Polysemy Resolution in Word Embeddings

Sanchit Agrawal, Ishu Dharmendra Garg,
Ujjawal Soni, and Shreyas Harish

Indian Institute of Technology Madras
{sanchit,ishugarg,ujjawals,shreyas}@cse.iitm.ac.in
{cs13b061,cs13b060,cs13b053,cs13b062}

Abstract. Traditional word embedding techniques like word2vec compute a single vector for each word. Most English words are polysemous (e.g. bank - financial institution or riverside slope), thus implying that various senses of a word are represented using a single vector. This limitation will detrimentally affect the quality of the embeddings, since components of other senses of a word will interfere when it is intended to be used with a particular sense. We propose a solution that involves using knowledge graphs like WordNet for computing sense embeddings instead of word embeddings, thus resolving difficulties arising due to polysemy. We show that our sense embeddings are superior to word embeddings in standard tasks like word similarity and relational similarity.

Key words: word2vec, word embeddings, sense embeddings, polysemy, WordNet

1 Introduction

1.1 Word Embeddings

Word embeddings are powerful tools in natural language processing, where words or phrases from the vocabulary are mapped to vectors of real numbers in a low-dimensional space relative to the vocabulary size. Methods to generate such mappings include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear.
Word and phrase embeddings, when used as the underlying input representation, have been shown to boost the performance in NLP tasks such as syntactic parsing and sentiment analysis.

Count Vectors These methods compute the word co-occurrence matrix (the term-document matrix) and then perform computations on it to reduce each row to a small, dense vector for each word. An example of this method is Latent Semantic Analysis.

Predict Vectors These methods jointly learn the word vectors and a language model by representing the probability of a sentence in terms of the vectors of its composite words, and maximizing the log likelihood of a large corpus. The probability function can be represented by a feedforward neural network. An example of this technique is the neural network language model (NNLM) [5].

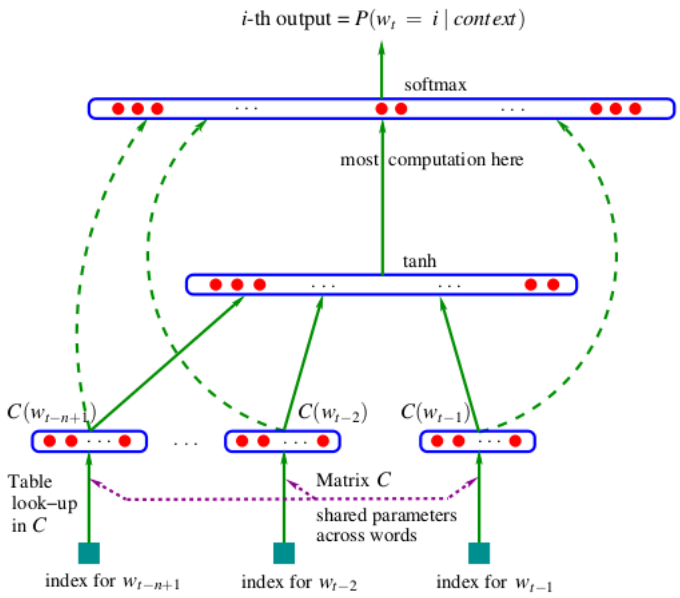

Fig. 1. Neural Network Language Model

1.2 word2vec

word2vec [1] is an efficient predictive technique for obtaining high quality word vectors from large corpora. The word embeddings obtained using word2vec are such that similar words tend to be close to each other.
The similarities go beyond simple syntactic regularities. Using simple algebraic operations are performed on the word vectors, it was shown for example that vector(King) - vector(Man) + vector(Woman) results in a vector that is closest to the vector representation of the word Queen.
The major contribution of word2vec was the proposal of two new model architectures for learning distributed representations of words that try to minimize computational complexity. The main observation was that most of the complexity is caused by the non-linear hidden layer in the model. While this is what makes neural networks so attractive, TODO:cite decided to explore simpler models that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently.

Continuous Bag of Words (CBOW) Model The first proposed architecture is similar to the feedforward neural network language model TODO:cite, where the non-linear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix). All words get projected into the same position (their vectors are averaged). The order of words in the history does not influence the projection, hence it is called a bag of words model. Words from the future are also used; for e.g. a log-linear classifier is built with four future and four history words at the input, where the training criterion is to correctly classify the current (middle) word.

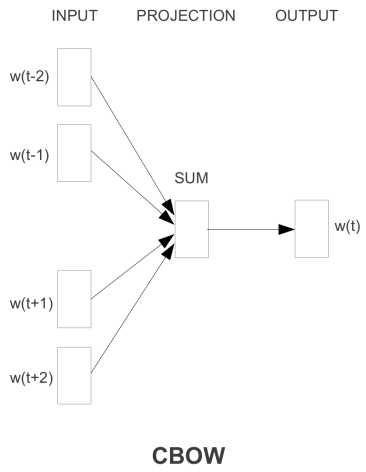

Fig. 2. Continuous Bag of Words Model

Continuous Skip-gram Model The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, it uses each current word as an input to a log-linear classifier with continuous projection layer, and predicts words within a certain range before and after the current word.

2 Polysemy in Word Embeddings

2.1 The Problem

Traditional word embeddings like Mikolov et. al's Word2Vec result in vectors for each string that is considered to be a word in the vocabulary.
One glaring drawback of this approach is that while most words are polysemous

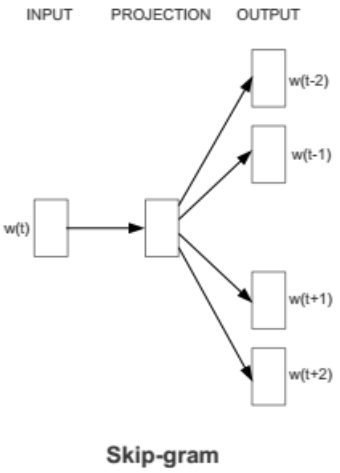

Fig. 3. Skip-gram Model

in languages like English, each word is represented by a single vector.
After applying the Word2Vec approach, we obtain vectors for all the words in the corpus. This might include highly polysemous words like "light" (electromagnetic waves or to set on fire or the opposite of heavy), "bank" (a riverside or a financial institution) or "top" (a spinning toy, or the upside, or an upper garment).
It is clear that the quality of the embeddings will be detrimentally affected as different meanings of a word are conflated into a single representation. We cannot reasonably hope that a single vector can effectively capture the "correct" meaning in all the contexts.
Secondly, word embeddings base their representations solely on the distributional statistics obtained from corpora, ignoring the wealth of information provided by existing semantic resources like WordNet.

