patches. We can also view these

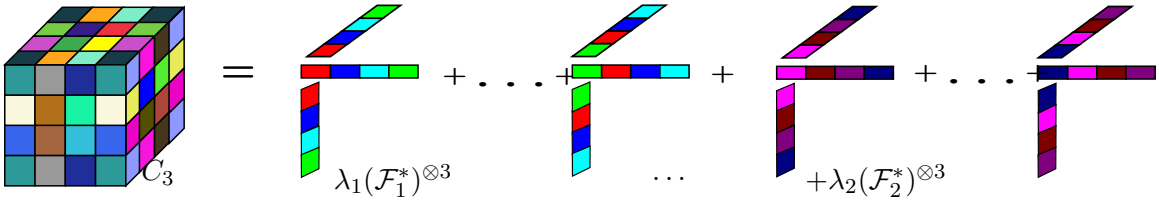


Figure 4: The third order cumulant is decomposed superposition of third order outer product of template phrases and third order outer product of shifted template phrases.

patches as phrases. A length n patch x is generated as the superposition of L phrase embeddings f_l^* convolved at L activation maps w_l^*, $\forall l \in [L]$. Due to the property of the convolution, the convolution is reformulated as the multiplication of $\mathcal{F}^*$ and w^*, where $\mathcal{F}^* := [\mathsf{Cir}(f_1^*), \mathsf{Cir}(f_2^*), \ldots, \mathsf{Cir}(f_L^*)]$ is the concatenation of circulant matrices and w^* is the *row-stacked* vector $w^* := \begin{bmatrix} w_1^* \\ w_2^* \\ \vdots \\ w_L^* \end{bmatrix} \in \mathbb{R}^{nL}$. To be precise, a patch

$$x = \sum_{l \in [L]} f_l^* * {w_l}^* = \mathcal{F}^* \cdot w^*, \tag{2}$$

This is illustrated in Fig 4(a). $\mathsf{Cir}(f_l^*)$ is circulant matrix corresponding to phrase template f_l^*, whose columns are shifted versions of f_l^* as shown in Fig 4(a). Note that although $\mathcal{F}^*$ is a n by nL matrix, there are only nL free parameters. Given access to the collection of word-sequence sample patches, $X := [x^1, x^2, \ldots]$, generated according to the above model, we aim to estimate the true template phrases f_i^*, for $i \in [L]$. In section 3 we will elaborate on our convolutional tensor decomposition dictionary learning method (ConvDic).

If the patches are in the same coordinate of the word sequence, these patches share a common set of phase templates, but their activation maps are different. The activation maps are the discriminative features that distinguish different patches. Once the template phrases are estimated, we can use standard decoding techniques, such as the square loss criterion in (1) to learn the activation maps for the individual maps.

2.3 Feature-extraction Phase – Word-sequence Embeddings

Activation maps in a coordination: After learning a good set of phrase templates $\{f_1, \ldots, f_L\}$ and thus $\mathcal{F}$, we use the deconvolutional decoding (DeconvDec) to obtain the activation maps for the j^{th} coordinate. For each observed coordinate of the word-sequence $y_i^{(j)}$, the activation map w_l^* in (2) indicates the locations where i^{th} template phrase f_l^* is activated and w^* is the *row-stacked* vector $w^* := [w_1^*; w_2^*; \ldots w_L^*]$. An estimation of w^*, $w_i^{(j)}$, is achieved as follows

$$w_i^{(j)} = \mathcal{F}^\dagger {y_i^{(j)}}^\top. \tag{3}$$

Note that the estimated phrase templates are zero padded to match the length of the word-sequence.

We assume that the elements of w^* are drawn from some product distribution, i.e. different entries are independent of one another, and we have the independent component analysis (ICA) model in (2). When the distribution encourages sparsity, e.g. Bernoulli-Gaussian, only a small subset of locations are active, and we have the *sparse coding* model in that case. We can also extend to dependent distributions such as Dirichlet for w^*, along the lines of Blei et al. (2003), but limit ourselves to ICA model for simplicity. This activation map $w_i^{(j)} \in \mathbb{R}^{N_i \cdot L}$ contains sequence embeddings from coordinate j only, and will be used as one coordinate of our final word-sequence embeddings.

