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

Furong Huang*[1] and Animashree Anandkumar†[2]

[1]Microsoft Research New York City
[2]University of California Irvine

May 4, 2017

Abstract

Unsupervised text embeddings extraction is crucial for text understanding in machine learning. Word2Vec and its variants have received substantial success in mapping words with similar syntactic or semantic meaning to vectors close to each other. However, extracting context-aware word-sequence embedding remains a challenging task. Training over large corpus is difficult as labels are difficult to get. More importantly, it is challenging for pre-trained models to obtain word-sequence embeddings that are universally good for all downstream tasks or for any new datasets. We propose a two-phased ConvDic+DeconvDec framework to solve the problem by combining a word-sequence dictionary learning model with a word-sequence embedding decode model. We propose a convolutional tensor decomposition mechanism to learn good word-sequence phrase dictionary in the learning phase. It is proved to be more accurate and much more efficient than the popular alternating minimization method. In the decode phase, we introduce a deconvolution framework that is immune to the problem of varying sentence lengths. The word-sequence embeddings we extracted using ConvDic+DeconvDec are universally good for a few downstream tasks we test on. The framework requires neither pre-training nor prior/outside information.

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 [4, 6, 7, 23, 27]. Word embeddings

*furongh@microsoft.com
†a.anandkumar@uci.edu

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. However word embeddings do not suffice for more advanced tasks which require context-aware information or word orders, for instance, paraphrase detection, sentiment analysis, plagiarism detection, information retrieval or 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 [25, 36, 30, 31, 3, 16, 26, 32, 34, 26]. This line of work is still a variant of the word embedding which is not context aware: the word embeddings learned without consideration of the surrounding texts are used along with the composition operators to map word embeddings to sentence embeddings. Furthermore, all these methods produce sentence representations that depend on a supervised task, and the class labels are back-propagated to update the composition weights [17].

Since most of the existing 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 they require intensive pre-training and hyper-parameter tuning. Unsupervised embedding learning remedies this problem. The state-of-the-art unsupervised framework is Skip-thought [20], 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 with no supervision/labels or annotated features such as parse trees? Can we avoid intensive pre-training? Also, can the sentence embeddings be context-aware, handle variable lengths, and not be limited to specific domains?

We propose an unsupervised ConvDic+DeconvDec framework: it requires neither pre-training nor prior information (or expert input such as parse trees); it is robust to varying word-sequence length; and it summaries template phrases for better comprehension of the corpus. Our framework is composed of two phases, a *comprehension phase*, which summarizes template phrases using *Convolutional Dictionary Model Learning*, followed by a *feature-extraction phase*, which extracts word-sequence embedding feature map using *Deconvolutional Decoding*. In the *comprehension phase*, convolutional dictionary model learns *shift-invariant* phrase embeddings. However learning a good set of phrase embeddings requires solving a non-convex optimization problem which is NP hard in general. We cast it as a spatial dictionary learning problem.

1.1 Related Works

The traditional dictionary learning problem involves detecting hidden dictionary elements from data points which are weighted sum of the shared dictionary elements. In this work, we extend the traditional dictionary learning model to incorporate **locations** of dictionary elements in addition to the **weights**. Convolutional model is natural to encode local objects and their locations. In NLP tasks, convolutional models are recently

introduced and have had revolutionary performance [16, 19, 9, 15].

A convolutional dictionary learning model posits that the observed sequence signal x is a linear combination of convolutions of unknown *phrase templates* (a.k.a spatial dictionary elements) ${f_l}^*$ vectors and unknown *activation maps* ${w_l}^*$ vectors:

$$x^j = \sum_{l=1}^{L} {f_l}^* * {w_l^j}^*. \tag{1}$$

The non-zero elements in vector ${w_l^j}^*$ indicate the locations where phrase embedding ${f_l}^*$ are active for sample j, and the amplitudes of the non-zero elements are the weights of the activations.

In order to learn the model in (1), a square loss reconstruction criterion is usually employed:

$$(\widehat{f}_l, \widehat{w}_l) = \arg\min_{\|f_l\|=1, w_l} \sum_j \|x^j - \sum_{l=1}^{L} f_l * w_l^j\|^2 \tag{2}$$

The constraints $\|f_l\| = 1$ are enforced, since otherwise, the scaling can be exchanged between the phrase embeddings f_l and the activation maps w_i. The activation maps $\{w_1^j, \ldots, w_L^j\}$ serve as the *phrase embedding* for sequence x^j.

A popular heuristic for solving (2) is based on alternate minimization (AM) [5], where the phrase embeddings f_l are optimized, while keeping the activations w_l^j fixed, and vice versa. 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, obtaining the global optimum of Equation (2) is NP-hard in general due to (1) the non-convexity of the objective function in Equation (2) and (2) shift invariance of the convolutional operator (shifting a phrase embedding f_l by some amount, and applying a corresponding negative shift on the activation w_l^j leaves the objective in Equation (2) unchanged). Therefore, solving Equation (2) is fundamentally *ill-posed* and has a large number of equivalent solutions.

1.2 Summary of Results

Can we design a method that efficiently incorporate the shift invariance constraints into the learning problem? Can it *break the symmetry* between the equivalent solutions? Is it scalable to huge datasets? In this paper, we provide positive answers to these questions. We propose a novel framework for learning convolutional models through tensor decomposition. We consider inverse method of moments to estimate the model parameters via decomposition of higher order (third or fourth order) input cumulant tensors. When the inputs x^j are generated from a convolutional model in Equation (1), with independent activation maps ${w_l^j}^*$, i.e. a convolutional ICA model, we show that the moment tensors have a CP decomposition, whose components form a stacked *circulant* matrix. We propose a novel method for tensor decomposition when such circulant (i.e. shift invariance) constraints are imposed. The *feature-extraction phase* following the *comprehension phase* uses a *Deconvolutional Decoding* framework to obtain word-sequence embeddings.

In the learning phase, we propose a novel approach to solve the non-convex optimization problem in equation (2) using tensor decomposition. Our method estimates the third order moments or approximately the cumulant tensor of the corpus (triple-wise statistics of the corpus aggregated over sentence patches) and efficiently solves an *constrained over-complete tensor decomposition* problem to obtain $\widehat{f_l}$ in the learning phrase, using the alternating least squares with circulant constraints (ALSwCC) method for tensor decomposition. The ALSwCC method for tensor decomposition is not to be confused with the AM method for solving Equation (2). One merit of this approach is that estimating ${f_l}^*$ is possible without specifying sample-specific weights ${w_l^j}^*$.

In the decoding phase, we estimate ${w_l^j}^*$ for a new sentence j^{th} as the *template phrases* $\widehat{f_l}$ are obtained in the learning phase. We emphasize that our method requires only one pass over data to compute the higher order cumulant of the input data. Decoding all the activation maps ${w_l^j}^*$ in each iteration by solving equation (2), which is the mechanism used in AM, is hugely expensive. Instead, our method avoids it by estimating only the phrase templates ${f_l}^*$ in the learning phase. In other words, the activation maps ${w_l^j}^*$'s are averaged out in the input cumulant. This is a huge saving in running time compared to AM method which requires a pass over data in each step.

