SAPIENZA
UNIVERSITÀ DI ROMA

NATURAL LANGUAGE PROCESSING HOMEWORK 2 REPORT

Creating Sense Embeddings

by

1871835

Submitted on

06/06/2019

1. The Problem

One of the limitation of word embeddings is the meaning conflation deficiency, which arises from representing a word with all its possible meanings as a single vector. This project is one step to address this; **representation of meanings**. Given a **corpus**, we will train a Word2Vec model to obtain its **lemma_synsetID** given a context of words.

2. The Dataset and Pre-processing

I made use of the compulsory **Eurosense** corpus and **Trainomatic** dataset using just the **english sentences**, there were many inconsistencies found in the corpus which I cleaned up. For nomenclature, babelnetIDs will be represented as bnIDs in this section. For both datasets, they were cleaned and put in the format example shown in 2a.

- Removing the bnIDs that are not found bn2wn_mapping.txt file. For Trainomatic, I swapped wordnetID with its respective bnIDs using same bn2wn_mapping.txt file.
- Removal of stop words from our English sentences using the nltk english list.
- Joining of each lemma in the annotation with its corresponding bnIDs since that's what we want our model to learn how to predict *i.e lemma_synsetID*. Every lemma was also converted to lowercase.
- Since our task is to predict a lemma_synset given its context of words, I replaced every anchor found in our annotation with its equivalent value from step III in a sentence. Fig.2a illustrates what a line of our training data looks like.

For performance boost, I also converted the final training to a **gz format** which took shorter time to train than the regular txt file.

> "NLP is a very interesting course to take, it teaches machines literature" **becomes**
> "NLP is a very_bn:00113573a interesting course_bn:03961126n etc……."

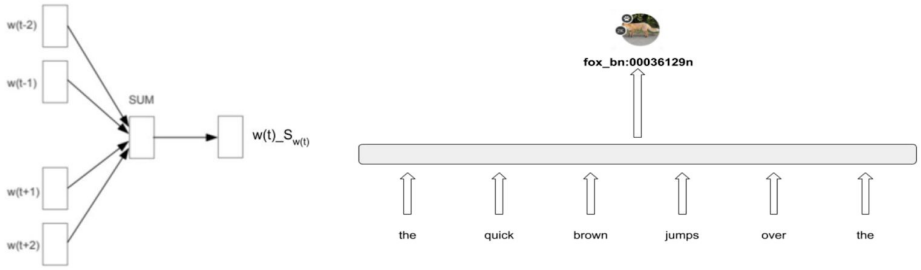
Fig 2a

It's also **worth mentioning** that I trained my model on **precision**, **coverage**, a **combination** of (precision & coverage) and a merged version of **precision**, **coverage** with **Trainomatic**. A table illustrating the performance of these datasets is shown in Fig. 5a.

3. The task and models explored

3.1. CBOW - Continuous bag of words:

Assuming the current word in a sentence is *wi*. The input to the model could be *wi*−2, *wi*−1 , *wi*+1, *wi*+2, the preceding and following words of the



current word we are at dependent on the windows size parameters. The output of the neural network will be *wi*. CBOW is simply predicting the word given its context. **This was the eventual model I used**. Detailed report of its performance is found in section 5.

3.2. **Skip Gram:** The opposite of CBOW such that the input to the model is *wi*, and the output could be *wi*−1, *wi*−2, *wi*+1,*wi*+2, the model predicts the context given a word. I also tried Skip gram for this task but it didn't perform well which is of no surprise because it's known to work well with small amount of the training data, which is not the same as our case.

3.3. **CBOW with keras tensorflow API:** I implemented my own CBOW in order to compare results, which required I build a vocab of my corpus, tokenize the sentence and a custom function to generate my data in batches. However, one iteration of this implementation took a very long time and due to insufficient computation resources, I observed that using existing libraries like gensim, fasttext, etc was more efficient.

4. Model Evaluation using similarity score

Word and sense embeddings can be evaluated using word similarities algorithms. I explored the cosine similarity measure according to the slide and also the gensim similarity function, both gave same results so I eventually settled with the manual implementation I did. I also made sure I removed the "context words" from my embeddings leaving only the lemma_synsetID embeddings.

To **penalize** words not found in embeddings vocabulary against the WordSimilarity-353 [8.4] human annotated score, I used a **cosine similarity score** of **-1** instead of skipping them. Eventually, each cosine score from the annotated scores(gold) were eventually used to compute my **spearman** rank score.

5. Model performance and parameters

At the beginning of training, I focused on experimenting with the datasets while observing the spearman correlation rank score, with the score function explained in section 4. Table 5a shows a summary of the experiment under same set of parameters, which helped make an early decision on the best training dataset to use.