Varying sentence length: One difficulty in learning the template phrases using our convolutional tensor decomposition model is that different word-sequence has a different length N_i, therefore the activation maps are of varying length as well. We resolved this problem by *max-k pooling*. In other words, we extract most informative global discriminative features from the activation maps, as illustrated in Figure 2. Finally, we concatenate all the max-k pooled coordinate sequence embeddings as a long vector as the final word-sequence embedding.

The overall framework flow is depicted in Fig 2.

3. Convolutional Dictionary Model Learning

In this section, we will focus on the comprehension phrase and propose a tensor decomposition dictionary learning method for learning the phrase templates. As we demonstrated earlier, the generative model for a patch from one coordinate, x, is illustrated in Fig 4. x is generated as the superposition of L phrase embeddings f_l^* convolved at L activation maps w_l^*, $\forall l \in [L]$. Let $f_l^* \in \mathbb{R}^n$ be the unknown template phrases, where $j \in [L]$ denotes the index of phrases. Under the convolution ICA model, we show that the third order cumulant has a nice tensor decomposition form Huang and Anandkumar (2015), as given below.

Lemma 1 (Decomposition of Cumulants) *The unfolded third order cumulant C_3 in* (6) *has the following decomposition form*

$$C_3 = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* (\mathcal{F}_j^* \odot \mathcal{F}_j^*)^\top = \mathcal{F}^* \Lambda^* (\mathcal{F}^* \odot \mathcal{F}^*)^\top, \quad \textit{where } \Lambda^* := \mathsf{diag}(\lambda_1^*, \lambda_2^*, \ldots, \lambda_{nL}^*) \quad (4)$$

where $\mathcal{F}_j^*$ denotes the j^{th} column of the column-stacked *circulant matrix $\mathcal{F}^*$ and λ_j^* is the third order cumulant corresponding to the (univariate) distribution of $w^*(j)$.*

The third order cumulant C_3 is a third order tensor, which could be empirically estimated using first three orders of moments. The form of the cumulant tensor is in Appendix A.1.

The decomposition form in (4) is known as the CANDECOMP/PARAFAC (CP) decomposition form Anandkumar et al. (2014) (the usual form has the decomposition of the tensor and not its unfolding, as above). We attempt to recover the unknown template phrases f_i^* through decomposition of the third order cumulants C_3. Our goal is to obtain template phrase estimates f_l's $\forall l \in [L]$ and weight estimates Λ such that the cumulant C_3 is decomposed as $\mathcal{F}\Lambda(\mathcal{F} \odot \mathcal{F})^\top$, as in equation 4. The formal statement is deferred to Appendix A.2.

We propose *convolutional tensor decomposition* using efficient Alternating Least Square with Circulant Constraint to solve the non-convex optimization problem. Consider the asymmetric relaxation and introduce separate variables $\mathcal{F}, \mathcal{G}$ and $\mathcal{H}$ for filter estimates along each of the modes to fit the third order cumulant tensor C_3. ALS iteratively alternates over the three variables and updates one mode by fixing the two other modes

$$\min_{\mathcal{F}} \quad \|C_3 - \mathcal{F}\Lambda\,(\mathcal{H} \odot \mathcal{G})^\top\|_F^2 \text{ s.t. } blk_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l \in [L] \quad (5)$$

Similarly, $\mathcal{G}$ and $\mathcal{H}$ have the same column-stacked circulant matrix constraint and are updated similarly in alternating steps. The diagonal matrix Λ is updated through normalization. The objective function is defined in Huang and Anandkumar (2015), refer to appendix A for details.

4. Experiments

We evaluate the quality of our word sequence embeddings using three challenging natural language process tasks: sentiment classification, paraphrase detection, and semantic textual similarity estimation. Eight datasets which cover various domains are used as shown in Table 1.