We show that the resulting optimization problem (in each ALSwCC step) can be solved in closed form, using simple operations such as Fast Fourier transforms (FFT) and matrix multiplications. These operations have a high degree of parallelism: for estimating L template phrases, each of length n (filter length), we require $O(\log n + \log L)$ time with $O(L^2 n^3)$ degree of parallelism. Note that the complexity is **independent** with number of sentence patches N. Thus, our method is embarrassingly parallel and scalable to huge datasets. We carefully optimize computation and memory costs by exploiting tensor algebra and circulant structure. We implicitly carry out many of the operations and never form large (circulant) matrices to minimize storage requirements.

Experiments further demonstrate superiority of our method compared to AM. Our algorithm converges accurately and much faster to the true underlying phrase embeddings compared to AM. Our algorithm is also orders of magnitude faster than AM.

We apply our method for the tasks of sentiment classification, semantic textual similarity estimation and paraphrase detection in natural language processing. These are challenging tasks, since a context-aware understanding of text, as opposed to simple word relationships, is usually needed to perform well. We learn the embeddings from scratch without using any auxiliary information. While previous works use information such as parse trees, Wordnet or pre-training on a much larger corpus, we obtain competitive results, which are close to state-of-art, from scratch. This is because our convolutional tensor framework efficiently learns the information present in the word order, which is crucial for phrase understanding. We evaluate our word-sequence embeddings using three downstream tasks which covers NLP tasks of varying complexity over eight datasets from various domains. Our unsupervised ConvDic+DeconvDec yields generic word-sequence representations that perform universally good across all tasks and datasets we listed.

2 Word-Sequence Modeling and Formulation

Convolutional dictionary model is indeed a one layer Convolutional Neural Network (CNN). CNN is the main reason of recent breakthrough success in computer vision. Extending the success of CNNs from computer vision to natural language processing is not obvious since the notion of filters or common patterns are not clear in a sentence. Yet machine learning researchers have successfully used CNNs for NLP tasks and have achieved substantial triumphs [16, 19, 9, 15].

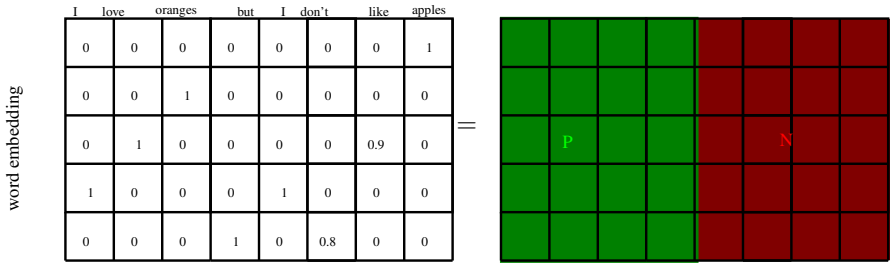


Figure 1: An toy example exhibiting matrix representation of a sequence of words. The phrase template marked as green represents positive opinions (P) whereas the red phrase template represents negative opinions (N). The word specific semantic meanings are captured vertically by word embeddings or one-hot encodings. The phrase level similarities are captured by extracting phrase templates horizontally across phrases. In practice, the interpretation of the phrase templates are not necessarily obvious.

There are some major differences between computer vision and natural language processing.

Sentence matrix representation The input data matrix in computer vision is the matrix of image pixels itself, whereas the input data in natural language process is the set of phrases, sentences, paragraphs or articles. It is common practice to represent sequence of words as a matrix whose columns correspond to tokens, and rows correspond to coordinates of the word embeddings [24, 27] or one-hot encoding vectors[1]. See Figure 1 (left matrix) for an example.

Single direction transition invariance The transition invariance or location invariance no longer exists vertically (across coordinates of word embedding)[2]. The coordinates/rows of the word embeddings serve jointly to distinguish the word level semantic meaning differences. The location invariance in the horizontal direction (across words) exists. For instance, "I love oranges" is equivalent to "oranges I love". The analogy of *image filters* is *phrase templates* in NLP and these phrase templates are *dictionary elements* across phrases in the corpus. Each phrase is a weighted sum of those phrase templates. For instance, our two phrase templates indicate positive opinions and negative opinions in Figure 1, the positive template is activated at the first half of the

[1]For a word, its corresponding one-hot encoding vector has a non-zero entry, 1, placed at the i-th position of the vector, where i is the index of the word in the vocabulary.

[2]We offer some computer vision transition invariance and max-k pooling background explanations in appendix E.

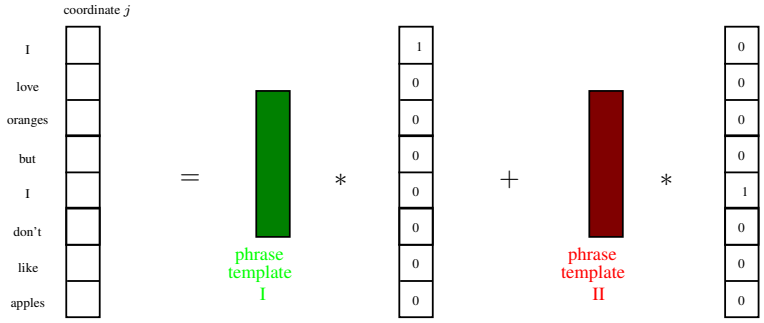


Figure 2: Generative model for one coordinate of the matrix representation of sequence of words.

sentence (the first phrase), while the negative template is activated at the second half of the sentence (the second phrase). This example is over-simplified, and should be served as just an intuition for understanding the modeling. In practice, the interpretation of the phrase templates are not necessarily obvious. Therefore visualization is difficult.

Varying sentence lengths Pooling in NLP tasks requires more thinking than in computer vision. In contrast to pooling in computer vision where maximum (or top-k) element(s) in the activation map matrix is/are picked out, pooling in NLP can be performed horizontally only or over both directions. The former preserves the word-level semantic differences, whereas the latter considers phrase-level semantic differences only.

Unclear compositionality Modeling compositionality plays an essential role in successful image classification tasks. In CNNs, each filter composes a local patch of low-level features into higher-level representation. The images are first decomposed into complex objects first, and complex objects are then decomposed into simpler objects. The hierarchical structure allows for hierarchical filters for the images. However, the meaning of higher-level representation over lower level phrases are not clear. And thus the compositionality in NLP is not as obvious as in computer vision. It is not clear whether the hierarchical structure in CNN will improve the performance. Additionally, CNNs are susceptible to the spurious local optima and saddle points.

We introduce a "convolutional dictionary learning model" to learn a good set of filters or phrases templates over word-sequences. Consider each coordinate (row) of the matrix in Figure 1 (left) independently, we posit a generative model on one coordinate of the word sequence as in Figure 2. Note that the row vector is drawn as a column for convenience of visualization. The learning problem is then to identify good set of phrase templates that are common to all phrases in the corpus for all coordinates/columns.

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 the input to the convolutional model and we discuss that below.