2.2 Proposed Solution

Our proposed solution is to learn vectors for senses (i.e. meanings) instead of words. One way to enumerate the various senses is to use the synsets defined in knowledge graphs like WordNet.
In order to make use of these sense vectors in a task, we can use one of the following methods:

1. If the appropriate sense of the word is known, then simply use the vector corresponding to that sense.
2. If a decent word sense disambiguation can be performed for the task at hand, then the senses of the words can be found, and the corresponding vectors can be used.

3. If an appropriate relation can be found that maps words to appropriate senses, then those vectors can be used (we use this method for word similarity).
4. If there is absolutely no way to ascertain which senses a word belongs to, then create a word vector as the weighted average of its sense vectors. Senses and weights (relative frequency counts) for a word are included in WordNet.

3 Related Work

3.1 SensEmbed

[4] obtains sense embeddings by training word2vec on large corpora annotated using automatic word sense disambiguation. To carry out the WSD task, they used Babelfy, which is a proprietary WSD algorithm based on BabelNet, a semantic graph formed by merging WordNet, Wikipedia and other knowledge sources.
They show that the sense embeddings obtained using this method are superior to word embeddings obtained using vanilla word2vec on similarity and relatedness tasks.

3.2 DeepWalk: Online Learning of Social Representations

[3] presents a novel method called DeepWalk for learning latent representations of vertices in a network (like a social network graph). The idea is to carry out a random walk rooted at a random vertex of the graph. The list of vertices visited during the walk represents an artificial "context". Collection of several such artificial contexts provides an artificial corpus, which is the primary source of extracting information from the network. The corpus is given as input to a word embedding method like word2vec, which outputs a vector for each vertex in the graph. The vectors summarize the vertices in a continuous low-dimensional space.
The vectors are then used in classification and clustering tasks, and are shown to be superior to other techniques.

4 Learning Sense Embeddings

To train sense embeddings, we experimented with the following two techniques:

4.1 Using WSD on a text corpus

Abstract Method

1. Obtain a large sense-annotated text corpus by running WSD
2. Run word2vec on the corpus to obtain sense vectors

WSD Details The WSD technique we chose was Babelfy (as used in [4]). Babelfy first models each concept in the network by leveraging a graph random-walk algorithm. Given an input text, the algorithm constructs a subgraph of the semantic network representing the input text. Babelfy then searches this subgraph for the intended sense of each content word using an iterative process and a dense subgraph heuristic.

Disadvantages This technique has the following disadvantages:

- WSD is one of the hardest problems in NLP. Having it a sub-task to our goal makes it considerably harder and likelier to affect the quality of embeddings.
- Despite Babelfy being one of the state-of-art methods for WSD, it is unable to disambiguate a large fraction of words for a given context.
- WSD techniques frequently assign the incorrect sense to a word.

Advantages This technique has the following advantages:

- Performing WSD allows us to exploit the knowledge hidden in very large corpora.
- When using the embeddings in a real task, if one is able to leverage a good WSD technique, then it provides a direct way to obtain word vectors from sense vectors.
- This technique can be applied well to technical domains where WSD is easy to perform, and corpora like manuals and books can be used to learn embeddings.
- This technique can easily generalize to new languages for which corpora and WSD are available.

4.2 Using DeepWalk on a knowledge graph

Abstract Method

1. Perform several random walks on a knowledge graph like WordNet
2. For each random walk, store the vertices visited in the knowledge graph
3. Run word2vec on the corpus to obtain sense vectors

Random Walk Details The random walk algorithm starts by randomly choosing a vertex from the graph. At each step, it either terminates with a probability α ($= 0.15$ for our experiments), or moves to a random adjacent vertex. We create an artificial corpus of $10,000,000$ such contexts. The knowledge graph we use is WordNet 3.0.

Disadvantages This technique has the following disadvantages:

- Since WordNet only contains the root words, we do not get sense vectors for lexical derivatives of a word.
- WordNet does not distinguish between the fine senses of similar words.
- For applying this method to a different language, we would need a new knowledge graph, which requires tremendous human effort.
- This technique may not be appropriate for technical domains, where terms have very specific connotations.
- It fails to capture the lexical & grammatical relations of the language, since the sentences are artificial.

Advantages This technique has the following advantages:

- It enables us to intelligently bypass the WSD step/
- Using DeepWalk, we are able to exploit the information of expertly curated knowledge graphs.
- It can potentially cover all aspects of human knowledge by using large graphs like Wikipedia concepts, which are continuously refined and updated by a large community.
- It captures semantic relations better due to the artificial nature of the random walks, in which similar synsets appear together.