Dataset	**Domain**	**Label**	**Label Distribution**	M
Review	Moview Reviews	{-1,1}	[0.49,0.51]	64720
SUBJ	Obj/Subj comments	{-1,1}	[0.50,0.50]	1000
MSRpara	news sources	{-1,1}	[0.33,0.67]	5801×2
STS-MSRpar	newswire	[0,5]	[0.00,0.02,0.10,0.24,0.47,0.17]	1500×2
STS-MSRvid	video caption	[0,5]	[0.13,0.21,0.14,0.16,0.21,0.14]	1500×2
STS-OnWN	glosses	[0,5]	[0.01,0.02,0.04,0.12,0.35,0.47]	750×2
STS-SMTeuroparl	machine translation	[0,5]	[0.01,0.00,0.00,0.02,0.19,0.78]	1193×2
STS-SMTnews	machine translation	[0,5]	[0.00,0.01,0.01,0.06,0.19,0.73]	399×2

Table 1: Summary statistics of the datasets used.

For all the datasets, we train a simple logistic regression model on the training samples and report test classification accuracy using a 10-fold cross validation. Sentiment analysis and paraphrase detection belong to binary classification tasks. In a binary classification task, either accuracy or F score is used as evaluate metric. Recall that F-score is the harmonic mean of precision and recall, i.e., $F = 2 \cdot (\text{precision} \cdot \text{recall})/\text{precision} + \text{recall}$. Precision is the number of true positives divided by the total number of elements labeled as belonging to the positive class, and recall is the number of true positives divided by the total number of elements that belong to the positive class.

Our ConvDic+DeconvDec learns word-sequence embeddings from scratch and requires no pre-training. When working on a new dataset from a new domain, we train fresh set of phrase templates as called domain phrase templates. Using these domain phrase templates, we decode activation maps and then form phrase-embeddings. Our approach is different from skip thoughts, where universal phrase embeddings are generated Kiros et al. (2015).

4.1 Evaluation Task: Sentiment Classification

Sentiment analysis is an important task in natural language process as automated labeling of word sequences into positive and negative opinions is used in various settings. We evaluate our sentence embeddings on two datasets from different domains, such as movie review and subjective and objective comments, as in Table 1. Using word-sequence embeddings combined with NB features, we obtain the state-of-the-art classification results for both these datasets as in Table 2.

Method	MR	SUBJ
NB-SVM Wang and Manning (2012)	79.4	93.2
MNB Wang and Manning (2012)	79.0	93.6
cBoW Zhao et al. (2015)	77.2	91.3
GrConv Zhao et al. (2015)	76.3	89.5
RNN Zhao et al. (2015)	77.2	93.7
BRNN Zhao et al. (2015)	82.3	94.2
CNN Kim (2014)	81.5	93.4
AdaSent Zhao et al. (2015)	83.1	95.5
Paragraph-vector Le and Mikolov (2014)	74.8	90.5
Skip-thought Kiros et al. (2015)	75.5	92.1
ConvDic+DeconvDec	**78.9**	**92.4**

Table 2: Binary classification tasks: sentiment analysis task of cataloging a word-sequence into two different categories. Classification accuracies in percentage on standard benchmarks (movie review and subject dataset) are displayed. The first group contains results using bag-of-words models; the second group exhibits some supervised compositional models; the third group is paragraph vector; the fourth is the skip-thought result.

Method	Outside Information [1]	F score
Vector Similarity Mihalcea et al. (2006)	word similarity	75.3%
ESA Hassan (2011)	word semantic profiles	79.3%
LSA Hassan (2011)	word semantic profiles	79.9%
RMLMG Rus et al. (2008)	syntacticinfo	80.5%
ConvDic+DeconvDec	**none**	**80.7%**
Skip-thought Kiros et al. (2015)	train large book corpus	81.9%

Table 3: Binary classification tasks: paraphrase detection task, which operates on pairs of word-sequences and decides on whether they are a paraphrase of each other or not. Comparison of F-score with other unsupervised sentence paraphrase approaches. Other methods use auxiliary information such as word similarities trained on Wikipedia or from *WordNet*. In contrast, our algorithm learns sentence embeddings from scratch.