2.1 From Rraw Text to Signals

Word encoding: A word is represented as a *one-hot encoding vector* of vocabulary length d. One could use *word2vec* embeddings alternatively. A sequence of words (could be multiple sentences) form an *encoding matrix* $\mathbf{S} \in \mathbb{R}^{d \times (\sum_{i=1}^{M} N_i)}$, assuming that there exists M sentences of varying lengths N_i. Now rows are the coordinates, and columns are words.

Principal components: The encoding matrix is extremely parsimonious and thus redundant and unstable. We perform dimensionality reduction through PCA and project the encoding matrix to a more compact and robust matrix $\mathbf{Y}$.

Let $\mathbf{P} \in \mathbb{R}^{d \times k}$ denote the top k (where $k \ll d$) left eigenvectors of $\mathbf{S}$. The encoding matrix is then projected to its principal directions $\mathbf{Y} = \mathbf{P}^\top \mathbf{S} \in \mathbb{R}^{k \times \sum_{i=1}^{M} N_i}$.

We treat the rows/coordinates of $\mathbf{Y}$, denoted as $\mathbf{Y}^{(j)}$, independently in parallel, fit convolutional model to each row and learn row-specific phrase templates.

Our goal in the learning phase is to learn template phrases for all coordinates $\forall j \in [k]$ of $\mathbf{Y}$. We now state the learning problem formally.

2.2 Comprehension Phase – Learning Phrase Templates

In this subsection, we extract overlapping patches from $Y^{(j)}$ and denote them as x for simplicity. Since all the coordinates are independent and the phrase templates are learned in parallel over all the coordinates, we drop the index j in this section for simple notations.

Our learning model posits that a length n patch x is generated as the superposition of L phrase embeddings $\{f_1^*, \ldots, f_L^*\}$ convolved at L activation maps $\{w_1^*, \ldots, w_L^*\}$:

$$x = \sum_{l=1}^{L} f_l^* * {w_l}^* \tag{3}$$

Due to the property of the convolution, the convolution is reformulated as the multiplication of $\mathcal{F}^*$ and w^*,

$$x = \sum_{l \in [L]} f_l^* * {w_l}^* = \mathcal{F}^* \cdot w^*. \tag{4}$$

where $\mathsf{Cir}(f_l^*)$ is a circulant matrix corresponding to phrase template f_l^*, whose columns are shifted versions of f_l^* as shown in Fig 3(b), $\mathcal{F}^* := [\mathsf{Cir}(f_1^*), \mathsf{Cir}(f_2^*), \ldots, \mathsf{Cir}(f_L^*)]$ is the concatenation of circulant matrices and w^* is the stacked column vector $w^* := [w_1^*; w_2^*; \ldots; w_L^*] \in \mathbb{R}^{nL}$. 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 $i \in [L]$.

Under the convolution Independent Component Analysis (ICA) model, the third order cumulant tensor has a nice decomposition form [13]

$$C_3 = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* (\mathcal{F}_j^* \odot \mathcal{F}_j^*)^\top = \mathcal{F}^* \Lambda^* (\mathcal{F}^* \odot \mathcal{F}^*)^\top, \tag{5}$$

where λ_j^* is the third order cumulant corresponding to the (univariate) distribution of $w^*(j)$, $\Lambda^* := \mathsf{diag}(\lambda_1^*, \lambda_2^*, \ldots, \lambda_{nL}^*)$, and $\mathcal{F}_j^*$ denotes the j^{th} column of the *column-stacked* circulant matrix $\mathcal{F}^*$.

The third order cumulant C_3 is an unfolded third order tensor, which could be empirically estimated using the samples as follows

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

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

The decomposition form in (5) is known as the CANDECOMP/PARAFAC (CP) decomposition form [2] (the usual form has the decomposition of the tensor and not its unfolding, as above). We attempt to recover the unknown template phrases f_l^* through decomposition of the third order cumulants C_3.

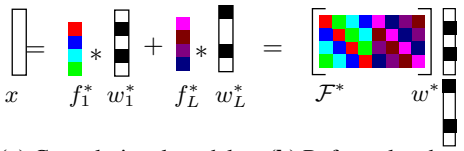
We propose *convolutional tensor decomposition* using efficient Alternating Least Square with Circulant Constraint (ALSwCC) to solve the non-convex optimization problem. Consider the asymmetric relaxation and introduce separate variables $\mathcal{F}, \mathcal{G}$ and $\mathcal{H}$ for filter estimates along each of the modes to fit the third order cumulant tensor C_3. ALSwCC iteratively alternates over the three variables and updates one mode by fixing the two other modes [13]

$$\begin{aligned} &\min_{\mathcal{F}} \quad \|C_3 - \mathcal{F}\Lambda\,(\mathcal{H} \odot \mathcal{G})^\top\|_F^2 \\ &\text{s.t. } \|f_l\|_2^2 = 1 \end{aligned} \tag{7}$$

where $\mathcal{F} = [blk_1(\mathcal{F}), \ldots, blk_L(\mathcal{F})] = [U\mathsf{diag}(\mathsf{FFT}(f_1))U^{\mathsf{H}}, \ldots, U\mathsf{diag}(\mathsf{FFT}(f_L))U^{\mathsf{H}}]$. U denotes the eigenvectors for circulant matrices.

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 patches in the same coordinate share a common set of phrase 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 to learn the activation maps for the individual maps as in the following section.



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

Figure 3: (a) The convolutional generative model with template phrases. (b) Reformulated multiplicative model where $\mathcal{F}^*$ is column-stacked circulant matrix of $f_1^*, \ldots, f_L^*$.

2.3 Feature-extraction Phase – Word-sequence Embeddings

After learning a good set of phrase templates for the j^{th} coordinate $\{f_1^{(j)}, \ldots, f_L^{(j)}\}$ and thus $\mathcal{F}^{(j)}$, we use the deconvolutional decoding (DeconvDec) to obtain the activation maps for the j^{th} coordinate. For a word-sequence $\mathbf{s}_i \in \mathbb{R}^{d \times N_i}$, we first project it onto the principle directions to get a compact representation $\mathbf{y}_i = \mathbf{U}^\top \mathbf{s}_i \in \mathbb{R}^{k \times N_i}$. For its j^{th} coordinate $y_i^{(j)}$, the activation map $\left(w^{(j)}\right)^*$ is the *stacked* vector $\left(w^{(j)}\right)^* := \left[\left(w_1^{(j)}\right)^*; \left(w_2^{(j)}\right)^*; \ldots \left(w_L^{(j)}\right)^*\right]$, where $\left(w_l^{(j)}\right)^*$ indicates the locations where template phrase $\left(f_l^{(j)}\right)^*$ is activated. An estimation of $\left(w_i^{(j)}\right)^*$ is achieved as follows

$$\widehat{w}_i^{(j)} = \left(\mathcal{F}^{(j)}\right)^\dagger {y_i^{(j)}}^\top. \tag{8}$$

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

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 solve this problem by *max-k pooling*. In other words, we extract most informative global discriminative features from the activation maps for each coordinate individually. Finally, we concatenate all the max-k pooled coordinate sequence embeddings as a long vector as the final word-sequence embedding.

