# X2VEC

Dan Kufra, Aviv Yaish

## Introduction

Our project implements, extends and reviews the model presented in the paper "Sense2Vec - A Fast and Accurate Method For Word Sense Disambiguation In Neural Word Embeddings"[1]. We'll go over the problems tackled by the paper, their difficulties, how the paper proposes to solve them, our implementation, our additions, connection to the course homework, and further proposals.

## Problem Description

### Word Sense Disambiguation

Given a word, a **word sense** is one of the meanings of the word. In a dictionary, there are can be several entries for each word, each one describing a different sense.

**Word sense disambiguation** is the task of identifying in a dictionary the sense that matches best the usage of a word in a certain given text.

#### Major Difficulties:

- Inter-annotator agreement - as we saw in class, a common benchmark for word sense disambiguation algorithms is comparison of their output to human output on the same task, which is usually the average of several human raters. But, not all humans know all senses of a given word, and even when they do - the agreement rate between them is quite far from 100%[2]. If agreement between humans isn't close to perfect, how can we measure the success rate of algorithms at the same task?

- What are the senses?

  - Different dictionaries can give different senses for the same words.

  - A dictionary can give very fine grained definitions of a given word, sometimes so fine that even human annotators can have a hard time understanding the distinctions between the different senses. Different granularities can be beneficial and harmful for different tasks[3].

### Word Representation & Embedding

The natural language representation of a word in a given language is the word itself, its spelling. As we saw, each word can have multiple meaning. How can we represent all words in the vocabulary of a given language in a way that retains the most information about them? For example, the similarities between them, their senses, etc'?

One solution "category" for this problem is embedding words into a vector space - each word is represented by a vector of real numbers. The embedding can be performed in many different ways, and we will see some interesting examples that use machine learning in our work.

#### Major Difficulties:

- Polysemy - using a single vector to represent a word can be problematic when that word has multiple senses[4].

- Curse of dimensionality - when using machine learning on this problem, we are trying to learn some natural distribution from a finite number of data samples. Natural language is very high dimensional - each word can have many meanings, and can be used in various contexts and in different positions in a sentence, etc'. As such, the learning algorithm needs several samples for each combination of said properties, and usually in order to receive a statistically significant result the amount of samples grows considerably with the number of dimensions.

### Word2Vec

Word2Vec[5] is a recent machine learning based word embedding model which has become widely used. It has two variants, and both rely on the famous adage coined by John Rupert Firth - "You shall know a word by the company it keeps":

1. The first looks at window chunks of size $2n$ of the text and tries to predict the word in the center of the window, meaning that its objective is $\arg\max Pr\left(w_k \mid w_{k-n}, \cdots, w_{k-1}, w_{k+1}, \cdots, w_{k+n}\right)$ .

2. The second does the opposite - it tries to predict the window from the word in its center, meaning its objective is $\arg\max Pr(w_{k-n}, \cdots, w_{k-1}, w_{k+1}, \cdots, w_{k+n} \mid w_k)$ .

Although this model performs admirably in certain use cases, it still generates only one representation for each word, irrespective of the number of senses it may have.

## Sense2Vec

### Premise

The work we are covering in our project is an extension of Word2Vec that is designed to tackle the problem of polysemy by relying on previous works[4] and porting their intuitions to Word2Vec. The main assumption underlying the paper is that the POS of a polysemous word in a certain text is strong hint to its correct sense in that text.

A classic example for the above assumption is the word "duck"; as a noun, duck means "an aquatic bird of the family Anatidae", but as a verb it means "To lower the head or body in order to prevent it from being struck by something".

So, the paper proposes looking at the POS tag of each word and using this additional information during the embedding process.

Note: throughout the paper the authors refer to the POS of the words as its sense (and hence the paper's name).

### Technique

Sense2Vec starts out by preprocessing the text: the POS of each word in the corpus is appended to the word itself.

After this, a Word2Vec model is trained on the modified corpus, and thus we get our desired embeddings.

As the state of the art POS taggers achieve very high accuracy, this technique can be used on untagged corpora, and hopefully with good results.

### Evaluation

The evaluation presented in the paper is mostly subjective. For example, several polysemous words were picked and their closest terms (as measure by cosine similarity) according to the Word2Vec model were presented, and then according to the Sense2Vec model after being tagged manually. For example:

| Word2Vec | | Sense2Vec | | | |
|---|---|---|---|---|---|
| apple | | apple_NOUN | | apple_PROPN | |
| word | similarity | | | | |
| iphone | .69 | apples | .64 | microsoft | .60 |
| ipad | .65 | pear | .58 | iphone | .59 |
| microsoft | .60 | peach | .58 | ipad | .59 |

The evaluation shows that the new model can separate different senses of the picked word based on their POS, if indeed the correct POS is given. But, these positive results were only shown for a very small number of hand picked words.

