IMPROVING QUALITY OF FEATURES IN VECTOR EMBEDDING MODELS BY USING SYNSETS

ABSTRACT. We present an innovative embedding word representation, which aims to address the flaws of the neural-network word embedding models popularized by word2vec research which do not accurately capture word semantic complexities. We start by detailing the reasons for our motivation based on search queries of WordNet, a widely-used thesaurus for representing word senses and relations between senses. We also present our methodology in a sense representation of a corpus used to benchmark Word2Vec and the results achieved with such representation on similarity and analogy tasks (Mikolov et al. 2013 [1]). Lastly, we demonstrate the usage of sense embedding vectors in sentence classification tasks.

1. MOTIVATION

Word2vec is a shallow neural network with one hidden layer allowing one-hot encoding of the words. It uses a vector representation of the words, and each word index (0 to the number of words in the vocabulary) is mapped to a low-dimensional vector-space from their distributional properties observed in the text corpus given as input to word2vec. The output feature vectors are the probabilities that each word is embedded near each other (semantic similarity). The weights or feature vectors are learned using two methods: CBOW or Skip-Gram [2] . Although different in nature, they both have the same end result: vectors of words judged similar by their context are nudged closer together. For faster training, instead of computing similarities of every word against every context of the text corpus, these models select noise contexts randomly, a method known as negative sampling. Words semantically close to each other are mapped to vectors with certain algebraic relations as exemplified by the relation: "v[man] - v[king] + v[queen] $\approx$ v[woman]" which is found in Skip-Gram using a cosine similarity metric.

The success of word2vec or similar approaches is two-fold:

(1) compact word representation, we have a low dimensional space which makes NLP tasks more computationally tractable. We can consider this dimensional space as a linear combination of the latent features of the vocabulary
(2) semi-unsupervised method, word2vec, using clever sampling method, produces a model in one training step leading to a faster learning time limited only by the amount of memory available which can be addressed by a framework like TensorFlow [1].

Despite these advancements, the algorithm has limitations. It is not the number of hyperparameters to be tuned. They have been tuned already by many and shown to lead to better optimization without questioning the validity of the algorithm itself. But it is that words with multiple meanings (or "senses") are squashed in the same embedding vector, which leads to less accurate semantic relations. In fairness, this limitation is mitigated by the fact that, in English, the majority of the words are not ambiguous.

2. PRELIMINARIES

Word2vec realizes a matrix decomposition $M = W \cdot C$ where **W is referred as the word matrix , and C as the context matrix**. Each row i of the word matrix W is the $1 \times d$ vector embedding V_i for word i in the vocabulary V. Each column i of the context matrix C is a $d \times 1$ vector embedding C_i for word i in the vocabulary. Each element of M, M_{ij}, reflects the strength of association between word i and context j. More specifically, it is found that $M_i^j = V_w^T \cdot V_c = \mathrm{PPMI}(V_w, V_c) - \log k$, k being the size of the context centered on the target word w, and where:

$$\mathrm{PPMI}(w_i, c_j) = \mathrm{PMI}_+(w_i, c_j) = \max(\mathrm{PMI}(w_i, c_j), 0)$$

$$\mathrm{PMI}(w_i, c_j) = \log \frac{\#(w_i, c_j)|D|}{\#(w_i)\#(c_j)}$$ where D is the set of all (w,c) in the corpus

CBOW computes the probability $p(w_i|w_j)$ by taking the softmax function:

$$p(w|c) = \frac{e^{V_w^T \cdot V_c}}{\sum_{w' \in T} e^{V_w^T \cdot V_c}}$$

And the learning task is:

$$\underset{\theta}{\operatorname{argmax}} \prod_{(w,c) \in D} p(w|c)$$

[1] https://www.tensorflow.org/

The maximum log likehood, is then, computed as:

$$
\begin{aligned}
L(\Theta) &= \prod_{w \in T} p(w|c) \\
l(\Theta) &= \log L(\Theta) = \sum_{w \in T} \log p(w|c) \\
l(\Theta) &= \sum_{w \in T} V_w^T \cdot V_c - log \sum_{w \in T} e^{V_w^T \cdot V_c} \\
\frac{\partial l(\Theta)}{\partial V_w} &= V_c - \frac{e^{V_w^T \cdot V_c}}{\sum_{w \in T} e^{V_w^T \cdot V_c}} \cdot V_c \\
\frac{\partial l(\Theta)}{\partial V_w} &= V_c - p(w|c) \cdot V_c
\end{aligned}
$$