1. The word similarities information they use are either trained in Wikipedia (4.4 million articles in contrast to the 4076 sentences of paraphrase dataset we use) or from *WordNet* with expert knowledge.

4.2 Evaluation Task: Paraphrase Detection

We consider the *paraphrase detection* task on the Microsoft paraphrase corpus Quirk et al. (2004); Dolan et al. (2004). We employ 4076 sentence pairs as training data to learn the sentence embeddings and regress on the ground truth binary labels with our learned sentence embeddings. The remaining test data is used to calculate classification error.

As discussed in Tai et al. (2015), we combine the pair of sentence embeddings produced earlier w_L and w_R, i.e., the embedding for the right and the left sentences. We generate features for classification using both the distance (absolute difference) and the product between the pair (w_L, w_R): $[w_L \odot w_R, \|w_L - w_R\|]$, where $\odot$ denotes the element-wise multiplication.

In contrast to other unsupervised methods which are trained using outside information such as wordnet and parse trees, our unsupervised approach use **no** extra information, and still achieves comparable results with the state of art Wiki (2014) as in table 3. We show some examples of paraphrase and non-paraphrase we identified.

Paraphrase detected: *(1) Amrozi accused his brother, whom he called "the witness", of deliberately distorting his evidence. (2) Referring to him as only "the witness", Amrozi accused his brother of deliberately distorting his evidence.* The two sentences are the "difficult sentence" to show how our algorithm detect paraphrases since they are not simple switching of clauses, and the sentence structures differ quite significantly in the two sentences.

Non-paraphrase detected : *(1) I never organised a youth camp for the diocese of Bendigo. (2) I never attended a youth camp organised by that diocese.* Similarly with non-paraphrase detection, the two sentences share common words such as youth camp and organized, but our method is able to successfully detect them as non-paraphrase.

4.3 Evaluation Task: Semantic Textual Similarity Estimation

For the Semantic Textual Similarity (STS) task, the goal is to predict a real-valued similarity score in a range $[1, K]$ given a sentence pair. We include datasets from STS task in various domains including news, image and video description, glosses from WordNet/OntoNotes, the output of machine translation systems with reference translation.

To frame semantic test similarity estimation task into the multi-class classification framework, the gold rating $\tau \in [K_1, K_2]$ is discretized as $p \in \Delta^{K_2-K_1}$ in the follow manner Tai et al. (2015), $p_i = \lfloor \tau \rfloor - \tau + 1$ if $i = \lfloor \tau \rfloor + 1 - K_1$, $p_i = \tau - \lfloor \tau \rfloor$ if $i = \lfloor \tau \rfloor + 2 - K_1$, and $p_i = 0$ otherwise. This reduces to finding a predicted $\hat{p}_\theta \in \Delta^{K_2-K_1}$ given model parameters θ to be closest to p in terms of KL divergence Tai et al. (2015). We use a logistic regression classifier to predict $\hat{p}_\theta$ and estimate $\hat{\tau}_\theta = [K_1, \ldots, K_2]\hat{p}$.

Results on STS task datasets are illustrated in Table 4. As in Wieting et al. (2015), Pearson's r of the median, 75th percentile, and highest score from the official task rankings are showed. We then compare our method against the performance of supervised models in Wieting et al. (2015): PARAGRAM-PHRASE (PP), projection (proj.), deep-averaging network (DAN), recurrent neural network (RNN) and LSTM; as well as the state-of-the-art unsupervised model skip-thought vectors Kiros et al. (2015).

As we can see from the table, LST is performing poorly even though a back-propagation after seeing the training labelings is carried out for sequence embedding learning. Our method is an unsupervised approach as in skip-thought vectors. However, our algorithm doesn't output universal word-sequence embeddings across domains. We train a fresh model

and a new set of domain phrase templates from scratch. Therefore our algorithm is performing better for these individual datasets on the STS task.