In addition to this, the model was quantitatively evaluated by using its embedding to train a dependency parse model[8] and comparing the results to the same dependency parse model when trained using Wang2Vec's[7] embeddings. Here the Sense2Vec showed a sizable improvement over Wang2Vec, but we won't delve into the results as it is out of the scope of our project.

### Comparison To Other Papers

The most obvious model to compare Sense2Vec to is Word2Vec, and we have implicitly done so in the "Premise" section.

Another interesting point of reference is "Multi-Prototype Vector-Space Models of Word Meaning" [9]. This work tackles embedding of polysemous words by collecting all the contexts of each occurrence of a specific word in the training corpus, clustering the contexts, and then for each resulting cluster averaging statistically generated vectors for all words in that cluster.

Note that [9] uses unsupervised learning at its core - the clusters themselves might not correspond to any clear-cut senses. On the other hand, if trained on a large enough corpus, this technique might even "discover" senses that are not yet encoded in any dictionary; for example, if trained on an ancient language which isn't yet understood, or if trained on Facebook/Twitter discourse that is heavy on slang terms that still haven't found their way into modern dictionaries.

This is in contrast to Sense2Vec, which uses POS tags which do have linguistic meaning and which are closely related to word senses, both potentially making constraints to one another.

## Our Implementation Of Sense2Vec

We implemented in Python a generic X2Vec class that can be inherited and expanded to handle any kind of NLP token, not necessarily only the POS tag. For example, we have created 3 inheriting classes:

1. Pos2Vec - an implementation of the model presented in the Sense2Vec paper. The POS tags are obtained from NLTK's default POS tagger.

2. Sense2Vec - this model uses the (real) sense as the NLP token, where the sense inventory is the WordNet definition set of each word. The sense is obtained by using the Simplified Lesk algorithm which we saw in exercise 1.

3. Sentiment2Vec - the NLP token for each word is its sentiment with regards to the sentence it is used in. The sentiment is obtained by using TextBlob's sentiment analyzer, which represents the sentiment as a real number $s \in [-1, 1]$. We take this number and assign to it a sentiment in the following way:

$$sentiment(s) = \begin{cases} NEUTRAL & s \in [-0.15, 0.15] \\ POSITIVE & s > 0.15 \\ NEGATIVE & s < -0.15 \end{cases}$$

In addition, we also allowed the use of the X2Vec model without any special NLP token, which is in essence exactly the same as Word2Vec.

### Evaluation

**Corpus and Processing:** We used the corpus supplied to us in exercise 2 for the evaluation of the various models. Before appending the appropriate NLP token, each model preprocess the corpus like so:

- Removes all stop words, punctuation and "bad" words[1].
- Converts all remaining words to lowercase.

Some notes: although the corpus is comparatively small (containing around $20,000,000$ words and weighing around $100mb$) it is practical for our purposes - we got interesting results, and training the models took an acceptable amount of time (Sense2Vec took around 24 hours to be trained on a $\frac{1}{4}$ of the corpus).

**Subjective Evaluation:** We prepared tools to subjectively evaluate the various models we implemented using techniques similar to the subjective evaluation performed in the paper. To do so, we created a file called run_models.py which trains all our models and then proceeds to evaluate them using text files contained in the evaluation_files folder. These files come in two variants:

1. *QA* files: these files contain examples of analogy question answering of the same kind that we saw in question 4 of exercise 2. Each such file contains multiple lines, where each line is of the form "X Y A B" where $X, Y, A, B$ contain the necessary NLP tokens for each model, and where the relation between $X$ and $Y$ is the same relation as between $A$ and $B$. Assuming that $v(X), v(Y), v(A), v(B)$ are the embeddings given for $X, Y, A, B$,run_models evaluates each model and verifies that indeed the embeddings given by the models answer that $v(B)$ is the closest vector to $v(X) + v(Y) - v(A)$.
2. *Sim* files: these files contain word pairs. run_models computes the similarity between them as given by the trained models.

The text files are human generated (a human has to pick the "questions" the model will be asked on). As a proof of concept, we have made several such test files, which you can find in our Github repository. We will go over a small part of our results, just to give a taste.

In table 1 on the following page are some of our results for the word similarity tests, where each cell is the word pair and the cosine similarity given to them by the model. You can see that in the examples we chose, Pos2Vec indeed gives more reasonable answers than Word2Vec. Suprisingly, on some of the examples, Sentiment2Vec outperforms Pos2Vec.

In table 2 on the next page are some of our results for the questions answering tests, where each cell is the question and the answer given by the model. In our chosen examples it seems that Word2Vec has the upper hand on both other models.

You can find more results (in files ending with the "res" suffix under evaluation_files) in our Github repository.

