Capturing and evaluating higher order relations in word embeddings using tensor factorization

A thesis submitted by

Eric Bailey

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Tufts University

May 2017

Adviser: Dr. Shuchin Aeron

Abstract

In Natural Language Processing, most popular word embeddings involve low-rank factorization of a word co-occurrence based matrix. We aim to generalize this trend by studying word embeddings given by low-rank factorization of word co-occurrence based higher-order arrays, or *tensors*. We present four novel word embeddings based on tensor factorization and show they outperform popular state-of-the-art baselines on a number of recent benchmarks, encoding useful properties in a new way. To create one of our word embeddings, we present a novel joint symmetric tensor factorization problem related to the idea of *coupled* tensor factorization. We also modify a recent embedding evaluation technique known as Outlier Detection to measure the degree to which an embedding captures N^{th} order information, showing that tensor embeddings (naturally) outperform popular pairwise embeddings at this task. Suggested applications of tensor factorization-based word embeddings are given, and all source code and pre-trained vectors are publicly available online.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

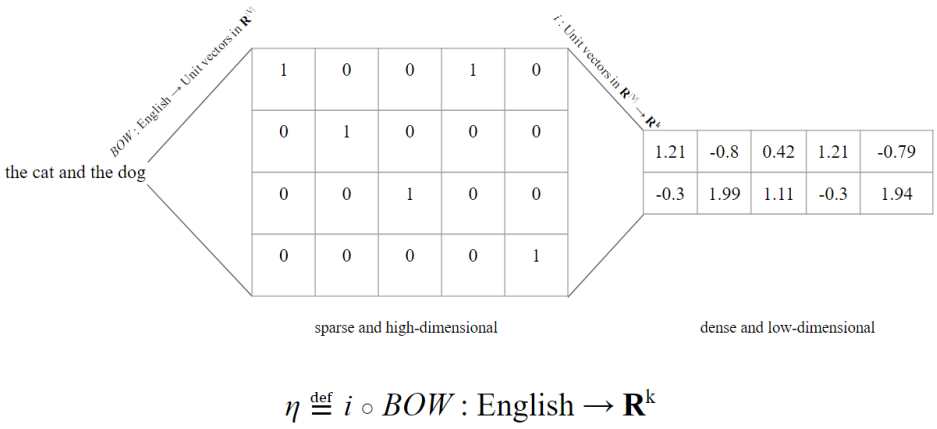
1.1 Word embeddings

Word embeddings are an important part of NLP, and can be used in almost every NLP task. They have been shown to improve the performance of many tasks, from language modelling [Bengio et al., 2003] to Machine Translation [Bahdanau et al., 2014] to sentiment analysis [Kim, 2014]. Their broad applicability to almost every problem in NLP and ease of training makes them a hot area of study, especially since [Mikolov et al., 2013b] recently showed great results on quickly extracting information on massive amounts of data with their `word2vec` model.

The word embedding problem is to learn a function η mapping words in a language to $\mathbb{R}^k$ ($k \approx$ 100-300 in most applications) that encodes meaningful semantic and/or syntactic information[1]. For example, typically words mean similar things if and only if their corresponding vectors appear nearby in the vector space. So for most modern word embeddings, $\eta(\text{car}) \approx \eta(\text{automobile})$, since the words mean similar things.

Models often encode many different types of semantic information. For instance, in the `word2vec` model [Mikolov et al., 2013b], we can answer analogy queries of the form $a : b :: c : ?$ using simple vector arithmetic: the answer to bed

[1] In NLP, *Semantic information* is the information related to intrinsic meaning. For example, the fact that *cat* and *dog* mean similar things (but not the same thing) is semantic information. *Syntactic information* is the information about the set of rules that govern the structure of sentences in a language. For example, the fact that *dog* is a noun (and that verbs can have subjects/objects that are nouns) is syntactic information.



$$\eta \stackrel{\text{def}}{=} i \circ BOW : \text{English} \rightarrow \mathbf{R}^k$$

Figure 1.1: Toy example encoding using BoW with $|V| = 4$ and $k = 2$

: sleep :: chair : x is given by the vector closest to $\eta(\text{sleep}) - \eta(\text{bed}) + \eta(\text{chair})$ ($\approx \eta(\text{sit})$). However, different word embeddings encode different types of information, or may encode such information in a nonlinear way [Jastrzebski et al., 2017].

η is typically learned by first creating a finite vocabulary V and then mapping words to a one-of-$|V|$ encoding (also known as a Bag of Words encoding). Then, the index of each word is used to index into a $|V| \times k$ matrix, where the i^{th} row of the matrix holds the vector corresponding to the i^{th} word in the vocabulary. So the word embedding problem can be (and often is) solved by finding a $|V| \times k$ matrix with rows that encode meaningful information about the corresponding vocabulary word. $|V|$ typically ranges from around 20,000 to 300,000 in production-ready word embeddings. This process is visualized in Figure 1.1.

Word embeddings are quite advantageous for many machine learning algorithms, which require a fixed-length vector representation of the input such as Logistic Regression or simple Artificial Neural Networks. For example, sentiment analysis of a full sentence can easily be solved by first mapping each word into is fixed-length vector representation and then summing each of the word vectors in the sentences elementwise. Then the entire sentence has a vector representation in $\mathbb{R}^k$ which can be fed into any off-the-shelf classification algorithm.

Word embedding can be applied to other areas of computer science – graph embeddings can be given via word embedding techniques [Grover and Leskovec, 2016]. Other fields of CS have also used ideas from word embeddings; for example, computational biology (dna2vec) [Ng, 2017], 3D Graphics (shape2vec) [Tasse and Dodgson, 2016], and Item Recommendation (item2vec) [Barkan and Koenigstein, 2016] all have contributions using word embeddings[2]. Thus, even incremental improvements on word embedding architectures may lead to further improvements across many areas.

Almost all of the recent word embeddings rely on the distributional hypothesis [Harris, 1954], which states that a word's meaning can be inferred from the words that surround it. For example, one can infer that the x in "he ate two full servings of x for breakfast" is something that is supposed to be eaten for breakfast. For instance, "cereal" and "granola" can reasonably be substituted for x, and we see that they mean similar things.

In this work we work with the distributional hypothesis, but work with higher-order co-occurrence information. Next, we will outline our main contributions.

1.2 Summary of main contributions

Our main contributions are fourfold:

1. **Four novel tensor-based word embeddings.** We apply the theory of tensor factorization to training generic word embeddings, studying many different types of tensor factorization that utilize different aspects of tensor structure – supersymmetry, sparsity, and nonnegativity. We call our new embeddings **CP, CP-S, CP-SN, JCP-S**. We motivate intuition behind some new properties these embeddings encode in Section 1.4 and show that they actually encode such properties in Section 4.4. We show experimentally that sparse embeddings can encode *more* (or encode information more readily) than dense embeddings. This suggests that sparse embeddings (which tend

[2]A list of many of the "*2vec"s can be found online: `https://gist.github.com/nzw0301/333afc00bd508501268fa7bf40cafe4e`

to be overlooked in the deep learning/NLP communities) should be considered and further studied.

Suggested applications for our embeddings are outlined in Section 4.6.

2. **A novel joint tensor factorization technique for simultaneously decomposing N supersymmetric tensors.** Joint Tensor Factorization (or Coupled Tensor Factorization [Acar et al., 2011, Naskovska and Haardt, 2016]) is a problem in which multiple tensors are being factorized sharing at least one factor matrix. We introduce and utilize a new joint symmetric tensor factorization problem in order to create a novel word embedding, which captures multiple different orders of relations between words. To the best of our knowledge, the special case of joint supersymmetric tensor decomposition has not been considered so far in the literature.

3. **New embedding evaluation metric to measure degree of capturing N^{th}-order information.** We modify the idea of Outlier Detection for evaluating word embeddings [Camacho-Collados and Navigli, 2016] to produce an N-way analog of this metric we call Outlier Detection (N) (ODN). This metric evaluates to what degree N-way information is captured by a given word embedding. We demonstrate that our third order models outperform state-of-the-art baselines (including `word2vec` [Mikolov et al., 2013b]) on OD2 and OD3.

4. **General-purpose TensorFlow framework for computing online CP decomposition.** While implementing our embeddings in TensorFlow, we found it useful to create a general framework for computing online CP Decomposition. To the best of our knowledge, such a library does not yet exist for TensorFlow, so we release it to the general public included in our word embedding code[3]. We have implemented asymmetric, symmetric, sparse, and joint online CP decomposition, including both GPU and CPU support.

[3] `https://github.com/popcorncolonel/tensor_decomp_embedding/blob/master/tensor_decomp.py`

1.3 Mathematical Background

In this section we describe the mathematical notation used in this work and outline the mathematical background required to understand our methodology.

1.3.1 Pointwise Mutual Information

Pointwise mutual information (PMI) is a measure very useful in NLP. We discuss its utility in Section 1.4 and we give some of its past uses in NLP in Section 2.3.

PMI is an extension of the notion of mutual information between two random variables [Cover and Thomas, 2006]. For two random variable X, Y, the Mutual Information between X and Y is given by:

$$I(X;Y) = \mathbb{E}\left[\log \frac{p(X,Y)}{p(X)p(Y)}\right]$$

where $\mathbb{E}$ is the expectation of a random variable. PMI is defined mathematically as

$$PMI(x,y) = \log \frac{p(x,y)}{p(x)p(y)}$$

PMI measures the degree of correlation between its arguments – it directly compares the *statistical dependence* between x and y (given by the numerator $p(x,y)$) vs. the *statistical independence* of x and y (given by the denominator $p(x)p(y)$).

Often in literature working with PMI, it is useful to observe the *positive* PMI (PPMI), defined as

$$PPMI(x,y) := \max(0, PMI(x,y))$$

[Bullinaria and Levy, 2007, Levy and Goldberg, 2014, Van de Cruys, 2009], as negative PMI values have little grounded interpretation.

Given an indexed finite set V, it is often useful to construct a $|V| \times |V|$ PPMI matrix $\mathbf{M}$ where $m_{ij} = PPMI(V_i, V_j)$. Many word embedding models (and more generally, problems in NLP) involve factorizing this PPMI matrix. For

example, [Levy and Goldberg, 2014] not only showed that `word2vec` *implicitly* factors a PMI-related matrix, but also *explicitly* factorized a shifted version of this PPMI matrix to produce a word embedding – see Section 1.4 for a more explicit formulation.

PMI can be generalized to N variables. There are multiple different ways to formulate N-way PPMI [Van de Cruys, 2011]. One is given (on three variables) by:

$$\begin{aligned} PMI_1(x,y,z) &= \log \frac{p(x,y)}{p(x)p(y)} - \log \frac{p(z)p(x,y,z)}{p(x,z)p(y,z)} \\ &= \log \frac{p(x,y)p(x,z)p(y,z)}{p(x,y,z)p(x)p(y)p(z)} \end{aligned}$$

And another given by:

$$PMI_2(x_1^N) = \log \frac{p(x_1,\ldots,x_n)}{p(x_1)\cdots p(x_N)}$$

At least based on notational similarity, we believe PMI_2 to be the more natural generalization of PMI for two variables. Also, it has been shown (albeit preliminarily) that PMI_2 performs better at extracting certain types of salient information than PMI_1 [Van de Cruys, 2011] based on their experiments on extracting subject-verb-object triples in NLP, so in this work we choose to use this version.

These different forms of PMI can be considered in a graph-based ideological framework: nodes are the variables and the edges are the strength of association between the variables. Under this framework, PMI_1 compares the strength of all pairwise connections in the system ($p(x_i, x_j)$) to the other connected components ($p(x_i), p(x_1, x_2, x_3)$), whereas PMI_2 compares the strength of the clique formed by the nodes compared (how related all n words are - $p(x_1, \ldots, x_N)$) to the disconnected graph ($p(x_1)\cdots p(x_N)$). Using this idea, perhaps PMI_1 may be better at capturing pairwise relations between words than PMI_2, but we leave different formulations of PMI to future research.

For intuition about some of the information PPMI encodes, we present a

list of some of the most interesting triples with high PPMI's we found in Table 1.1. As we can see, word neighbor information (*las, vegas, poker*), phrase-based information (*teenage, mutant, ninja*), and relational syntax information (i.e., subject-verb-object triples: *enzyme, catalyzes, reaction*) are captured.

High PPMI Triples
(*las, vegas, poker*)
(*buick, cadillac, chevrolet*)
(*jesus, christ, saints*)
(*teenage, mutant, ninja*)
(*enzyme, catalyzes, reaction*)
(*sonic, hedgehog, knuckles*)
(*dorsal, fin, spine*)

Table 1.1: Meaningful triples with some of the the highest PPMIs.

In the same way one can construct a PPMI matrix, we study N-way PPMI *tensors* $\mathcal{M}$, where $m_{ijk} = PPMI(w_i, w_j, w_k)$.

1.3.2 Tensors

Basic tensor notation

A tensor for our purpose is simply an N-way array [Kolda and Bader, 2009]. For example, vectors can be seen as tensors of order 1, and matrices can be seen as tensors of order 2. Following the standard notation in the literature, throughout this work we will write scalars in lowercase italics α, vectors in lowercase bold letters $\mathbf{v}$, matrices with uppercase bold letters $\mathbf{M}$, and tensors (of order $N > 2$) with Euler script notation $\mathcal{X}$.

The mode-n product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1,\ldots,I_N}$ and a matrix $\mathbf{M} \in \mathbb{R}^{J \times I_n}$ is of size $I_1, \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$ and is given by

$$(\mathcal{X} \times_n \mathbf{M})_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \ldots i_N} m_{j i_n}$$

In other words, each mode-n fiber ($\in \mathbb{R}^{I_n}$) of $\mathcal{X}$ is left-multiplied by $\mathbf{M}$ ($\in \mathbb{R}^{J \times I_n}$).

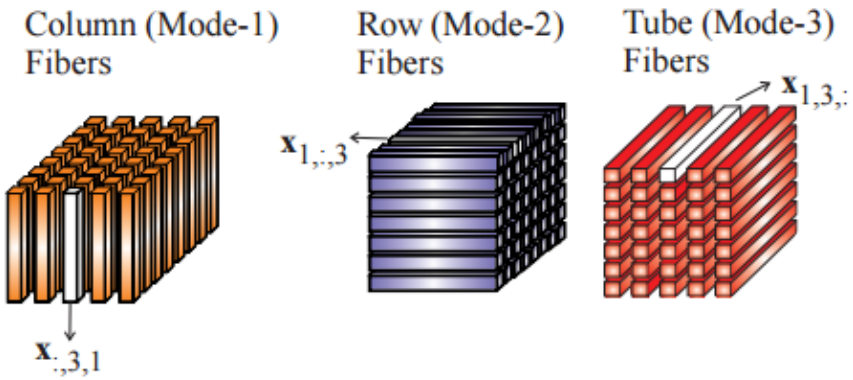


Figure 1.2: Mode-n fibers for a third order tensor ($n = 1, 2, 3$) [Cichocki et al., 2009]

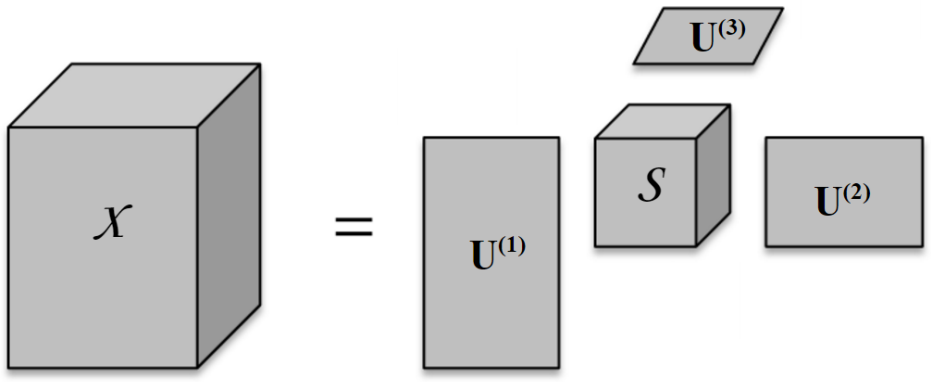


Figure 1.3: Tucker decomposition of a third order tensor

Tensor factorization

Now we will describe the most common tensor factorization techniques. For ease of notation, we will discuss factorization of a 3-way tensor, but it will be obvious how to generalize to the case of an N-way tensor.