The pseudo-code of our ConvDic+DeconvDec is in Algo 1 and Algo 2.

Algorithm 1 ConvDic

Input: Encoding matrix $\mathbf{S}$

Output: Phrase templates $\widehat{f}_l^{(j)}$, $\forall l \in [L]$, in all coordinates $\forall j \in [k]$

1: Compute top k left singular vectors $\mathbf{P}$ of $\mathbf{S}$
2: $\mathbf{Y} \leftarrow \mathbf{P}^\top \mathbf{S}$
3: **for** $j = 1$ **to** k **in parallel do**
4: Collect patches X=$[x^1, \ldots, x^N]$ of length n from $\mathbf{Y}^{(j)}$
5: $\widehat{f}_l^{(j)} \leftarrow$ Algo 3(X) , $\forall l \in [L]$
6: **end for**

3 Experiments

The performance of the convolutional tensor decomposition framework is first evaluated on simulated data, and compare against solving equation (2) using alternating minimization (AM) method where gradient descent is employed to update f_i and w_i alternatively. The performance is illustrated in [13]and in Appendix B. The convolutional tensor decomposition using ALSwCC method is both accurate and efficient compared to AM.

Algorithm 2 DeconvDec

Input: $\widehat{f}_l^{(j)}$, $\forall l \in [L]$ and word-sequence $\mathbf{s}_i \in \mathbb{R}^{d \times N_i}$
Output: Word sequence embedding $\widehat{w}_i$

Zero pad templates $f_l^{(j)} \leftarrow [\widehat{f}_l^{(j)}; \mathbf{0}] \in \mathbb{R}^{N_i}$
$\mathcal{F}^{(j)} \leftarrow [\mathsf{Cir}(f_1^{(j)}), \mathsf{Cir}(f_2^{(j)}), \ldots, \mathsf{Cir}(f_L^{(j)})]$
$[y_i^{(1)}; y_i^{(2)}; \ldots y_i^{(k)}] \leftarrow \mathbf{P}^\top \mathbf{s}_i$
$\widehat{w}_i \leftarrow \varnothing$
for $j = 1$ **to** k **in parallel do**
    $\widehat{w}_i \leftarrow [\widehat{w}_i; \text{max-}k \text{ pool}({\mathcal{F}^{(j)}}^\dagger {y_i^{(j)}}^\top)]$
end for

Algorithm 3 Convolutional Tensor Decomposition Algorithm via ALS update

Input: Samples $X = [x^1, \ldots, x^N]$. Define column-stacked identity matrix $\mathbf{I} := [I, \ldots, I] \in \mathbb{R}^{n \times nL}$, where I is $n \times n$ identity. Define block diagonal matrix $\mathbf{U} := Blkdiag(U, U, \ldots U) \in \mathbb{R}^{nL \times nL}$ with U along the diagonal.
Output: f_l, $\forall l \in [L]$.

Estimate $\hat{C}_3 \leftarrow X(X \odot X)^\top - \hat{Z}$
for $m = 1$ **to** n **in parallel do**
    $\Phi^{(m)} \leftarrow \mathbf{U}^{\mathsf{H}} \mathbf{I}^\top \Gamma^{(m)} \mathbf{I} \mathbf{U}$, where $[\Gamma^{(m)}]_j^i := [\hat{C}_3]_{i+(j-1)n}^m$, $\forall i, j \in [n]$
end for
Initialize random f_l, g_l, h_l, $\forall l \in [L]$
while not converged **do**
    $f_l' \leftarrow$ Algo 4($\Phi^{(m)}$, g_l,h_l)
    $g_l' \leftarrow$ Algo 4($\Phi^{(m)}$,h_l,f_l')
    $h_l' \leftarrow$ Algo 4($\Phi^{(m)}$,f_l',g_l')
    $f_l \leftarrow f_l'$,$g_l \leftarrow g_l'$,$h_l \leftarrow h_l'$
end while

Dataset	**Domain**	**Label**	**Label Distribution**	**Size** M
MR	Movie Reviews	{-1,1}	[0.49,0.51]	64720
SUBJ	Objective/Subjective 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 output	[0,5]	[0.01,0.00,0.00,0.02,0.19,0.78]	1193×2
STS-SMTnews	machine translation output	[0,5]	[0.00,0.01,0.01,0.06,0.19,0.73]	399×2

Table 1: Summary statistics of the datasets used.

Algorithm 4 One Mode Update

Input: $\Phi^{(m)}$, g_l and h_l $\forall l \in [L]$. Define $blk_l^j(\mathbf{\Psi})$ as the $(j,l)^{\text{th}}$ block of $\mathbf{\Psi} \in \mathbb{R}^{nL \times nL}$ and $blk_l(M)$ as the l^{th} block of $M \in \mathbb{R}^{n \times nL}$.

Output: $f_l, \forall l \in [L]$

$v \leftarrow \varnothing$, $z \leftarrow \varnothing$

for $l = 1$ **to** L **in parallel do**

  $v \leftarrow [v; \mathsf{FFT}(g_l)]$ and $z \leftarrow [z; \mathsf{FFT}(h_l)]$

end for

$blk_l^j(\mathbf{\Psi}) \leftarrow \mathsf{diag}^{\mathsf{H}}(\mathsf{FFT}(g_j))\mathsf{diag}^{\mathsf{H}}(\mathsf{FFT}(h_j))\ \mathsf{diag}(\mathsf{FFT}(g_l))\mathsf{diag}(\mathsf{FFT}(h_l))$.

$M \leftarrow \varnothing$

for row $m = 1$ **to** n **in parallel do**

  $M \leftarrow [M; \sum_{j \in [nL]} \mathbf{U}^j \mathsf{diag}^{\mathsf{H}}(z)\, \Phi^{(m)} \mathsf{diag}(v)\, (\mathbf{U}^j)^{\mathsf{H}} \mathbf{U}^j \mathbf{\Psi}^{\dagger} \mathbf{U}^{\mathsf{H}}]$

end for

$f_l \leftarrow \varnothing$

for $p = 1$ **to** n **in parallel do**

  $f_l \leftarrow \left[f_l; \frac{\sum_{i,j \in [n]} \|blk_l(M)_j\|^{-1} \cdot blk_l(M)_j^i \cdot I_{p-1}^q}{\sum_{i,j \in [n]} I_{p-1}^q} \right]$, where $q = (i-j) \mod n$

end for

We evaluate the quality of our word sequence embeddings using three challenging natural language process tasks: sentiment classification, paraphrase detection and semantic test similarity estimation. Eight datasets which cover various domains are used as shown in Table 1. We emphasize that our method requires no pre-training on rich or consecutive corpus in contrast with Skip-thought. This is extremely useful when learning word sequence embeddings for domains that are not trained previously. Although we exhibit the power of our algorithm by training on small set of training samples, our algorithm has the potential to learn universal templates if trained on large corpora.

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.,

$$F = 2 \cdot \frac{(\text{precision} \cdot \text{recall})}{\text{precision} + \text{recall}}. \tag{9}$$

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 actually 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 [20].

[3]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.