**t-Distributed Stochastic Neighbor Embedding:** t-SNE is a visualization technique for high-dimensional data[6] that puts an emphasis on keeping the low dimensional representations of similar datapoints close together (as opposed to other traditional techniques, that try to keep the low dimensional representations of dissimilar datapoints far apart).

We implemented this technique in the function visualize_embedding which is located in the base $X2Vec$ class, and as such is available to all inheriting classes.

Using this technique, we can find interesting examples that shed light on the effectiveness of the various models. For example, in figure 1 on the following page you can see a small cluster obtained by running TSNE on all of Sense2Vecs embeddings. In it we get that indeed "republican.a.01" (according to WordNet: "relating to or belonging to the Republican Party") is associated with politics, and not any other definition of republican.

You can find more photos in our Github repository.

[1] Fun fact: during our work on the project we found a bug in one of the Python NLP libraries that we used; it would fail on certain sentences that contain specific words in specific locations, for example the word "oed". We located the bug's location in their code and sent them a notice about it.

| Word2Vec | Pos2Vec | Sentiment2Vec |
|---|---|---|
| monitor : check, 0.25 | monitor_VBP : check_VBP, 0.48 | monitor_NEUTRAL : check_NEUTRAL, 0.49 |
| handle : lever, 0.47 | handle_NN : lever_NN, 0.55 | handle_NEUTRAL : lever_NEUTRAL, 0.57 |
| handles : supervises, 0.39 | handles_VBP : supervises_VBP, 0.47 | handles_NEUTRAL : supervises_NEUTRAL, 0.28 |
| watch : look, 0.39 | watch_VB : look_VB, 0.43 | watch_NEUTRAL : look_NEUTRAL, 0.428428 |

Table 1: Word similarity comparison

| Word2Vec | Pos2Vec |
|---|---|
| man : uncle, woman : aunt ? **false** | man_NN : uncle_NN, woman_NN : aunt_NN ? **true** |
| evening : dinner, afternoon : lunch ? **true** | evening_NN : dinner_NN, afternoon_NN : lunch_NN ? **false** |
| music : composer, play : playwright ? **false** | music_NN : composer_NN, play_NN : playwright_NN ? **false** |
| warm : winter, cold : summer ? **true** | warm_JJ : winter_NN, cold_JJ : summer_NN ? **false** |

| Sentiment2Vec |
|---|
| man_NEUTRAL : uncle_NEUTRAL, woman_NEUTRAL : aunt_NEUTRAL ? **false** |
| evening_NEUTRAL : dinner_NEUTRAL, afternoon_NEUTRAL : lunch_NEUTRAL ? **false** |
| music_NEUTRAL : composer_NEUTRAL, play_NEUTRAL : playwright_NEUTRAL ? **false** |
| warm_NEUTRAL : winter_NEUTRAL, cold_NEUTRAL : summer_NEUTRAL ? **true** |