	Supervised	+	Unsupervised	Supervised	Methods		Unsupervised	Methods
Dataset	50%	75%	Max	DAN	RNN	LSTM	Skip-thought	**ConvDic+DeconvDec**
MSRpar	51.5	57.6	73.4	40.3	18.6	9.3	16.8	**36.0**
MSRvid	75.5	80.3	88.0	70.0	66.5	71.3	41.7	**61.8**
SMT-eur	44.4	48.1	56.7	43.8	40.9	44.3	35.2	**37.5**
OnWN	60.8	65.9	72.7	65.9	63.1	56.4	29.7	**33.1**
SMT-news	40.1	45.4	60.9	60.0	51.3	51.0	30.8	**72.1**

Table 4: STS task results: Pearson's $r \times 100$ on MSRpar, MSRvid, OnWN, SMTeuroparl and SMTnews dataset. The first three columns are official rankings reported in the STS2012 official website, so it combines both supervised and unsupervised methods. The second three columns are reported by Wieting et al. (2015). Our comparison against the state-of-the-art unsupervised word-sequence embedding method is in the last two columns.

5. Conclusion

Our unsupervised efficiently ConvDic+DeconvDec yields word-sequence representations that perform well across a wide range of NLP tasks over datasets from various domains. At the same time, our efficient tensor learning algorithm requires a relatively small amount of data and computation. In the future, we plan to investigate the use of ConvDic+DeconvDec for other domains such as images and videos, as well obtaining joint text-image embeddings.

References

Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

David Belanger and Sham Kakade. A linear dynamical system model for text. *arXiv preprint arXiv:1502.04081*, 2015.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Hilton Bristow and Simon Lucey. Optimization methods for convolutional sparse coding. *arXiv preprint arXiv:1406.2407*, 2014.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*, page 350. Association for Computational Linguistics, 2004.

Samer Hassan. *Measuring semantic relatedness using salient encyclopedic concepts*. University of North Texas, 2011.

Furong Huang and Animashree Anandkumar. Convolutional dictionary learning through tensor factorization. In *Conference and Workshop Proceedings of JMLR*, pages 116–129, 2015.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 655–665. The Association for Computer Linguistics, 2014a. ISBN 978-1-937284-72-5. URL `http://aclweb.org/anthology/P/P14/P14-1062.pdf`.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014b.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems*, pages 3276–3284, 2015.

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

Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780, 2006.

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

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

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

Chris Quirk, Chris Brockett, and William B Dolan. Monolingual machine translation for paraphrase generation. In *EMNLP*, pages 142–149, 2004.

Vasile Rus, Philip M McCarthy, Mihai C Lintean, Danielle S McNamara, and Arthur C Graesser. Paraphrase identification with lexico-syntactic graph subsumption. In *FLAIRS conference*, pages 201–206, 2008.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*, 2015.

ACL Wiki. Paraphrase identification (state of the art), 2014. URL `http://aclweb.org/aclwiki/index.php?title=Paraphrase_Identification_%28State_of_the_art%29`.

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

Han Zhao, Zhengdong Lu, and Pascal Poupart. Self-adaptive hierarchical sentence model. *arXiv preprint arXiv:1504.05070*, 2015.

Appendix A. Convolutional Tensor Decomposition For Learning Convolutional Dictionary Model

A.1 Cumulant Form Huang and Anandkumar (2015)

Let $C_3 \in \mathbb{R}^{n \times n^2}$ denote the unfolded version of third order cumulant tensor, it is given by

$$C_3 := \mathbb{E}[x(x \odot x)^\top] - unfold(Z) \tag{6}$$

where $[Z]_{a,b,c} := \mathbb{E}[x_a]\mathbb{E}[x_b x_c] + \mathbb{E}[x_b]\mathbb{E}[x_a x_c] + \mathbb{E}[x_c]\mathbb{E}[x_a x_b] - 2\mathbb{E}[x_a]\mathbb{E}[x_b]\mathbb{E}[x_c],\ \forall a, b, c \in [n]$.