Method	MR	SUBJ
NB-SVM [33]	79.4	93.2
MNB [33]	79.0	93.6
cBoW [37]	77.2	91.3
GrConv [37]	76.3	89.5
RNN [37]	77.2	93.7
BRNN [37]	82.3	94.2
CNN [19]	81.5	93.4
AdaSent [37]	83.1	95.5
Paragraph-vector [21]	74.8	90.5
Skip-thought [20]	75.5	92.1
ConvDic+DeconvDec	**78.9**	**92.4**

Table 2: Sentiment analysis: Classification accuracies on standard benchmarks on movie review and subject dataset. The first group contains results using bag-of-words models; the second group exhibits some supervised compositional models.

Method	Outside Info [3]	F score
Vector Similarity [22]	word similarity	0.753
ESA [12]	word semantic profiles	0.793
LSA [12]	word semantic profiles	0.799
RMLMG [29]	syntactic info	0.805
ConvDic+DeconvDec	**none**	**0.807**
Skip-thought [20]	train on large book corpus	0.819

Table 3: Paraphrase detection: 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 embedding from scratch.

3.1 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 dataset from different domains, movie review and subjective/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.

3.2 Evaluation Task: Paraphrase Detection

The challenging *paraphrase detection* task is implemented using Microsoft paraphrase corpus [28, 8] to evaluate the quality of our sentence embeddings. We consider the *paraphrase detection* task on the Microsoft paraphrase corpus [28, 8]. 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 [32], 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

Dataset	DAN	RNN	LSTM	Siamese CBOW	Skip-thought	**ConvDic+ DeconvDec**
MSRpar	40.3	18.6	9.3	43.8	16.8	**36.0**
MSRvid	70.0	66.5	71.3	45.2	41.7	**61.8**
SMT-eur	43.8	40.9	44.3	45.0	35.2	**37.5**
OnWN	65.9	63.1	56.4	64.4	29.7	**33.1**
SMT-news	60.0	51.3	51.0	39.0	30.8	**72.1**

Table 4: STS: Pearson's $r \times 100$ on MSRpar, MSRvid, OnWN, SMTeuroparl and SMTnews dataset results. The first 3 columns (supervised methods) are reported by Wieting et al. Our comparison against the state-of-the-art unsupervised method is in the last two columns.

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 [35] as in table 3. We show some examples of paraphrase and non-paraphrase we identified in 4076 pairs of sentences.

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.

3.3 Evaluation Task: Semantic Test Similarity Estimation

For the Semantic Test Similarity (STS) task, our goal is to predict a real-valued similarity score in range [1,K] given a sentence pair. We include datasets from STS task in various different domains including news, image and video description, glosses from WordNet/OntoNotes, 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 [32], $p_i = \lfloor \tau \rfloor - \tau + 1$ 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 [32]. 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 [34], Pearson's r are showed. We then compare our method against the performance of supervised models in [34]: PARAGRAM-PHRASE (PP), projection (proj.), deep-averaging network (DAN) [14], recurrent neural network (RNN) and LSTM; as well as the state-of-the-art

unsupervised model skip-thought vectors [20].

As we can see from the table, LSTM is performing poorly even though a back-propagation after seeing the training labeling is carried out for sequence embedding learning. Optimizing over word embeddings (Siamese CBOW) performs better in some of these datasets [18]. Our method is an unsupervised approach as in skip-thought vectors. However, our algorithm outputs domain specific word-sequence embeddings as we only train the model on the training samples from that domain. In other words, we train a fresh model and a new set of domain phrase templates from scratch, although our method is amendable to training on large and rich corpus to get universal phrase templates and thus embeddings. Therefore our algorithm is performing better for these individual datasets on the STS task.

4 Conclusion

Our unsupervised ConvDic+DeconvDec efficiently 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. Although we exhibit the power of our algorithm by training on small set of training samples, our algorithm has the potential to learn universal templates if trained on large corpora. Whereas Skip-thought requires pre-training on consecutive sentences, our ConvDic+DeconvDec requires no ordering among sentences and still learn rich set of phrase templates as long as the corpus contains rich contents. 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

[1] Anima Anandkumar, Rong Ge, and Majid Janzamin. Guaranteed non-orthogonal tensor decomposition via alternating rank-1 updates. *arXiv preprint arXiv:1402.5180*, 2014.

[2] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.

[10] Maria Gabriela Eberle and Maria Cristina Maciel. Finding the closest toeplitz matrix. *Computational & Applied Mathematics*, 22(1):1–18, 2003.

[11] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

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

[13] Furong Huang and Animashree Anandkumar. Convolutional dictionary learning through tensor factorization. In *Proceedings of The 1st International Workshop on Feature Extraction: Modern Questions and Challenges, NIPS*, pages 116–129, 2015.

[14] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the Association for Computational Linguistics*, 2015.

[15] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*, 2014.

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

[17] 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, 2014.

[18] Tom Kenter, Alexey Borisov, and Maarten de Rijke. Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv preprint arXiv:1606.04640*, 2016.

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

[20] 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.

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

[22] 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.

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

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

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

[26] Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural language inference by tree-based convolution and heuristic matching. In *The 54th Annual Meeting of the Association for Computational Linguistics*, volume 130, 2016.

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

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] ACL Wiki. Paraphrase identification (state of the art), 2014.

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

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

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

A Convolutional Tensor Decomposition For Learning Convolutional Dictionary Model

A.1 Cumulant Form

Let $C_3 \in \mathbb{R}^{n \times n^2}$ denote the unfolded version of third order cumulant tensor, it is given by [13]

$$C_3 := \mathbb{E}[x(x \odot x)^\top] - unfold(Z) \tag{10}$$

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

Objective Function: [13] 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,

$$\begin{aligned} &\min_{\mathcal{F}} \quad \|C_3 - \mathcal{F}\Lambda\,(\mathcal{F} \odot \mathcal{F})^\top\|_F^2, \\ &\text{s.t. } blk_l(\mathcal{F}) = U\mathsf{diag}(\mathsf{FFT}(f_l))U^{\mathsf{H}},\ \|f_l\|_2 = 1, \quad \forall l \in [L], \quad \Lambda = \mathsf{diag}(\lambda). \end{aligned} \tag{11}$$

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 (10) 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 (10).

To solve the non-convex optimization problem in (10), we propose the alternating least squares with circulant constraints (ALSwCC) method with *column stacked* circulant constraint. We first consider the asymmetric relaxation of (10) 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,

$$\min_{\mathcal{F}} \quad \|C_3 - \mathcal{F}\Lambda\,(\mathcal{H} \odot \mathcal{G})^\top\|_F^2 \text{ s.t. } blk_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l \in [L] \tag{12}$$

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 (11) 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 (11) is equivalent to

$$\min_{\mathcal{F}} \|C_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger \Lambda^\dagger - \mathcal{F}\|_F^2 \text{ s.t. } blk_l(\mathcal{F}) = U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}},\ \|f_l\|_2^2 = 1, \forall l \in [L] \tag{13}$$

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

$$M := C_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger, \tag{14}$$

where † denotes pseudoinverse. Let $blk_l(M)$ and $blk_l(\Lambda)$ denote the l^{th} blocks of M and Λ.Since (12) has block constraints, it can be broken down in to solving L independent sub-problems

$$\min_{f_l} \left\| blk_l(M) \cdot blk_l(\Lambda)^\dagger - U \cdot \mathsf{diag}(\mathsf{FFT}(f_l)) \cdot U^{\mathsf{H}} \right\|_F^2 \quad s.t. \quad \|f_l\|_2^2 = 1, \forall l \in [L] \tag{15}$$

We present the main result now.