Considering the advantages and disadvantages in both the methods, and after experimenting with them, we decided to finally pursue the DeepWalk approach. The word2vec model parameter details are:

- Embedding size = 100
- Window size = 7
- Training Rate = 0.04 (with exponential decay)
- Number of epochs = 10
- Model Type: Skip-gram
- Output Layer: Hierarchical Softmax

5 WordNet Relations

The WordNet graph on which we performed random walks captures the following major relations between senses in the form of edges:

- Hypernymy / Hyponymy: This represents the is-a relation. It links general synsets to more specific ones. e.g. {dog} is hyponym of {canine}. This relationship is transitive.
- Meronymy: This represents the part-of relation. Parts are inherited from their superordinates. e.g. {back, backrest} is a part of {chair}. This relationship is non-transitive.
- Troponymy: This relation represents specificity in verbs. Troponyms express increasingly specific ways of doing things. e.g. {communicate}-{talk}-{whisper}

- Antonymy for adjectives: Includes direct antonyms like {wet}-{dry}, {old}-{young}.
- Semantically similar antonyms: e.g. {dry} to {parched}, {arid}, {wet} to {soggy}, {waterlogged} etc.
- Pertainyms: This relation points adjectives to nouns from which they are derived. e.g. {criminal}-{crime}.
- Other Cross POS relations: e.g. words having the same stem words like {observe}-{observer}-{observant}, miscellaneous tagged verb-noun relations etc.

While performing the random walks, all relations are treated in the same manner, having equal weights.

6 Model Evaluation

We computed sense embeddings using the DeepWalk approach, and word embeddings using vanilla word2vec trained over the Brown Corpus. We then evaluated both models on two tasks: word similarity and word relatedness, and compared the results.

6.1 Word Similarity

Description The task involves computing the similarity between 2 words, a metric that lies in the range $[0, 1]$. Average human judgements are considered as ground truth.

Datasets Used

- WordSim-353-Sim: 203 pairs of words & similarity scores in the range $[0, 10]$
- MEN: 3000 pairs of words & similarity scores in the range $[0, 50]$
- RG-65: 65 pairs of words & similarity scores in the range $[0, 5]$

Note: Another drawback of word2vec is highlighted here. Due to the limitations of corpora, not all words present in the datasets are present in the trained word2vec model. We ignore these pairs while evaluating both models.

Algorithm Used We used the max similarity algorithm for judging similarity scores between words when using the sense embeddings. The algorithm is described below:

1. Given 2 words w_1, w_2, find the sets of all senses corresponding to these words. Let these sense sets be S_1, S_2 respectively.
2. Given 2 senses s_1 & s_2, define $sim(s_1, s_2) = (cosine(V(s_1), V(s_2)) + 1)/2$, where $cosine(u, v)$ is the cosine distance between vectors u and v, and $V(s)$ is the sense vector for sense s.
3. Output $max\{sim(s_1, s_2) : s_1 \in S_1, s_2 \in S_2\}$.

The justification for using this algorithm is that it approximates how humans might make judgements while considering word similarity. We believe that the closest senses of the words get triggered when humans carry out this task. For e.g. when judging (bark, husk), most humans will consider bark to have the sense meaning the skin of a tree's trunk. However, when judging (bark, woof), we consider bark to mean the noise made by a dog.
When using the word2vec word embeddings, we simply return $sim(w_1, w_2)$, where w_1, w_2 are the words being judged.
Additionally, we also test a hybrid model that uses both the embeddings. There are 2 ways to do this: we could either concatenate the word and sense vectors to get a large vector, or we could take a weighted relatedness score using the scores of both the models. For the following task, we use the weighted similarity scores.

Results The results for various metrics on which the models were evaluated are shown below:

	MEN	RG-65	WordSim-353-Sim
word2vec	14.1588	1.2306	2.8249
sense embeddings	10.6590	0.9055	1.8451
hybrid model	10.6453	0.8031	1.8409

Table 1. RMS error on various datasets (lower is better)

	MEN	RG-65	WordSim-353-Sim
word2vec	0.3907	0.4545	0.4946
sense embeddings	0.6423	0.8142	0.7144
hybrid model	0.6489	0.8310	0.7220

Table 2. Pearson correlation on various datasets (higher is better)

We see that the sense embedding model is significantly better than the word2vec model over all datasets. While the hybrid model is better than the word2vec model, it is still worse than the pure sense model.

6.2 Word Relatedness

Description The task involves computing the how related 2 words are, a metric that lies in the range $[0, 1]$. Average human judgements are considered as ground truth.

Datasets Used

– WordSim-353-Rel: 252 pairs of words & relatedness scores in the range $[0, 10]$

Note: Another drawback of word2vec is highlighted here. Due to the limitations of corpora, not all words present in the datasets are present in the trained word2vec model. We ignore these pairs while evaluating both models.

Algorithm Used We used the same algorithms as used for the word similarity task.
Additionally, we also test a hybrid model that uses both the embeddings. There are 2 ways to do this: we could either concatenate the word and sense vectors to get a large vector, or we could take a weighted relatedness score using the scores of both the models. For the following task, we use the concatenated hybrid model.

Results The results for various metrics on which the models were evaluated are shown below:

	WordSim-353-Rel
word2vec	3.2309
sense embeddings	2.3684
hybrid model	2.3346

Table 3. RMS error on various datasets (lower is better)

	WordSim-353-Rel
word2vec	0.2254
sense embeddings	0.4584
hybrid model	0.4647

Table 4. Pearson correlation on various datasets (higher is better)

We see that the sense embedding model is significantly better than the word2vec model over all datasets. For this task, the hybrid model outperforms the sense embedding model.

7 Visualization of word2vec Embeddings

In order to visualize the word2vec embeddings, we performed a Principal Component Analysis on the embedding matrix to reduce the dimensionality to three,

and plotted the vectors for some words. The plots give some insight to how various words are represented in the vector space.

