Towards better word embeddings for polysemic words

Shikhar Murty, Sai Praneeth Reddy
{shikhar.murty, saipraneet}@gmail.com

Abstract

Word embeddings are utilized ubiquitously in NLP tasks as a way to overcome sparsity as well as to encode semantic and syntactic information. Most of the work largely assumes that each word in the corpus encodes only one vector. This assumption does not make sense when words could have multiple meanings (polysemy). We implement and extend the state of the art word embeddings using two approaches - a)We use ideas from linear chain CRFs to sequentially perform simultaneous sense disambiguation while learning the word embeddings and b) In cases where POS taggers are available, we show that simply appending the POS tag along with using Noun chunking is a simple straight forward way of tackling polysemy.

Introduction

NLP suffers from the problem of sparsity because the number of words are really large - most of which are tiny variations of each other. Dense continuous word embeddings of words in a low dimensional space are used as a way to tackle the sparsity problem. In such an approach, each word w is represented by a vector $v(w) \in \mathbb{R}^d$ such that words w_i, w_j which are syntactically and semantically close to each other also have close vector representations. Thus one could then harness this similarity to either substitute each new unseen word with the closest seen word, or cluster the words, replacing all words with their cluster ids [Dhi+12], or devise a more sophisticated approach using neural networks. Many state of the art results in downstream NLP tasks in fact use these vectors as features [Dhi+12].

Due to their importance, word embeddings have received considerable attention so far [Mik+13]. However most of the current approaches attach a single vector to each word - polysemy and hononymy are ignored (See Table 1). In many applications, especially in specialized domains, words take on meanings other than the usual English one (e.g. *fork* in git). Thus capturing the different meanings might be crucial to the application. Moreover, the triangle inequality constrains the distance between two unrelated group of words. Thus even though there are relatively few polysemic words,

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

I APPROACHED the brand new JAGUAR
The hunter eyed the JAGUAR with caution
This APPROACH is unreasonable for this problem
..Cause you are a sky full of STARS
This movie STARS very famous STARS

Table 1: Some examples showing polysemic words

they can potentially disturb the groups of vectors surrounding them . For example the words *sky* and *star* are pulled together as $d(sky, actor) \leq d(sky, star) + d(star, actor)$. The solution is then, clearly, to use multiple vectors per word depending on the sense.

We demonstrate two approaches for performing this task. 1) We extend the traditional skip gram model, given a context $C = \{c_1 \dots c_k\}$ and a word w_i, we use the current global word embeddings learnt to determine the context vector. This context vector is then used to determine the sense of c_1. We use this to obtain a better context vector c. This process is repeated till the word sense of the entire context is disambiguated. Then we can use the sense vectors of both the context and the word to determine the log likelihood, and perform the relevant update step. 2) In the second approach, we use the insight that the different senses of a word often arises due to ambiguity about their part of speech label. For example, *bank* (VERB) the chopper hard versus robbing the *bank* (NOUN). However this may not always be enough. Most such cases arise due to the noun requiring further disambiguation; for eg., *river bank* versus *reserve bank*. Here *noun chunking* could be used to distinguish between senses (see). Further chunking also solves the issue of compound nouns for eg. *ice cream* should be treated as a single word as should *Fullham drive*. As we will see, in cases where accurate POS taggers are available, which is the case for general English text, adding POS tags to words is an extremely simple yet remarkably effective way of tackling polysemy.

Related Work and Our Contribution

The idea of distributional similarity of words, has recently become extremely popular and has been used for a variety of downstream NLP tasks from Named entity recognition, to sentiment analysis and paraphrase detection.

There has been some work on learning sense specific

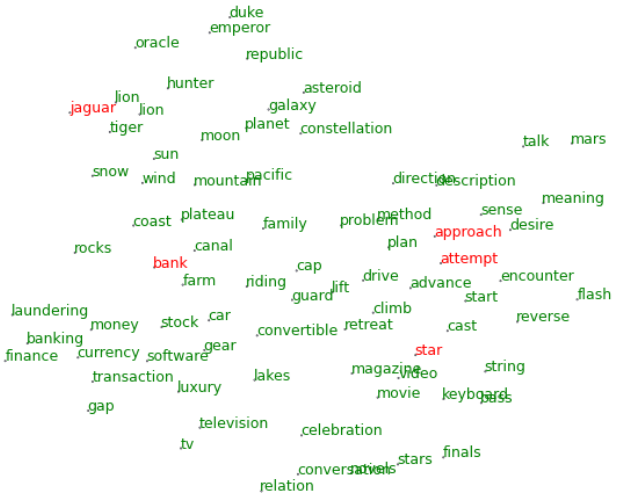

Figure 1: The two clusters around 'bank', one containing 'finance' etc. and the other containing 'rocks', etc. have been pulled together. We use word2vec to obtain the embeddings and then project them onto the 2D plane using T-SNE.