Assume $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. The most common tensor factorizations algorithms decompose $\boldsymbol{\mathcal{X}}$ into a small core tensor $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{I' \times J' \times K'}$ where $I' \ll I, J' \ll J, K' \ll K$, and 3 factor matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)} \in \mathbb{R}^{x' \times x}$ (for $x \in \{I, J, K\}$) such that

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{S}} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}$$

Such a decomposition is called the *Tucker decomposition*. This is visualized in Figure 1.3.

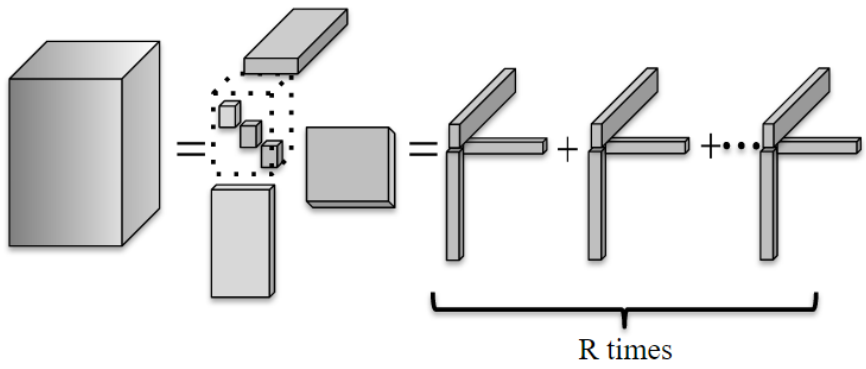


Figure 1.4: Rank-R CP decomposition of a third order tensor

In this work, we study a special case of the Tucker decomposition called the *CANDECOMP/PARAFAC (CP) decomposition*, in which $\boldsymbol{\mathcal{S}}$ is a superdiagonal tensor (i, j, k not equal $\Leftrightarrow s_{ijk} = 0$), and $I' = J' = K' =: R$. This is a *rank-R* CP decomposition. This is visualized in Figure 1.4.

Notice that in the case of a 2-way tensor (matrix), the CP decomposition is given by

$$\mathbf{M} \approx \boldsymbol{\Sigma} \times_1 \mathbf{U} \times_2 \mathbf{V} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$

where $\boldsymbol{\Sigma}$ is a diagonal matrix. Thus, the optimal rank-R 2-way CP decomposition is given by the R-truncated SVD of $\mathbf{M}$.

Unfortunately, in general it is NP-hard to compute the optimal rank-R decomposition for N-way tensors ($N > 2$) [Håstad, 1990]. Further, an optimal rank-R decomposition for a given tensor may *not even exist*, although you may be able to get arbitrarily close to the optimum setting [de Silva and Lim, 2008]. Thus, we have to resort to using approximate solutions. Nonetheless, as we will show, it can still be fruitful to consider approximations to a rank-R decomposition, and we can still approximate tensors quite well in practice.

Because the core tensor is superdiagonal, the CP Decomposition can also be viewed as the sum of the outer product of rank-one factors. Just as the matrix

SVD can be written as:

$$\begin{aligned}\mathbf{M} &\approx \sum_{r=1}^{R} \sigma_r \mathbf{u}_r \circ \mathbf{v}_r \\ &=: [\![\sigma; \mathbf{U}, \mathbf{V}]\!]\end{aligned}$$

The rank-R CP decomposition can be written as:

$$\begin{aligned}\boldsymbol{\mathcal{X}} &\approx \sum_{r=1}^{R} \lambda_r \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \mathbf{u}_r^{(3)} \\ &=: [\![\lambda; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]\end{aligned}$$

where λ is a vector $\in \mathbb{R}^R$ of weights of importance of each rank-1 factor. When λ is equal to the 1 vector in $\mathbb{R}^R$ (i.e., all features are weighted equally), we write $\boldsymbol{\mathcal{X}} \approx [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$.

The factor matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}$ given by the CP decomposition reveal linear relationships between the tensors that correspond to the reconstruction of $\boldsymbol{\mathcal{X}}$, and the superdiagonal core $\boldsymbol{\mathcal{S}}$ (= diag(λ)) gives the corresponding importance weights to each factor, much like the singular values in the matrix SVD.

Symmetric CP Decomposition. There is a variant of CP Decomposition specific to supersymmetric tensors $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{I \times I \times I}$, where $x_{ijk} = x_{\sigma(i)\sigma(j)\sigma(k)}$ for any permutation $\sigma \in S_3$.

In this case, instead of having 3 factor matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}$ and factorizing $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{I \times I \times I}$ such that $\boldsymbol{\mathcal{M}} \approx [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$, we instead consider the problem of factorizing $\boldsymbol{\mathcal{M}}$ into the triple product of a single factor matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ such that

$$\boldsymbol{\mathcal{M}} \approx [\![\mathbf{U}, \mathbf{U}, \mathbf{U}]\!]$$

This works because, if $\boldsymbol{\mathcal{M}}$ is symmetric, $m_{ijk} = m_{\sigma(i)\sigma(j)\sigma(k)}$ so choice of the factor matrix is irrelevant.

The CP Decomposition (and Symmetric CP Decomposition) can be computed in multiple ways, the most common of which is CP-ALS, in which each factor matrix is independently optimized until convergence. An explicit description of the CP-ALS algorithm can be found in [Kolda and Bader, 2009].

There are more advanced methods to compute the CP decomposition [Chi and Kolda, 2012, Sharan and Valiant, 2017], but for the amount of data we use, existing implementations of such methods require more memory than our computing server allows, so we rely on either CP-ALS or our own implementations, which we will describe in detail in Chapter 3.

For a more in-depth overview of tensor factorization and its numerous applications, see [Kolda and Bader, 2009].

1.4 Background and Intuition

1.4.1 The Utility of PPMI

Recall that:

$$\begin{aligned} PMI(x,y) &:= \log \frac{p(x,y)}{p(x)p(y)} \\ &= \log \frac{\frac{\#(x,y)}{|D|}}{\frac{\#(x)}{|D|}\frac{\#(y)}{|D|}} \\ &= \log \frac{\#(x,y)|D|}{\#(x)\#(y)} \end{aligned}$$

where $\#(x,y)$ is defined as the number of times x and y both occur within a k-word window of the text.

Words x and y will have high 2-way PPMI if they frequently appear together (more so than not appearing together). For example, the pair $(puerto, rico)$ is likely to have a large PPMI since the words almost only appear together, but (the, of) will have a low (or zero) PPMI, since they are very likely to co-occur in separate contexts (even though they probably also co-occur in the same context). Also, note that $(wolf, chase)$ will have a high PPMI, as well as $(dog, chase)$ - and *dog* and $wolf$ mean similar things.

Based on this observation, we see that PPMI can be used to encode semantic information - if x tends to co-occur with the same words that y co-occurs with (i.e., $PPMI(x,z) \approx PPMI(y,z)\ \forall\ z \in V$), x and y probably have a similar meaning.

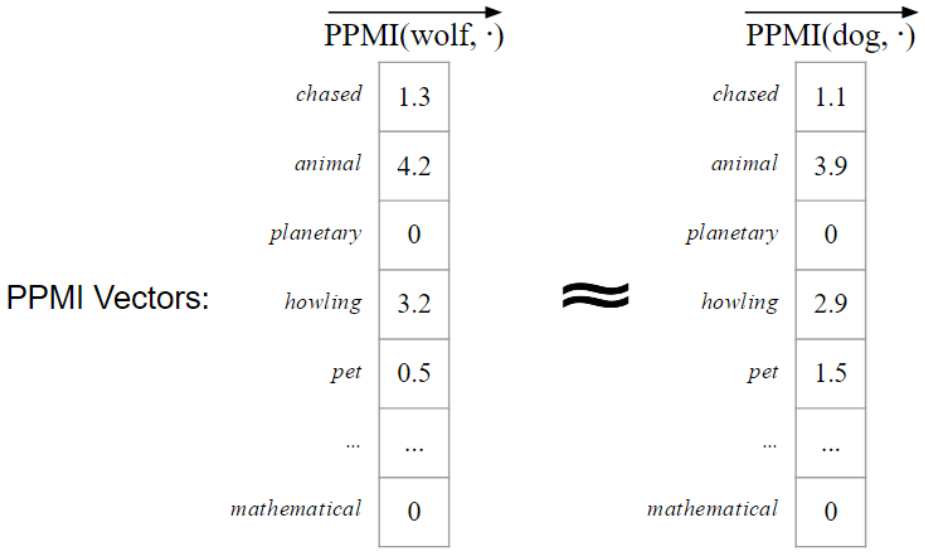


Figure 1.5: The type of information PPMI encodes using the distributional hypothesis

Using this idea, [Levy and Goldberg, 2014] showed that one can create a good word embedding by first creating a (Shifted) PPMI matrix $\mathbf{M}$ where

$$m_{ij} = PPMI(w_i, w_j) - \log k$$

with k being a hyperparameter (corresponding to the number of negative samples in SGNS). Then, they factorize

$$\mathbf{M} \approx \mathbf{W} \cdot \mathbf{C}^\top$$

setting W to be the word embedding. Recall that if words $i, j \in V$ have similar meanings, $PPMI(i, x) \approx PPMI(j, x)\ \forall x \in V$ as observed earlier. Since $\mathbf{w}_i$ and $\mathbf{w}_j$ are used to reconstruct the approximate PPMI vectors $PPMI(i, \cdot)$ and $PPMI(j, \cdot)$, $\mathbf{w}_i \approx \mathbf{w}_j$, as desired. This is illustrated in Figure 1.5.

We aim to generalize the notion of factorizing a PPMI matrix by using tensor factorization on a PPMI tensor.

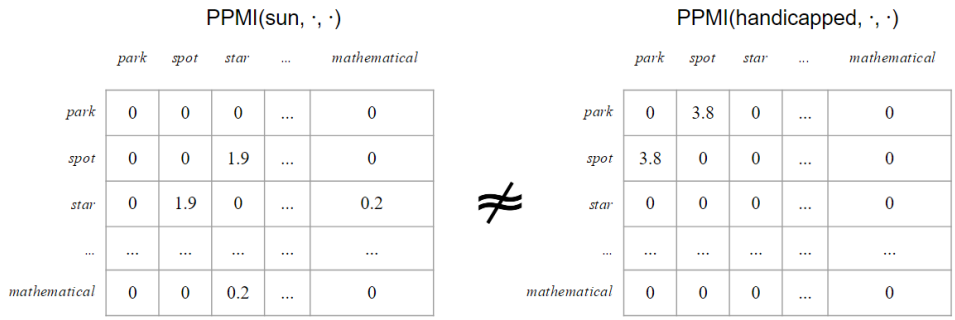


Figure 1.6: Discriminatory power of N-way PPMI

1.4.2 Why N-way PPMI?

Similar information can be encoded with N-way PPMI as that of 2-way PPMI. For example, *cereal* tends to occur in the context of both $breakfast$ and *crunchy*, and the word *granola* also appears in those contexts (and not too many other contexts). And we see that *cereal* and *granola* have similar meanings.

Further, N-way PPMI can encode more information than simply 2-way PPMI. That is, information held by an N-way PPMI tensor cannot be inferred from a 2-way PPMI tensor. For instance, using third order information we can encode more discriminatory information about words. Consider the words *handicapped*, *park*, *spot*, and *sun*. The word *sun* tends to co-occur with both *spot* and *park* (in the separate contexts of solar phenomena (sunspots) and a grassy park), but the triple (*sun*, *spot*, *park*) is much less likely to appear in a single context.

Since *handicapped* tends to also pairwise co-occur with the words *spot* and *park*, a simply pairwise model might (wrongly) assign *handicapped* a similar vector to the vector it assigns *sun*. But a third order model may discriminate between the two, since the triple (*handicapped*, *spot*, *park*) will often co-occur, but the triple (*sun*, *spot*, *park*) will not. The discriminatory power of 3-way PPMI is illustrated in Figure 1.6.

Thus, given enough data, a higher order model is better able to discriminate between words which happen to appear near the same words, but in different contexts. As we will show later, such information can be used to discern multiple meanings of *polysemous* words, which have multiple meanings in different

contexts (such as *spot* and *park*).

The downside here is that tensor methods require much more data than the matrix case. Luckily, unsupervised text data is rather plentiful (thanks to Wikipedia), so lack of data isn't too much of an issue.

Now that we have gone over the required background to understand our work and motivated the reasoning behind our approach, we will now go over the literature related to this work.

Chapter 2

Review of Related Literature

2.1 Word Embeddings

2.1.1 word2vec

Due in part to proper advertisement, proper naming, ease of use, and embedding quality, `word2vec` is the most popular word embedding in use today. In 2013, [Mikolov et al., 2013b] produced two generic word embeddings known as Skip-Gram (SG) and Continuous Bag-of-Words (CBOW), supporting a vocabulary of over 3 million words and phrases embedded into $\mathbb{R}^{300}$, under the name `word2vec`. Their main contribution was a method of computing generic word vectors in a computationally cheap way in order to utilize huge masses of information.

Their model learns from a corpus of sentences, setting a word vector approximately equal to the average of the context vectors of all the words that surround it. Explicitly, for a given $(word, context)$ pair (w, c), SG (the more popular model) maximizes the log probability:

$$\log p(c|w) := \log \frac{\exp(\vec{w} \cdot \vec{c})}{\sum_{c' \in V} \exp(\vec{w} \cdot \vec{c'})} \tag{2.1}$$

for each "context" c that appears surrounding w in a sentence. A *context* is the averaged (or summed) word vectors in a fixed-length window around w. CBOW maximizes $\log p(w|c)$, which is defined symmetrically. A visualization of SG and the context of a word is given in Figure 2.1. In CBOW, the arrows just flow

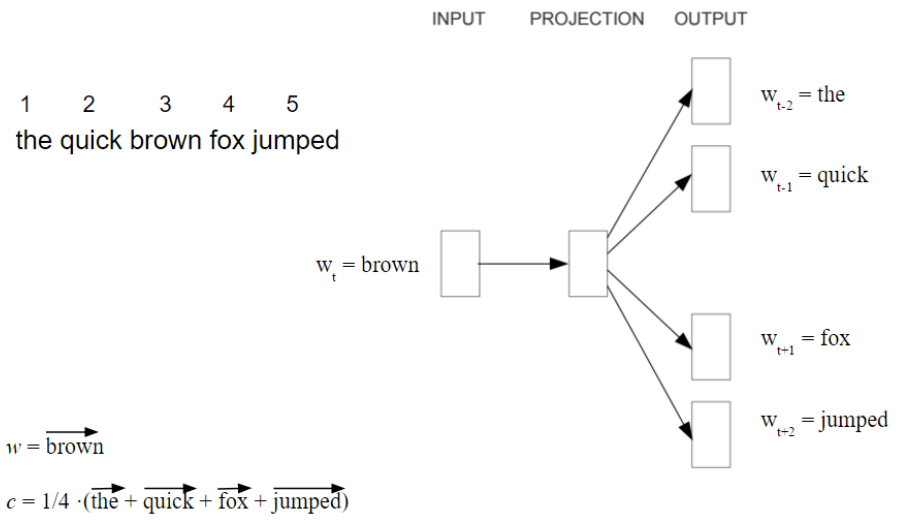


Figure 2.1: A single instance of SG's prediction scheme (at $t = 3$). [Mikolov et al., 2013a]

in the opposite direction, and w and c are switched.