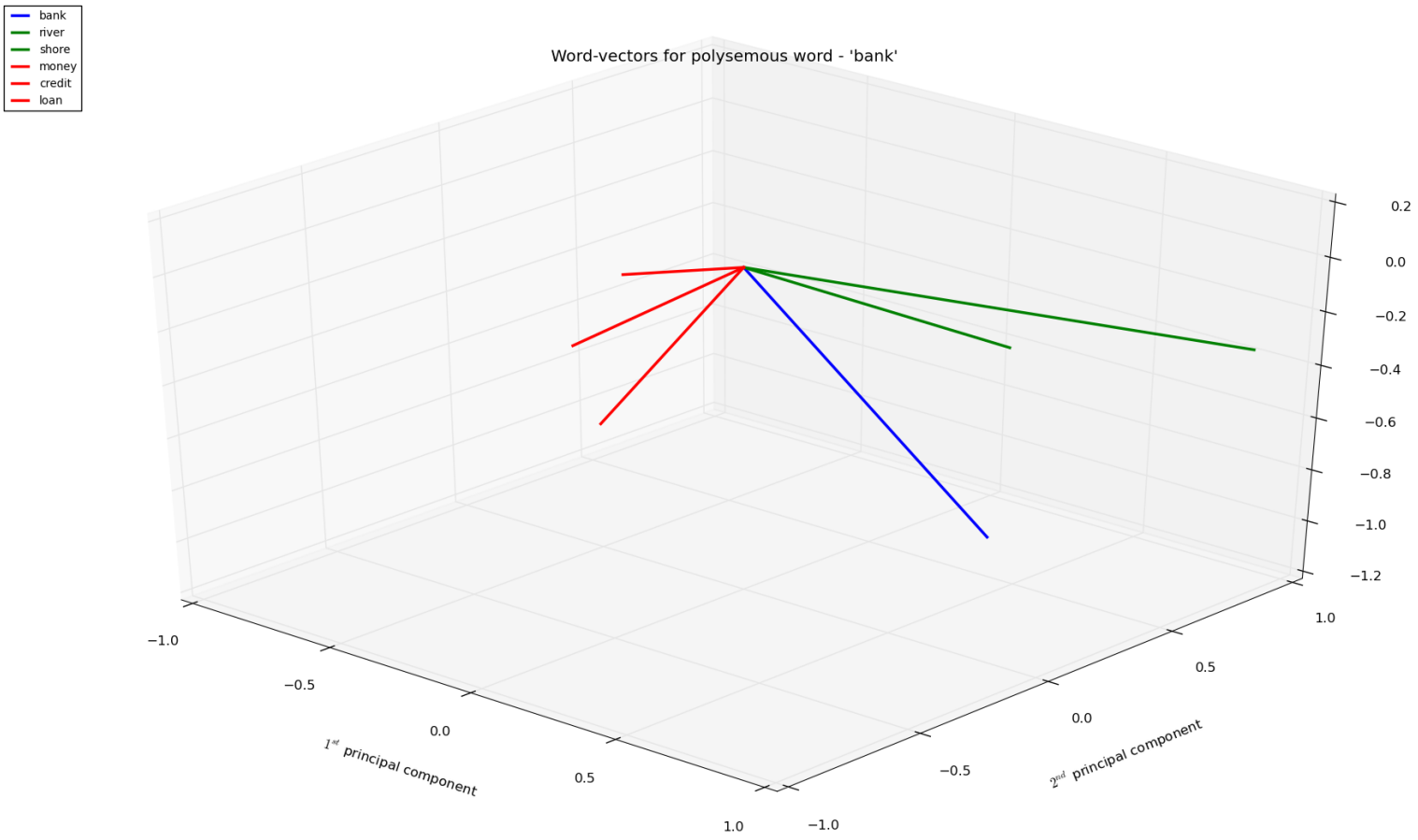

Fig. 4. "bank" vector plotted with its senses

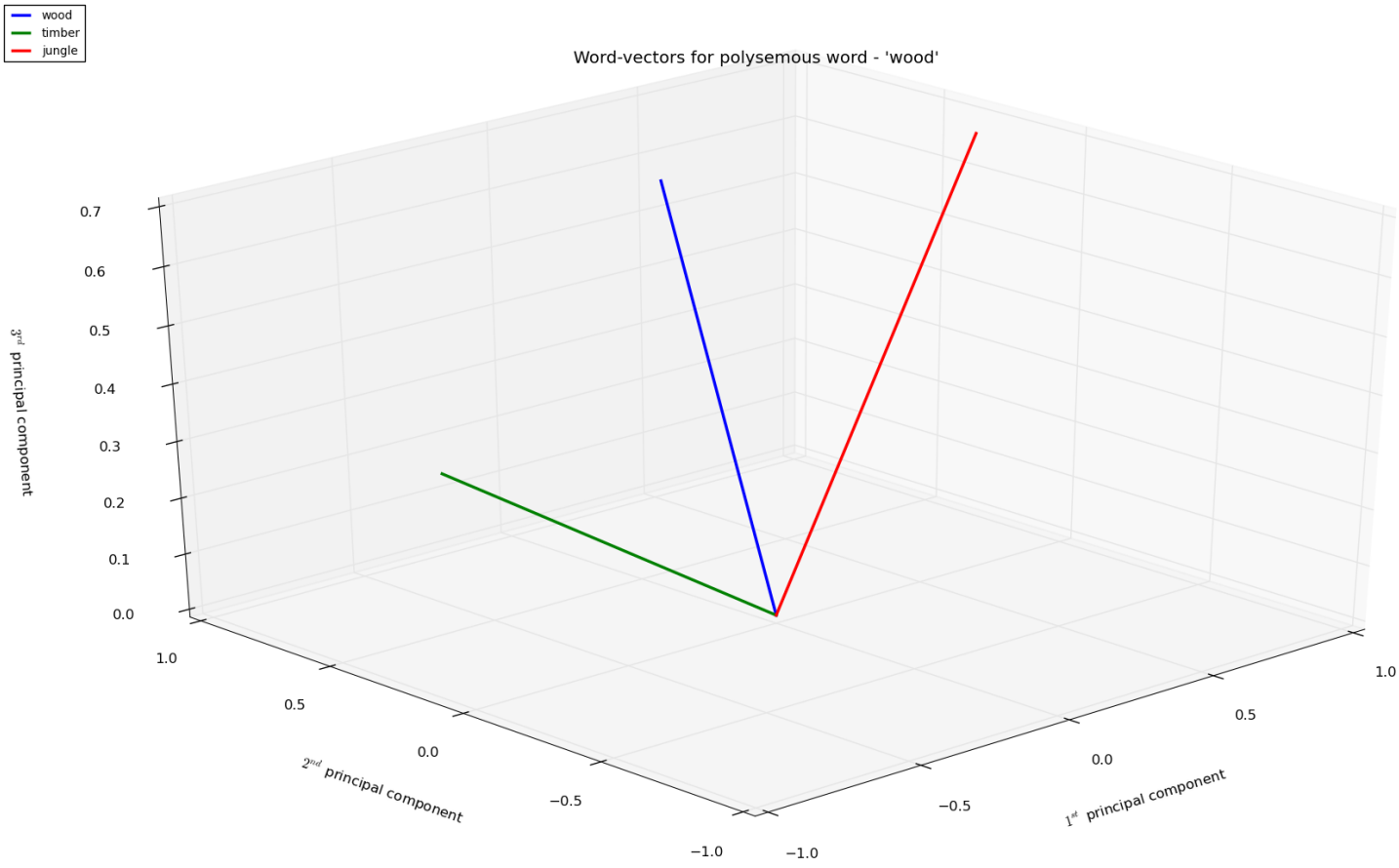

Fig. 5. "wood" vector plotted with its 2 senses

From the visualizations, we can conclude that the word2vec embeddings for polysemous words are confused combinations of their various senses.

8 Visualizations of Sense Embeddings