Theorem A.1 (Closed form updates)**.** *The optimal solution f_l^{opt} for* (14) *is given by*

$$f_l^{opt}(p) = \frac{\sum\limits_{i,j \in [n]} \|blk_l(M)_j\|^{-1} \cdot blk_l(M)_j^i \cdot I_{p-1}^q}{\sum\limits_{i,j \in [n]} I_{p-1}^q}, \quad \forall p \in [n], q := (i-j) \mod n. \tag{16}$$

Further $\Lambda = \mathsf{diag}(\lambda)$ is updated as $\lambda(i) = \|M_i\|$, for all $i \in [nL]$.

If $\mathsf{Cir}(f_l)$ has linear independent columns, previous method provably recovers the filter [1]. The blind deconvolution problem however cannot achieve this identifiability.

Thus, the reformulated problem in (14) can be solved in closed form efficiently [10]. A bulk of the computational effort will go into computing M in (13). Computation of M requires $2L$ fast Fourier Transforms of length n filters and simple matrix multiplications without explicitly forming $\mathcal{G}$ or $\mathcal{H}$. We make this concrete in the next section. The closed form update after getting M is highly parallel. With $O(n^2 L / \log n)$ processors, it takes $O(\log n)$ time. In order to establish the equivalence between the original ALS update and the reformulated problem in (14), we require that $(\mathcal{H} \odot \mathcal{G}) \in \mathbb{R}^{n^2 \times nL}$ and $\Lambda \in \mathbb{R}^{nL \times nL}$ be full column rank. This requires $nL < n^2$ or $L < n$, 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.

The alternating least square update for the above problem requires $O(L)$ time with $O(n^2)$ processes and $2L$ fast Fourier Transforms per iteration.

Remark. (1)*We require $L < n$ for $(\mathcal{H} \odot \mathcal{G})$ to be full column rank. This is a mild condition as requiring non-degeneracy on the filters are reasonable, otherwise one filter can be expressed through other filters.* **(2)** *Although we decompose the optimization problem into L sub ones for notation simplicity, the algorithm consider the stacked version.*

A.3 Algorithm Optimization for Reducing Memory and Computational Costs

We now focus on estimating $M := C_3((\mathcal{H} \odot \mathcal{G})^\top)^\dagger$ in (13). If done naively, this requires inverting $n^2 \times nL$ matrix and multiplication of $n \times n^2$ and $n^2 \times nL$ matrices with $O(n^6)$ time. However, forming and computing with these matrices is very expensive when n (and L) are large. Instead, we utilize the properties of circulant matrices and the Khatri-Rao product $\odot$ to efficiently carry out these computations implicitly. We present our final result on computational complexity of the proposed method.

Lemma A.2 (Computational Complexity). *With multi-threading, the running time is $O(\log n + \log L)$ per iteration using $O(L^2 n^3)$ processes.*

We now describe how we utilize various algebraic structures to obtain efficient computation.

Property 1 (Khatri-Rao product): $((\mathcal{H} \odot \mathcal{G})^\top)^\dagger = (\mathcal{H} \odot \mathcal{G})((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger$, where .$\star$ denotes element-wise product.

Computational Goals: Find $((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger$ first and multiply the result with $C_3(\mathcal{H} \odot \mathcal{G})$ to find M. We now describe in detail how to carry out each of these steps.

A.3.1 Challenge: Computing $((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger$

A naive implementation to find the matrix inversion $((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger$ is very expensive. However, we incorporate the stacked circulant structure of $\mathcal{G}$ and $\mathcal{H}$ to reduce computation. Note that this is not completely straightforward since although $\mathcal{G}$ and $\mathcal{H}$ are column stacked circulant matrices, the resulting product whose inverse is required, is *not* circulant. Below, we show that however, it is partially circulant along different rows and columns.

Property 2 (Block circulant matrix):The matrix $(\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G})$ consists of row and column stacked circulant matrices.

We now make the above property precise by introducing some new notation. Define column stacked identity matrix $\mathbf{I} := [I, \ldots, I] \in \mathbb{R}^{n \times nL}$, where I is $n \times n$ identity matrix. Let $\mathbf{U} := Blkdiag(U, U, \ldots U) \in \mathbb{R}^{nL \times nL}$ be the block diagonal matrix with U along the diagonal. The first thing to note is that $\mathcal{G}$ and $\mathcal{H}$, which are column stacked circulant matrices, can be written as

$$\mathcal{G} = \mathbf{I} \cdot \mathbf{U} \cdot \mathsf{diag}(v) \cdot \mathbf{U}^{\mathsf{H}}, \quad v := [\mathsf{FFT}(g_1); \mathsf{FFT}(g_2); \ldots; \mathsf{FFT}(g_L)], \qquad (17)$$

where $g_1, \ldots, g_L$ are the filters corresponding to $\mathcal{G}$, and similarly for $\mathcal{H}$, where the diagonal matrix consists of FFT coefficients of the respective filters $h_1, \ldots, h_L$.

By appealing to the above form, we have the following result. We use the notation $blk^i_j(\mathbf{\Psi})$ for a matrix $\mathbf{\Psi} \in \mathbb{R}^{nL \times nL}$ to denote $(i,j)^{\text{th}}$ block of size $n \times n$.

Lemma A.3 (Form of $(\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G})$)**.** *We have*

$$((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger = \mathbf{U} \cdot \mathbf{\Psi}^\dagger \cdot \mathbf{U}^{\mathsf{H}}, \tag{18}$$

where $\mathbf{\Psi} \in \mathbb{R}^{nL \times nL}$ has L by L blocks, each block of size $n \times n$. Its $(j,l)^{th}$ block is given by

$$blk^j_l(\mathbf{\Psi}) = \mathsf{diag}^{\mathsf{H}}(\mathsf{FFT}(g_j)) \cdot \mathsf{diag}^{\mathsf{H}}(\mathsf{FFT}(h_j)) \cdot \mathsf{diag}(\mathsf{FFT}(g_l)) \cdot \mathsf{diag}(\mathsf{FFT}(h_l)) \in \mathbb{R}^{n \times n} \tag{19}$$

Therefore, the inversion of $(\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G})$ can be reduced to the inversion of row-and-column stacked set of diagonal matrices which form $\mathbf{\Psi}$. Computing $\mathbf{\Psi}$ simply requires FFT on all $2L$ filters $g_1, \ldots, g_L$ and $h_1, \ldots, h_L$, i.e. $2L$ FFTs, each on length n vector. We propose an efficient iterative algorithm to compute $\mathbf{\Psi}^\dagger$ via block matrix inversion theorem[11] in Appendix D.