In large-scale applications, $|V| \approx 10^5 - 10^7$, so calculating the denominator in Equation 2.1 is infeasible for all word/context pairs that occur. To get around this, they introduce an approximation to this objective called *negative sampling* in which only a small subset of V is considered in the denominator, under the assumption that most words will likely not co-occur with w. They also now use the sigmoidal function σ popular in neural networks given by $\sigma(x) = \frac{1}{1+e^{-x}}$. Their modified objective (called Skip-Gram with negative sampling (SGNS)) to maximize is given by:

$$\log p(c|w) \approx \log \sigma(\vec{w} \cdot \vec{c}) - \sum_{i=1}^{k} \mathbb{E}_{c_i \sim P_D(w)} \left[\log \sigma(\vec{w} \cdot \vec{c_i}) \right]$$

where k is the number of negative samples (a hyperparameter) and the probability distribution P_D is given by the unigram distribution[1]:

$$P_D(w) = \frac{\#(w)}{|D|}$$

where $|D|$ is the number of observed datapoints (number of tokens in the corpus).

[1]Technically the best setting for P_D they found is according to $(\frac{\#(w)}{|D|})^{.75}$, but for ease of discussion, we use the simpler pure unigram distribution.

If k is small ($\sim 5 - 15$), maximizing this objective is much more efficient than the original SG objective yet still approximates the original objective quite decently.

2.1.2 word2vec as PPMI matrix factorization

[Levy and Goldberg, 2014] showed that SGNS is *implicitly* factorizing a PPMI matrix. Their argument is as follows:

Since the probabilities are given by inner products $P(w_i, c_j) \sim \exp(w_i \cdot c_j)$, this objective is implicitly factorizing some sort of matrix $\mathbf{M}$ such that $\mathbf{M} \approx \mathbf{W} \cdot \mathbf{C}^\intercal$ (so each entry $m_{ij} \approx \mathbf{w}_i^\intercal \mathbf{c}_j$ for $\mathbf{w}_i$ the i^{th} row in a word embedding matrix $\mathbf{W}$ and $\mathbf{c}_j$ the j^{th} row in a context embedding matrix $\mathbf{C}$). We will now derive the actual contents of $\mathbf{M}$.

Now consider the global objective given by SGNS:

$$
\begin{aligned}
\max_{\mathbf{W},\mathbf{C}} \ell &= \sum_{w,c} \Bigg(\log \sigma(w \cdot c) + k \mathbb{E}_{c_{neg} \sim P_D} \Big[\log \sigma(-w \cdot c_{neg}) \Big] \Bigg) \\
&= \sum_{w \in \mathbf{W}} \sum_{c \in \mathbf{C}} \#(w,c) \Bigg(\log \sigma(w \cdot c) + k \mathbb{E}_{c_{neg} \sim P_D} \Big[\log \sigma(-w \cdot c_{neg}) \Big] \Bigg) \\
&= \sum_w \sum_c \#(w,c) \log \sigma(w \cdot c) + \sum_w \sum_c \#(w,c) k \mathbb{E}_{c_{neg} \sim P_D} \Big[\log \sigma(-w \cdot c_{neg}) \Big] \\
&= \sum_w \sum_c \#(w,c) \log \sigma(w \cdot c) + \sum_w \#(w) k \mathbb{E}_{c_{neg} \sim P_D} \Big[\log \sigma(-w \cdot c_{neg}) \Big] \\
&= \sum_w \sum_c \#(w,c) \log \sigma(w \cdot c) + \sum_w \#(w) k \sum_{c_{neg} \in C} \frac{\#(c_{neg})}{|D|} \log \sigma(-w \cdot c_{neg}) \\
&= \sum_w \sum_c \#(w,c) \log \sigma(w \cdot c) \\
&+ \sum_w \#(w) k \Bigg(\frac{\#(c)}{|D|} \log \sigma(-w \cdot c) + \sum_{c_{neg} \neq c} \frac{\#(c_{neg})}{|D|} \log \sigma(-w \cdot c_{neg}) \Bigg)
\end{aligned}
$$

Now consider optimizing this objective for a specific (w, c) pair:

$$
\ell(w,c) = \#(w,c) \log \sigma(w \cdot c) + k \frac{\#(w)\#(c)}{|D|} \log \sigma(-w \cdot c)
$$

Defining $x := w \cdot c$ for ease of notation, consider

$$\frac{\partial \ell(w,c)}{\partial x} = \frac{\partial}{\partial x}\#(w,c)\log\sigma(x) + k\frac{\#(w)\#(c)}{|D|}\log\sigma(-x)$$

$$= \#(w,c)\sigma(-x) + \frac{k\#(w)\#(c)}{|D|}\sigma(x) \qquad \left(\text{Since } \frac{\partial \log\sigma(x)}{\partial x} = \sigma(-x)\right)$$

Setting $\frac{\partial \ell(w,c)}{\partial x} = 0$, we get (after some algebraic manipulation using $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$):

$$\frac{\partial \ell(w,c)}{\partial x} = e^{2x} - \left(\frac{\#(w,c)|D|}{k\#(w)\#(c)} - 1\right)e^x - \frac{\#(w,c)|D|}{k\#(w)\#(c)} = 0$$

Setting $y := e^x$, we get a quadratic equation in y with two solutions:

1. $y = -1$ (which is invalid given the problem)

2. $y = \frac{\#(w,c)|D|}{k\#(w)\#(c)}$

Using the second solution, we get the optimal setting for $x = w \cdot c$ to minimize the loss for a single (w, c) pair is:

$$e^x = \frac{\#(w,c)|D|}{k\#(w)\#(c)}$$

which gives:

$$x = \log\frac{\#(w,c)|D|}{k\#(w)\#(c)} = \log\frac{\#(w,c)|D|}{\#(w)\#(c)} - \log k = PMI(w,c) - \log k$$

And thus the optimal setting for $\mathbf{M} \approx \mathbf{W}\cdot\mathbf{C}^\intercal$ is $m_{ij} := w_i \cdot c_j = PMI(w_i, c_j) - \log k$, where k is the number of negative samples in SGNS. This implies that SGNS is implicitly factorizing a *shifted* PMI matrix, shifted by the number of negative samples in SGNS.

[Levy and Goldberg, 2014] goes further and *explicitly* factorize a shifted PPMI matrix using SVD, reporting improvements on a number of benchmarks (primarily based on word similarity). However, in some evaluation metrics, SGNS outperformed their SVD-based embeddings particularly in the context of encoding syntactic information. We will discuss `word2vec`'s ability to encode

rich syntactic information in Section 4.2.3.

2.1.3 GloVe (Global Vectors)

Another popular word embedding is known as GloVe [Pennington et al., 2014], which is also based on the idea of matrix factorization.

Specifically, GloVe collects a co-occurrence count matrix $\mathbf{X}$ where $x_{ij} = \#$ times w_i occurs within a k-word window of w_j, where k is a hyperparameter. Technically, no symmetry is assumed in this definition, but often is it useful to consider symmetric definitions of context, as we will later in this work.

GloVe then factorizes $\mathbf{X}$, minimizing the cost function:

$$\mathcal{L}(\mathbf{W}, \mathbf{C}) = \sum_{i,j=1}^{|V|} f(x_{ij})(\mathbf{w}_i \cdot \mathbf{c}_j - \log x_{ij})^2$$

(with some added biases). Here, f is a weighting function that is a hyperparameter - in their experiments, they set f to be the piecewise function defined by:

$$f(x) = \begin{cases} \left(\frac{x}{100}\right)^{\frac{3}{4}} & x < 100 \\ 1 & \text{otherwise} \end{cases}$$

To minimize $\mathcal{L}$, one must minimize $(\mathbf{w}_i \cdot \mathbf{c}_j - \log x_{ij})^2$, meaning GloVe is computing a weighted factorization given by $\log \mathbf{X} \approx \mathbf{W} \cdot \mathbf{C}^\intercal$, where the log is taken elementwise.

GloVe is named as such since, unlike `word2vec`, it takes global information into account via the matrix $\mathbf{X}$ of global co-occurrence counts. Our methodologies will be like GloVe in that we first capture global (N-way) co-occurrence information and then factorize a weighted, related tensor.

2.1.4 NNSE

In our experiments, we also consider sparse embeddings, in which most of the entries in the embedding matrix $\mathbf{W}$ are zero. Also, to make the vectors more interpretable, it is useful to consider forcing all vector entries to be positive. In this setting, the dot product of two vectors will be high if and only if they align

positively in a number of dimensions. Since most vectors will be sparse, this means dimensions will tend to take on meaning and interpretability.

The Nonnegative Sparse Embedding (NNSE) [Murphy et al., 2012] is an example of such an embedding. NNSE is based on nonnegative matrix factorization. Later, we will present a sparse nonnegative embedding based on nonnegative *tensor* factorization.

Also, the entries in the matrix in NNSE are weighted by PPMI, so NNSE is another form of PPMI matrix factorization.

In order to fairly compare the nature of our sparse nonnegative embedding, we will use NNSE as a baseline in our experiments along with the dense, potentially negative vectors given by `word2vec` (CBOW) and our other popular embeddings.

2.1.5 Orth-ALS

Recently, tensor factorization has been applied to create a generic word embedding [Sharan and Valiant, 2017], but the idea was not explored extensively – the authors gave just a preliminary glance towards using CP decomposition on a co-occurrence count-based tensor. Their formulation is as follows. Following GloVe [Pennington et al., 2014], they compute a third-order co-occurrence tensor, where each co-occurrence x_{ijk} is weighted by the function $f(x_{ijk}) = \log(1+x_{ijk})$. Then, they factorize the co-occurrence tensor using CP Decomposition to rank $\frac{k}{3}$ (where the embedding dimension is k) and simply concatenate the resulting embedding matrices together.

Their results are not as competitive with pairwise baselines as ours, and they only explore evaluation on two common, yet fairly outdated, tasks. Our work studies word embeddings given by CP decomposition much more in-depth, and we are factorizing a PPMI tensor rather than a weighted co-occurrence-count tensor.

2.2 Word embedding evaluation

Evaluation of word embeddings is still a very open problem. The most common metric for evaluating word embeddings is Word Similarity, in which a list of word pairs are created and human judges score each pair on how similar they are on a score of 1-10, then human scores are averaged and normalized to be in $[0, 1]$. Then, word similarity for each word pair is computed via vector cosine similarity ($sim(v, w) = \frac{v \cdot w}{||v||||w||}$), and Spearman's correlation coefficient ρ [Spearman, 1904] between the predicted scores and the human scores is computed. The higher the similarity, the "better" the word embedding. Word similarity is fast and easy to evaluate a set of word embeddings on, and the results are interpretable. However, there have been a number of problems pointed out about evaluation with word similarity evaluation [Faruqui et al., 2016]. Two big problems with word similarity evaluation that have been solved by more recent suggested evaluation tasks are:

1. **Low correlation with performance on real NLP tasks.** If a certain word embedding performs better than another at word similarity, this by no means indicates that it will do better at a downstream NLP task. This is by far the most important metric for word embedding quality – how well a word embedding can aid the performance on real NLP tasks. Instead of using word similarity, a better idea is to evaluate word embeddings on a set of representative downstream NLP tasks to actually indicate how a word embedding will aid performance on specific tasks.

2. **Low human accuracy.** Humans are often not that good at predicting word similarity, as the task is somewhat poorly defined. For example, many human judges would assign *coffee* and *mug* to have a high similarity score, when the two ideas really mean different things (a bean vs. a drinking container). This leads to low human judge agreement on the actual similarity between words. So even if a word embedding matches the human judgements perfectly, those judgements themselves could be flawed. A more ideal task would be one on which humans have perfect or near-perfect

performance, so we can always judge against the human baseline.

The evaluation task known as Outlier Detection solves these two rather large issues.

2.2.1 Outlier Detection

Recently, [Camacho-Collados and Navigli, 2016] suggested the idea of Outlier Detection as a metric for evaluating generic word embeddings. In this task, a word embedding is presented with a list of $n+1$ words, where the first n words in the list mean semantically similar things, and the last word is unrelated to the first n. For example, a possible list could be:

$$\{\textit{Mercury, Venus, Earth, Mars, Jupiter, lunch}\}$$

To a human, it is clear that *lunch* is the outlier, as the first 5 words are planets, but the word embedding must automatically detect this using vector similarity. After all, computers have no world knowledge.

The outlier is detected via a measure called *compactness*. Ideally, the n words which share semantic meaning should form a tight cluster in the embedded vector space, and the last would should be unrelated to the first n. A *compactness score* for a given word w is given by:

$$c(w) = \frac{1}{Z} \sum_{w_{i_1} \neq w} \left(\sum_{w_{i_2} \neq w, w_{i_1}} sim(w_{i_1}, w_{i_2}) \right)$$

where Z is the appropriate scaling factor to make this the mean similarity between all word pairs of the other n words that are not w.

There are two metrics are associated with this method:

1. *Accuracy* (which gives how often the outlier has the highest compactness score out of the $n+1$ words)

2. *Outlier Position Percentage (OPP)* (which gives how close the outlier is to having the highest compactness score)

Let us explicitly define these two metrics. First, we define Outlier Position

(OP) for a set of words W of length n as the position of the outlier on a scale of 0 to n sorted by compactness score. For example, OP=0 indicates that the outlier had the lowest compactness score (which should never be the case) and OP=n indicates that the outlier (correctly) had the highest compactness score.

Let D be the set of of ordered lists of words. Accuracy is defined as:

$$\frac{\sum_{W \in D} \mathbb{1}(OP(W) = n)}{|D|}$$

and OPP is defined as:

$$\frac{\sum_{W \in D} \frac{OP(W)}{|W|-1}}{|D|}$$

This evaluation metric solves some of the problems listed above: Humans perform nearly perfectly at the task, and performance at outlier detection was shown to be well-correlated with performance on sentiment analysis (a common NLP task) [Blair et al., 2016].

[Blair et al., 2016] made a gold standard dataset for Outlier Detection automatically using Wikipedia. When evaluating word embeddings using outlier detection, we use this dataset.

Also, [Schnabel et al., 2015] points out that word similarity only measures *pairwise* information captures by a word embedding, rather than understanding groups of words. They found that evaluation of set-based properties produced different results than evaluation of pairwise properties, so it is important to take into account N-way relations between words when evaluating a set of word embeddings.

A primary contribution of this thesis is a novel N-way analog to Outlier Detection computation we call ODN (for $N \in \mathbb{N}$). This is meant to measure degree of N-way relations captured by a word embedding. We will discuss this formulation explicitly in Section 4.2.2.

2.3 Tensor Factorization for NLP

Tensor factorization has been studied in the context of NLP in the past. [Van de Cruys et al., 2013] consider the problem of modelling the interaction between

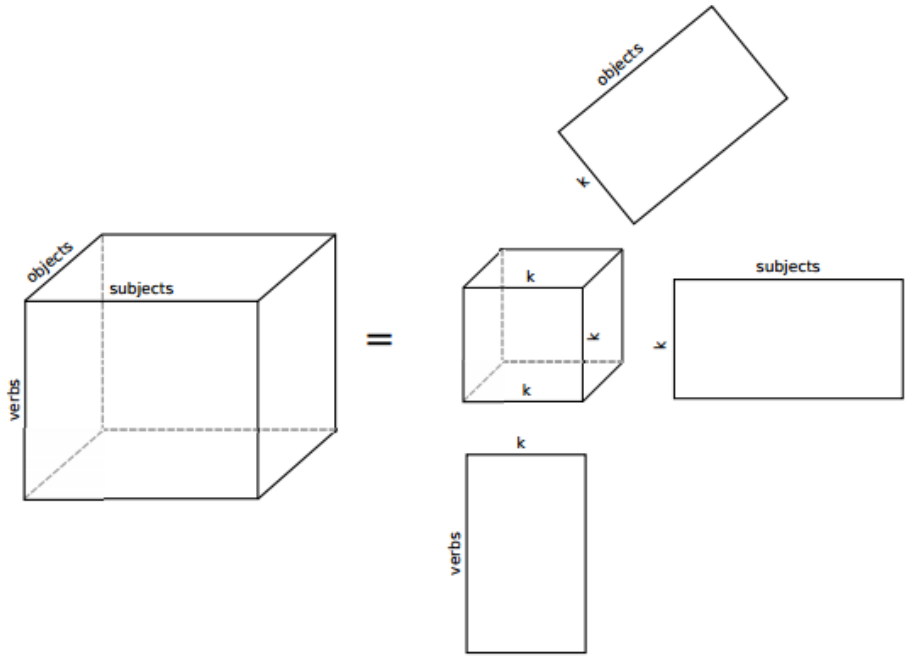


Figure 2.2: Tucker Decomposition of *SVO* triples [Van de Cruys et al., 2013]

(subject, verb, object) (*SVO*) triples. For example, *(man, drove, car)* would be likely, but *(car, drove, man)* would be unlikely.

