Abstract

We present an unsupervised method to generate \textbf{Word2Sense} word embeddings that are interpretable — each dimension of the embedding space corresponds to a fine-grained \textit{sense}, and the non-negative value of the embedding along the $j$-th dimension represents the relevance of the $j$-th sense to the word. The underlying LDA-based generative model can be extended to refine the representation of a polysemous word in a short context, allowing us to use the embeddings in contextual tasks. On computational NLP tasks, Word2Sense embeddings compare well with other word embeddings generated by unsupervised methods. Across tasks such as word similarity, entailment, sense induction, and contextual interpretation, Word2Sense is competitive with the state-of-the-art method for that task. Word2Sense embeddings are at least as sparse and fast to compute as prior art.

1 Introduction

Several unsupervised methods such as \textit{SkipGram} (Mikolov et al., 2013) and \textit{Glove} (Pennington et al., 2014) have demonstrated that co-occurrence data from large corpora can be used to compute low-dimensional representations of words (a.k.a. embeddings) that are useful in computational NLP tasks. While not as accurate as semi-supervised methods such as BERT (Devlin et al., 2018) and ELMO (Peters et al., 2018) that are trained on various downstream tasks, they do not require massive amounts of compute unaccessible to all but few.

Nearly all such methods produce dense representations for words whose coordinates in themselves have no meaningful interpretation. The numerical values of a word’s embedding are meaningful only in relation to representations of other words. A unitary rotation can be applied to many of these embeddings retaining their utility for computational tasks, and yet completely changing the values of individual coordinates. Can we design an interpretable embedding whose coordinates have a clear meaning to humans?

Ideally such an embedding would capture the multiple \textit{senses} of a word, while being effective at computational tasks that use inter-word spacing of embeddings. Loosely, a sense is a set of semantically similar words that collectively evoke a bigger picture than individual words in the reader’s mind. In this work, we mathematically define a sense to be a probability distribution over the vocabulary, just as topics in topic models. A human can relate to a sense through the words with maximum probability in the sense’s probability distribution. Table 1 presents the top 10 words for a few senses.

We describe precisely such an embedding of words in a space where each dimension corresponds to a sense. Words are represented as probability distributions over senses so that the magnitude of each coordinate represents the relative importance of the corresponding sense to the word. Such embeddings would naturally capture the polysemous nature of words. For instance, the embedding for a word such as \textit{cell} with many senses – e.g. “biological entity”, “mobile phones”, “excel sheet”, “blocks”, “prison” and “battery” (see Table 1) – will have support over all such senses.

To recover senses from a corpus and to represent word embeddings as (sparse) probability distributions over senses, we propose a generative model (Figure 1) for the co-occurrence matrix: (1) associate with each word $w$ a sense distribution $\theta_w$ with Dirichlet prior; (2) form a context around a target word $w$ by sampling senses $z$ according to $\theta_w$, and sample words from the distribution of sense $z$. This allows us to use fast inference tools such as WarpLDA (Chen et al., 2016) to recover few thousand fine-grained senses from large cor-
Table 1: Top senses of polysemous words as identified Word2Sense embeddings. Each row lists the rank of the sense in terms of its weight in the word’s embedding, and the top 10 words in the senses’ probability distribution.

<table>
<thead>
<tr>
<th>Word</th>
<th>Rank</th>
<th>Top 10 words with the highest probability in the sense’s distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tie</td>
<td>1</td>
<td>hitch, tying, magnet, tied, knots, tie, loops, rope, knot, loop</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>shirts, wore, shoes, jacket, trousers, worn, shirt, dress, wearing, wear</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>against, scored, round, 2-1, champions, match, finals, final, win, cup</td>
</tr>
<tr>
<td>Bat</td>
<td>1</td>
<td>species, myotis, roosts, pipistrelle, reservoir, roost, dam, horseshoe, bats, bat</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>smuggling, smoked, cigars, smokers, cigar, smoke, smoking, cigarette, cigarettes, tobacco</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>bowled, bowler, first-class, bowling, batsman, wicket, overs, innings, cricket, wickets</td>
</tr>
<tr>
<td>Apple</td>
<td>1</td>
<td>player, micro, zen, portable, shuffle, mini, nano, mp3, apple, ipod</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>graphics, g4, pc, hardware, pci, macintosh, intel, os, apple, mac</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>vegetables, lemon, grapes, citrus, orange, apple, apples, fruits, juice, fruit</td>
</tr>
<tr>
<td>Star</td>
<td>1</td>
<td>vulcan, archer, picard, enterprise, voyager, starship, spock, kirk, star trek</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>obi-wan, luke, anakin, skywalker, sith, vader, darth, star, jedi, wars</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>cluster, nebula, dwarf, magnitude, nge, constellation, star, stars, galaxies, galaxy</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>inn, guest, star, b&amp;b, rooms, bed, accommodation, breakfast, hotels, hotel</td>
</tr>
<tr>
<td>Cell</td>
<td>1</td>
<td>plasma, cellular, membranes, molecular, cells, molecules, cell, protein, membrane, proteins</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>kinase, immune, gene, activation, proteins, receptors, protein, receptor, cell, cells</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>transfusion, renal, liver, donor, transplantation, bone, kidney, marrow, transplant, blood</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>top, squares, stack, bottom, the, table, columns, row, column, rows</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>inmate, correctional, workhouse, jail, prisoner, hmp, inmates, prisons, prisoners, prison</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>aaa, powered, lin, mains, lithium, aa, rechargeable, charger, batteries, battery</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>handset, bluetooth, ericsson, ringtones, samsung, mobile, phones, phone, motorola, nokia</td>
</tr>
</tbody>
</table>

Our methodology does not suffer from computational constraints unlike Word2GM (Athiwaratkun and Wilson, 2017) and MSSG (Neelakantan et al., 2015) which are constrained to learning 2-3 senses for a word. The key idea that gives us this advantage is that rather than constructing a per-word representation of senses, we construct a global pool of senses from which the senses a word takes in the corpus are inferred. Our methodology takes just 5 hours on one multicore processor to recover senses and embeddings from a concatenation of UKWAC (2.5B tokens) and Wackypedia (1B tokens) co-occurrence matrices (Baroni et al., 2009) with a vocabulary of 255434 words that occur at least 100 times.

Our major contributions include:

- A single prototype word embedding that encodes information about the senses a word takes in the training corpus in a human interpretable way. This embedding outperforms Word2vec in rare word similarity task and word relatedness task and is within 2% in other similarity and relatedness tasks; and outperforms Word2GM on the entailment task of (Baroni et al., 2012).
- A generative model that allows for disambiguating the sense of a polysemous word in a short context that outperforms the state-of-the-art unsupervised methods on Word Sense Induction for Semeval-2010 (Manandhar et al., 2010) and MakeSense-2016 (Mu et al., 2017) datasets and is within 1% of the best models for the contextual word similarity task of (Huang et al., 2012).
2 Related Work

Several unsupervised methods generate dense single prototype word embeddings. These include Word2vec (Mikolov et al., 2013), which learns embeddings that maximize the cosine similarity of embeddings of co-occurring words, and Glove (Pennington et al., 2014) and Swivel (Shazeer et al., 2016) that learn embeddings by factorizing the word co-occurrence matrix. (Dhillon et al., 2015; Stratos et al., 2015) use canonical correlation analysis (CCA) to learn word embeddings that maximize correlation with context. (Levy and Goldberg, 2014; Levy et al., 2015) showed that SVD based methods can compete with neural embeddings. (Lebret and Collobert, 2013) use Hellinger PCA, and claim that Hellinger distance is a better metric than Euclidean distance in discrete probability space.

Multiple works have considered converting the existing embeddings to interpretable ones. Murphy et al. (2012) use non-negative matrix factorization of the word-word co-occurrence matrix to derive interpretable word embeddings. (Sun et al., 2016; Han et al., 2012) change the loss function in Glove to incorporate sparsity and non negativity respectively to capture interpretability. (Faruqui et al., 2015) propose Sparse Overcomplete Word Vectors (SPOWV), by solving an optimization problem in dictionary learning setting to produce sparse non-negative high dimensional projection of word embeddings. (Subramanian et al., 2018) use a $k$-sparse denoising autoencoder to produce sparse non-negative high dimensional projection of word embeddings, which they called SParse Interpretable Neural Embeddings (SPINE). However, all these methods lack a natural extension for disambiguating the sense of a word in a context.

In a different line of work, Vilnis and McCallum (2015) proposed representing words as Gaussian distributions to embed uncertainty in dimensions of the embedding to better capture concepts like entailment. However, Athiwaratkun and Wilson (2017) argued that such a single prototype model can’t capture multiple distinct meanings and proposed Word2GM to learn multiple Gaussian embeddings per word. The prototypes were generalized to elliptical distributions in (Muzellec and Cuturi, 2018). A major limitation with such an approach is the restriction on the number of prototypes per word that can be learned, which is limited to 2 or 3 due to computational constraints.

Many words such as ‘Cell’ can have more than 5 senses. Another open issue is that of disambiguating senses of a polysemous word in a context – there is no obvious way to embed phrases and sentences with such embeddings.

Multiple works have proposed multi-prototype embeddings to capture the senses of a polysemous word. For example, Neelakantan et al. (2015) extends the skipgram model to learn multiple embeddings of a word, where the number of senses of a word is either fixed or is learned through a non-parametric approach. Huang et al. (2012) learns multi-prototype embeddings by clustering the context window features of a word. However, these methods can’t capture concepts like entailment. Tian et al. (2014) learns a probabilistic version of skipgram for learning multi-sense embeddings and hence, can capture entailment. However, all these models suffer from computational constraints and either restrict the number of prototypes learned for each word to 2-3 or restrict the words for which multiple prototypes are learned to the top $k$ frequent words in the vocabulary.

Prior attempts at representing polysemy include (Pantel and Lin, 2002), who generate global senses by figuring out the best representative words for each sense from co-occurrence graph, and (Reisinger and Mooney, 2010), who generate senses for each word by clustering the context vectors of the occurrences of the word. Further attempts include Arora et al. (2018), who express single prototype dense embeddings, such as Word2vec and Glove, as linear combinations of sense vectors. However, their underlying linearity assumption breaks down in real data, as shown by Mu et al. (2017). Further, the linear coefficients can be negative and have values far greater than 1 in magnitude, making them difficult to interpret. Neelakantan et al. (2015) and Huang et al. (2012) represent a context by the average of the embeddings of the words to disambiguate the sense of a target word present in the context. On the other hand, Mu et al. (2017) suggest representing sentences as a hyperspace, rather than a single vector, and represent words by the intersection of the hyperspaces representing the sentences it occurs in.

A number of works use naïve Bayesian method (Charniak et al., 2013) and topic models (Brody and Lapata, 2009; Yao and Van Durme, 2011; Pedersen, 2000; Lau et al., 2012, 2013, 2014) to learn senses from local contexts, treating each in-
stance of a word within a context as a pseudo-document, and achieve state of the art results in WSI task (Manandhar et al., 2010). Since this approach requires training a single topic model per target word, it does not scale to all the words in the vocabulary.

In a different line of work, (Tang et al., 2014; Guo and Diab, 2011; Wang et al., 2015; Tang et al., 2015; Xun et al., 2017) transform topic models to learn local context level information through sense latent variable, in addition to the document level information through topic latent variable, for producing more fine grained topics from the corpus.

3 Notation

Let \( V = \{ w_1, w_2, .. w_{|V|} \} \) denote the set of unique tokens in corpus (vocabulary). Let \( C \) denote the word-word co-occurrence matrix constructed from the corpus, i.e., \( C_{ij} \) is the number of times \( w_j \) has occurred in the context of \( w_i \). We define a context around a token \( w \) as the set of \( n \) words to the left and \( n \) words to the right of \( w \). We denote the size of context window by \( n \). Typically \( n = 5 \).

Our algorithm uses LDA to infer a sense model \( \beta \) – essentially a set of \( k \) probability distributions over \( V \) – from the corpus. It then uses the sense model to encode a word \( w \) as a \( k' \)-dimensional \( \mu \)-sparse vector \( \hat{\theta}_w \). Here, we use \( \alpha \) and \( \gamma \), respectively, to denote the Dirichlet priors of \( \theta_w \), the sense distribution of a word \( w \), and \( \beta_z \), context word distribution in a sense \( z \). \( JS \) is a \( k \times k \) matrix that measures the similarity between senses. We denote the \( z^{th} \) row of a matrix \( M \) by \( M_z \).

4 Recovering senses

To recover senses, we suppose the following generative model for generating words in a context of size \( n \) (see Figure 1).

1. For each word \( w \in V \), generate a distribution over senses \( \theta_w \) from the Dirichlet distribution with prior \( \alpha \).

2. For each context \( c_w \) around target word \( w \), and for each of the \( 2n \) tokens \( \in c_w \), do

   (a) Sample sense \( z \sim \text{Multinomial}(\theta_w) \).
   (b) Sample token \( c \sim \text{Multinomial}(\beta_{zn}) \).

Such a generative model will generate a co-occurrence matrix \( C \) that can also be generated by another model. \( C \) is a matrix whose columns \( C_w \) are interpreted as a document formed from the count of all the tokens that have occurred in a context centered at \( w \). Given a Dirichlet prior of parameter \( \alpha \) on sense distribution of \( C_w \) and \( \beta \), the distribution over context words for each sense, document \( C_w \) (and thus the co-occurrence matrix \( C \)) is generated as follows:

1. Generate \( \theta_w \sim \text{Dirichlet}(\alpha) \).

2. Repeat \( N \) times to generate \( C_w \):

   (a) Sample sense \( z \sim \text{Multinomial}(\theta_w) \).
   (b) Sample token \( c \sim \text{Multinomial}(\beta_z) \).

Based on this generative model, given the co-occurrence matrix \( C \), we infer the matrix \( \beta \) and the maximum a posteriori estimate \( \theta_w \) for each word using a fast variational inference tool such as WarpLDA (Chen et al., 2016).

5 Word2Sense embeddings

Word2Sense embeddings are probability distributions over senses. We discuss how to use the senses recovered by inference on the generative model in section 4 to construct word embeddings. We demonstrate that the embeddings so computed are competitive with various multi-modal embeddings in semantic similarity and entailment tasks.

5.1 Computing Word2Sense embeddings

Denote the probability of occurrence of a word in the corpus by \( p(w) \). We approximate the probability of the word \( p(w) \) by its empirical estimate \( \| C_w \|_1 / \sum_{w' \in V} \| C_{w'} \|_1 \). We define the global probability \( p_Z(z) \) of a sense \( z \) as the probability that a randomly picked token in the corpus has that sense in it’s context window. We approximate the global distribution of generated senses using the following formulation.

\[
p_Z(z) = \sum_{w \in V} \theta_w[z] p(w) \quad \forall z \in \{1..k\}.
\]
Then, for each word \( w \in V \), we compute \( p_c(w) \), its sense distribution (when acting as a context word) as follows:

\[
p_c(z|w) = \frac{p(w|z)p_Z(z)}{p(w)} = \frac{\beta_{w,z}p_Z(z)}{p(w)}.
\]

Eliminating redundant senses. LDA returns a number of topics that are very similar to each other. Examples of such topics are given in Table 11 in appendix. These topics need to be merged, since inferring two similar words against such senses can cause them to be (predominantly) assigned to two different topic ids, causing them to look more dissimilar than they actually are. In order to eliminate redundant senses, we use the similarity of topics according to the Jensen Shannon (JS) divergence. We construct the topic similarity matrix \( JS \in \mathbb{R}^{k \times k} \), whose \( [i,j] \)-th entry \( JS[i,j] \) is the JS divergence between senses \( \beta_i \) and \( \beta_j \). Recall that JS divergence \( JSdiv(p,q) \) between two multinomial distributions \( p,q \in \mathbb{R}^k \) is given by

\[
\sum_{i=1}^{k} -p_i \log \frac{2p_i}{p_i + q_i} - q_i \log \frac{2q_i}{p_i + q_i}.
\]

We run agglomerative clustering on the JS matrix to merge similar topics. We use the following distance metric to merge two clusters \( D_i \) and \( D_j \):

\[
d(D_i,D_j) = \frac{1}{|D_i||D_j|} \sum_{a \in D_i, b \in D_j} JS[a,b]^{0.5}
\]

Let \( D_{i=1..k'} \) denote the final set of \( k' \) clusters obtained after clustering. We approximate the occurrence probability of the merged cluster of senses \( D_i \) by \( p_D(D_i) = \sum_{a \in D_i} p_Z(a) \). Table 11 in appendix shows some clusters formed after clustering. Using the merged senses, we compute the embedding \( \hat{v}_x \) of word \( w \) — a distribution over senses indexed by \( z \in \{1..k\} \) — as follows:

\[
\hat{v}_x[z] = p_c(z|w) + \theta_w[z]
\]

\[
v'_x = \text{Truncate}_\mu(\text{Project}(\hat{v}_x) \odot p_D(\cdot))
\]

\[
v_x = v'_x/\|v'_x\|_1.
\]

5.2 Evaluation

We compare Word2Sense embeddings with the state-of-the-art on word similarity and entailment tasks as well as on benchmark downstream tasks.

5.2.1 Hyperparameters

We train Word2vec Skip-Gram embeddings with 10 passes over the data, using separate embeddings for the input and output contexts, 5 negative samples per positive example, window size \( n = 2 \) and the same sub-sampling and dynamic window procedure as in (Mikolov et al., 2013). For Word2GM, we make 5 passes over the data (due to very long training time of the published code\(^1\)), using 2 modes per word, 1 negative sample per positive example, spherical covariance model, window size \( n = 10 \) and the same sub-sampling and dynamic window procedure as in (Athiwaratkun and Wilson, 2017). Since there is no recommended dimension in these papers, we report the numbers for the best performing embedding size. We report the performance of Word2vec and Word2GM at dimension 500 and 400 respectively\(^2\). We report the performance of SPOWV and SPINE in benchmark downstream tasks, that use Word2vec as base embeddings, using the recommended settings as given in (Faruqui et al., 2015) and (Subramanian et al., 2018) respectively\(^3\). For Multi-Sense Skip-Gram model (MSSG) (Neelakantan et al., 2015), we use pre-trained word and sense representations\(^4\).

We found \( k = 3000, \alpha = 0.1 \) and \( \gamma = 0.001 \) to be good hyperparameters for WarpLDA to recover fine-grained senses from the corpus. A choice of \( k' \approx \frac{3}{4} k \) that merges \( k/4 \) senses improved results. We use a context window size \( n = 5 \) and truncation parameter \( \mu = 75 \). We think \( \mu = 75 \) works best because we found the average sparsity of \( p_c(\cdot|w) \) to be around 100. Since we decrease the number of senses by \( 1/4^{th} \) after post-processing, the average sparsity reduces to close to 75. If a word is not present in the vocabulary, we take an embedding on the unit simplex, that contains equal values in all the dimensions.

5.2.2 Word Similarity

We evaluate our embeddings at scoring the similarity or relatedness of pairs of words on several

\(^{1}\)https://github.com/benathi/word2gm

\(^{2}\)We tried 100, 200, 300, 400, 500 dimensions for Word2vec, and 50, 100, 200, 400 dims for Word2GM

\(^{3}\)The two models don’t perform better than Word2vec in similarity tasks and don’t show performance in entailment.

\(^{4}\)bitbucket.org/jeevan_shankar/multi-sense-skipgram
datasets annotated with human scores: Simlex-999 (Hill et al., 2015), WS353-S and WS353-R (Finkelstein et al., 2002), MC (Miller and Charles, 1991), RG (Rubenstein and Goodenough, 1965), MEN (Bruni et al., 2014), RW (Luong et al., 2013) and MT-771 (Radinsky et al., 2011; Halawi et al., 2012).

We predict similarity/relatedness score of a pair of words \(\{w_1, w_2\}\) by computing the JS divergence (see Equation 1) between the embeddings \(\{v_{w_1}, v_{w_2}\}\) as computed in Equation 2. For other embeddings, we use cosine similarity metric to measure similarity between embeddings. The final prediction effectiveness of an embedding is given by computing Spearman correlation between the predicted scores and the human annotated scores.

Table 2 compares our embeddings to multimodal Gaussian mixture (Word2GM) model (Athiwaratkun and Wilson, 2017) and Word2vec (Mikolov et al., 2013). We extensively tune hyperparameters of prior work, often achieving better results than previously reported. We concluded from this exercise that SkipGram (Word2vec) is the best among all the unsupervised embeddings at similarity and relatedness tasks. We see that while being interpretable and sparser than the 500-dimensional Word2vec, Word2Sense embeddings is competitive with Word2vec on all the datasets.

### 5.2.3 Word entailment

Given two words \(w_1\) and \(w_2\), \(w_2\) entails \(w_1\) (denoted by \(w_1 \models w_2\)) if all instances of \(w_1\) are \(w_2\). We compare Word2Sense embeddings with Word2GM on the entailment dataset provided by (Baroni et al., 2012). We use KL divergence to generate entailment scores between words \(w_1\) and \(w_2\). For Word2GM, we use both cosine similarity and KL divergence, as used in the original paper. We report the F1 scores and Average Precision(AP) scores for reporting the quality of prediction. Table 3 compares the performance of our embedding with Word2GM. We notice that Word2Sense embeddings with \(µ = k'\) (denoted Word2Sense-full in the table), i.e., with no truncation, yields the best results. We do not compare with hyperbolic embeddings (Tifrea et al., 2019; Dhingra et al., 2018) because these embeddings are designed mainly to perform well on entailment tasks, but are far off from the performance of Euclidean embeddings on similarity tasks.

### 5.2.4 Downstream tasks

We compare the performance of Word2Sense with Word2vec, SPINE and SPOWV embeddings on the following downstream classification tasks: sentiment analysis (Socher et al., 2013), news classification, noun phrase chunking (Lazaridou et al., 2013) and question classification (Li and Roth, 2006). We do not compare with Word2GM and MSSG as there is no obvious way to compute sentence embeddings from multi-modal word embeddings. The sentence embedding needed for text classification is the average of the embeddings of words in the sentence,
as in (Subramanian et al., 2018). We pick the best among SVMs, logistic regression and random forest classifier to classify the sentence embeddings based on accuracy on the development set. Table 4 reports the accuracies on the test set. More details of the tasks are provided in Appendix E.

6 Interpretability

We evaluate the interpretability of the Word2Sense embeddings against Word2vec, SPINE and SPOWV models using the word intrusion test following the procedure in (Subramanian et al., 2018). We select the 15k most frequent words in the intersection of our vocabulary and the Leipzig corpus (Goldhahn et al., 2012). We select a set $H$ of 300 random dimensions or senses from 2250 senses. For each dimension $h \in H$, we sort the words in the 15k vocabulary based on their weight in dimension $h$. We pick the top 4 words in the dimension and add to this set a random intruder word that lies in the bottom half of the dimension $h$ and in the top 10 percentile of some other dimension $h' \in H \setminus \{h\}$ (Fyshe et al., 2014; Faruqui et al., 2015). For the dimension $h$ to be claimed interpretable, independent judges must be able to easily separate the intruder word from the top 4 words.

We split the 300 senses into ten sets of 30 senses, and assigned 3 judges to annotate the intruder in each of the 30 senses in a set (we used a total of 30 judges). For each question, we take the majority voted word as the predicted intruder. If a question has 3 different annotations, we count that dimension as non interpretable6. Since, we followed the procedure as in (Subramanian et al., 2018), we compare our performance with the results reported in their paper. Table 5 shows that Word2Sense is competitive with the best interpretable embeddings.

6(Subramanian et al., 2018) used a randomly picked intruder in this case.

<table>
<thead>
<tr>
<th>Task</th>
<th>Word 2Sense</th>
<th>Word 2Vec</th>
<th>SPOWV</th>
<th>SPINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sports news</td>
<td>0.865</td>
<td>0.826</td>
<td>0.834</td>
<td>0.810</td>
</tr>
<tr>
<td>Computer news</td>
<td>0.861</td>
<td>0.838</td>
<td>0.862</td>
<td>0.886</td>
</tr>
<tr>
<td>Religion news</td>
<td>0.965</td>
<td>0.975</td>
<td>0.966</td>
<td>0.936</td>
</tr>
<tr>
<td>NP Bracketing</td>
<td>0.693</td>
<td>0.686</td>
<td>0.687</td>
<td>0.665</td>
</tr>
<tr>
<td>Sentiment analysis</td>
<td>0.815</td>
<td>0.812</td>
<td>0.816</td>
<td>0.778</td>
</tr>
<tr>
<td>Question clf.</td>
<td>0.970</td>
<td>0.969</td>
<td>0.980</td>
<td>0.940</td>
</tr>
</tbody>
</table>

Table 4: Comparison on benchmark downstream tasks.

Table 5: Comparison of embeddings on for Word Intrusion tasks. The second column indicates the inter annotator agreement – the first number is the fraction of questions for which at least 2 annotators agreed and the second indicates the fraction on which all three agreed. The last column is the precision of the majority vote.

6.1 Qualitative evaluation

We show the effectiveness of our embeddings at capturing multiple senses of a polysemous word in Table 1. For e.g. "tie" can be used as a verb to mean tying a rope, or drawing a match, or as a noun to mean clothing material. These three senses are captured in the top 3 dimensions of Word2Sense embedding for "tie". Similarly, the embedding for "cell" captures the 5 senses discussed in section 1 within the top 15 dimensions of the embedding. The remaining top senses capture fine grained senses such as different kinds of biological cells – e.g. bone marrow cell, liver cell, neuron – that a subject expert might relate to.

7 WordCtx2Sense embeddings

A word with several senses in the training corpus, when used in a context, would have a narrower set of senses. It is therefore important to be able to refine the representation of a word according to its usage in a context. Note that Word2vec and Word2GM models do not have such a mechanism. Here, we present an algorithm that generates an embedding for a target word $\hat{w}$ in a short context $T = \{w_1, \ldots, w_N\}$ that reflects the sense in which the target word was used in the context. For this, we suppose that the senses of the word $\hat{w}$ in context $T$ are an intersection of the senses of $\hat{w}$ and $T$. We therefore infer the sense distribution of $T$ by restricting the support of the distribution to those senses $\hat{w}$ can take.

7.1 Methodology

We suppose that the words in the context $T$ were picked from a mixture of a small number of senses. Let $S_k = \{\psi = (\psi_1, \psi_2, \ldots, \psi_k) : \psi_z \geq 0; \sum_z \psi_z = 1\}$ be the unit positive simplex. The generative model is as follows. Pick a $\psi \in S_k$, and let $P = \beta \psi$, where $\beta$ is the collection of sense probability distributions recovered by LDA from
the corpus. Pick \( N \) words from \( P \) independently.

Let \( A \sim P = \beta \psi \), \( \psi \in S_k \),

\[ (3) \]

where \( A \) is a vocabulary-sized vector containing the count of each word, normalized to sum 1. We do not use the Dirichlet prior over sense distribution as in the generative model in section 4, as we found its omission to be better at inferring the sense distribution of contexts.

Given \( A \) and \( \beta \), we want to infer the sense distribution \( \psi \in S_k \) that minimizes the log perplexity \( f(\psi; A, \beta) = -\sum_i |V|A_i \log(\beta \psi)_i \) according to the generative model in Equation 3. The MWU – multiplicative weight update – algorithm (See Appendix A for details) is a natural choice to find such a distribution \( \psi \), and has an added advantage. The MWU algorithm’s estimate of a variable \( \psi \) w.r.t. a function \( f \) after \( t \) iterations (denoted \( \psi^{(t)} \)) satisfies

\[ \psi^{(t)}[i] = 0, \text{ if } \psi^{(0)}[i] = 0 \forall i \in \{1..k\} \text{ and } \forall t \geq 0. \]

Therefore, to limit the set of possible senses in the inference of \( \psi \) to the \( \mu \) senses that \( \hat{w} \) can take, we initialize \( \psi^{(0)} \) to the embedding \( v_{\hat{w}} \). We used the embedding obtained in Equation 2 without the Project operator that adds probabilities of similar senses, to correspond with the use of the original matrix \( \beta \) for MWU.

Further, to keep iterates close to the initial \( \psi^{(0)} \), we add a regularizer to log perplexity. This is necessary to bias the final inference towards the senses that the target word has higher weights on. Thus the loss function on which we run MWU with starting point \( \psi^{(0)} = v_{\hat{w}} \) is

\[ f(\psi; A, \beta) = -\sum_i |V|A_i \log(\beta \psi)_i + \lambda KL(\psi, \psi^{(0)}) \]

\[ (4) \]

where the second term is the KL divergence between two distributions scaled by a hyperparameter \( \lambda \). Recall that \( KL(p, q) = -\sum_{i=1}^k p_i \log(p_i/q_i) \) for two distributions \( p, q \in \mathbb{R}^k \). We use the final estimate \( \psi^{(t)} \) as the Word-Ctx2Sense distribution of a word in the context.

7.2 Evaluation

We demonstrate that the above construction of a word’s representation disambiguated in a context is useful by comparing with state-of-the-art unsupervised methods for polysemy disambiguation on two tasks: Word Sense Induction and contextual similarity. Specifically, we compare with MSSG, the \( K \)-Grassmeans model of \( (\text{Mu et al., 2017}) \), and the sparse coding method of \( (\text{Arora et al., 2018}) \).^7

7.2.1 Hyperparameters

We use the same hyperparameter values for \( \alpha, \beta, k \) and \( n \) as in section 5.2.1. We use \( \mu = 100 \) since we do not merge senses in this construction. We tune the hyperparameter \( \lambda \) to the task at hand.

7.2.2 Word Sense Induction

The WSI task requires clustering a collection of (say 40) short texts, all of which share a common polysemous word, in such a way that each cluster uses the common word in the same sense. Two datasets for this task are Semeval-2010 (Manandhar et al., 2010) and MakeSense-2016 (Mu et al., 2017). The evaluation criteria are F-score (Artiles et al., 2009) and V-Measure (Rosenberg and Hirschberg, 2007). V-measure measures the quality of a cluster as the harmonic mean of homogeneity and coverage, where homogeneity checks if all the data-points that belong to a cluster belong to the same class and coverage checks if all the data-points of the same class belong to a single cluster. F-score is the harmonic mean of precision and recall on the task of classifying whether the instances in a pair belong to the same cluster or not. F-score tends to be higher with a smaller number of clusters and the V-Measure tends to be higher with a larger number of clusters, and it is important to show performance in both metrics.

For each text corresponding to a polysemous word, we learn a sense distribution \( \psi \) using the steps in section 7.1. We tuned the parameter \( \lambda \) and found the best performance at \( \lambda = 10^{-2} \). We use hard decoding to assign a cluster label to each text, i.e., we assign a label \( k^* = \arg\max_k \psi_k \) to a text with inferred sense vector \( \psi_k \).

Suppose that this yields \( \hat{k} \) distinct clusters for the instances corresponding to a polysemous word. We cluster them using agglomerative clustering into a final set of \( K \) clusters. The distance metric used to group two clusters \( D_i \) and \( D_j \) is

\[ d(D_i, D_j) = \max_{a \in D_i, b \in D_j} (JS[a, b])^{0.5} \]

\[ (5) \]

Note that we report baseline numbers from the original papers. These papers have trained their models on newer versions of Wikipedia dump that contain more than 3 billion tokens (MSSG uses a 1 billion token corpus). However, our model has been trained on a combined dataset of wiki-2009 dump and ukWaC, which contains around 3B tokens. Hence, there might be minor differences in comparing our model to the baseline models.
Table 6: Comparison on the SCWS task. Setting (a) for WordCtx2Sense uses $\lambda = 0.1$ for all pairs, and setting (b) uses $\lambda = 10^{-3}$ for pairs containing same target words and $\lambda = 0.1$ for all other pairs. Word2Sense, Word2Vec and Word2GM neglect context and compare target words. Numbers reported from (Mu et al., 2017) whose experimental setup we could replicate.

<table>
<thead>
<tr>
<th>Method</th>
<th>MakeSense-2016</th>
<th>SemEval-2010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F-scr</td>
<td>V-msr</td>
</tr>
<tr>
<td>(Huang et al., 2012)</td>
<td>-</td>
<td>47.40</td>
</tr>
<tr>
<td>(Neelakantan et al., 2015)</td>
<td>-</td>
<td>54.49</td>
</tr>
<tr>
<td>300D.30K.key</td>
<td>-</td>
<td>57.91</td>
</tr>
<tr>
<td>300D.6K.key</td>
<td>-</td>
<td>64.66</td>
</tr>
<tr>
<td>(Mu et al., 2017)</td>
<td>2</td>
<td>58.25</td>
</tr>
<tr>
<td>(Arora et al., 2018)</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>WordCtx2Sense</td>
<td>2</td>
<td>63.71</td>
</tr>
<tr>
<td>$\lambda = 0.0$</td>
<td>5</td>
<td>59.75</td>
</tr>
<tr>
<td>6</td>
<td>59.13</td>
<td>34.20</td>
</tr>
<tr>
<td>WordCtx2Sense</td>
<td>2</td>
<td>65.27</td>
</tr>
<tr>
<td>$\lambda = 10^{-2}$</td>
<td>5</td>
<td>62.88</td>
</tr>
<tr>
<td>6</td>
<td>61.43</td>
<td>35.30</td>
</tr>
</tbody>
</table>

Table 6: Comparison of WordCtx2Sense with the state-of-the-art methods for Word Sense Induction on MakeSense-2016 and SemEval-2010 dataset. We report F-score and V-measure scores by a considerable margin. We observe similar improvements on the MakeSense-2016 dataset.

Results Table 6 shows the results of clustering on WSI SemEval-2010 dataset. WordCtx2Sense outperforms (Arora et al., 2018) and (Mu et al., 2017) on both F-score and V-measure scores by a considerable margin. We observe similar improvements on the MakeSense-2016 dataset.

7.2.3 Word Similarity in Context

The Stanford Contextual Word Similarity task (Huang et al., 2012) consists of 2000 pairs of words, along with the contexts the words occur in. Ten human raters were asked to rate the similarity of each pair of words according to their use in the corresponding contexts, and their average score (on a 1 to 10 scale) is provided as the ground-truth similarity score. The goal of a contextual embedding would be to score these examples to maximize the correlation with this ground-truth.

We compute the WordCtx2Sense of each word in its respective context as in section 7.1. For comparing the meaning of two words in context, we use the JS divergence between their WordCtx2Sense embeddings. We report the coefficient between the ground-truth and WordCtx2Sense according to two different settings of $\lambda$, (a) $\lambda = 0.1$, and b) $\lambda = 10^{-3}$ for inferring the contextual embedding of a word in those pairs that contain same target words, and $\lambda = 0.1$ for all other pairs. The main idea is to reduce unnecessary bias for comparing sense of a polysemous word in two different contexts.

Results Table 7 shows that sense embeddings using context information perform better than all the existing models, except MSSG models (Neelakantan et al., 2015). Also, computing the embeddings of a word using the contextual information improves results by aprox. 0.025, compared to the case when words embeddings are used directly.

8 Conclusion and future work

We motivated an efficient unsupervised method to embed words, in and out of context, in a way that captures their multiple senses in a corpus in an interpretable manner. We demonstrated that such interpretable embeddings can be competitive with dense embeddings like Word2vec on similarity tasks and can capture entailment effectively. Further, the construction provides a natural mechanism to refine the representation of a word in a short context by disambiguating its senses. We have demonstrated the effectiveness of such contextual representations.

A natural extension to this work would be to capture the sense distribution of sentences using the same framework. This will make our model more comprehensive by enabling the embedding of words and short texts in the same space.

9 Acknowledgements

We thank Monojit Choudhury, Ravindran Kannan, Aditya Pratapa and Anshul Bawa for many helpful discussions. We thank all anonymous reviewers for their constructive comments.
References


Lushan Han, Tim Finin, Paul McNamee, Anupam Joshi, and Yelena Yesha. 2012. Improving word similarity by augmenting PMI with estimates of word polysemy. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1307–1322.


A Multiplicate Weight Update

Algorithm 1 Multiplicative Weight update

1: function MWU(k, L, f, θ(0), ITER)
   ▷ k denotes dimension of variable θ, f denotes a function of θ, L if lipschitz constant of f, θ0 denotes initial starting point of θ, ITER denotes the number of iterations to run
2: for t = 1 .. ITER do
3:     η = \frac{1}{k} \sqrt{2 \log(\frac{k}{t})}
4:     \hat{θ}^{(t)} = \hat{θ}^{(t-1)} \exp (\eta \nabla f(θ)|_{θ=θ^{(t-1)}})
5:     θ_t = \hat{θ}^{(t)}/‖\hat{θ}^{(t)}‖_1
6: end for
7: end function

B Hyper-parameter tuning for Word2vec

We use the default hyperparameters for training Word2vec, as given in Mikolov et al. (2013). We tuned the embedding size, to see if the performance improves with increasing number of dimensions. Table 8 shows that there is minor improvement in performance in different similarity and relatedness tasks as the embedding size is increased from 100 to 500.

C Hyper-parameter tuning for Word2GM

We use the default hyperparameters for training Word2GM, as given in Athiwaratkun and Wilson (2017). We tuned the embedding size, to see if the performance improves with increasing number of dimensions. Table 9 shows that there is minor improvement in performance of Word2GM, when the embedding size is increased from 100 to 400.

D Hyper-parameter tuning for Word2Sense

For generating senses, we use WarpLDA that has 3 different hyperparameters, a) Number of topics k b) α, the dirichlet prior of sense distribution of each word and c) γ, the dirichlet prior of word distribution of each sense. We keep k fixed at 3000 and vary α and β. We show a small subset of the hyperparameter space searched for α and β. We report the performance of word embeddings computed by Equation 3, without the Project step, in different similarity tasks. Table 10 shows that the performance slowly decreases as we increase β and somewhat stays constant with α. Hence, we choose α = 0.1 and γ = 0.001 for carrying out our experiments.

E Benchmark downstream tasks

In this section, we discuss about the different downstream tasks considered. We follow the same procedure as (Faruqui et al., 2015) and (Subramanian et al., 2018)\(^8\).

- **Sentiment analysis** This is a binary classification task on Sentiment Treebank dataset (Socher et al., 2013). The task is to give a sentence a positive or a negative sentiment label. We used the provided train, dev. and test splits of sizes 6920, 872 and 1821 sentences respectively.

- **Noun phrase bracketing** NP bracketing task (Lazaridou et al., 2013) involves classifying a noun phrase of 3 words as left bracketed or right bracketed. The dataset contains 2,227 noun phrases split into 10 folds. We append the word vectors of three words to get feature representation (Faruqui et al., 2015). We report 10-fold cross validation accuracy.

- **Question classification** Question classification task (Li and Roth, 2006) involves classifying a question into six different types, e.g., whether the question is about a location, about a person or about some numeric information. The training dataset consists of 5452 labeled questions, and the test dataset consists of 500 questions.

- **News classification** We consider three binary categorization tasks from the 20 Newsgroups dataset. Each task involves categorizing a document according to two related categories (a) Sports: baseball vs. hockey (958/239/796) (b) Comp.: IBM vs. Mac (929/239/777) (c) Religion: atheism vs. christian (870/209/717), where the brackets show training/dev./test splits.

\(^8\)We use the evaluation code given in https://github.com/harsh19/SPINE
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SCWS</td>
<td>0.638</td>
<td>0.646</td>
<td>0.648</td>
<td>0.649</td>
<td>0.651</td>
</tr>
<tr>
<td>Simlex-999</td>
<td>0.365</td>
<td>0.388</td>
<td>0.387</td>
<td>0.393</td>
<td>0.393</td>
</tr>
<tr>
<td>MEN</td>
<td>0.749</td>
<td>0.760</td>
<td>0.763</td>
<td>0.767</td>
<td>0.780</td>
</tr>
<tr>
<td>RW</td>
<td>0.361</td>
<td>0.361</td>
<td>0.363</td>
<td>0.365</td>
<td>0.365</td>
</tr>
<tr>
<td>MT-771</td>
<td>0.684</td>
<td>0.685</td>
<td>0.681</td>
<td>0.681</td>
<td>0.688</td>
</tr>
<tr>
<td>WS353</td>
<td>0.705</td>
<td>0.719</td>
<td>0.721</td>
<td>0.733</td>
<td>0.732</td>
</tr>
<tr>
<td>WS353-S</td>
<td>0.744</td>
<td>0.766</td>
<td>0.768</td>
<td>0.768</td>
<td>0.769</td>
</tr>
<tr>
<td>WS353-R</td>
<td>0.669</td>
<td>0.679</td>
<td>0.670</td>
<td>0.696</td>
<td>0.703</td>
</tr>
</tbody>
</table>

Table 8: Performance of Word2vec at different embedding size, in similarity tasks.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>0.345</td>
<td>0.385</td>
<td>0.398</td>
</tr>
<tr>
<td>WS353</td>
<td>0.664</td>
<td>0.672</td>
<td>0.669</td>
</tr>
<tr>
<td>WS353-S</td>
<td>0.727</td>
<td>0.735</td>
<td>0.751</td>
</tr>
<tr>
<td>WS353-R</td>
<td>0.626</td>
<td>0.625</td>
<td>0.607</td>
</tr>
<tr>
<td>MEN</td>
<td>0.740</td>
<td>0.755</td>
<td>0.761</td>
</tr>
<tr>
<td>MC</td>
<td>0.812</td>
<td>0.802</td>
<td>0.826</td>
</tr>
<tr>
<td>RG</td>
<td>0.730</td>
<td>0.772</td>
<td>0.750</td>
</tr>
<tr>
<td>MT-771</td>
<td>0.638</td>
<td>0.664</td>
<td>0.682</td>
</tr>
<tr>
<td>RW</td>
<td>0.303</td>
<td>0.338</td>
<td>0.338</td>
</tr>
</tbody>
</table>

Table 9: Performance of Word2GM, with spherical covariance matrix for each embedding, at different embedding sizes in similarity tasks.

<table>
<thead>
<tr>
<th>$\alpha$, $\gamma$</th>
<th>SCWS</th>
<th>MT-771</th>
<th>WS353</th>
<th>RG</th>
<th>MC</th>
<th>WS353-S</th>
<th>WS353-R</th>
<th>MEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1, 0.001</td>
<td>0.596</td>
<td>0.623</td>
<td>0.654</td>
<td>0.794</td>
<td>0.767</td>
<td>0.685</td>
<td>0.662</td>
<td>0.754</td>
</tr>
<tr>
<td>0.1, 0.005</td>
<td>0.595</td>
<td>0.625</td>
<td>0.647</td>
<td>0.809</td>
<td>0.758</td>
<td>0.699</td>
<td>0.638</td>
<td>0.748</td>
</tr>
<tr>
<td>0.1, 0.1</td>
<td>0.584</td>
<td>0.609</td>
<td>0.601</td>
<td>0.733</td>
<td>0.671</td>
<td>0.618</td>
<td>0.626</td>
<td>0.738</td>
</tr>
<tr>
<td>1.0, 0.001</td>
<td>0.596</td>
<td>0.607</td>
<td>0.651</td>
<td>0.815</td>
<td>0.700</td>
<td>0.692</td>
<td>0.658</td>
<td>0.743</td>
</tr>
<tr>
<td>1.0, 0.005</td>
<td>0.613</td>
<td>0.620</td>
<td>0.640</td>
<td>0.792</td>
<td>0.691</td>
<td>0.676</td>
<td>0.653</td>
<td>0.749</td>
</tr>
<tr>
<td>1.0, 0.05</td>
<td>0.559</td>
<td>0.562</td>
<td>0.583</td>
<td>0.730</td>
<td>0.742</td>
<td>0.609</td>
<td>0.581</td>
<td>0.711</td>
</tr>
<tr>
<td>1.0, 0.1</td>
<td>0.587</td>
<td>0.602</td>
<td>0.602</td>
<td>0.755</td>
<td>0.720</td>
<td>0.641</td>
<td>0.605</td>
<td>0.727</td>
</tr>
<tr>
<td>10.0, 0.001</td>
<td>0.595</td>
<td>0.610</td>
<td>0.628</td>
<td>0.822</td>
<td>0.772</td>
<td>0.664</td>
<td>0.639</td>
<td>0.747</td>
</tr>
<tr>
<td>10.0, 0.005</td>
<td>0.608</td>
<td>0.635</td>
<td>0.657</td>
<td>0.808</td>
<td>0.826</td>
<td>0.708</td>
<td>0.648</td>
<td>0.739</td>
</tr>
<tr>
<td>10.0, 0.05</td>
<td>0.562</td>
<td>0.539</td>
<td>0.544</td>
<td>0.786</td>
<td>0.710</td>
<td>0.573</td>
<td>0.551</td>
<td>0.717</td>
</tr>
<tr>
<td>10.0, 0.1</td>
<td>0.573</td>
<td>0.606</td>
<td>0.570</td>
<td>0.773</td>
<td>0.696</td>
<td>0.612</td>
<td>0.593</td>
<td>0.724</td>
</tr>
</tbody>
</table>

Table 10: Performance of Word2Sense as computed in eq. 3 without the Project step in similarity tasks, at different hyperparameter settings.

<table>
<thead>
<tr>
<th>Cluster size</th>
<th>Top 10 words with the highest probability in the sense’s distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>tennessee, kentucky, alabama, mississippi, georgia, arkansas, nashville, memphis, louisville, atlanta state, idaho, oregon, montana, wisconsin, utah, nevada, wyoming, states, california</td>
</tr>
<tr>
<td>19</td>
<td>lol, im, thats, dont, mrplow, yeah, cant, it, ive, ur im, dont, ive, cant, didnt, thats, lol, ur, my, cos my, ve, have, it, me, n’t, ll, just, blog, but</td>
</tr>
<tr>
<td>5</td>
<td>lgame, games, adventure, gameplay, 3d, players, play, arcade, of, fun game, multiplayer, games, gameplay, gaming, xbox, shooter, gamers, mode, halo cheats, mario, super, game, arcade, unlock, mode, nintendo, cheat, bros</td>
</tr>
<tr>
<td>7</td>
<td>charlton, striker, midfield, defender, leads, midfielder, darren, goal, bowyer, danny swanse, derby, leicester, city, wolves, watford, burnley, boss, stoke, bos swans manager, albion, club, coach, season, football, boss, fa, robson, gary</td>
</tr>
</tbody>
</table>

Table 11: Examples of clusters formed after agglomerative clustering. Each group of rows shows a randomly picked cluster, it’s size and top 10 words of 3 randomly picked senses from the cluster. The clusters represent U.S. states, generic words, video games, and soccer respectively.