word embeddings, and much of it has focused on clustering word contexts to induce word types. For example, Reisenger and Mooney precluster the contexts of a word and relabel the word tokens according to the closest cluster, given the context in which it appears.

On the other hand, the recent sense2vec model proposed by [TML15] uses supervised sense disambiguation to train a word2vec model, and we explore their work as well.

Very recently, [Nee+15] proposed a joint model for learning multiple word vectors per sense, by using a skip-gram style model. Their model is very similar to skipgram except that they use a simple sense disambiguation technique to determine the right sense of a word, given its context, and use the sense vector corresponding to that sense as the input to the skipgram.

Our work builds on this, by proposing a slightly different sense disambiguation technique and achieves better scores on a limited subset of the wikipedia dataset (with roughly 16 million tokens as opposed to 990 million used by [Nee+15]). Instead of using the global word vectors for the context, we apply a simple algorithm that performs a first order sense disambiguation of the context, and uses that to perform the final disambiguation of the center word to pick the correct sense for input to the word2vec model.

Results

To evaluate our algorithms, we train embeddings on the first billion characters from Wikipedia (and use the preprocessing perl script from the bottom of Matt Mahoney's page). This is a rather small dataset with only 16M tokens as opposed to the 990 million token corpus used by [Nee+15], making our results directly incomparable. For this reason, and also because the original code was written in Scala, we implement and train all the algorithms ourselves in Python. For both LeftRight algorithm and MSSG, we fix

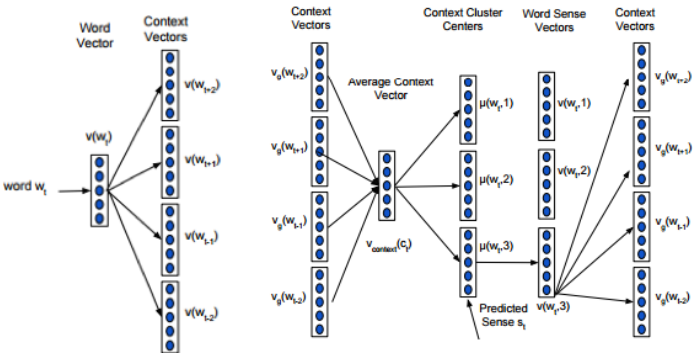

Figure 2: Architecture of the word2vec skipgram model on the left vs. the architecture of the MSSG and the LeftRight model on the right. The LeftRight algorithm involves an additional step of sense disambiguation.

Algorithm 1 Context Disambiguation

procedure CONTEXTVECTOR(i, K)
  $\epsilon := 0.1$
  $x := globalVectors(i, K)$
  for $j := i - K$ **to** $i + K$
    $s := \text{argmax}_k\ senseVec(j, k) \cdot x$
    $l := \text{argmax}_{k, k \neq s}\ senseVec(j, k) \cdot x$
    $bestScore := senseVec(j, s) \cdot x$
    $secondBestScore := senseVec(j, l) \cdot x$
    if $\|bestScore - secondBestScore\| > \epsilon$ **then**
      $x := x - globalVector(j) + senseVec(j, l)$
    end if
  end for
  return x ▷ The context vector
end procedure

the total number of senses to 4 (including the global sense) and leave all other values at their default as in the original paper. For TAG, we group similar POS tags together. When performing noun chunking, we restrict the chunking to only 2 words, and then retain only the top 3000 chunks. The Stanford's Contextual Word Similarity (SCWS), developed by [Hua+12], consists of 2003 pairs of words with context (i.e. used in sentences). These pairs and sentences were chosen to bring out the ambiguity in meaning and the importance of context. The similarity between each pair was then evaluated by humans. The distribution of the words can be seen in 2 and the results of the algorithms in 3. The prevalence of nouns in the test dataset is probably the reason

POS Tag	Count
ADV	35
NOUN	2723
VERB	790
ADJ	294

Table 2: Distribution of major POS tags of words in the SCWS dataset.