To do so, they first create a $|V| \times |N| \times |N|$-dimensional tensor $\mathcal{X}$, where $|V|$ is the number of verbs and $|N|$ is the number of nouns. Then, they decompose $\mathcal{X}$ using a k^{th} order Tucker decomposition so

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{W} \times_3 \mathbf{W}$$

with $\mathcal{G} \in \mathbb{R}^{k \times k \times k}$. They then combine $\mathcal{G} \times_1 \mathbf{A} =: \mathcal{G}^{(v)}$ to be a verb tensor modelling the latent interactions between subjects and objects. Each $\mathbf{w}_i \in \mathbf{W}$ is a latent noun vector for the i^{th} noun, and each slice $\mathbf{G}_j^{(v)}$ is a $k \times k$ matrix modelling the latent interactions between subjects and objects for a given verb. Their factorization is visualized in Figure 2.2.

Similar to our approach, they weight the entries in $\mathcal{X}$ by the multivariate PMI_2. The latent factor vectors are used to predict similarity scores between different *subject verb object* triples which may or may not be similar. A correlation coefficient is computed between human-judged similarity scores and latent

vector similarity predictions, meaning a higher score is better. At the time they released this paper, this method outperformed the state-of-the-art at the SVO-modelling task.

Also, [Van de Cruys, 2009] presents nonnegative tensor factorization (NTF) in the context of NLP. Particularly, they apply NTF to the Selectional Preference Induction task, in which a model must accurately model the interactions between SVO triples, where a likely triple will be have a high score and an unlikely triple will have a low score.

To model triples, they again score SVO triples in a 3-way tensor $\mathcal{T}$, factorize it using non-negative CP decomposition (to create Subject embeddings $\mathbf{S}$, Verb embeddings $\mathbf{V}$, and Object embeddings $\mathbf{O}$ in the same space) and then model interactions by elementwise multiplication and summation.

Explicitly,

$$\mathcal{T} \approx \sum_{i=1}^{k} \mathbf{s}_i \circ \mathbf{v}_i \circ \mathbf{o}_i$$

subject to $\mathbf{S}, \mathbf{V}, \mathbf{O} \in \mathbb{R}_{\geq 0}^{x \times k}$, where $x = |S|, |V|, |O|$ ranges between the number of subjects, verbs, and objects. They decompose the tensor using an Alternating Least Squares algorithm.

Since the factorization is given by CP Decomposition, this is a specific instance of Tucker Decomposition (where the core tensor is superdiagonal), and so the same visual presented in Figure 2.2 applies. In this case, however, the factor matrices all have nonnegative entries, and the diagonal core tensor only has 1's on the superdiagonal and 0's everywhere else.

They evaluate their SVO embeddings on a pseudo-disambiguation task, in which a model is presented with two different SVO triples (with varying subjects and objects) and the model must decide which is more likely: (s, v, o) (which was drawn from the corpus), (s', v, o) or (s, v, o'), with s' and o' being randomly drawn from the noun vocabulary. When this paper was released, the nonnegative tensor model outperformed the current state of the art at this task.

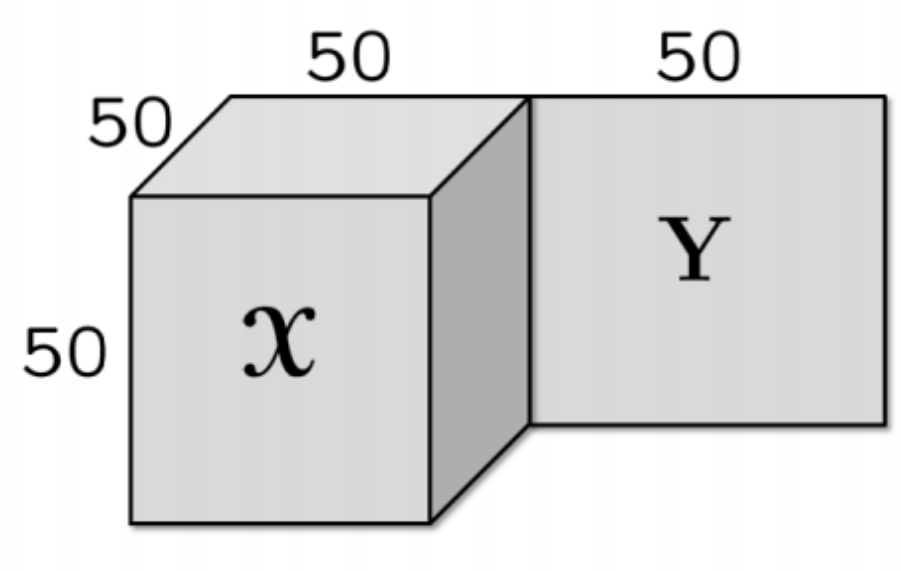


Figure 2.3: Coupled tensor and matrix. In this case, the tensor and matrix share their first axes

2.4 Joint Tensor Factorization

In our embeddings, we not only consider tensor factorizations of a single tensor [Kolda and Bader, 2009], but we also consider the problem of *joint* symmetric CP decomposition, in which multiple tensors are being decomposed using a single factor matrix. This is related to the idea of *Coupled/Linked/Multiple/Multi-* CP decomposition [Acar et al., 2011, Naskovska and Haardt, 2016], in which multiple tensors are simultaneously being decomposed sharing one or more factor matrix, but often there are factor matrices other than the shared one.

[Acar et al., 2011] presents an algorithm to jointly factorize a N-way tensor $\mathcal{X}$ and a single matrix $\mathbf{Y}$ (that has the n^{th} mode in common with $\mathcal{X}$ such that the objective function

$$f(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}, \mathbf{V}) = ||\mathcal{X} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]||^2 + ||\mathbf{Y} - \mathbf{U}^{(n)}\mathbf{V}^\intercal||^2$$

is minimized (with respect to $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}, \mathbf{V}$). This is visualized in Figure 2.3.

In Section 3.2.2 we will present a form of joint tensor factorization in which N supersymmetric tensors are being simultaneously decomposed using a sin-

gle factor matrix $\mathbf{U}$. To the best of our knowledge, this specific case of joint supersymmetric decomposition has not yet been considered.

Now that we have described the related work to our methodologies, in the next chapter we will explicitly describe two primary contributions: the new word embeddings presented and the new joint symmetric tensor factorization problem.

Chapter 3

Methodology

In the next section we explicitly present our novel word embeddings based on CP Decomposition.

3.1 CP Decomposition model (CP)

We follow a similar construction to that of [Levy and Goldberg, 2014]. Reformulated in CP Decomposition notation, they decompose a shifted PPMI matrix $\mathbf{M}$ using the k-truncated SVD:

$$\mathbf{M} \approx [\![\sigma_k; \mathbf{U}_k, \mathbf{V}_k]\!]$$

and set

$$\mathbf{W} := \mathbf{U}_k \sqrt{\text{diag}(\sigma_k)}$$

$$\mathbf{C} := \mathbf{V}_k \sqrt{\text{diag}(\sigma_k)}$$

We generalize this notion by first constructing a 3-way PPMI tensor $\boldsymbol{\mathcal{M}}$, and factorize it via the rank-k CP decomposition such that

$$\boldsymbol{\mathcal{M}} \approx [\![\lambda; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$$

Where $\mathbf{U}^{(i)} \in \mathbb{R}^{|V| \times k}$ has orthonormal columns and $\lambda \in \mathbb{R}^k$ is the vector of weights corresponding to each rank-one approximation.

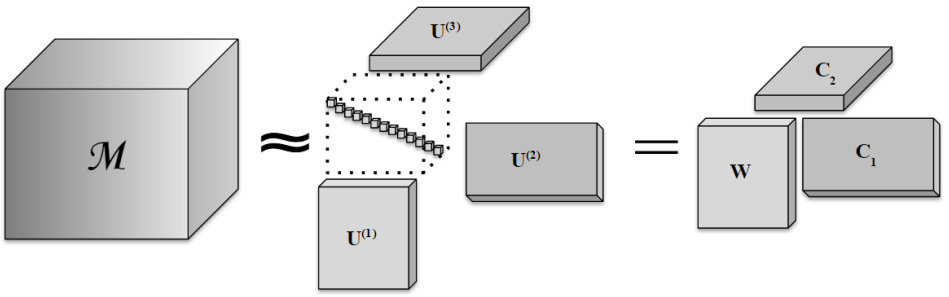


Figure 3.1: CP model

We then set the word embedding to be:

$$\mathbf{W} := \mathbf{U}^{(1)} \sqrt[3]{\text{diag}(\lambda)}$$

$$\mathbf{C_1} := \mathbf{U}^{(2)} \sqrt[3]{\text{diag}(\lambda)}$$

$$\mathbf{C_2} := \mathbf{U}^{(3)} \sqrt[3]{\text{diag}(\lambda)}$$

so $\mathcal{M} \approx [\![\mathbf{W}, \mathbf{C_1}, \mathbf{C_2}]\!]$ (where $\mathbf{C}_i$ is an embedding of the context words).

We will refer to this method as **CP**.

We can also consider *shifting* the PPMI tensor by a constant factor, following the example of [Levy and Goldberg, 2014]. We show (empirically) that using a shift of $\log 15$ produces a better word embedding on almost all metrics than not shifting for CP (and CP-S), so in our experiments we factorize a shifted tensor.

Note that this method is different from the CP Decomposition embedding given in [Sharan and Valiant, 2017], as they decompose a different tensor (based on co-occurrence counts), and they also use the concatenation of the three factor matrices as the word embedding rather than just using (a properly scaled version of) the first factor matrix.

3.2 Symmetric CP Decomposition (CP-S)

Since PMI is supersymmetric by definition ($PMI(x, y, z) = PMI(y, x, z) = ... = PMI(z, y, x)$), the PPMI tensor is supersymmetric, so we take advantage of symmetric tensor factorization.

This is advantageous for a number of reasons:

1. There are N times fewer parameters to learn (for an N-dimensional tensor)

2. We need to feed in $N!$ fewer entries we need to feed into the algorithm (all permutations of the indices can be inferred from one, using the supersymmetry).

and so the decomposition algorithm will have faster convergence, and be a better fit to the tensor. We also found that it is generally advantageous to take advantage of input structure when we have such structural information.

To train our symmetric CP Decomposition, we first define a loss function (given $\mathcal{M}^t$, a random sample of $\mathcal{M}$ at time t) as the squared error between the predicted values and the nonzero tensor values in $\mathcal{M}^t$:

$$\begin{aligned}\mathcal{L}(\mathcal{M}^t, \mathbf{U}) &= \sum_{i,j,k \,:\, m^t_{ijk} \neq 0} (m^t_{ijk} - \sum_{r=1}^{R} u_{ir} u_{jr} u_{kr})^2 \\ &= \sum_{i,j,k \,:\, m^t_{ijk} \neq 0} (m^t_{ijk} - \widehat{m}_{ijk})^2\end{aligned}$$

We learn $\mathbf{U}$ by minimizing the loss function with respect to $\mathbf{U}$ using an off-the-shelf stochastic optimization algorithm such as Adam [Kingma and Ba, 2014] or Adagrad [Zeiler, 2012]. The entire sparse PPMI tensor would not fit in our GPU's memory (of 12GB), so we have to feed in random samples of the tensor in minibatches in an online fashion.

And so we assign our word embedding $\mathbf{W}$ to be:

$$\mathbf{W} := \underset{\mathbf{U}}{\operatorname{argmin}} \, ||\mathcal{M} - [\![\mathbf{U}, \mathbf{U}, \mathbf{U}]\!]||_F$$

We will refer to this method as **CP-S**.

3.2.1 Sparse Nonnegative Symmetric CP Decomposition (CP-SN)

We further study specific instances of the CP Decomposition and explore the quality of word embeddings produced by a *nonnegative* symmetric CP decomposition.

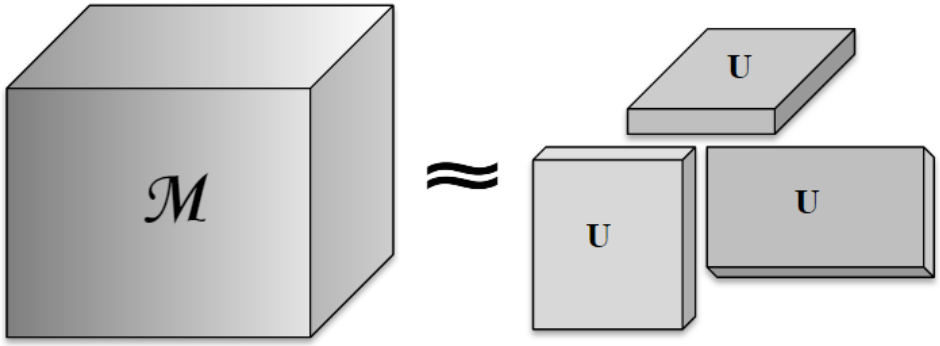


Figure 3.2: Symmetric CP model. In CP-SN, $\mathbf{U}$ is set to $\max(\mathbf{U}, 0)$

In this formulation of symmetric CP decomposition, only the positive values of $\mathbf{U}$ are taken into account when formulating the prediction $\widehat{m}_{ijk}$, and the values less than zero in $\mathbf{U}$ are set to zero. The idea is that only using positive values for prediction provide a more interpretable explanation for similarity between words. This decomposition method is applicable because PPMI is always nonnegative, so negative values should not be needed to describe the tensor.

The formulation for this model is the same as that of the symmetric CP decomposition, but at tensor prediction time, $\mathbf{U}$ is set to $\max(\mathbf{U}, 0)$, where max is taken elementwise. This is otherwise known as the well-known *ReLU* (Rectified Linear Unit) activation function from modern neural network architectures.

To encourage sparsity, we add on L_1 regularization (sum of absolute values) to our loss function, so

$$\mathcal{L}_{\text{reg}}(\mathcal{M}^t, \mathbf{U}) = \mathcal{L}(\mathcal{M}^t, ReLU(\mathbf{U})) + \lambda||\mathbf{U}||_1$$

Given a large enough regularization parameter λ, $\mathbf{U}$ will be mostly zero. The ideal choice of λ is one which sets most values in $\mathbf{U}$ to be zero, yet still predicts $\mathcal{M}$ relatively well.

So we propose a new word embedding based on this training method given by zeroing out the negative entries in $\mathbf{U}$. Here, we assign our word embedding $\mathbf{W}$ to be:

$$\mathbf{W} := ReLU(\operatorname*{argmin}_{\mathbf{U}} \mathcal{L}_{\text{reg}}(\mathcal{M}, \mathbf{U}))$$

We will refer to this method as **CP-SN**.