Table 2: Question answering comparison

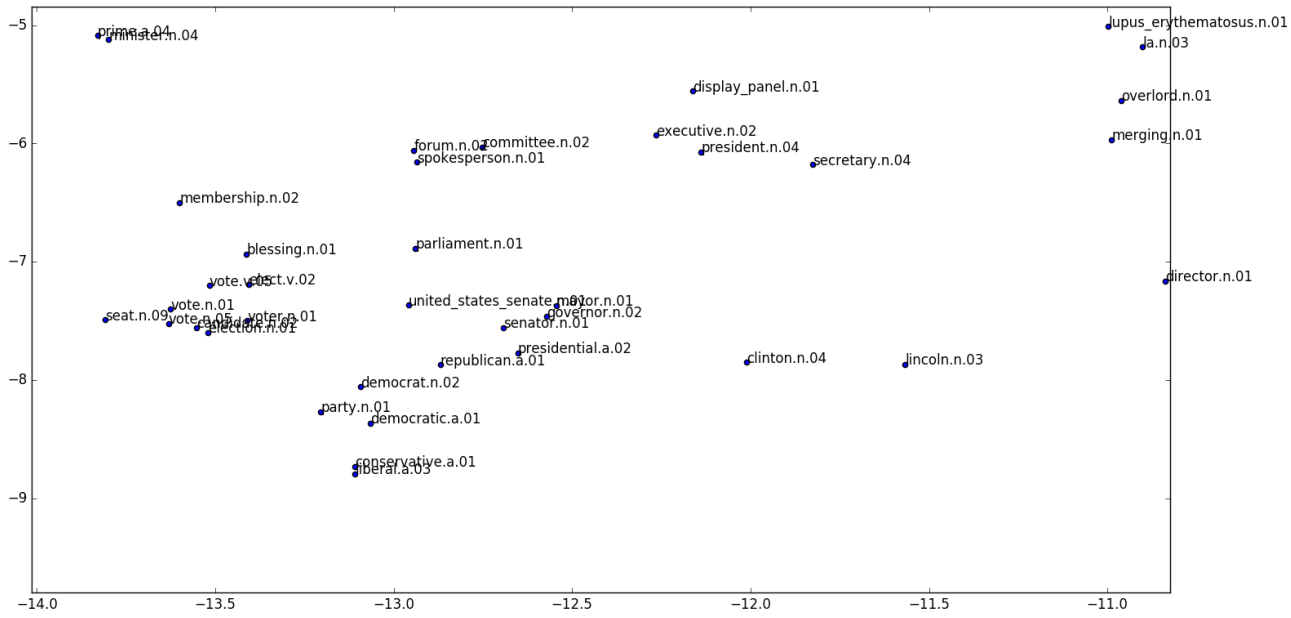


Figure 1: Sense2Vec, politics cluster

**Notes Regarding the Code**

Our code is well documented and is accompanied by a README which explains how to install all necessary software and actually run the code.

All models used in the above evaluations were generated by using the run_models script and can be downloaded here by logging in with a HUJI account.

# Future Work

## Model Improvement Proposals

1. The model proposed in Sense2Vec has its merits, but as we proposed implicitly in our paper, it can be generalized - the NLP token used in the preprocessing doesn't have to be the POS tag, but can also be the actual sense, sentiment, and any other NLP token. To further this generalization, we also propose using multiple NLP tokens at the same time; for example, tagging each word in the corpus both with its POS and its sentiment.

   Training the embedding model when given these additional tags might retain more information about the words in each use case.

2. Regarding the Sentiment2Vec model - one can increase/decrease the granularity of the sentiments. For example, in addition to the 3 basic sentiments we introduced, one can also add "Very positive" and "Very negative" sentiments. Evaluations should be performed in order to determine the correct number of sentiments and their boundaries.

## Experiment Proposals

1. As was said earlier, most of the evaluation performed in the paper was subjective. The only quantitative evaluation performed was in a single NLP task (dependency parsing) against a single relatively not well known model, training both on Wikipedia articles. It seems reasonable that a true test of efficacy would be to compare the model against more widely used models (for example, Word2Vec), in a variety of NLP tasks, and when trained on different types of corpora that are not limited to the encyclopedic genre.

2. Unsupervised evaluation of word embedding models: this is a generic method we thought of that can be applied to any word embedding model, and not necessarily only the various X2Vec models. After training the model, use some clustering algorithm on the embeddings of all words known by the model. Now, for each cluster go over all the (untagged) words assigned to that cluster, and compute the average of the human rated similarity between all words (this is the clusters similarity to itself). Also, for each two clusters, compute the average human rated word similarity between them (this is the similarity between each two clusters). A good word embedding model should have a high first average (because in essence it means that according to humans words that received similar embeddings are indeed similar) and a low second average (likewise, because according to humans words that received dissimilar embedding are indeed dissimilar)

   This method might benefit if PCA or some kind of dimensionality reduction is used before the actual clustering. Then again, this might hint that the models dimensions was too high to begin with.

# Conclusions

In this report we reviewed the model proposed in [1], compared it to other papers, implemented and proposed extensions for it, implemented the extensions (and the original idea presented in the paper), evaluated them, and suggested further methods of evaluation that can give a clearer picture regarding their efficacy. We believe that the paper presented an interesting idea that can prove to be very useful, but needs further and more in depth evaluation.

# Acknowledgments

We would like to give our thanks to:

Andrew Trask, Phil Michalak, and John Liu for writing the paper this project is based on.

The authors of all Python libraries we used, especially gensim, nltk, textblob, pywsd, which are all extremely useful for NLP tasks.

# References

[1] Andrew Trask, Phil Michalak, and John Liu, 2015. Sense2Vec - A Fast and Accurate Method For Word Sense Disambiguation In Neural Word Embeddings.

[2] Benjamin Snyder and Martha Palmer, 2004. The English All-Words Task.

[3] Philip Edmonds and Adam Kilgarriff, 2002. Introduction to the Special Issue on Evaluating Word Sense Disambiguation Systems.

[4] Eric Huang, Richard Socher, Christopher Manning, Andrew Ng, 2012. Improving word representations via global context and multiple word prototypes.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 2013. Efficient Estimation of Word Representations in Vector Space.

[6] Laurens van der Maaten and Geoffrey Hinton, 2008. Visualizing Data using t-SNE.

[7] Wang Ling, Chris Dyer, Alan Black, Isabel Trancoso, 2015. Two/Too Simple Adaptations of Word2Vec for Syntax Problems.

[8] Danqi Chen and Christopher Manning, 2014. A fast and accurate dependency parser using neural networks.

[9] Joseph Reisinger and Raymond Mooney, 2010. Multi-prototype vector-space models of word meaning.