In order to visualize the sense embeddings, we performed a Principal Component Analysis on the embedding matrix to reduce the dimensionality to three, and

plotted the vectors for some senses. The plots give some insight to how various senses are represented in the vector space.

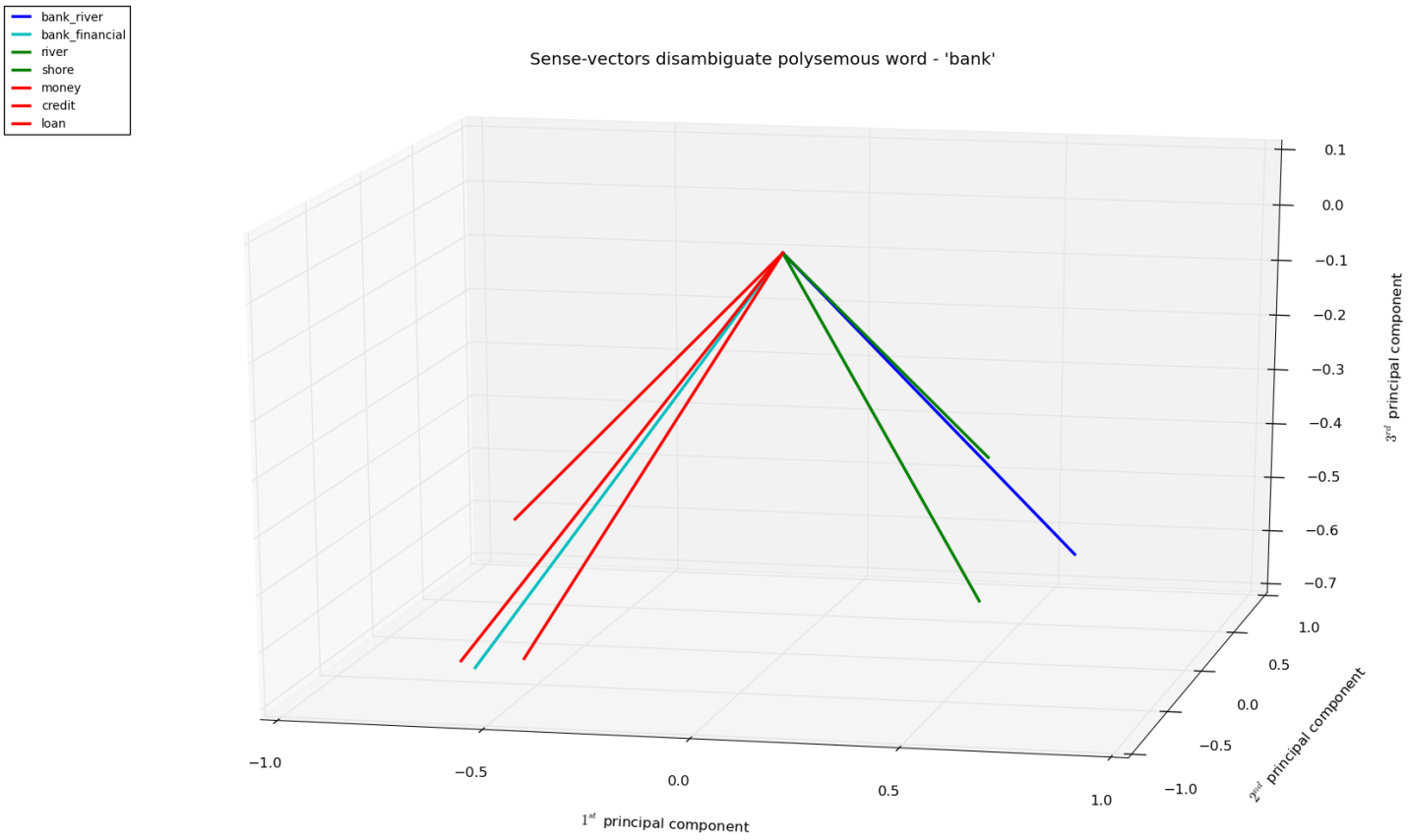

Fig. 6. Disambiguating the senses of "bank"