Sparse nonnegative tensor factorization has also been (in general) studied in the past [Liu et al., 2012], also using the L_1 norm penalty to encourage sparsity. Although, instead of using a specific CP decomposition algorithm for nonnegativity, we simply use a stochastic optimizer to minimize our objective.

3.2.2 Joint Symmetric CP Decomposition (JCP-S)

A problem we noticed with the CP decomposition is that factorizing a N-way tensor *only* trains on N-way information. While matrix factorization only takes into account pairwise information, pairwise information is often important and cannot be missed.

To rectify this issue, we propose a novel joint tensor factorization problem we call *Joint Symmetric CP Decomposition*. In this problem, the input is list of n supersymmetric tensors $\boldsymbol{\mathcal{M}}^{(n)}$ of different orders but whose axis lengths all equal I, each of which is to be simultaneously factorized to rank-R by a *single* $I \times R$ factor matrix $\mathbf{U}$.

Explicitly, we first define a loss function:

$$\mathcal{L}_{\text{joint}}((\boldsymbol{\mathcal{M}}^t)_{n=2}^N, \mathbf{U}) = \sum_{n=2}^{N} \mathcal{L}(\boldsymbol{\mathcal{M}}_n^t, \mathbf{U})$$

where $\boldsymbol{\mathcal{M}}_n \in \mathbb{R}^{\overbrace{|V| \times \cdots \times |V|}^{n \text{ times}}}$ is an n-dimensional symmetric PPMI tensor, so we are decomposing $N-1$ such tensors. We then minimize the loss with respect to $\mathbf{U}$.

For example, in our experiments, we consider joint factorization of 2-dimensional and 3-dimensional tensors, given by a loss function of:

$$\mathcal{L}_{\text{joint}}(\mathbf{M}_2^t, \boldsymbol{\mathcal{M}}_3^t, \mathbf{U}) = \mathcal{L}(\mathbf{M}_2^t, \mathbf{U}) + \mathcal{L}(\boldsymbol{\mathcal{M}}_3^t, \mathbf{U})$$

Here, we assign our word embedding $\mathbf{W}$ to be:

$$\mathbf{W} := \underset{\mathbf{U}}{\operatorname{argmin}}\, \mathcal{L}_{\text{joint}}((\boldsymbol{\mathcal{M}})_{n=2}^N, \mathbf{U})$$

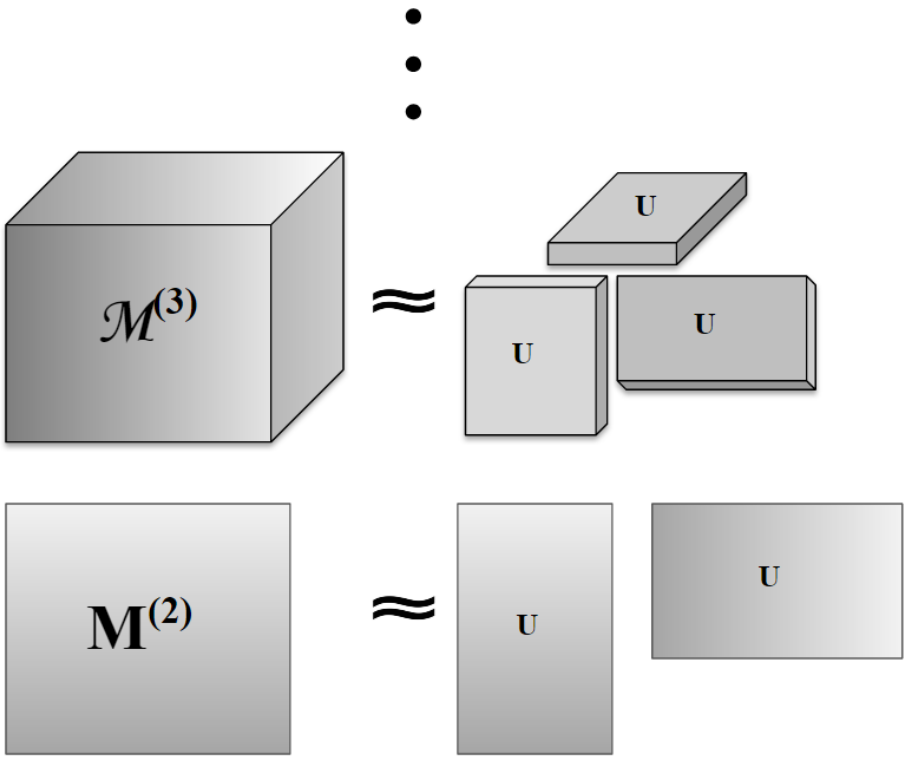


Figure 3.3: JCP-S model

We will refer to this method as **JCP-S**.

3.3 Implementation Details

For vocabulary management, we use Gensim [Řehůřek and Sojka, 2010]. For our first embedding suggested (CP), we use CP-ALS as implemented in the Tensor Toolbox [Bader et al., 2015]. We implement all variants of symmetric CP decomposition directly in TensorFlow [Abadi et al., 2015]. All of our code is publicly available on GitHub[1].

[1] `https://github.com/popcorncolonel/tensor_decomp_embedding`. Currently we only release vectors pre-trained on our development set of 10 million sentences of Wikipedia due to training time limitations and the scale of the data. However, we will soon release a pre-trained set of vectors for these methods to the public after upgrading our computing server, and after finding the optimal settings for a problem of such scale required to make a high-quality large embedding.

3.3.1 Computational issues

There were a number of computational limitations prohibiting the direct implementation of our ideas. First, the most obvious limitation is the size of the data we're working with. Unlike some other tensor factorization approaches in NLP which typically use only third order tensors on the scale of $1,000 \times 1,000 \times 10,000$ which can fit in memory on even a modest performance computing server (assuming floats, that is 40 GB), our tensors are of size $|V|^n$, where $|V|$ can vary from $10,000$ to $300,000$. For $n = 3$, even a modest vocabulary size of $10,000$ entries fills up $10,000^3 * 4$ bytes of floats, which is 4 TB of data. So clearly, all the tensors we are working with must be stored in a sparse fashion, where only the nonzero entries are stored.

On our third order experiments using the first 10 million sentences of Wikipedia, the 3-way PPMI tensors store 396 million nonzero entries, meaning only $\frac{3.96e8}{8000^3} = 0.00077$ (or 0.077%) of the tensor data is nonzero. So not only do we have to deal with the massive scale of data, we have to face the vast sparsity of the PPMI tensor.

Also, as previously stated, the CP Decomposition in general cannot be computed exactly, as the problem is NP-hard [Håstad, 1990], and potentially ill-posed [de Silva and Lim, 2008]. So not only do we have to use an approximation, many of such existing approximate methods require too much storage. For instance, the CP-APR algorithm of [Chi and Kolda, 2012] requires the storage of a matrix of dimension #nonzeroes $\times$ k (where k is the embedding dimension), which cannot even be stored in memory itself – there are 396 million nonzero unique entries for our moderate third order development case, which is $396,000,000 * 300 * 4$ bytes = 475 GB of 300-dimensional float vectors.

Thus, we often have to resort to online approximations, where nonzero subsets of the tensor are fed in in minibatches. Online methods scale better with data, since we could theoretically feed as much data as we want into them. CP Decomposition solvers which take in the entire tensor (such as CP-ALS) are limited by how much nonzero data can fit in memory, but online methods can theoretically take in all the data that exists in batches.

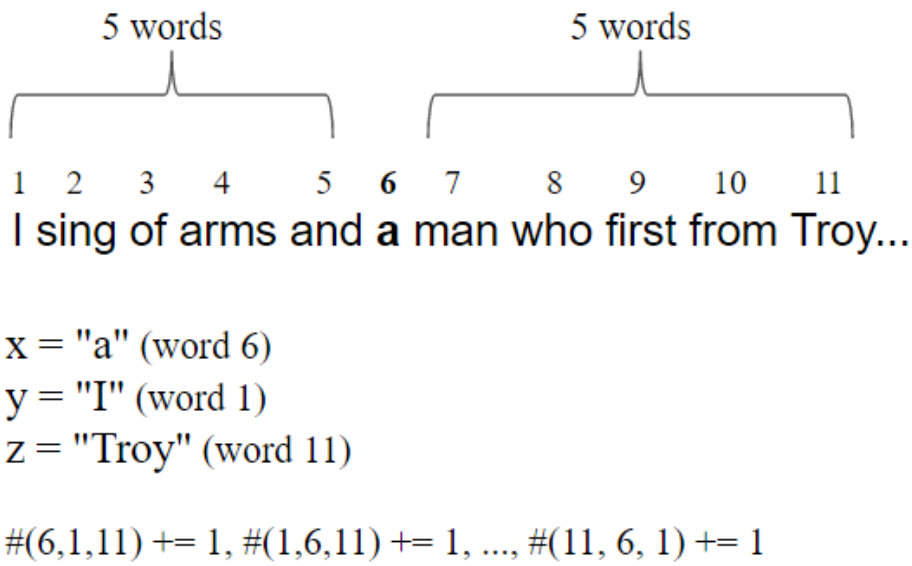


Figure 3.4: Enforcing n-way PPMI supersymmetry

3.3.2 Computational notes

Recall that $PMI(x, y, z)$ is defined as:

$$\begin{aligned} PMI(x, y, z) &= \log \frac{p(x, y, z)}{p(x)p(y)p(z)} \\ &= \log \frac{\frac{\#(x,y,z)}{|D|}}{\frac{\#(x)}{|D|}\frac{\#(y)}{|D|}\frac{\#(z)}{|D|}} \\ &= \log \frac{\#(x, y, z)|D|^2}{\#(x)\#(y)\#(z)} \end{aligned}$$

Where $\#(x, y, z)$ is the number of times the both y and z appear within a window of 5 words around x. Technically this is not a symmetric definition, since y and z could appear on different sides of x, and thus would not be within a window of 5 words of each other. However, up to doubling the window size, this is a symmetric definition, so to ensure good quality of our Symmetric CP Decomposition, we enforce supersymmetry by only counting $\#(x, y, z)$ and then setting $\#(y, x, z) := \cdots := \#(z, y, x) := \#(x, y, z)$. We visualize the problem and our solution in Figure 3.4.

We first gather the counts $\#(\cdot)$ on the entire corpus and then create the PPMI values in another pass to be fed into a CP decomposition algorithm. In

order to feasibly store all the counts in memory, while gathering the counts of all triples we occasionally delete half of the triples with count 1, assuming those instances are just noise. After gathering all the counts, we delete all triples with count < 5, assuming again that those occurrences were simply due to random chance.

Also, instead of just feeding in the positive values of the tensor (and thus incorrectly not capturing the sparsity of the tensor), we use "negative sampling" for our online tensor decompositions. What we mean by this is that in addition to feeding in the nonzero entries of a minibatch, we also feed in a certain percentage of the entries in each batch as random indices that do not co-occur in the text, and thus have zero PPMI. We found that putting too many negative samples caused our systems to learn the zero embedding, and a good amount of negative samples is around 10% of the entries per minibatch should be zero "noise" entries.

Now that we have explicitly laid out our word embedding formulations, we will discuss evaluation of word embeddings in general and present both quantitative results and some of our most interesting qualitative results.

Chapter 4

Evaluation and Findings

In this section we will discuss word embedding evaluation and present our results on a number of benchmarks. We will also qualitatively evaluate our word embeddings, showing some interesting properties that are encoded in our embeddings but not in other existing embeddings.

Unlike many machine learning tasks, there is no single standard, agreed-upon way of evaluating word embeddings as a benchmark to declare one word embedding better than another. In fact, such a benchmark may not even exist [Jastrzebski et al., 2017]. In the past, the most popular metric for evaluating the quality of word embeddings has been word similarity, but it has been shown that performance quality on word similarity has low correlation with performance on actual downstream NLP tasks, among other problems [Faruqui et al., 2016].

There have been many methods proposed as gold standards for word embedding evaluation[1], so we will go over a few of these evaluation methods and their motivations, and present comparative results for each of them.

4.1 Embeddings to evaluate

Our baselines are:

1. **Random** (embedding with entries distributed via $\mathcal{N}(0, \frac{1}{2})$), for comparing against a model with no information encoded

[1] `https://xkcd.com/927/`

2. **CBOW** (`word2vec`) [Mikolov et al., 2013b], for comparison against the most commonly used embedding method

3. **NNSE**[2] [Murphy et al., 2012], for comparing against an explicit matrix factorization embedding, as well as a sparse nonnegative embedding

With our own contributions being **CP, CP-S, CP-SN, JCP-S**. We will refer to CBOW and NNSE as *pairwise models* (as they are specifically trained on pairs of words), and we will refer to our models as *tensor-based models*. We refer to CP-S, CP-SN, and JCP-S as *symmetric CP decomposition based models*. For a fair comparison, we trained them on the exact same data and vocabulary – 10 million randomly shuffled sentences of Wikipedia (130 million tokens) with stopwords and words that appear fewer than 2,000 times removed – and implemented all methods using the recommended hyperparameters.

Following [Levy and Goldberg, 2014], we consider factorization of a *shifted* PPMI tensor $\mathcal{M}$, where

$$m_{w_1,w_2,w_3} = PPMI(w_1, w_2, w_3) - \log k$$

We find that the optimal number of negative samples depends on the specific model, but in all our experiments we report results using the best setting of k we found for each embedding. We will discuss our findings for the shifts later in this section.

Unless otherwise stated, all CP methods are trained with third order tensors, and JCP-S with tensors of order 2 and 3. All embeddings are 300-dimensional.

4.2 Quantitative evaluation

4.2.1 Word Similarity

A description of the Word Similarity problem was given in Section 2.2. As stated previously, there have been many problems pointed out with evaluating

[2]The input to NNSE is an $m \times n$ matrix, where there are m words and n co-occurrence patterns. In our experiments, we set $m = n = |V|$ and set the co-occurrence information to be the number of times w_i appears within a window of 5 words of w_j. As stated in the paper, the matrix entries are weighted by PPMI.

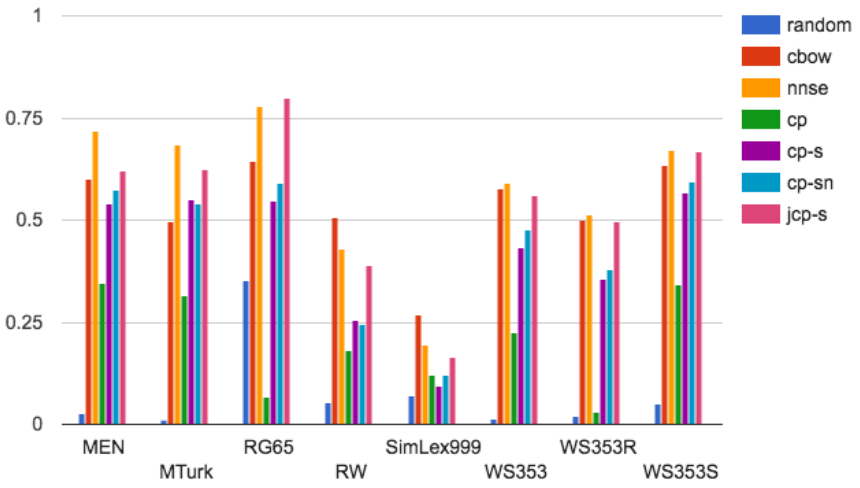


Figure 4.1: Word similarity scores

word embeddings solely using word similarity [Faruqui et al., 2016], so we treat word similarity more as a heuristic to verify that the word embedding is actually capturing some semantic information rather than using it as a gold standard to say which embeddings are overall superior.

The descriptions of specific word similarity datasets can be found in the Word Embedding Benchmarks software package[3] from the authors of [Jastrzebski et al., 2017]. All word similarity evaluation code is using evaluating using this package.