A.3.2 Challenge: Computing $M = C_3(\mathcal{H} \odot \mathcal{G}) \cdot ((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger$

Now that we have computed $((\mathcal{H}^\top \mathcal{H}). \star (\mathcal{G}^\top \mathcal{G}))^\dagger$ efficiently, we need to compute the resulting matrix with $C_3(\mathcal{H} \odot \mathcal{G})$ to obtain M. We observe that the m^{th} row of the result M is given by

$$M^m = \sum_{j \in [nL]} \mathbf{U}^j \mathsf{diag}^{\mathsf{H}}(z)\, \Phi^{(m)} \mathsf{diag}(v)\, (\mathbf{U}^j)^{\mathsf{H}} \mathbf{U}^j \mathbf{\Psi}^\dagger \mathbf{U}^{\mathsf{H}}, \quad \forall m \in [nL], \tag{20}$$

where $v := [\mathsf{FFT}(g_1); \ldots; \mathsf{FFT}(g_L)]$, $z := [\mathsf{FFT}(h_1); \ldots; \mathsf{FFT}(h_L)]$ are concatenated FFT coefficients of the filters, and

$$\Phi^{(m)} := \mathbf{U}^{\mathsf{H}} \mathbf{I}^\top \Gamma^{(m)} \mathbf{I} \mathbf{U}, \quad [\Gamma^{(m)}]^i_j := [C_3]^m_{i+(j-1)n}, \quad \forall i,j,m \in [n] \tag{21}$$

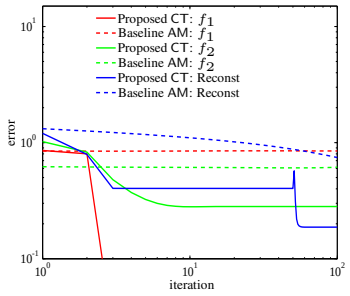
Note that $\Phi^{(m)}$ and $\Gamma^{(m)}$ are fixed for all iterations and need to be computed only once. Note that $\Gamma^{(m)}$ is the result of taking m^{th} row of the cumulant unfolding C_3 and matricizing it. Equation (19) uses the property that $C_3^m(\mathcal{H} \odot \mathcal{G})$ is equal to the diagonal elments of $\mathcal{H}^\top \Gamma^{(m)} \mathcal{G}$.

We now bound the cost for computing (19). (1) Inverting $\mathbf{\Psi}$ takes $O(\log L + \log n)$ time with $O(n^2L^2/(\log n + \log L))$ processors according to appendix D. (2) Since $\mathsf{diag}(v)$ and $\mathsf{diag}(z)$ are diagonal and $\mathbf{\Psi}$ is a matrix with diagonal blocks, the overall matrix multiplication in equation (19) takes $O(L^2n^2)$ time serially with $O(L^2n^2)$ degree of parallelism for each row. Therefore the overall serial computation cost is $O(L^2n^3)$ with $O(L^2n^3)$ degree of parallelism. With multi-threading, the running time is $O(1)$ per iteration using $O(L^2n^3)$ processes. (3) FFT requires $O(n \log n)$ serial time, with $O(n)$ degree of parallelism. Therefore computing $2L$ FFT's takes $O(\log n)$ time with $O(Ln)$ processors.

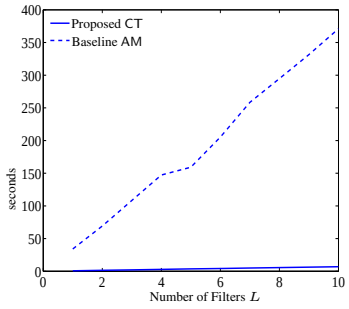
Combining the above discussion, it takes $O(\log L + \log n)$ time with $O(L^2n^3)$ processors.

B Synthetic Experiments

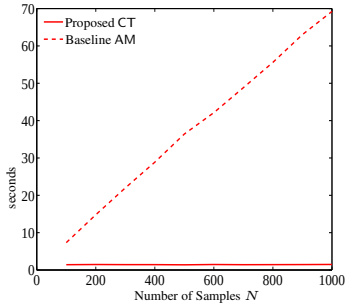
We compare our convolutional tensor decomposition framework with solving equation (2) using alternating minimization method where gradient descent is employed to update f_i and w_i alternatively. The error comparison between our proposed convolutional tensor algorithm and the alternate minimization algorithm is in figure 4(a). We evaluate the errors for both algorithms by comparing the reconstruction of error and template recovery error. Our algorithm converges much faster to the solution than the alternate minimization algorithm. In fact, the alternate minimization leads to spurious solution where the reconstruction error decreases but template estimation error increases. The running time is also reported in figure 4(b)(c) between our proposed convolutional tensor algorithm and the alternate minimization. Our algorithm is orders of magnitude faster than the alternate minimization. Both our algorithm and alternate minimization scale linearly with number of templates. However convolutional tensor algorithm scales constantly with the number of samples whereas the alternate minimization scales linearly.



(a) Reconstruction Error



(b) Running Times Scale with L



(c) Running Times Scale with N

Figure 4: (a) Error comparison (on filters and reconstruct tensor) between our convolutional tensor method (proposed CT) and the baseline alternate minimization method (baseline AM) on synthetic data. (b)(C)Running time comparison between our proposed CT method and the baseline AM method on synthetic data.

C Cumulant Form

In [2], it is proved that in ICA model, the cumulant of observation x is decomposed into multi-linear transform of a diagonal cumulant of h. Therefore, we aim to find the third order cumulant for input x.

As we know that the r^{th} order moments for variable x is defined as

$$\mu_r := \mathbb{E}[x^r] \in \mathbb{R}^{n \times n \times n} \tag{22}$$

Let us use $[\mu_3]_{i,j,k}$ to denote the $(i,j,k)^{\text{th}}$ entry of the third order moment. The relationship between 3^{th} order cumulant $\kappa_3$and 3^{th} order moment $\mu_3$is

$$[\kappa_3]_{i,j,k} = [\mu_3]_{i,j,k} - [\mu_2]_{i,j}[\mu_1]_k - [\mu_2]_{i,k}[\mu_1]_j - [\mu_2]_{j,k}[\mu_1]_i + 2[\mu_1]_i[\mu_1]_j[\mu_1]_k \tag{23}$$

Therefore the shift tensor is in this format: We know that the shift term

$$[Z]_{a,b,c} := \mathbb{E}[x_a^i]\mathbb{E}[x_b^i x_c^i] + \mathbb{E}[x_b]\mathbb{E}[x_a x_c^i] + \mathbb{E}[x_c]\mathbb{E}[x_a x_b] - 2\mathbb{E}[x_a]\mathbb{E}[x_b]\mathbb{E}[x_c], \quad a,b,c \in [n] \tag{24}$$

It is known from [2] that cumulant decomposition in the 3 order tensor format is

$$\mathbb{E}[x \otimes x \otimes x] - Z = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* \otimes \mathcal{F}_j^* \otimes \mathcal{F}_j^* \tag{25}$$

Therefore using the Khatri-Rao product property,

$$unfold(\sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* \otimes \mathcal{F}_j^* \otimes \mathcal{F}_j^*) = \sum_{j \in [nL]} \lambda_j^* \mathcal{F}_j^* (\mathcal{F}_j^* \odot \mathcal{F}_j^*)^\top = \mathcal{F}^* \Lambda^* \left(\mathcal{F}^* \odot \mathcal{F}^*\right)^\top \tag{26}$$

Therefore the unfolded third order cumulant is decomposed as $C_3 = \mathcal{F}^* \Lambda^* \left(\mathcal{F}^* \odot \mathcal{F}^*\right)^\top$.