For example, if the l^{th} activation is drawn from a Poisson distribution with mean $\tilde{\lambda}$, we have that $\lambda_l^* = \tilde{\lambda}$. Note that if the third order cumulants of the activations, i.e. λ_j^*'s, are zero, we need to consider higher order cumulants. This holds for zero-mean activations and we need to use fourth order cumulant instead. Our method extends in a straightforward manner for higher order cumulants.

A.2 Alternating Least Squares for Convolutional Tensor Decomposition Huang and Anandkumar (2015)

Objective Function: Our goal is to obtain template phrase estimates f_i's which minimize the Frobenius norm $\|\cdot\|_{\mathbb{F}}$ of reconstruction of the cumulant tensor C_3,

$$\begin{aligned} &\min_{\mathcal{F}} \quad \|C_3 - \mathcal{F}\Lambda\,(\mathcal{F} \odot \mathcal{F})^\top\|_F^2, \\ &\text{s.t. } blk_l(\mathcal{F}) = U\mathsf{diag}(\mathsf{FFT}(f_l))U^{\mathsf{H}},\ \|f_l\|_2 = 1, \quad \forall l \in [L], \quad \Lambda = \mathsf{diag}(\lambda). \end{aligned} \tag{7}$$

where $blk_l(\mathcal{F})$ denotes the l^{th} circulant matrix in $\mathcal{F}$, i.e., $\mathcal{F} = [blk_1(\mathcal{F}), \ldots, blk_L(\mathcal{F})]$. The conditions in (7) enforce $blk_l(\mathcal{F})$ to be circulant and for the template phrases to be normalized. Recall that U denotes the eigenvectors for circulant matrices. Now we explain our proposed convolutional tensor decomposition using efficient Alternating Least Square with Circulant Constraint to solve (7).

To solve the non-convex optimization problem in (7), we consider the alternating least squares (ALS) method with *column stacked* circulant constraint. We first consider the asymmetric relaxation of (7) and introduce separate variables $\mathcal{F}, \mathcal{G}$ and $\mathcal{H}$ for filter estimates along each of the modes to fit the third order cumulant tensor C_3. We then perform alternating updates by fixing two of the modes and updating the third one. For instance,

$$\min_{\mathcal{F}} \quad \|C_3 - \mathcal{F}\Lambda\,(\mathcal{H} \odot \mathcal{G})^\top\|_F^2 \text{ s.t. } blk_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l \in [L] \tag{8}$$

Similarly, $\mathcal{G}$ and $\mathcal{H}$ have the same column-stacked circulant matrix constraint and are updated similarly in alternating steps. The diagonal matrix Λ is updated through normalization.

We now introduce the *Convolutional Tensor* (CT) Decomposition algorithm to efficiently solve (8) in closed form, using simple operations such as matrix multiplications and fast Fourier Transform (FFT). We do not form matrices $\mathcal{F}, \mathcal{G}$ and $\mathcal{H} \in \mathbb{R}^{n \times nL}$, which are large, but only update them using filter estimates $f_1, \ldots, f_L, g_1, \ldots, g_L, h_1, \ldots h_L$.

Using the property of least squares, the optimization problem in (8) is equivalent to

$$\min_{\mathcal{F}} \|C_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger \Lambda^\dagger - \mathcal{F}\|_F^2 \text{ s.t. } blk_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l \in [L] \tag{9}$$

when $(\mathcal{H} \odot \mathcal{G})$ and Λ are full column rank. The full rank condition requires $nL < n^2$ or $L < n$, and it is a reasonable assumption since otherwise the filter estimates are redundant. In practice, we can additionally regularize the update to ensure full rank condition is met. Denote

$$M := C_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger, \tag{10}$$

where † denotes pseudoinverse. Let $blk_l(M)$ and $blk_l(\Lambda)$ denote the l^{th} blocks of M and Λ.Since (9) has block constraints, it can be broken down in to solving L independent sub-problems

$$\min_{f_l} \left\| blk_l(M) \cdot blk_l(\Lambda)^\dagger - U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}} \right\|_F^2 \quad s.t. \quad \|f_l\|_2^2 = 1, \forall l \in [L] \tag{11}$$