Results and discussion. A summary of word similarity results is given in Figure 4.1. As we can see, all tensor methods perform considerably better than the random embedding, so similarity information is definitely being encoded. NNSE appears to encode the most amount of similarity information (on average), often performing best on all datasets. Still, JCP-S still occasionally outperforms NNSE (RG65), and outperforms CBOW a number of times (MEN, MTurk, RG65, WS353S). Pairwise similarity is naturally correlated with how much pairwise information a word embedding captures, whereas the tensor-based embeddings are all trained on third order information. It is thus logical that JCP-S (which is trained on both pairwise information and 3-way information) would outperform the other tensor-based methods at this task (which is only trained on 3-way information). We suspect tensor-based methods would

[3] `https://github.com/kudkudak/word-embeddings-benchmarks/blob/master/web/datasets/similarity.py`

excel at groupwise word similarity, but we are unaware of such a dataset.

In fact, it is somewhat surprising that these third order methods do well on these metrics at all: the word similarity predictions are given by pairwise relationships between words. Since 3-way methods are performing decently on these metrics than the random embedding, these methods are still learning some pairwise information - just not as well as the methods whose objective it is to explicitly capture such pairwise information.

As we will see Section 4.2.2, CBOW and other 2-way methods are still capturing some higher order information, just not as well as the CP methods.

4.2.2 Outlier Detection

A description of the existing method to compute Outlier Detection scores was given in Section 2.2.1. Recall that, unlike word similarity, Outlier Detection was shown to be correlated with downstream tasks such as Sentiment Analysis. We will now describe our proposal for N-way outlier detection.

***N*-way Outlier Detection.** In order to measure how well an embedding captures N-way information, we suggest a natural N-way extension of the existing outlier detection method. Instead of computing the compactness score by the mean similarity between *pairs* of words, we compute compactness score by the mean similarity between *groups of* N words. Similarity between a group of N vectors is defined to be the inverse of the mean distance between each vector and the average of the N vectors:

$$sim(v_1, \ldots, v_N) = \left(\frac{1}{N} \sum_{i=1}^{N} ||v_i - \frac{1}{N} \sum_{j=1}^{N} v_j||_2 \right)^{-1}$$

and compactness score for a word w is given by:

$$\frac{1}{Z} \sum_{w_{i_1} \neq w} \sum_{w_{i_2} \neq w, w_{i_1}} \cdots \sum_{w_{i_N} \neq w, \ldots, w_{i_{N-1}}} sim(w_{i_1}^{i_N})$$

where Z is the appropriate normalization factor. We call this N-way evaluation method Outlier Detection (N) (ODN). Thus, the method suggested in [Camacho-Collados and Navigli, 2016] is a special case of ODN, where $N = 2$.

To implement ODN, we modify the code released in [Blair et al., 2016].

ODN can be seen as as a way of evaluating an embedding's ability of capturing N-way information, as it computes similarity between groups of N words. We will show that tensor methods perform relatively (and absolutely) better on OD3, and pairwise methods perform relatively better on OD2.

Dataset. Outlier detection datasets are comprised of a set D of lists ($W \in D$) of words ($w_i \in W, i = 1, \ldots, n+1$), where in each list the first n words all share semantic meaning and the last word (the outlier) is unlike the first n. These datasets can be constructed by humans [Camacho-Collados and Navigli, 2016] or automatically [Blair et al., 2016]. The specific dataset we use for ODN is the WikiSem500 dataset, which was gathered automatically based on Wikipedia articles [Blair et al., 2016]. In this dataset, $|D| = 13,314$, before filtering by out-of-vocabulary words.

(Method)	**OD2 OPP**	**OD2 acc**	**OD3 OPP**	**OD3 acc**
Random	0.6123	0.2765	0.5345	0.1950
CBOW	0.6542	0.3731	0.6162	0.3034
NNSE	0.6998	0.4288	0.6292	0.3190
CP	0.6532	0.3586	0.6190	0.2795
CP-S	**0.7078**	**0.4370**	0.6741	**0.3597**
CP-SN	0.6874	0.3950	**0.6747**	0.3545
JCP-S	0.7017	0.4242	0.6666	0.3201

Table 4.1: Outlier detection scores across all embeddings

Results and discussion. A comparison of outlier detection performance is given in Table 4.1. Recall that OPP measures how close the outlier is to being detected as the outlier, and accuracy (acc) is the proportion of times it was correctly detected as the outlier.

At first glance, we notice CP-S outperforms every other embedding at most of these tasks. Only on OD3 OPP does CP-SN barely beat it. We will now discuss the way ODN measures N-way information.

For OD2, the pairwise models perform quite competitively with the symmetric CP Decomposition based models. However, when OD3 is considered, the latter methods clearly shine: for OD2 OPP, the mean difference between the symmetric CP decomposition scores and pairwise model scores is 2.2%, whereas

for OD3 OPP the mean difference is 4.9%.

Based on this, we can clearly see that the CP decomposition models perform comparatively much better than the pairwise methods on the groupwise metric compared to the pairwise metric. Also, notice that JCP-S performs better than the pairwise embeddings at OD3 and (mostly) better at the other tensor-based embeddings at OD2. Given this, we see JCP-S is successfully capturing both pairwise and 3-way information.

We notice that CP does not perform as well (absolutely) as any of the other ODN benchmarks on average. We believe this is due to the poor tensor fit the CP-ALS algorithm gives. We outline some suggested improvements to the CP model in Section 5, which will likely increase its performance across all tasks.

However, we can look at its relative performance on OD2 and OD3 to deduce information about ODN. The relative difference between CP's OD2 OPP and NNSE's OD2 OPP is 6.9%, which is a rather stark difference. However, on OD3, their OPP only relatively differs by 1.6%. Also, CBOW outperforms CP at OD2 OPP, but the opposite is true for OD3 OPP.

Since CP is trained using third order information and it performs relatively better on OD3 compared to its performance on OD2, this indicates that OD3 is accurately measuring the degree to which third order information is captured. We believe similar results will old ODN and N^{th} order information, but we leave experiments on higher-order tensors to future work.

As we can see, the plain CP method (which was found using the simple CP-ALS algorithm) does not do as well on the others (other than Random). We conjecture that this is because the tensor fit is not as good as the other algorithms due to the naive decomposition algorithm which has very weak convergence properties [Sharan and Valiant, 2017]. We believe that the other tensor methods tend to perform better at capturing this N-way information because their respective tensor factorization algorithms have more structure and fewer parameters. It would be interesting to see how much the quality of results increase given a better CP decomposition than that given by CP-ALS, such as that given by [Chi and Kolda, 2012].

This phenomenon is also shown in [Sharan and Valiant, 2017], where they

compare word embeddings produced by CP-ALS compared to the CP decomposition algorithm they provide in their paper. As they report, a better tensor fit leads to a better embedding, and this is what we believe is happening here. However, our results are not directly comparable because we are using a different vocabulary of words which may have different embedding dimensions.

4.2.3 Simple supervised tasks

Description. [Jastrzebski et al., 2017] discusses the importance of using simple supervised machine learning tasks when evaluating word embeddings. Their main idea is that the primary application of word embeddings is in *transfer learning* to machine learning tasks which do not have much supervised data (the pre-trained word embeddings are already trained on a large corpus of unsupervised data). If supervised training data for a given task were plentiful, one could learn their own task-specific word embeddings, which would almost certainly outperform any possible generic embedding at that specific task.

So when evaluating the quality of generic word embeddings, one should measure the *accessibility of information* to a supervised task. To do so, one must cite the performance of simple supervised tasks that do not have much data to learn from, thereby measuring the ease of transfer learning given by a specific embedding. Also, it is important to measure the progress of a supervised learning algorithm as training set size increases, as is commonly done in transfer learning evaluation [Jastrzebski et al., 2017]. If an algorithm performs well with just a small subset of all of the available training data for a particular supervised problem, then the information encoded in the word embeddings must be easily accessible, which is beneficial for successful transfer learning. This is not the case for word embeddings that perform well only by the end of the supervised training – in this case, the information was harder for the supervised model to learn.

The simple supervised downstream tasks we used to evaluate the embeddings are as follows:

1. **Supervised Analogy Recovery.** Analogy recovery is the task of answering a query of the form *Japan : yen :: France :* ?, where the answer is *euro*.

[Mikolov et al., 2013b] showed (empirically) that their `word2vec` models can answer such queries *linearly* (i.e., $\overrightarrow{yen} - \overrightarrow{japan} + \overrightarrow{france} \approx \overrightarrow{euro}$).

As shown in [Jastrzebski et al., 2017], analogy information need not be encoded linearly in word embeddings. Indeed, answers to such queries can be found using a neural network even in embeddings that do not encode analogy linearly. The specific analogy neural network we use is defined as follows for an analogy query of the form $\mathbf{v}_1 : \mathbf{v}_2 :: \mathbf{v}_3 : \mathbf{v}_4$ (all vectors are normalized to unit length):

$$\widehat{\mathbf{v}_4} = \frac{\mathbf{W}_2\mathbf{v}_2 - \mathbf{W}_1\mathbf{v}_1 + \mathbf{W}_3\mathbf{v}_3}{||\mathbf{W}_2\mathbf{v}_2 - \mathbf{W}_1\mathbf{v}_1 + \mathbf{W}_3\mathbf{v}_3||}$$

Where the $\mathbf{W}_i$'s are initialized to be the identity matrix $\mathbf{I}_k$, then learned using a stochastic optimizer (in minibatches) to minimize the loss:

$$\mathcal{L}(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4) = ||\mathbf{v}_4 - \widehat{\mathbf{v}_4}||_2 + \lambda_{reg}||\mathbf{W}_3||_2^2$$

Notice that in this formulation, since $\mathbf{W}_i$ is initialized to be the identity, this formulation is equivalent to using simple vector addition to recover the analogies learning, as is encoded in many popular word embeddings such as `word2vec` and GloVe [Mikolov et al., 2013a, Pennington et al., 2014].

We should note that our implementation slightly differs from the one originally suggested in [Jastrzebski et al., 2017] – in their formulation, they consider an affine solution, where a bias vector $\mathbf{b}$ (initialized to be the zero vector) is added to the numerator and denominator. Also, the objective they optimize is to maximize the cosine similarity between $\mathbf{v}_4$ and $\widehat{\mathbf{v}_4}$. We found that our neural network, given enough data, can perfectly recover such analogies in all of our embeddings, so an affine transformation is not needed. Also, we noticed that the random embedding was performing quite well at this task (likely due to overfitting[4]), so we introduced regularization

[4] λ_{reg} is a regularization parameter to prevent overfitting due to the amount of repetition in the Google analogy testbed (the query $a : b :: dublin : ireland$ shows up 22 times). Because of this repetition, the neural network can simply learn the mapping $c \rightarrow d$ in $a : b :: c : d$ without taking a and b into account. $\mathbf{W}_3$ encodes this mapping, and regularizing $\mathbf{W}_3$ fixed the issue we were noticing. In our experiments, we set λ_{reg} to be 10^{-3}.

to prevent this.

The analogy dataset we use is from the Google analogy testbed [Mikolov et al., 2013b], and we filter out all analogies where all 4 words are not in the vocabulary, leaving a total of 4,530 analogies to train and test on. 85% of the analogies (randomly shuffled) are used for training and the rest are used for testing.

The Google testbed is separated into semantic queries (like *Cairo : Egypt :: Paris :: France*) and syntactic queries (like *has : had :: run : ran*). We report scores on both semantic and syntactic queries[5].

2. **Sentiment Analysis.** Another simple supervised task we use is Sentiment Analysis, as described by [Schnabel et al., 2015].

   In this task, a document is given by a weighted sum of the unique words in the document (weighted by the number of times each word appears). This weighted sum is then fed into a Logistic Regression model for classification as either positive sentiment or negative sentiment.

   As stated in [Schnabel et al., 2015], performance on this task should be correlated with (or at least an indication of) *semantic* information encoded, since nearly all syntactic information in the document is lost by summing the word vectors, and sentiment is a semantic category.

   The specific dataset we use is that of the *Polarity Dataset v2.0* [Pang and Lee, 2004], which contains 1,000 positive movie reviews and 1,000 negative movie reviews. 85% of the reviews (randomly shuffled) are used for training and the rest are used for testing.

3. **Word Classification.** The final simple supervised task we mention from [Jastrzebski et al., 2017] is the task of single word classification. Specifically, the classification problem we use is word part of speech classification, where each word is classified as noun, adjective, adverb, verb, etc. The input to the supervised model is the corresponding word vector.

[5] Recall that *Semantic information* is the information related to meaning, and *Syntactic information* is the information about the set of rules that govern the structure of sentences in a language.

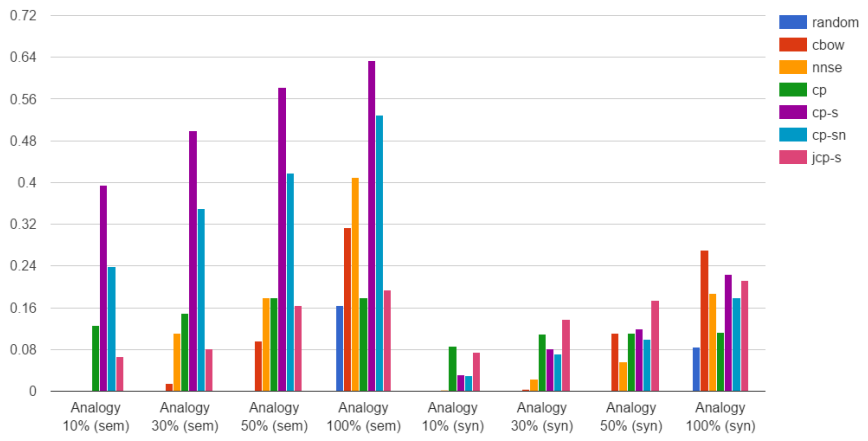


Figure 4.2: Supervised analogy task performance vs. % training data

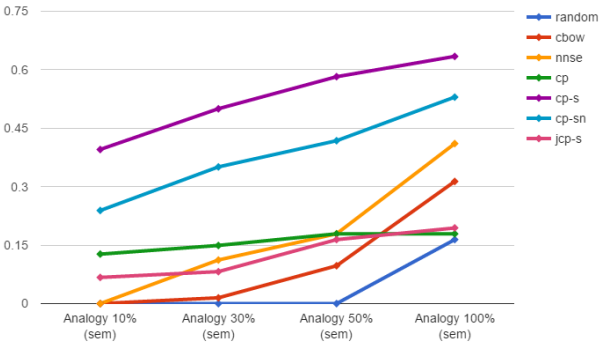


Figure 4.3: Semantic analogy accuracy vs. % training data

Performance on this task should an indication of the amount of *syntactic* information encoded rather than semantic information, since part of speech is a syntactic category.

Using as simple a model as possible, we train a Logistic Regression model using the single word as the only feature. After filtering out words that are out of the vocabulary, there were 4330 total words for training and 765 words for testing.

The numerical results presented are the mean scores of 5 random restarts.

All code is implemented based on the description in the papers referenced in each individual task, using scikit-learn [Pedregosa et al., 2011] or TensorFlow [Abadi et al., 2015].

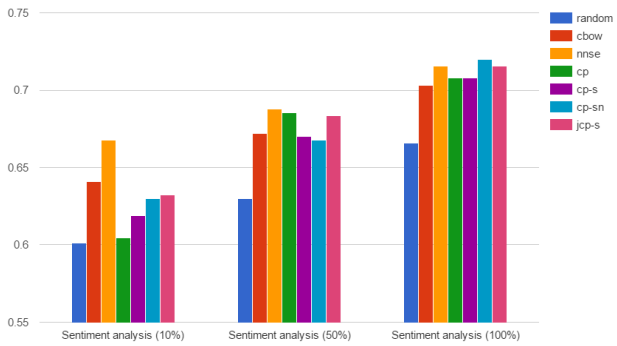


Figure 4.4: Sentiment analysis task performance vs. % dataset size

Results and Discussion

Again, to measure quality of transfer learning, we report on the performance of each supervised model as training dataset size increases.