	Dataset used	Using score function explained in section 4	Skipping words not found	Number of pairs of words found out of 353 pairs in the combined.tab file
Training before removing stopwords	precision	0.179	0.256	313 pairs found
	coverage	0.199	0.298	316 pairs found
	combined	0.226	0.331	319 pairs found
	combined + trainomatic	0.204	0.273	319 pairs found
Training after removing stopwords	precision	0.170	0.245	313 pairs found
	coverage	0.186	0.282	316 pairs found
	combined	0.204	0.305	319 pairs found
	combined + trainomatic	0.200	0.292	319 pairs found

Fig 5a

The above experiment made me realize using the combination(precision + coverage) **without removing stopword** as the training data was the best choice, Trainomatic data didn't add any improvement as seen above, and I think this is because the pairs of words found was still same as the combined.

I initially started training using the parameters on the reference paper[8.2] which didn't give great results. After training with different parameters of the models mentioned in section 3, the best model performance

was the **CBOW** which wasn't surprising because of our large corpus vocabulary size. Fig 5b shows the parameters of my final sense embeddings from the combined data and the score obtained.

Here are the final parameters used for my best embeddings:

My final embeddings spearman score result from section 4 was **0.226**					
Model:	**CBOW using gensim and hierarchical softmax**	**window**	*5*	**epochs**	**20**
ns_exponent	**10e-3**	**embeddings size**	**100**	**alpha**	**0.025**

6. Model Visualization

6.1. Similar words and t-SNE(T-distributed Stochastic Neighbor Embedding)

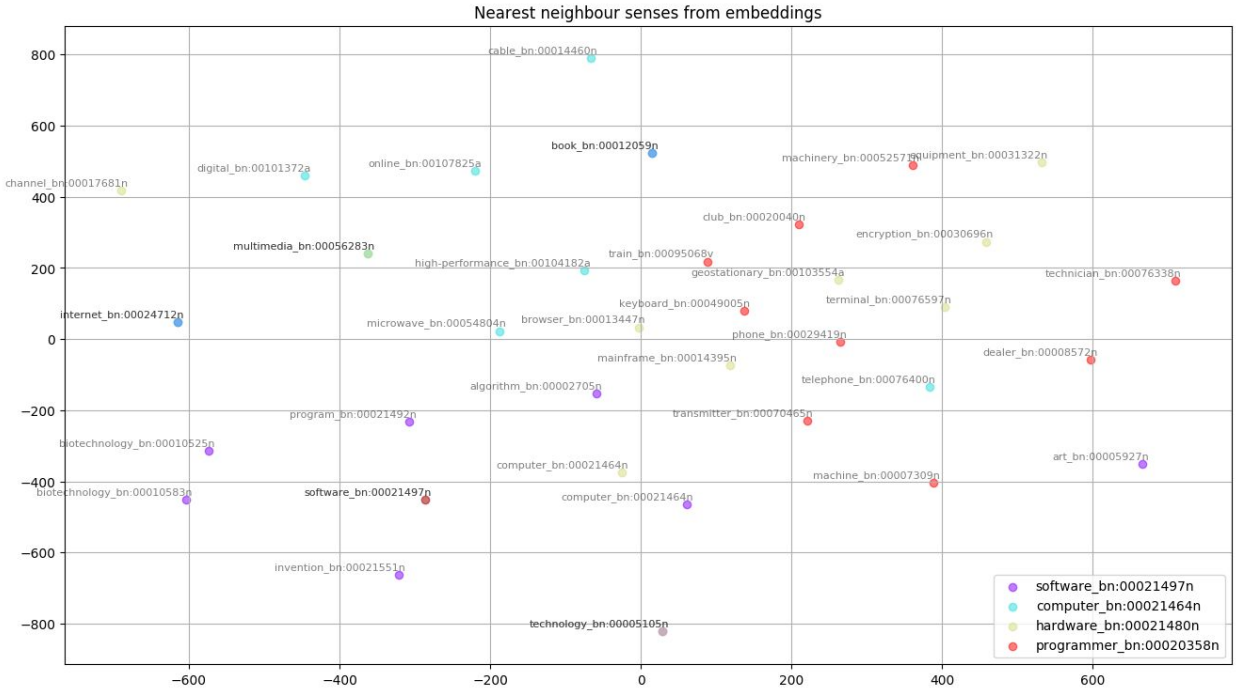
T-SNE is a data visualization algorithm which is quite useful in our case because it is necessary to visualize similarity between objects located in multidimensional space. With a large dataset like ours, it is difficult to make an easy-to-read t-SNE plot, so it is common practice to visualize groups of the most similar words.