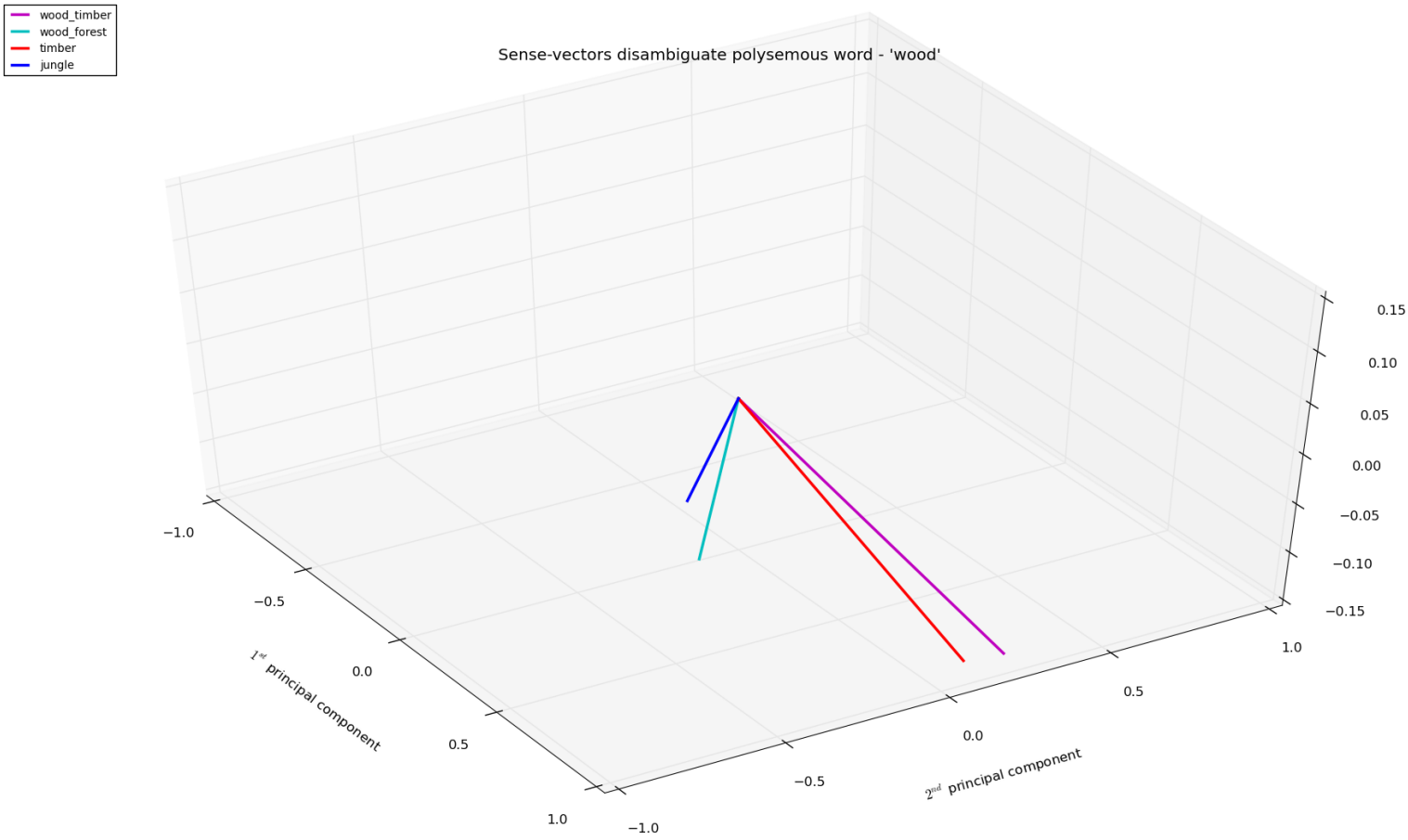

Fig. 7. Disambiguating the senses of "wood"

From these plots we see that sense embeddings indeed do help in disambiguating the various senses of a word. Each sense is closer to other senses that are similar to it, as opposed to unrelated senses.