D Parallel Inversion of Ψ

We propose an efficient iterative algorithm to compute $\mathbf{\Psi}^\dagger$ via block matrix inversion theorem [11].

Lemma D.1. *(Parallel Inversion of row and column stacked diagonal matrix) Let $J^L = \mathbf{\Psi}$ be partitioned into a block form:*

$$J^L = \begin{bmatrix} J^{L-1} & O \\ R & blk_L^L(\mathbf{\Psi}) \end{bmatrix}, \tag{27}$$

where $O := \begin{bmatrix} blk_L^1(\mathbf{\Psi}) \\ \vdots \\ blk_L^{L-1}(\mathbf{\Psi}) \end{bmatrix}$, and $R := \left[blk_{L-1}^1(\mathbf{\Psi}), \ldots, blk_{L-1}^L(\mathbf{\Psi})\right]$. After inverting $blk_L^L(\mathbf{\Psi})$ which takes $O(1)$ time using $O(n)$ processors, there inverse of $\mathbf{\Psi}$ is

achieved by

$$\mathbf{\Psi}^{\dagger} = \begin{bmatrix} (J^{L-1} - Oblk_L^L(\mathbf{\Psi})^{-1}R)^{-1} & -(J^{L-1})^{-1}O(blk_L^L(\mathbf{\Psi}) - R(J^{L-1})^{-1}O)^{-1} \\ -blk_L^L(\mathbf{\Psi})^{-1}R(J^{L-1} - Oblk_L^L(\mathbf{\Psi})^{-1}R)^{-1} & (blk_L^L(\mathbf{\Psi}) - R(J^{L-1})^{-1}O)^{-1} \end{bmatrix} \tag{28}$$

assuming that J^{L-1} and $blk_L^L\mathbf{\Psi}$ are invertible.

This again requires inverting R, O and J^{L-1}. Recursively applying these block matrix inversion theorem, the inversion problem is reduced to inverting L^2 number of n by n diagonal matrices with additional matrix multiplications as indicated in equation (27).

Inverting a diagonal matrix results in another diagonal one, and the complexity of inverting $n \times n$ diagonal matrix is $O(1)$ with $O(n)$ processors. We can simultaneous invert all blocks. Therefore with $O(nL^2)$ processors, we invert all the diagonal matrices in $O(1)$ time. The recursion takes L steps, for step $i \in [L]$ matrix multiplication cost is O($\log nL$) with $O(n^2L/\log(nL))$ processors. With L iteration, one achieves $O(\log n + \log L)$ running time with $O(n^2L^2/(\log L + \log n))$ processors.

E Local Objects – Filters in Computer Vision

We offer some details of transition invariance in computer vision as a background for better understanding of the invariance in natural language processing. Consider a simple image where there are stars, squares and circles as in Figure 5. Convolution is the key to model the images as common patterns across images and the activation of the patterns for individual image.

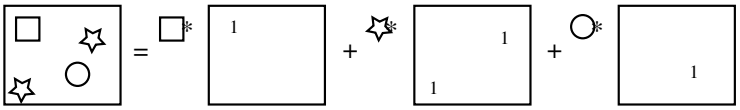


Figure 5: Convolutional generative model for an image.

The convolutional generative model posits that an image is generated through a linear combination of filters (squares, stars and circles) convolved with their corresponding activation maps. The activation map, which is usually sparse, encodes the location of the filters.

Now consider three images that contain squares and stars as in Figure 6. The activation maps corresponding to square and star filters are depicted for each of the three images. For each image, there are activation maps (across filters) which serve as the informative discriminative features. As we see in Figure 6, both image 1 and image 3 contain non-zero elements only in activation maps for the star filter, whereas image 2 contains non-zero elements only in activation map for the square filter. Therefore, a max-pooling layer as in Figure 6 or a max-3-pooling layer as in Figure 7 is helpful to guarantee location invariance in the discriminative features.

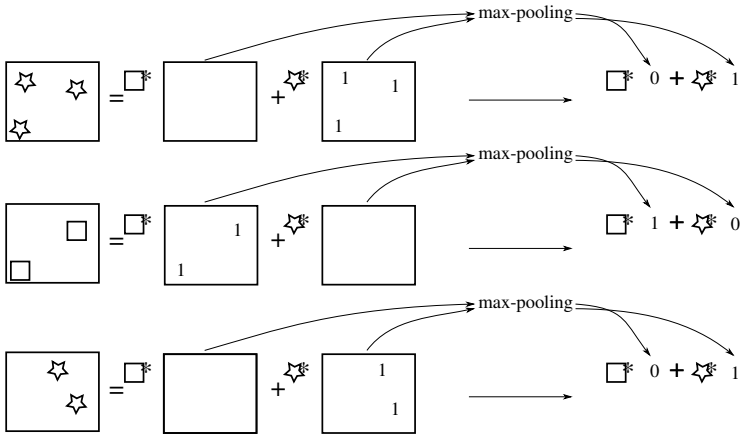


Figure 6: The activation maps for image 1,2 and 3. Max-pooling of the activation map results in discriminative features that ignore the number of times each filter is activated in the image. Image 1 and 3 are classified as one category and image 2 as another category.

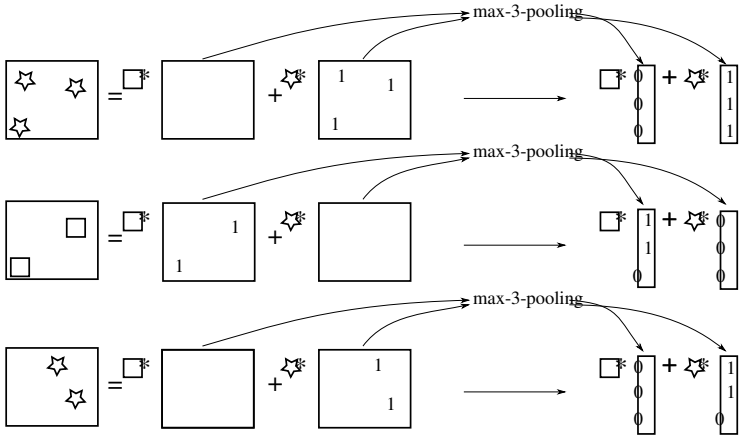


Figure 7: Max-3-pooling of the activation map results in discriminative features which contain the number of times (saturated at 3) each filter is activated in the image.

Therefore, image classification is the process of finding good set of filters and their corresponding discriminative features. CNNs encode additional non-linearity and hierarchical structure.