Fig.6a shows the similar words from our model, and as seen below using **king** as a case study, we find lemmas like **throne, queen, prince** associated with the sense "king". This I believe is a good benchmark of a good sense embeddings. Fig 6a shows other examples and their top 3 sense embeddings.

king_bn:00049141n	transplantation_bn:00059470n	bank_bn:00008364n
throne_bn:00077086n	*donation_bn:00022271n*	*banker_bn:00008406n*
queen_bn:00034025n	*therapy_bn:00076843n*	*taxpayer_bn:00076252n*
prince_bn:00064405n	*vaccine_bn:00079447n*	*statistics_bn:00074061n*

Fig. 6a

6.2. Nearest Neighbours



Clusters of similar embeddings which also shows the nearest neighbours for 4 embeddings. We will see **algorithm, browser, keyboard, mainframe**, are neighbors even though they are from different senses.

6.3. 3D t-SNE

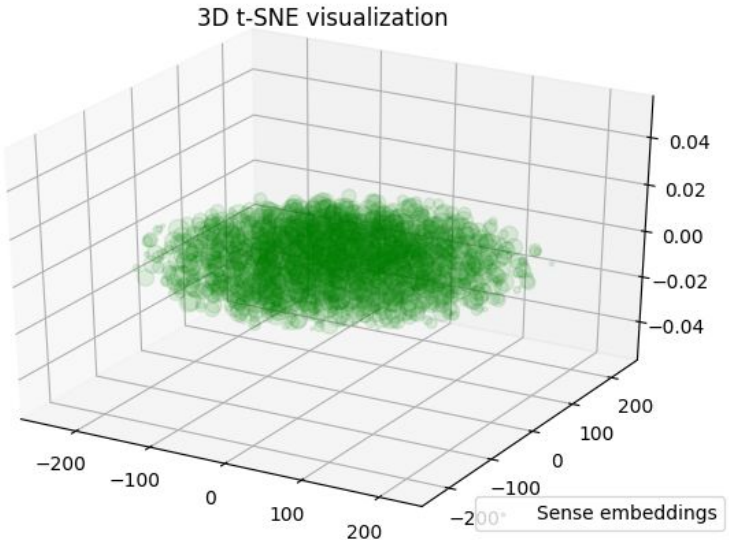


Fig. 6a

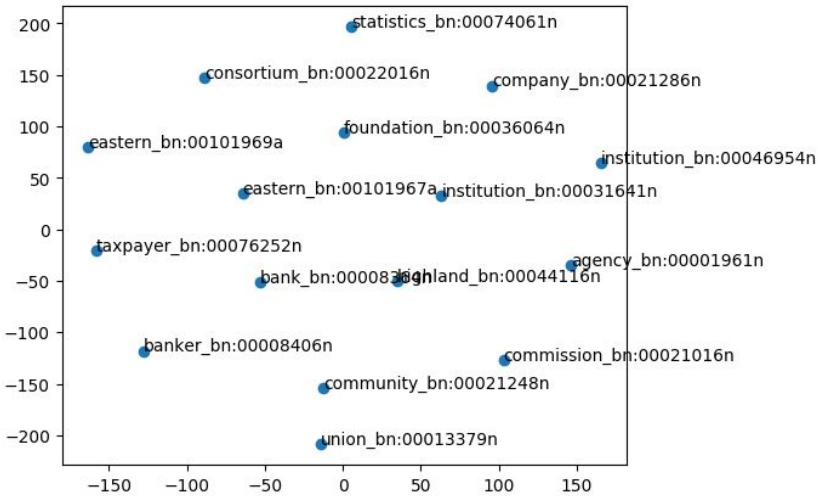


Fig. 6b

Fig 6a shows a 3D visualization of our embeddings on a **random space size of 3000**, this shows how diverse and the distribution of our embeddings in different spaces and clusters. Plotting for all our embeddings would take a huge computational time, hence why we randomly selected a small size.

Fig 6b shows a visual of the table in Fig 6a, we show the similar senses for **bank_bn:00008364n.** As observed above, we can see senses like **statistics, company, banker, taxpayer** which displays the right senses expected for **bank.**

7. Conclusion

We don't have a perfect sense embeddings with a score of **0.226**, and there's room for improvement. Improvement includes using more data like the SEW data which would account for the 34 pairs of OOV's which affected the score a great deal. Also, more hyper parameter tuning using grid search and an increase in epochs should achieve a better sense embedding.

However, we have been able to establish meanings and similarity for senses disambiguation using the visualizations above, this means senses disambiguation can be extended to senses for other languages besides English.

8. References

8.1. Sascha Rothe, and Hinrich Schutze 2015. AutoExtend: Extending Word Embeddings to Embeddings for Synsets and Lexemes

8.2. Ignacio Iacobacci, Mohammad Taher Pilehvar and Roberto Navigli 2015. SENSEMBED: Learning Sense Embeddings for Word and Relational Similarity

8.3. Jose Camacho-Collados, Mohammad Taher Pilehvar 2018. From Word to Sense Embeddings: A Survey on Vector Representations of Meaning

8.4. http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/wordsim353.zip