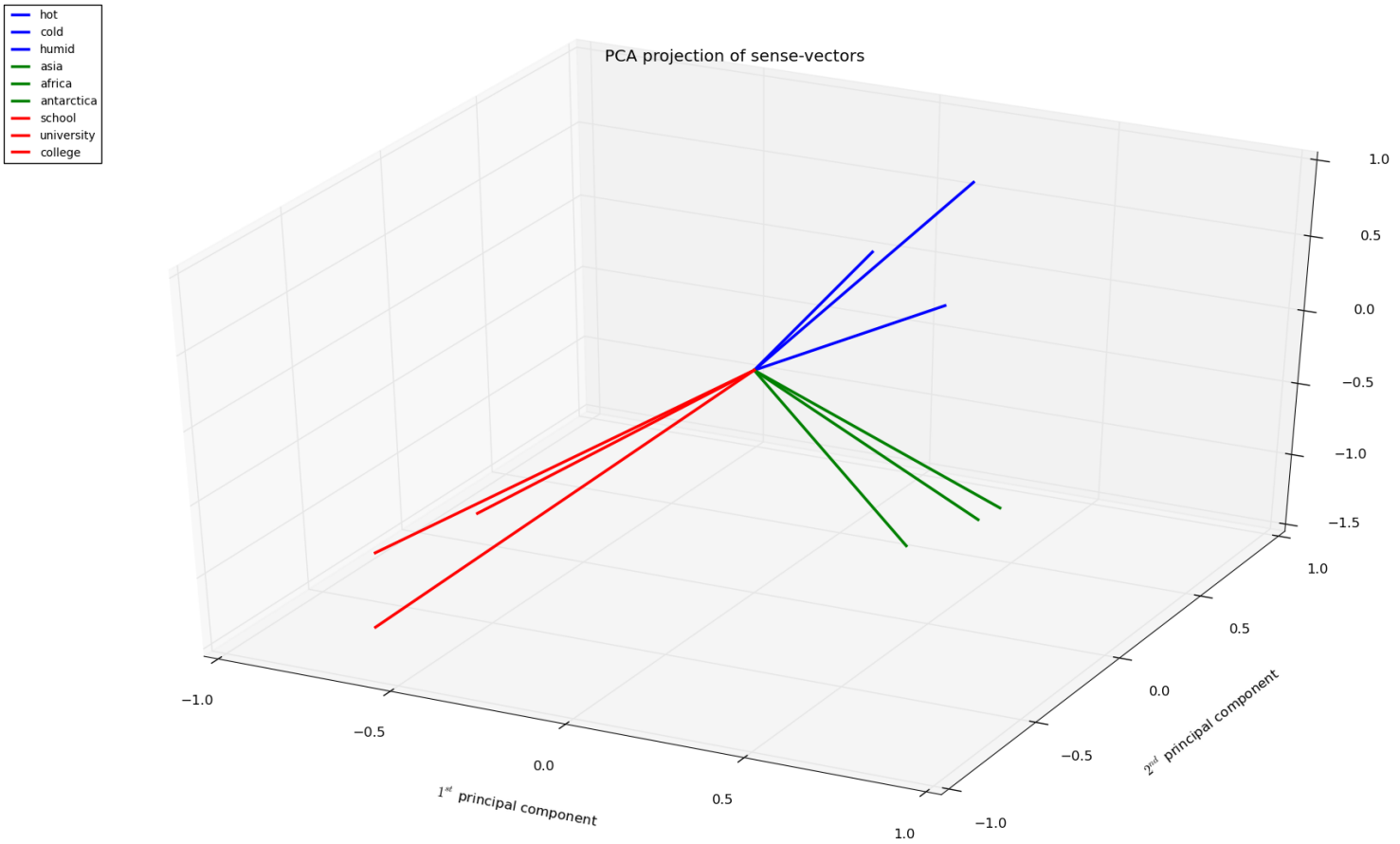

Fig. 8. Visualizing the relatedness between weathers, continents, and educational institutes