Supervised analogy recovery. A comparison of the results on the supervised analogy task is found in Figure 4.2. "Analogy x%" means only x% of the training data was used (3,850 analogies total) for the supervised analogy NN. Accuracy is always calculated on the entire test set (680 analogies).

As we can see, tensor methods are very strong in a limited data setting at encoding semantic relational information as they all outperform the pairwise methods at analogy tasks when trained on 10% of the data (fewer training examples than the size of the test set!). Even with such little data, the NN trained using CP-S's vectors still manage to achieve an accuracy of 39.55% on the semantic analogy testbed, and CP achieves a syntactic accuracy of 8.6%.

To further visualize the difference in information accessibility between the embeddings, we present a graph of semantic analogy accuracy vs. training data in Figure 4.3. Notice that both pairwise models have a sharp jump in quality when going from 50% of the training data to 100% of the training data – such a jump indicates that this semantic information in the pairwise models is less easily accessible than in the tensor-based models, as supervised models trained on pairwise data require more data to perform as competitively with the supervised models trained on the tensor-based data. For these reasons, we can conclude that this form of semantic information encoded in the tensor-based embeddings is more easily accessible than that of CBOW and NNSE.

We observe that, following the trends in other papers, all of these models across the board have a harder time encoding syntactic information than syntactic information. Even so, we can see that such information is more easily encoded in the tensor-based methods, as they perform better (than the other models) on a small subset of the data. However, they are still eventually surpassed by CBOW when data gets to 100%.

We will discuss CBOW's ability to encode syntactic information compared to the other methods later in this section, and suggest further improvements to the syntactic information encoded in our models in Section 5.1.

Supervised sentiment analysis. A performance comparison on the supervised sentiment analysis task is presented in Figure 4.4.

At sentiment analysis, notice that NNSE outperforms all others given little training data. Also, CP-SN performs on top when 100% of the training data is presented, with NNSE trailing close behind. Given this information, we see that sparse embeddings (such as NNSE and CP-SN) do not necessarily encode less information, and may actually encode *more* information (or present such information more readily).

This is something that the NLP/deep learning community tends to overlook. Often, dense embeddings are often favored presumably because of the idea that they encode much more information, but as we see from CP-SN's performance on various supervised tasks, this is not necessarily the case.

Also, all tensor-based methods outperform CBOW at this task when training data is 100%, speaking to the quality of semantic information they encode.

Supervised word classification. A comparison of performance on part of speech classification is presented in Figure 4.5. As we can see, CBOW clearly outperforms all other methods at this task, which is indicative of syntactic information encoded. Following the trend set by analogy recovery, the tensor-based methods tend to perform better on semantic tasks compared to syntactic tasks, especially when all the data is presented. However, at least in the case of syntactic analogy recovery, the syntactic information seems to be more efficiently encoded in the tensor-based methods as they outperform the other baselines when training data is very sparse.

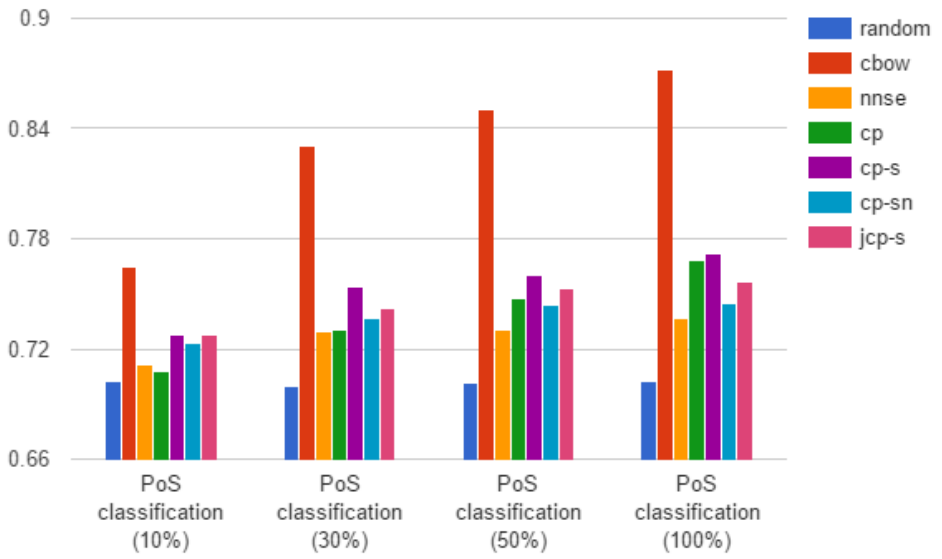


Figure 4.5: Scores on word Part of Speech Classification (a syntactic task)

We conjecture that this vast discrepancy in syntactic information encoded is due to the highly *local* training nature of `word2vec` compared to the global nature of the other methods being compared here. Recall that Syntax is defined as the rules that govern how sentences are constructed and how words are placed in relation to each other in a sentence.

The word embedding methods trained on global information (NNSE and tensor-based methods) all follow two steps:

1. Gather some global count-based data structure (i.e., a tensor)

2. Factorize it under some constraints

In our case, the global data structure is the N-way PPMI tensor of word pairs/word groups. In contrast, `word2vec` is trained using a local training procedure: the model loops over all sentences and records information about each word and its surrounding context. Because of this, we conjecture that `word2vec` gets a much better sentence-level picture of how sentences are constructed, how nouns and verbs interact, etc. All of this information feeds into the quality of syntactic information encoded. With the global count methods, we are destroying this sentence-level information by storing all co-occurrence information in one global data structure and thereby ignoring the information of how the sentence

is constructed.

To illustrate this point, let us consider an example sentence:

the frosting was just the icing on the cake.

In global-based models, the training procedure for this sentence would be to first gather all pairs (or groups of N) words close enough in this sentence ((*frosting, was*), (*icing, cake*), (*it, cake*), ...). CBOW, however, loops through this entire sentence one word w at a time, averages the context words surrounding w, and predicts w given its surrounding words. For example, for the word *was*, the input to the neural network is (*the, frosting, just, the, icing, on, the, cake*). We can see that sentence-level information about subjects, objects, etc. (indicative of syntactic information) is taken into account in a way that is not by methods that only consider global counts of words.

Thus, when choosing embedding for a syntactic task (such as Part of Speech Tagging), a more locally trained embedding method such as `word2vec` will likely perform better. Further, embeddings which are trained specifically using syntactic or morpheme/character-level information such as Morpheme-Enhanced Word Embedding (MWE), have been shown to encode even more syntactic information than `word2vec` [Xu and Liu, 2017], so that could be an even better embedding for syntactic information.

We will discuss potential improvements to our tensor-based methods to encode more syntactic information in Section 5.1, but that is left to future work.

Key points. Based on these results, we make three observations:

1. Tensor-based methods encode useful information readily, potentially more so than existing methods

2. Sparse embeddings should be seriously considered for downstream tasks, as embedded information quality is not sacrificed for sparsity, and in fact can be *improved* by sparsity

3. When solving a syntactic task, use a locally trained embedding such as `word2vec` or MWE [Mikolov et al., 2013a, Xu and Liu, 2017]

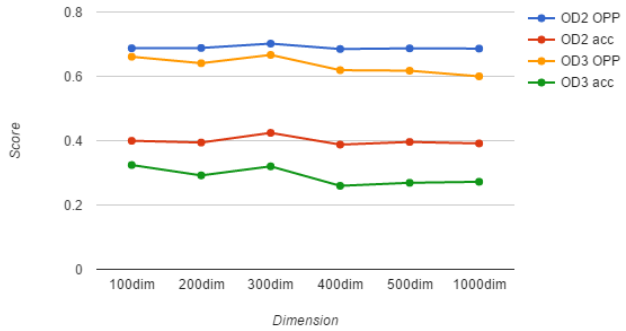


Figure 4.6: Outlier detection quality vs. embedding dimension for JCP-S

4.3 Choice of hyperparameters

Embedding Dimension. The choice of embedding dimension for a word embedding is often difficult. Interestingly, it has been empirically shown that embedding quality does *not* monotonically increase with embedding dimension [Murphy et al., 2012], and a sweet spot for many (if not all) embeddings appears to be around $k = 300$.

A comparison of outlier detection quality vs. embedding dimension for JCP-S can be found in Figure 4.6.

As we can see, outlier detection quality does not at all seem positively correlated with embedding dimension, and in fact for OD3 OPP, the score actually seems to *decrease* as the embedding dimension increases. Still, we can see 300-dimensional vectors work best overall at outlier detection, aligning with similar results found in many other papers. Perhaps some theoretical analysis is needed to state why word vectors tend to work best around dimension 300.

(Shift)	**OD2 OPP**	**OD2 acc**	**OD3 OPP**	**OD3 acc**
$-\log 1$	0.6723	0.3850	0.6547	0.3535
$-\log 5$	0.7047	0.4200	0.6725	*0.3566*
$-\log 10$	*0.7074*	*0.4279*	0.6736	0.3483
$-\log 15$	**0.7078**	**0.4370**	*0.6741*	**0.3597**
$-\log 20$	0.7060	0.4243	0.6693	0.3452
$-\log 25$	0.7042	0.4142	0.6645	0.3420
$-\log 50$	0.6960	0.4224	**0.6762**	0.3545

Table 4.2: Outlier detection scores for various PPMI shifts of CP-S. Second highest values are italicized.

Shifted PPMI. Also, we will compare the quality of embeddings given by factorization of a Shifted PPMI tensor, varying with shift amount. In this formulation (using the idea from [Levy and Goldberg, 2014]), the PPMI tensor $\mathcal{M}$ is shifted so each value gets reassigned by:

$$m_{ijk}^{(new)} := m_{ijk}^{(old)} - \log k$$

We present the resulting outlier detection scores in Table 4.2 across various shifts for CP-S (the embedding for which shifting was most influential), and observe that a shift of $k = 15$ is about optimal for this embedding. A similar increase in performance for CP-S was shown across the other evaluation metrics. $k = 15$ was the best shift setting we found for our CP embedding as well.

It is interesting to notice how sensitive CP-S is to differently shifted values – the best and worst performances were given by shifts of $k = 15$ and $k = 1$ respectively, resulting in an absolute accuracy increase of 5.2% (a relative increase of 12.7%). Such a difference is quite considerable for such a simple hyperparameter.

Interestingly, we also observed that a shift of $k = 15$ is *not* optimal for all of our embeddings. In fact, across all of the various levels of shifting we tried for JCP-S and CP-SN, any shift other than $k = 1$ (equivalent to not shifting at all) caused the produced embedding to perform worse across nearly all metrics. Thus, shift is specific to word embedding method, and experiments should be done for each different type of embedding to find an optimal setting. And as shown by CP-S, very significant quality gains can be attained by finding proper hyperparameter settings.

4.4 Qualitative evaluation & properties

Word embeddings given by tensor factorization have a number of interesting properties. We will now go over a few of the properties that we found.

4.4.1 Polysemy

A word is *polysemous* if it means different things in different contexts. For example, the word 'bank' can be used as a reference to a financial institution or the side of a river.

As discussed briefly in Section 1.4, word embeddings trained by tensor factorization can have more discriminatory power for polysemous words compared to word embeddings trained on pairwise data. We will demonstrate that this information is actually encoded directly into the vectors themselves in a natural way.

Recall that the PPMI for a triple (u, v, w) is predicted by a tensor model by:

$$PPMI(w_u, w_v, w_w) \approx \sum_{i=1}^{k} u_i v_i w_i$$

Another way to write this is:

$$PPMI(w_u, w_v, w_w) \approx \langle \mathbf{u} * \mathbf{v}, \mathbf{w} \rangle$$

where multiplication is taken elementwise. Using this fact, we can deduce a word's multiple meanings. For example, the word *pearl* can be used in the context of the band *Pearl Jam* or in context of *Pearl Harbor* (amongst others). While *bombardment* may be likely to appear in the context of *Pearl Harbor*, it is unlikely to appear in the context of *Pearl Jam*. In other words, based on PPMI prediction, $\langle \overrightarrow{pearl} * \overrightarrow{harbor}, \overrightarrow{bombardment} \rangle$ is high, whereas $\langle \overrightarrow{pearl} * \overrightarrow{jam}, \overrightarrow{bombardment} \rangle$ is low. Using this fact, we can discern that *pearl* has multiple meanings. In fact, $\overrightarrow{pearl} * \overrightarrow{harbor}$ gives a vector that can be used for the word *pearl* which has the property that, when dotted with another word w, approximates PPMI(*pearl, w*), where *pearl* is in the *Pearl Harbor* sense. This is an example of a "phrase vector" for the phrase *pearl harbor*.

Specific examples of phrase vectors given by $u * v$ along with their nearest neighbors (in cosine similarity) for JCP-S and CBOW are presented in Table 4.3. As shown in the table, CBOW does not have the same property. Similar results hold for other pairwise embeddings we tried.

Thus, we can compute multiple different vectors of meaning for polysemous words by simple elementwise multiplication in a way that cannot be done with existing methods.

Phrase vector	**Nearest neighbors (JCP-S)**	**Nearest neighbors (CBOW)**
$\overrightarrow{pearl} * \overrightarrow{bombardment}$	$\overrightarrow{marines}, \overrightarrow{airborne}, \overrightarrow{carriers}$	$\overrightarrow{mainland}, \overrightarrow{maiden}, \overrightarrow{touring}$
$\overrightarrow{pearl} * \overrightarrow{jam}$	$\overrightarrow{drummer}, \overrightarrow{recording}, \overrightarrow{bassist}$	$\overrightarrow{mainland}, \overrightarrow{abroad}, \overrightarrow{maiden}$
$\overrightarrow{bat} * \overrightarrow{baseball}$	$\overrightarrow{leagues}, \overrightarrow{teams}, \overrightarrow{clubs}$	$\overrightarrow{ago}, \overrightarrow{workers}, \overrightarrow{baron}$
$\overrightarrow{bat} * \overrightarrow{creature}$	$\overrightarrow{bird}, \overrightarrow{aerial}, \overrightarrow{tall}$	$\overrightarrow{celtic}, \overrightarrow{brazil}, \overrightarrow{calendar}$
$\overrightarrow{star} * \overrightarrow{wars}$	$\overrightarrow{fictional}, \overrightarrow{comics}, \overrightarrow{character}$	$\overrightarrow{exchange}, \overrightarrow{encouraging}, \overrightarrow{tree}$
$\overrightarrow{star} * \overrightarrow{planet}$	$\overrightarrow{galaxy}, \overrightarrow{earth}, \overrightarrow{minor}$	$\overrightarrow{fingers}, \overrightarrow{layer}, \overrightarrow{arm}$

Table 4.3: Phrase vectors and their nearest neighbors

4.4.2 Nearest neighbors

Also, it is interesting to note the difference in nearest neighbors between tensor factorization embeddings and pairwise embeddings. It is well-known that popular existing embeddings cluster words based on shared meaning. For example, the nearest neighbors in CBOW of *queen* include *regent, lady, duchess, monarch.*

However, an interesting property that we noticed about vectors given by tensor factorization is that their neighbors are not only words that have similar meaning but also words that appear nearby in the corpus. For example, the nearest neighbors in CP-S of *queen* are *elizabeth, monarch, charles, lady, scotland.* Similarly, the nearest neighbors in CP-SN of *professor* are *visiting, associate, lecturer, harvard.* This is potentially advantageous in applications which must detect and/or classify common phrases in text, as vectors not only carry semantic meaning but vectors for words which appear nearby will be nearby in the vector space as well.

Other potential applications will be discussed in Section 4.6.

4.5 Advantages and Disadvantages of tensor-based word embeddings

4.5.1 Advantages

Naturally, much more information is encoded in a N-way tensor than a matrix. If a set of vectors capture the information in a co-occurrence-based tensor well, they can theoretically encode much more information than a set of vectors of the same size that just capture the information stored in a matrix. This is evidenced by the fact that supervised models trained using tensor-based embeddings performed better (in data-poor domains) than those trained using pairwise information: more information is encoded to reconstruct a tensor compared to a matrix, and such information is readily available for supervised tasks to learn from.

Also, our implementation of CP-SN produced an embedding which was only 35% dense, compared to CBOW which is 100% dense. Despite CBOW requiring more space, CP-SN still outperformed CBOW on a number of metrics (such as MTurk, Outlier Detection, and Sentiment analysis). In this sense, CP-SN encodes certain types of information more effectively (105 nonzeroes out of 300), since CBOW needed about 3 times the amount of information (all 300 entries are nonzero for each vector). We are thus in a way compressing the information encoded in dense vectors into a sparse representation which requires less memory.

Tensor factorization methods come with the added benefit of being able to utilize elementwise multiplication to create a "pair vector" which can induce a vector of single meaning for a polysemous word. As illustrated in Table 4.3, different meanings of a word can be inferred quite naturally given some basic information about what other words appear nearby that word.

Also, unlike CBOW, our Tensor factorization methods encode global information rather than purely local information, since the full PPMI tensor (based on global co-occurrence counts) is constructed before vector training. Our tensor methods therefore more effectively utilize the global statistics of the corpus than local methods like CBOW. Because of this, tensor factorization methods tend to encode more semantic information than existing local methods such as `word2vec`. However, this comes at the cost of destroying much of the syntactic

information that is locally encoded in small asymmetric context windows.

4.5.2 Disadvantages

A clear disadvantage to tensor factorization for word embeddings is the size of the data when working with tensor problems. The tensors we have to use in NLP are incredibly large and sparse, and the tensor size and sparsity both scale superlinearly with vocabulary size (and thus corpus size). Because of the nature of higher order information, it requires much more space and time to gather, store, and factorize such data.

However, the redeeming quality of this is that pre-trained vectors only need to be trained once – after finding optimal training settings, we can train the model once on a very large dataset over a long period of time, then release those pre-trained vectors to the public once finished.

Also, as the number of dimensions increases, the quality of the embedding by no means increases. For instance, consider a dimension-10 PPMI tensor. It is highly unlikely that *any* groups of 10 words will have a high PMI, since there probably aren't even any groups of 10 words that co-occur in different sentences *at all*. So we are still limited to a small set of tensor dimensions that make sense to factorize.

Also, we noticed that the quality of embeddings we got from tensor factorization-based methods is highly sensitive to choice of hyperparameters. For example, as observed earlier, shifting the PPMI tensor from 0 to $\log 15$ caused a relative increase in performance on some tasks of over 12% for CP-S, whereas any shift for CP-SN or JCP-S only decreased performance. Also, regularizing too heavily (even slightly) or adding too many negative samples (as described earlier) can cause the model to learn the zero embedding (in which every vector is the zero vector). Further, time required to train each model (2-3 hours on the development set of 130 million tokens) prohibits any sort of grid search for optimal hyperparameters. Because of this, all hyperparameter settings we found were found via a combination of intuition and trial-and-error.

Recall that all of our tensor-based methods do worse at encoding syntactic information than semantic information. We will suggest future improvements to

tensor-based embeddings to encode more syntactic information in Section 5.1.

4.6 Recommended applications

Recommended applications of tensor decomposition embeddings are, quite naturally, those that require relations between groups of words.

Named Entity Recognition. Named Entity Recognition (NER) is the problem of detecting and classifying named entities in a body of text. An example sentence to classify is "The New York Times released a scathing article about Donald Trump" which would be correctly classified as "The [New York Times]$_{\text{Organization}}$ released a scathing article about [Donald Trump]$_{\text{Person}}$". In NER, if one knows the maximum length of any named entity is, say, 4 words, perhaps JCP-S trained to jointly factorize 2-order, 3-order, and 4-order tensors would be appropriate to capture the many relations that different-order groups of words can have. Further, PPMI prediction could be used (or learned) as a feature to predict likelihood of n-way co-occurrence. In our example, PPMI(*New*, *York*, *Times*) is high, meaning $\langle \overrightarrow{New} * \overrightarrow{York}, \overrightarrow{Times} \rangle \approx$ PPMI(*New*, *York*, *Times*) would also be high, potentially indicating that those words constitute a named entity.

Identifying word sense. Also, as shown previously, our vectors directly capture word sense information (without being explicitly trained to do so) in a way that is not captured in CBOW/matrix factorization-based methods. Because of this, a good application for these vectors could be word sense induction (where all senses of a word must be *gathered* from a corpus of documents) or word sense disambiguation (where the specific sense of a word must be *classified* given context).

Limited data domains. Finally, as shown previously in this section, our tensor-based methods (especially CP-SN) useful encode information very readily, moreso than some current state-of-the-art embeddings such as CBOW. Thus, applications with little training data may benefit from tensor-based methods.

Chapter 5

Conclusion

In this work, we have generalized the popular notion of matrix factorization in word embedding literature to tensor factorization, presenting four competitive generic word embeddings. We have explored properties of such embeddings, and showed experimentally that they outperform state-of-the-art baselines such as `word2vec` on a number of recently proposed evaluation tasks, especially at encoding useful semantic information for supervised tasks when training data is limited.

We have also presented a new method to evaluate the degree to which an embedding captures N-way information, and showed that, as intuition would suggest, tensor-based methods perform better at capturing higher-order relations than embeddings trained solely on second order information.

To construct our new word embeddings, we formulated a novel joint tensor factorization problem in which multiple supersymmetric tensors with the same axis lengths are jointly decomposed using a single factor matrix. This is meant to capture in the factor matrix differently-ordered relations among the information in the tensors. We implemented a generic online CP Decomposition library written in TensorFlow, and released all source code and pre-trained vectors to the public.

5.1 Future Work

There are multiple promising directions for future work in this area.

1. **Other tensor decompositions.** The only tensor decomposition we explored in this work is the CP Decomposition – however, there are many more tensor factorizations available that all have their own different advantages and disadvantages. It would be interesting to compare the embeddings produced by various types of different tensor factorizations, such as HOSVD [Kolda and Bader, 2009], Tensor Train [Oseledets, 2011], etc. and compare to the CP Decomposition-based embeddings provided in this thesis.

2. **More accurate tensor decomposition.** Also, in our experiments, generating a better tensor fit almost always led to a better word embedding. In this respect, our CP embedding could be improved, as we only implemented it using the CP-ALS algorithm due to the high memory required by other less naive algorithms. More direct/theoretically grounded decomposition methods such as that of [Chi and Kolda, 2012, Sharan and Valiant, 2017] would be desirable in achieving a better tensor fit.

   [Chi and Kolda, 2012] suggests the use of a Poisson KL divergence-based loss function, based on the idea that the Poisson model better models sparse nonnegative data, as in the case of our tensors. We tried experimenting with this loss function rather than squared error loss, and found that embedding quality was unchanged across all methods when comparing training objectives, so we continued to use the simpler to interpret squared error loss function. However, we were unable to utilize their specific CP-APR algorithm due to its memory requirements which are infeasible because of how many nonzero entries we have. Because the Poisson model naturally better suits the data in such sparse nonnegative tensors, it is likely that either an implementation that has feasible memory requirements would lead to gains in tensor fit, and thus embedding quality.

3. **JCP-SN.** Also, in the spirit of turning CP-S into CP-SN, we would like to create a sparse nonnegative version of JCP-S in order to capture multiple dimensions of relations in a sparse nonnegative manner. We briefly experimented with nonnegative joint embeddings, but we could not produce

a meaningful embedding from such experiments. Further experiments on hyperparameter settings are required, as we demonstrated how sensitive these embeddings can be to hyperparameter choice.

4. **Hyperparameters.** A more theoretical understanding of hyperparameter choice would be ideal. Our best hyperparameter settings were found primarily by intuition and trial-and-error. We have also only extensively studied tensors of order 3 or less. It would be interesting and most likely fruitful in some respects to study tensors of higher order, at least up to order 4 or 5.

5. **Asymmetric tensors.** In this work, we have only considered factorization of symmetric tensors. Construction of an asymmetric tensor to apply asymmetric tensor decomposition would be an interesting problem. For example, a tensor of n-grams counts ($x_{ijk} = f(\#(w_i, w_j, w_k \text{ appears in a sentence}))$, where f could be $f(x) = \log(1 + x)$ or some other appropriately scaled function such as that of [Pennington et al., 2014]) would be a reasonable n-dimensional asymmetric tensor to factorize.

   Such asymmetric approaches would likely encode much more syntactic information than current approaches, since syntactic information is highly dependant on word order and local sentence information (which is currently ignored by our global co-occurrence based models).

6. **Non tensor-based higher-order word embedding ideas.** We have discussed a number of other ideas which would produce a word embedding using higher-order information that are not necessarily based on tensor factorization.

   a. **Tensor eigenvector contraction.** We noticed that the word-word co-occurrence matrix can be computed from a word-word-word co-occurrence tensor by summing along a single axis. This is equivalent to contracting the tensor by a vector of all 1's (weighting all slices equally). Instead of this, we could first compute the leading eigenvector of a supersymmetric co-occurrence tensor and then contract along an axis using that eigen-

vector as corresponding weights. This would give us a more intelligently weighted word-word co-occurrence matrix, from which we could compute a PPMI matrix, log-based co-occurrence matrix, etc. to factorize using traditional methods such as SVD, following the example of many matrix-based word embeddings [Murphy et al., 2012, Pennington et al., 2014, Levy and Goldberg, 2014].

b. **Hypergraph embedding theory.** A Hypergraph is a tuple of vertex and edge sets (V, E) where the edges connect *groups* of vertices rather than just pairs of vertices (as in the case of a normal graph). Just as there is a theory of graph embedding, there is a theory of hypergraph embedding [Zhou et al., 2006]. Using this theory (barring any computational constraints), we could first construct a hypergraph based on sentences (i.e., $(w_i, w_j, w_k) \in E \Leftrightarrow w_i, w_j, w_k$ appear in the same sentence) and use hypergraph embedding to embed this hypergraph into $\mathbb{R}^k$. Utilizing such a hypergraph could also lead to encoding many different orders of information by considering hypergraphs which simultaneously have edges with pairs of nodes, groups of 3 nodes, etc. (as opposed to a k-regular hypergraph in which each edge connects k vertices).

5.2 Closing statement

Tensor factorization appears to be a highly applicable and effective approach to learning generic word embeddings. It is advantageous to encode many different types of information in word embeddings, so we should not restrict ourselves to solutions which only are trained on pairwise word relations. There seems to be much information encoded in tensor-based word embedding vectors that is not encoded in the vectors produced by matrix-based word embeddings, and such information will likely prove useful in downstream NLP tasks.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Acar et al., 2011] Acar, E., Kolda, T. G., and Dunlavy, D. M. (2011). All-at-once optimization for coupled matrix and tensor factorizations. *CoRR*, abs/1105.3422.

[Bader et al., 2015] Bader, B. W., Kolda, T. G., et al. (2015). Matlab tensor toolbox version 2.6. Available online.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Barkan and Koenigstein, 2016] Barkan, O. and Koenigstein, N. (2016). Item2vec: Neural item embedding for collaborative filtering. *CoRR*, abs/1603.04259.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

[Blair et al., 2016] Blair, P., Merhav, Y., and Barry, J. (2016). Automated generation of multilingual clusters for the evaluation of distributed representations. *CoRR*, abs/1611.01547.

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

[Camacho-Collados and Navigli, 2016] Camacho-Collados, J. and Navigli, R. (2016). Find the word that does not belong: A framework for an intrinsic evaluation of word vector representations. In *ACL Workshop on Evaluating Vector Space Representations for NLP*, pages 43–50. Association for Computational Linguistics.

[Chi and Kolda, 2012] Chi, E. C. and Kolda, T. G. (2012). On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299.

[Cichocki et al., 2009] Cichocki, A., Zdunek, R., Phan, A. H., and Amari, S.-i. (2009). *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley Publishing.

[Cover and Thomas, 2006] Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience.

[de Silva and Lim, 2008] de Silva, V. and Lim, L.-H. (2008). Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Anal. Appl.*, 30(3):1084–1127.

[Faruqui et al., 2016] Faruqui, M., Tsvetkov, Y., Rastogi, P., and Dyer, C. (2016). Problems with evaluation of word embeddings using word similarity tasks. *CoRR*, abs/1605.02276.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653.

[Harris, 1954] Harris, Z. (1954). Distributional structure. *Word*, 10(23):146–162.

[Håstad, 1990] Håstad, J. (1990). Tensor rank is np-complete. *Journal of Algorithms*, 11(4):644 – 654.

[Jastrzebski et al., 2017] Jastrzebski, S., Lesniak, D., and Czarnecki, W. M. (2017). How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM REVIEW*, 51(3):455–500.

[Levy and Goldberg, 2014] Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 2177–2185, Cambridge, MA, USA. MIT Press.

[Liu et al., 2012] Liu, J., Liu, J., Wonka, P., and Ye, J. (2012). Sparse non-negative tensor factorization using columnwise coordinate descent. *Pattern Recogn.*, 45(1):649–656.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

[Murphy et al., 2012] Murphy, B., Talukdar, P. P., and Mitchell, T. M. (2012). Learning effective and interpretable semantic models using non-negative sparse embedding. In Kay, M. and Boitet, C., editors, *COLING*, pages 1933–1950. Indian Institute of Technology Bombay.

[Naskovska and Haardt, 2016] Naskovska, K. and Haardt, M. (2016). Extension of the semi-algebraic framework for approximate CP decompositions via simultaneous matrix diagonalization to the efficient calculation of coupled CP decompositions. In *50th Asilomar Conference on Signals, Systems and Computers, ACSSC 2016, Pacific Grove, CA, USA, November 6-9, 2016*, pages 1728–1732.

[Ng, 2017] Ng, P. (2017). dna2vec: Consistent vector representations of variable-length k-mers. *CoRR*, abs/1701.06279.

[Oseledets, 2011] Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317.

[Pang and Lee, 2004] Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

[Schnabel et al., 2015] Schnabel, T., Labutov, I., Mimno, D. M., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In Màrquez, L., Callison-Burch, C., Su, J., Pighin, D., and Marton, Y., editors, *EMNLP*, pages 298–307. The Association for Computational Linguistics.

[Sharan and Valiant, 2017] Sharan, V. and Valiant, G. (2017). Orthogonalized als: A theoretically principled tensor decomposition algorithm for practical use. *arXiv preprint arXiv:1703.01804*.

[Spearman, 1904] Spearman, C. (1904). "General Intelligence," objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292.

[Tasse and Dodgson, 2016] Tasse, F. P. and Dodgson, N. A. (2016). Shape2vec: semantic-based descriptors for 3d shapes, sketches and images. *ACM Transactions On Graphics*.

[Van de Cruys, 2009] Van de Cruys, T. (2009). A non-negative tensor factorization model for selectional preference induction. In *Proceedings of the Workshop on Geometrical Models of Natural Language Semantics*, GEMS '09, pages 83–90, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Van de Cruys, 2011] Van de Cruys, T. (2011). Two multivariate generalizations of pointwise mutual information. In *Proceedings of the Workshop on Distributional Semantics and Compositionality*, DiSCo '11, pages 16–20, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Van de Cruys et al., 2013] Van de Cruys, T., Poibeau, T., and Korhonen, A. (2013). A tensor-based factorization model of semantic compositionality. In *Proc. of NAACL-HLT*, pages 1142–1151.

[Xu and Liu, 2017] Xu, Y. and Liu, J. (2017). Implicitly incorporating morphological information into word embedding. *CoRR*, abs/1701.02481.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.

[Zhou et al., 2006] Zhou, D., Huang, J., and Schölkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in Neural Information Processing Systems (NIPS) 19*, page 2006. MIT Press.