Algorithm	Time	GlobalSim	AvgSim	AvgSimC
word2vec	30mins	36	-	-
MSSG	3hrs	50	48.2	44.8
LeftRight	3hrs	**50**	**48.9**	45.62
sense2vec	1hr	-	-	51.16*
TAG	1hr	-	-	**54.96***

Table 3: Results of different embeddings tested on the SCWS dataset. AvgSim takes an average of all the sense vectors whereas the AvgSimC takes an average weighted by the probability of a word being of that sense. In sense2vec and TAG, this probability is either 0 or 1 i.e. they are forced to choose a particular sense. This is sometimes called LocalSim and is a strictly harder metric than AvgSimC.

noun chunking gives such a big performance boost. Note that the sense vectors of LeftRight are better than the sense vectors of MSSG, but their global vectors perform the same. This is probably because the training data was insufficient - global vectors converge must faster than the sense vectors. Thus, we hypothesize that training the algorithms on the full 990M tokens dataset would result in a much better performance of the sense vectors.

Note that the AvgSim metric, defined as

$$AvgSim(w, w') = \frac{1}{K^2} \sum_{i=1}^{K} \sum_{j=1}^{K} d(v_s(w, i), v_s(w, j))$$

calculates the average similarity over all embeddings for each word, while ignoring context information.

On, the other hand, the AvgSimC score defined as,

$$AvgSimC(w, w', c, c') = \frac{1}{K^2} \sum_{i=1}^{K} \sum_{j=1}^{K} P(w, c, i) P(w, c', j) \times d(v_s(w, i), v_s(w', j))$$

takes context into account by assigning weights to each sense based on how probable it is, given the context. We report the Spearman correlation between the model's similarity scores and human judgement as in the SCWS dataset.

Conclusion

We proposed an extension to the word by [Nee+15], and also explored the sense2vec idea by [TML15]. Our model performs better than Neelakantan's model on a smaller dataset. However, on implementing the sense2vec model and running on the same dataset achieves a better localSimC score than all previous methods. We modify the sense2vec algorithm slightly to use noun chunking and retain the top 3000 most frequent chunks.

The extension to the Neelakantan model modifies the joint training procedure by picking the sense based on a context vector, which instead of being the average of just the global vectors of the context tokens, uses a simple disambiguation algorithm, that proceeds from left to right and iteratively improves the aggregated context vector.

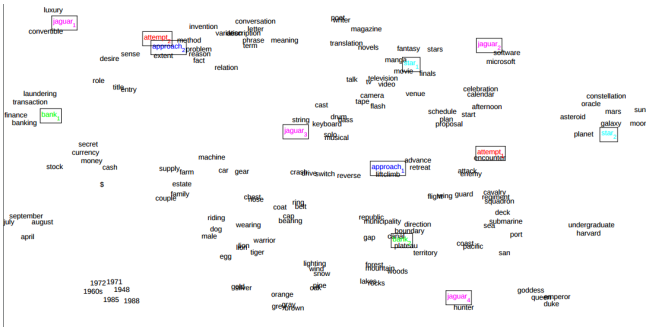

Figure 3: This provides of a qualitative demonstration of the advantage of multiple sense embeddings showing well separated clusters of polysemic words. The projections are performed using tsne.

References

[CLS14] Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. "A Unified Model for Word Sense Representation and Disambiguation". In: *EMNLP*. 2014.

[Dhi+12] Paramveer Dhillon et al. "Two Step CCA: A new spectral method for estimating vector models of words". In: *arXiv preprint arXiv:1206.6403* (2012) (cit. on p. 1).

[Hua+12] Eric H Huang et al. "Improving word representations via global context and multiple word prototypes". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics. 2012, pp. 873–882 (cit. on p. 2).

[Mik+13] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119 (cit. on p. 1).

[Nee+15] Arvind Neelakantan et al. "Efficient non-parametric estimation of multiple embeddings per word in vector space". In: *arXiv preprint arXiv:1504.06654* (2015) (cit. on pp. 2, 3).

[TML15] Andrew Trask, Phil Michalak, and John Liu. "sense2vec-A Fast and Accurate Method for Word Sense Disambiguation In Neural Word Embeddings". In: *arXiv preprint arXiv:1511.06388* (2015) (cit. on pp. 2, 3).