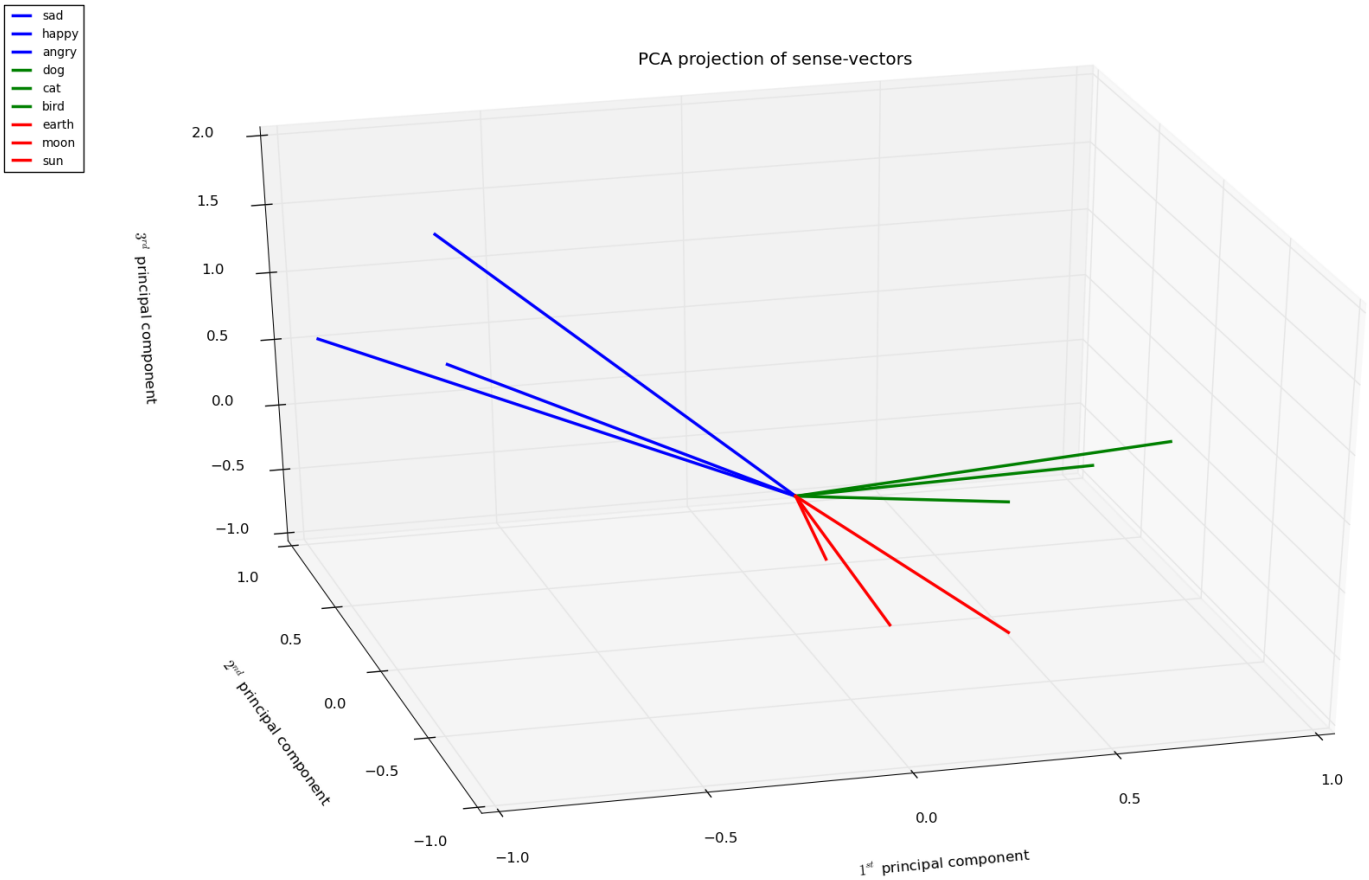

Fig. 9. Visualizing the relatedness between moods, animals, and celestial bodies

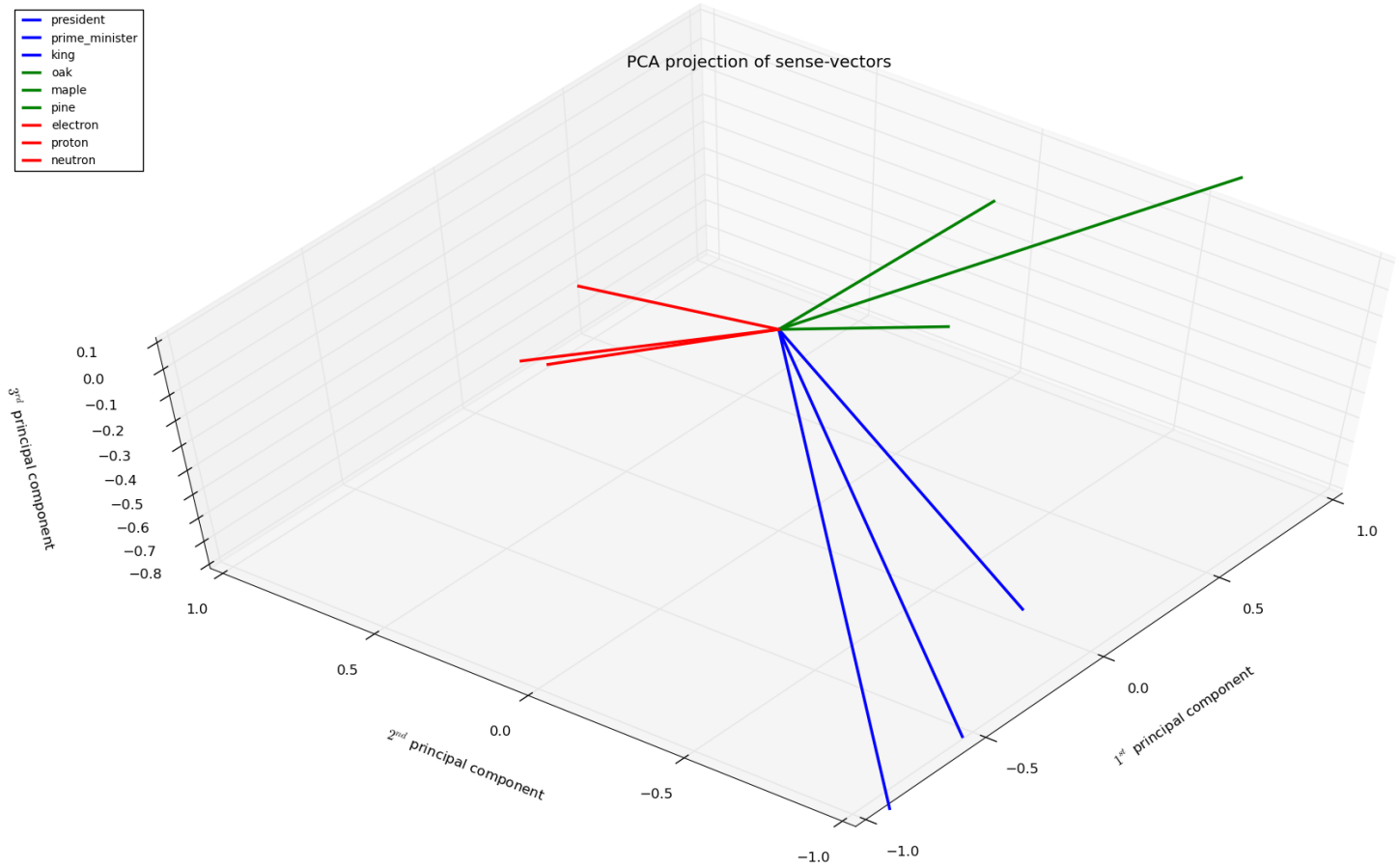

Fig. 10. Visualizing the relatedness between positions of power, trees, and atomic particles

From the above plots, we observe that senses belonging to the same logical category tend to be aligned towards the same direction, while unrelated categories do not show any alignment properties.

9 Observations & Inferences

- We observe that the sense embeddings clearly outperform the standard word2vec embeddings in both the tasks. This can be partly attributed to the implicit disambiguation provided by separate sense vectors, which allowed us to design an intelligent similarity algorithm.
- For both the tasks, the hybrid model outperforms both - the sense embeddings, and the word2vec model. We justify this by saying that word2vec and sense embeddings probably capture complementary information about the words. When they are combined, the cumulative information outperforms both of the composite models.
- From the low-dimensional visualizations of the word2vec vectors, we conclude that for polysemous words, the vectors are combinations of their various senses.
- From the low-dimensional visualizations of the sense vectors, we conclude that sense vectors are indeed markedly separate for polysemous words. This is a significant advantage against word2vec, where all senses are conflated into a single vector.
- Observing the PCA plots, we conclude that senses belonging to a single category are aligned towards the same general direction, while senses in separate logical categories are unaligned.
- In natural corpora, similar words do not appear close to each other in the same sentence, while in our artificial corpus, most words in a context would be similar to each other. e.g. We do not expect a sentence like "animal pet dog pet cat feline" in a natural corpus, but it is likely to appear in the artificial corpus.
- Since similar words are explicitly close to each other in our artificial corpus, while similar words have to be "inferred" in a natural corpus by observing similar sentences, we expect the DeepWalk model to perform much better in similarity and relatedness tasks, which it does.
- In natural corpora, we expect similar words to play the same roles in different sentences. e.g. "The cat walked across the room" and "The dog ran across the bedroom". This property does not necessarily hold in the artificial corpus, since the contexts are not bound by grammatical rules.
- Since our artificial corpus is generated by a random walk method, we do not expect it to have learned any grammar regularities of English. We expect the embeddings to perform poorly if used for syntactic tasks.

10 Bibliography

References

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

2. Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In Advances in neural information processing systems, pp. 3111-3119. 2013.
3. Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 701-710. ACM, 2014.
4. Iacobacci, Ignacio, Mohammad Taher Pilehvar, and Roberto Navigli. "SensEmbed: learning sense embeddings for word and relational similarity." In Proceedings of ACL, pp. 95-105. 2015.
5. Bengio, Yoshua, Rjean Ducharme, Pascal Vincent, and Christian Jauvin. "A neural probabilistic language model." journal of machine learning research 3, no. Feb (2003): 1137-1155.
6. National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov`