Word Embeddings via Tensor Factorization

Eric Bailey
Tufts University
Medford, MA
ericbailey94@gmail.com

Shuchin Aeron
Tufts University
Medford, MA
shuchin@ece.tufts.edu

Abstract

Most popular word embedding techniques involve implicit or explicit factorization of a word co-occurrence based matrix into low rank factors. In this paper, we aim to generalize this trend by using numerical methods to factor higher-order word co-occurrence based arrays, or *tensors*. We present four word embeddings using tensor factorization and analyze their advantages and disadvantages. One of our main contributions is a novel joint symmetric tensor factorization technique related to the idea of coupled tensor factorization. We show that embeddings based on tensor factorization can be used to discern the various meanings of polysemous words without being explicitly trained to do so, and motivate the intuition behind why this works in a way that doesn't with existing methods. We also modify an existing word embedding evaluation metric known as Outlier Detection [Camacho-Collados and Navigli, 2016] to evaluate the quality of the order-N relations that a word embedding captures, and show that tensor-based methods outperform existing matrix-based methods at this task. Experimentally, we show that all of our word embeddings either outperform or are competitive with state-of-the-art baselines commonly used today on a variety of recent datasets. Suggested applications of tensor factorization-based word embeddings are given, and all source code and pre-trained vectors are publicly available online.

Keywords: Natural langauge processing, word embedding, tensor factorization, CP decomposition

1 Introduction

1.1 Word embeddings

Word embeddings are an important part of NLP, and are used almost everywhere. 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., 2013] 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. For example, typically words w_1 and w_2 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 also encode many different types of semantic information. For instance, in the `word2vec` model [Mikolov et al., 2013], we can answer analogy queries of the form $a : b :: c : ?$ using simple vector arithmetic: the answer to bed : sleep :: chair : x is given by the vector closest to $\eta(\text{sleep}) - \eta(bed) + \eta(\text{chair})$ ($\approx \eta(\text{sit})$). 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.

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 solutions using word embeddings. A list of many of the "*2vec"s can be found online[1].

Thus, even incremental improvements on word embedding architectures may lead to further improvements across many areas. However, many recent word embedding papers have been adapted to a specific task, and there has been evidence that no single embedding performs best across all benchmarks [Jastrzebski et al., 2017].

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 paper we work with the distributional hypothesis, but work with higher-order co-occurrence information. Next, we will outline the work related to our methodology, then we will outline our main contributions.

1.2 Related Work

In the context of word embeddings, `word2vec` [Mikolov et al., 2013] is by far the most the popular and most widely used word embedding. Specifically, in their experiments, the Skip-gram model with Negative Sampling (SGNS) was shown to perform the best on a number of benchmarks, compared to their other model in the same paper (CBOW). While there have been advances in the field of generic word embeddings, `word2vec` is still the most widely used.

[Levy and Goldberg, 2014] approaches SGNS from a matrix factorization perspective - they show SGNS is implicitly factoring a certain type of matrix (PPMI). They further *explicitly* decompose this matrix and compare the results, showing comparable (and in some cases superior) performance to SGNS. We will discuss their explicit factorization methodology (and generalize it) in Sections 3 and 4.

Tensor factorization has been directly applied in the past to NLP, particularly in the domain of identifying and understanding (Subject, Verb, Object) (SVO) triples [Hashimoto and Tsuruoka, 2016, Fried et al., 2015, Van de Cruys, 2009]. n-way PPMI has also been applied in the context of NLP [Van de Cruys, 2011, Villada Moiron and Maria Begona, 2005].

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. Our work studies word embeddings given by CP decomposition much more in-depth, and we are factoring a different kind of tensor.

1.3 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 the intuition behind some of the new properties in Section 3 and show that they actually encode such properties in Section 6. Suggested applications for such embeddings are outlined in Section 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 cre-

[1] `https://gist.github.com/nzw0301/333afc00bd508501268fa7bf40cafe4e`

ate 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., 2013]) on OD3. Further, our joint tensor factorization method (meant to capture multiple different orders of information) performs competitively with or outperforms other state-of-the-art models on both 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[2]. We have implemented asymmetric, symmetric, and joint online CP decomposition, including both GPU and CPU support.

1.4 Paper organization

The rest of the paper is organized as follows. In Section 2 we will explicitly lay out the mathematical formulations required to understand our methodology. We then motivate the intuition behind our novel word embedding architectures in Section 3. Then in Section 4 we will explicitly lay out our new word embedding formulations, and introduce the idea of *Joint Symmetric Tensor Factorization*. In Section 5 we will give explicit implementation details of our approach, including computational optimizations, specific software packages used, and hyperparameter suggestions. Finally, in Section 6 we will outline the current state of evaluation of word embeddings, generalize Outlier Detection [Camacho-Collados and Navigli, 2016] to determine how much n-way information ($n \geq 2$) a word embedding captures, report our results on a number of evaluation metrics, and qualitatively show some of the sorts of new information that is captured by higher-order models.

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

2 Mathematical preliminaries

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

2.1 Pointwise Mutual Information

Pointwise mutual information (PMI) is a measure very useful in NLP. We give some of its past uses in NLP in Section 1.2 and we will discuss its utility in Section 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], 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) deal with 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 3 for a more explicit formulation.

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 paper we choose to use this version.

In the same way one can construct a PPMI matrix, we study n-way PPMI *tensors*.

2.2 Tensors

2.2.1 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 paper 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 $\mathbf{M}$ ($\in \mathbb{R}^{J \times I_n}$).

2.2.2 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 $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$. The most common tensor factorizations algorithms decompose $\mathcal{X}$ into a small core tensor $\mathcal{S} \in \mathbb{R}^{I' \times J' \times K'}$ 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

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

Such a decomposition is called the *Tucker decomposition*.

In this paper, we study a special case of the Tucker decomposition called the *CANDECOMP/PARAFAC (CP) decomposition*, in which $\mathcal{S}$ 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. Notice that in the case of a 2-way tensor (matrix), the CP decomposition is given by

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

where $\mathbf{\Sigma}$ 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, as 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:

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

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 $\mathcal{X} \approx [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$.

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 $\mathcal{X}$, and the superdiagonal core $\mathcal{S}$ (= 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 $\mathcal{M} \in \mathbb{R}^{I \times I \times I}$, 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 $\mathcal{M} \in \mathbb{R}^{I \times I \times I}$ such that $\mathcal{M} \approx [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$, we instead consider the problem of factorizing $\mathcal{M}$ into the triple product of a single factor matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ such that

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

This works because, if $\mathcal{M}$ 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. There are more advanced methods to compute the CP decomposition [Chi and Kolda, 2012, Sharan and Valiant, 2017], but 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 Sections 4 and 5.

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

3 Background and Intuition

3.1 The Utility of PPMI

Words x and y will have high 2-way PPMI if they frequently appear together (moreso 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.

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.

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

3.2 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, infor-

mation 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 *handicap*, *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 *handicap* tends to also pairwise co-occur with the words *spot* and *park*, a simply pairwise model might (wrongly) assign *handicap* a similar vector to the vector it assigns *sun*. But a third order model may discriminate between the two, since the triple (*handicap*, *spot*, *park*) will often co-occur, but the triple (*sun*, *spot*, *park*) will not.

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*).

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.

4 Methodology

In this section we present our novel word embeddings based on CP Decomposition.

4.1 CP Decomposition model (CP)

We follow a similar construction to that of [Levy and Goldberg, 2014].

Re-formulated in CP Decomposition notation, they decompose the 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 $\mathcal{M}$, and factorize it via the rank-k CP decomposition such that

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

Where $\mathbf{U}^{(i)}$ has orthonormal columns and λ is the vector of weights corresponding to each rank-one approximation.

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 will discuss the actual contents of the tensor in Section 6 . We can also consider shifting the PPMI tensor, following the example of [Levy and Goldberg, 2014]. We show (empirically) that using a shift of $\log 15$ produces a better word embedding than not shifting for CP (and CP-S), so we also consider factorization of 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.

4.2 Symmetric CP Decomposition (CP-S)

Since PMI is symmetric 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 learning 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**.

4.2.1 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.

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:

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

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

Nonnegative word embeddings have been studied in the past [Murphy et al., 2012]. Previous work presents a nonnegative sparse embedding called NNSE, which is based on sparse nonnegative matrix factorization. CP-SN can be seen as a generalization of NNSE, as CP-SN is given by a sparse nonnegative PPMI *tensor* factorization.

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

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.

4.2.2 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 $\mathcal{M}^{(n)}$ 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:

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

where $\mathcal{M}_n \in \mathbb{R}^{\overbrace{|V| \times \cdots \times |V|}^{n \text{ times}}}$ is an n-dimensional symmetric PPMI tensor, so we are decomposing $N-1$ such tensors.

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

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

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

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

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

In general, Joint Tensor decomposition has been studied in the past [Acar et al., 2011, Naskovska and Haardt, 2016] (also known as *Coupled/Linked/Multiple/Multi-* Tensor Decomposition), where the goal is to simultaneously factorize multiple tensors using one (or more) shared factor matrix. However, this is usually in the context of decomposing a primary tensor with some auxiliary tensors (or matrices) that augment some of the information in the original tensor, where the auxiliary tensors only share indices along a single axis, and only one factor matrix is shared across the decomposition of all tensors, but there are other factor matrices involved.

In our framework, there is only one factor matrix (since all tensors are supersymmetric with the same dimension length), and all tensors being decomposed are treated equally (unless they are separately weighted to more specifically focus on certain orders of relations). To the best of our knowledge, joint *symmetric* tensor factorization (where multiple symmetric tensors are being factorized using a single factor matrix) has not yet been studied.

5 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[3].

[3] `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.

5.1 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.

5.2 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 $\#(y, x, z) := \cdots := \#(z, y, x) := \#(x, y, z)$.

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.

6 Evaluation

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 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[4], so we will go over a few of these evaluation methods and their motivations, and present comparative results for each of them.

6.1 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
2. **CBOW** (`word2vec`) [Mikolov et al., 2013], for comparison against the most commonly used embedding method
3. **NNSE**[5] [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.

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

[5] 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.

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.

6.2 Quantitative evaluation

6.2.1 Outlier Detection

Description. A more recent form of embedding evaluation known as Outlier Detection has been suggested as a gold standard of evaluation [Blair et al., 2016, Camacho-Collados and Navigli, 2016].

In many word embeddings, it is advantageous to cluster words that mean similar things together – Outlier Detection can be seen as a measure of how good word embeddings are at doing this. The word embedding is presented with a list of $n+1$ words, where the first n words share semantic meaning, and the last word is unrelated to all of the first n. The outlier is then detected using vector similarity – the idea is that the first n words should form a tight cluster in the vector space, and the outlier should be unrelated to that cluster.

For each word in each list, a *compactness score* is assigned to the other n words given by the mean cosine similarity between all the other pairs of words. The compactness score should be highest for the outlier, since the n words should form a semantic cluster in the vector space. 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)

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. These outlier detection 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].

Results and discussion. A comparison of outlier detection performance is given in Table 1.

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.

(Method)	OD2 OPP	OD2 acc	OD3 OPP	OD3 acc
Random	0.5819	0.2591	0.5405	0.2127
CBOW	0.6542	0.3731	0.6162	0.3034
NNSE	0.6998	0.4288	0.6292	0.3190
CP	0.6404	0.3421	0.6084	0.2638
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 1: Outlier detection scores across all embeddings

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 7, 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 difference between CP's OD2 OPP and NNSE's OD2 OPP is 5.1%, which is a rather stark difference. However, on OD2, their OPP only differs by 2.1%. Similar observations hold when comparing CP to CBOW on OD2.

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, though we have not yet tried such experiments on higher-order tensors.

6.2.2 Simple supervised tasks

Description. [Jastrzebski et al., 2017] discusses the importance of using simple supervised tasks when evaluating word embeddings. Their main idea is that the primary application of word embeddings is in *transfer learning* to tasks which do not have much supervised data – if supervised data 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 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 – 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., 2013]

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 neural networks even in embeddings that do not encode analogy linearly. For details of the neural network, see [Jastrzebski et al., 2017].

The analogy dataset we use is from the Google analogy testbed [Mikolov et al., 2013], and we filter out all analogies where all 4 words are not in the vocabulary, leaving a total of 4,596 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.

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 is lost by summing the word vectors.

   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.

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

We implemented all code 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].

Results and Discussion. A comparison of the results on the supervised analogy task is found in Figure 1. "Analogy x%" means only x% of the training data was used (3,906 analogies total). Accuracy is calculated on the entire test set (690 analogies).

As we can see, tensor methods are very strong in a limited data setting 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-SN's vectors still manage to achieve an accuracy of 36.49% on the semantic analogy testbed.

To further visualize the difference in information accessibility between the embeddings, we present a graph of semantic analogy accuracy vs. training data in Figure 2. For example, CBOW only starts performing competitively with the others when the NN gets near 100% of the training data, whereas tensor methods started on top. Notice how 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 analogy information encoded in the pairwise models is less easily accessible than in the tensor-based models. For these reasons, we can conclude that the analogy information encoded in the tensor-based embeddings is more easily accessible than that of CBOW.

When trained on 100% of the dataset, CP-SN still outperforms all opposing methods on the semantic testbed but is surpassed by NNSE on the syntactic testbed (and nearly surpassed by CBOW).

A performance comparison on the supervised sentiment analysis is presented in Figure 3.

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. 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).

Also, we notice a similar trend for CBOW as we did in the supervised analogy tasks – to perform at its best comparative ranking, CBOW requires 100% of the training data. Perhaps this is

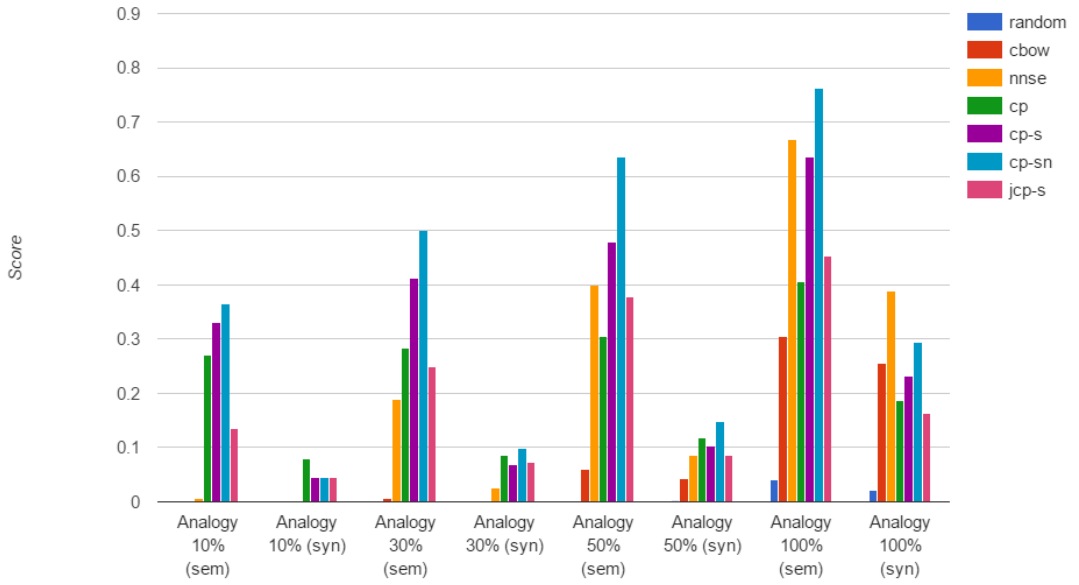

Figure 1: Supervised analogy task performance vs. % training data

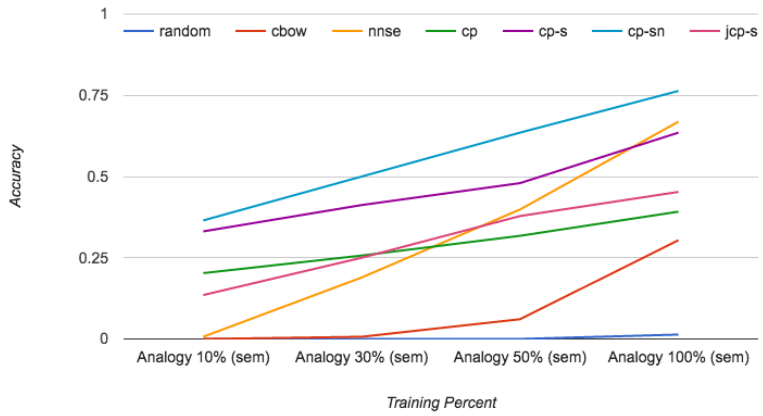

Figure 2: Semantic analogy accuracy vs. % training data

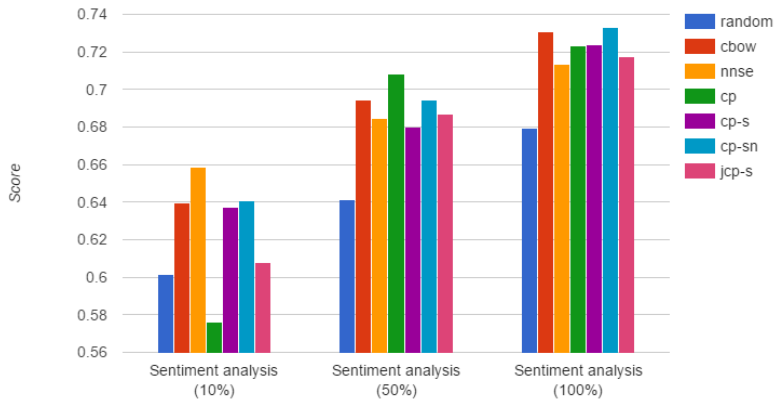

Figure 3: Sentiment analysis task performance vs. % dataset size

an indication of the lack of data accessibility inherent in `word2vec`, at least compared to sparse or tensor-based methods.

Based on these results, we make two observations:

1. Tensor-based methods encode useful information readily
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

6.3 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.

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.

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 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.

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.

6.4 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.

6.4.1 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 3, 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

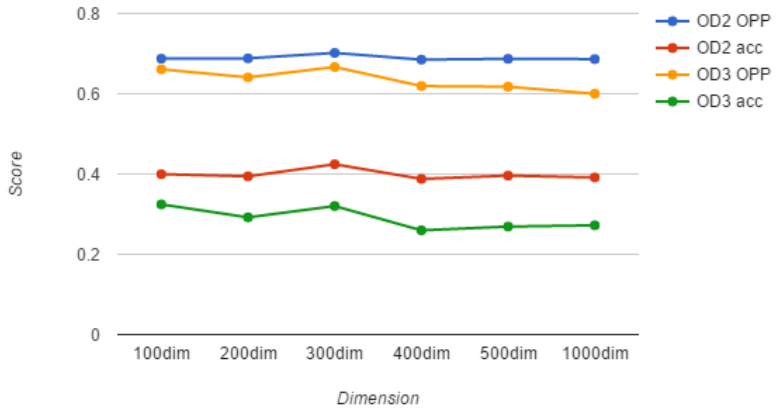

Figure 4: Outlier detection quality vs. embedding dimension for JCP-S

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

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 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.

6.4.2 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

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 3: Phrase vectors and their nearest neighbors

vectors for words which appear nearby will be nearby in the vector space as well.

This brings us to the next section: suggested applications of tensor-based embeddings.

6.5 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.

Word sense problems. 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.

7 Conclusion and Future work

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 information in a setting where 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 expected) 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 ten-

sors with the same axis lengths are jointly decomposed using a single factor matrix. This is meant to capture 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.

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 paper 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, Tensor Train, etc. and compare to the CP Decomposition-based embeddings provided in this paper.

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 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.

Closing remarks. 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.

References

[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.

[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.

[Fried et al., 2015] Fried, D., Polajnar, T., and Clark, S. (2015). Low-rank tensors for verbs in compositional distributional semantics.

[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.

[Hashimoto and Tsuruoka, 2016] Hashimoto, K. and Tsuruoka, Y. (2016). Adaptive joint learning of compositional and non-compositional phrase embeddings. *CoRR*, abs/1603.06067.

[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., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). 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.

[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.

[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*.

[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.

[Villada Moiron and Maria Begona, 2005] Villada Moiron and Maria Begona (2005). *Data-driven identification of fixed expressions and their modifiability*. PhD thesis, University of Groningen.

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