The gradient descent update is therefore $V_w = V_w - \eta \cdot V_c \cdot [1 - p(w|c)]$. Skip-gram algorithm is roughly the mirror image of the CBOW model, but this time predicting the surrounding words within the context from the target word. The model as it is, requires one to compute the dot product against each word in the dictionary, which is very expensive. Instead, a more optimized version of skip-gram is used, skip-gram with negative sampling. From the paper written by Mikolov et al. 2013 [1], the optimization is:

$$
\operatorname{argmax} \log \sigma(v_{wo}'^T v_{wI}) + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w_t \sim P_n(w)} [\log \sigma(-v_{wi}'^T v_{wI})]
$$

Note that assuming we have a bijective function $\mathscr{F} : w \to s$, then the previous objective functions remain the same, they are now solved in a different feature space. The function $\mathscr{F}$ can be the mapping between a word and its sense within the context. During our experimentation, we found that such mapping is not always straightforward.

3. Related Work

There are different efforts to address word2vec 's word ambiguity limitation. They can be split into two categories: one which learns the word sense representation before using an embedding transformation and one which learns senses and vectors simultaneously.

Research in the first category has used a clustering-based approach, where a cluster is defined by sense membership. The cluster membership is determined using different graph based methods. The initial step involves finding word similarities using various similarity metrics.

In the second category, input word cluster membership is determined online, from the current context window. Bartunov et al. (2015)s AdaGram

model [5] can be seen as part of this category. Adaptive Skip-Gram (Adagram) proposes a Bayesian nonparametrics Skip-gram model. It is an extension of Skip-gram, but differs in the expression of the loss function. Adagram tries to capture prior meanings of a word using a Dirichlet Process (DP), which leads to a loss function optimized using stochastic gradient w.r.t a global posterior approximation β, where the prior probability of a k-th meaning of the word w is:

$$p(z = k|w, \beta) = \beta_{wk} \prod_{r=1}^{k-1} (1 - \beta_{wr}) p(\beta_{wk}|\alpha) = Beta(\beta_{wk}|1, \alpha) k = 1, \cdots$$

And the Adagram model is:

$$p(Y, Z, \beta|X, \alpha, \theta) = \prod_{w=1}^{V} \prod_{k=1}^{\infty} p(\beta_{wk}|\alpha) \prod_{i=1}^{N} [p(z_i|x_i, \beta) \prod_{j=1}^{C} p(y_{ij}|z_i, x_i, \theta)]$$

In a way similar to AdaGram, the iSGM model defined by Nalisnick et al. [6], expands the embeddings using a joint distribution over a word vector and its context, hence, expending to infinite their dimensionality. Embedding vectors growing more easily allows one to capture polysemy and homonymy without explicit definitions of these relations within their model. They prove that iSGM is an upper-bound in expectation of the traditional Skip-Gram objective. However, they show that with the appropriate partition function, the expectation becomes finite and with more relaxations, the optimization of the expectation is computationally feasible.

It is also worth mentioning sense2vec. Given a corpus C, sense2vec counts the number of uses of a word and then generates a random "sense embedding " for each use. A sense2vec model is then trained using CBOW or Skip-Gram which can predict a word sense surrounding its senses.

Our approach is similar to sense2vec. It leverages supervised NLP Word disambiguation algorithms by preprocessing a corpus and from it, emitting a "senses" corpus, which is then used as input to word2vec. One could argue that the initial word disambiguation which transforms the words into senses is imperfect. However, the methodology we are proposing is an iterative process where the initial sense embeddings are used to improve the word sense disambiguation step, which then generates better sense embedding vectors.

4. EXPERIMENTATION

WORDNET

To disambiguate words and map them to senses we use Wordnet. Wordnet is a large database of sense relations between words. The core concept is the synset, which is an unordered set of synonyms representing the same concept. These synsets are grouped into nouns, verbs, adjectives, adjective satellites, and adverbs. Some of these semantic relations are: **hyponym/hyperonymy** (ex.animal/dog), **meronym/holonym** (wheel/car).

The WordNet 3.0 release has 117,798 nouns, 11,529 verbs, 22,479 adjectives, and 4,481 adverbs. The average noun has 1.24 senses, and the average verb has 2.17 senses. As you can see below in Figure 1, most of the words have a one-to-one relationship with synsets, hence most of them are not polysemeous.

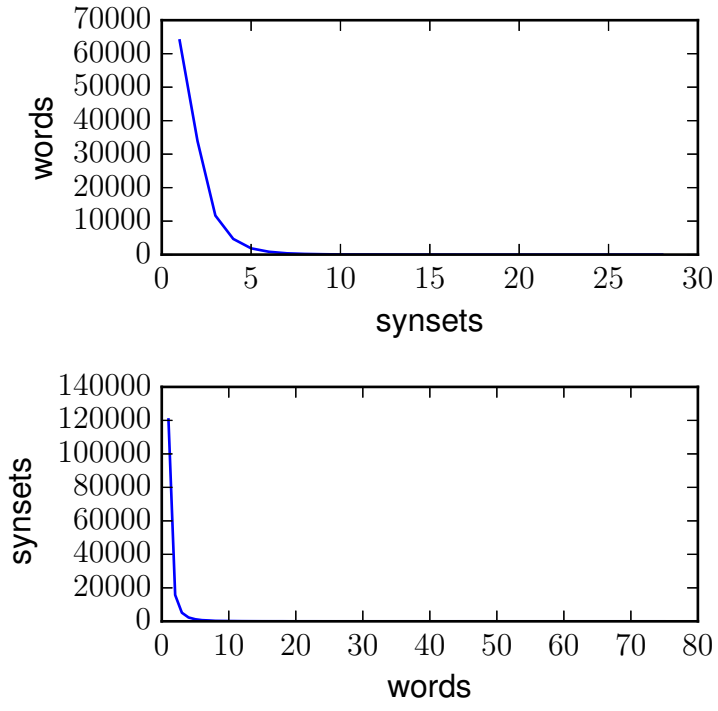

FIGURE 1. Synsets vs. words and words vs synsets distributions

The most ambiguous word is **break** with 75 synsets.

For example the noun "bass" has 8 senses or sensekeys in WordNet:

(1) bass (bass%1:07:01::) (the lowest part of the musical range)
(2) bass (bass%1:10:01::), bass part (bass_part%1:10:00::) (the lowest part in polyphonic music)
(3) bass (bass%1:18:00::), basso (basso%1:18:00::) (an adult male singer with the lowest voice)
(4) sea bass (sea_bass%1:13:00::), bass (bass%1:13:02::) (the lean flesh of a saltwater fish of the family Serranidae)
(5) freshwater bass (freshwater_bass%1:13:00::), bass (bass%1:13:01::) (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
(6) bass (bass%1:13:01::), bass voice (bass_voice%1:10:00::), basso (basso%1:10:00::) (the lowest adult male singing voice)
(7) bass (bass%1:06:02::) (the member with the lowest range of a family of musical instruments)
(8) bass (bass%1:05:00::) (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

The adjective "bass" has 1 sense in WordNet.

(1) bass (bass%3:00:00:low:03), deep (deep%3:00:00:low:03) (having or denoting a low vocal or instrumental range) "a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"

TABLE 1. **Ambiguity by POS**

POS	Min	Max
$verb$	1	59
$noun$	1	33
$adjective\ satellite$	1	21
$adverb$	1	13
$adjective$	1	11

WordNet is quite powerful, but still has a lot of limitations. It does not have relations between different parts of speech. For example, *heart* and *heartily* do not have graph connectivity, making similarity measurement difficult, although you can work around this limitation with a good lemmatizer. The other problem with WordNet is its incompleteness. It seems

that a more recent thesaurus like BabelNet [2], which is a multilingual thesaurus integrating WordNet, Wikipedia, Wikidata, addresses some of these limitations. Nevertheless, due to the lack of non-commercial offering for Babelnet, we decided to use WordNet.

SET UP

In order to validate the proposed methodology, our initial effort was to build a platform to transform an initial corpus into a synset training data set for word2vec. The process involves four different transformation steps:

(1) words to lemma using nltk lemmatizer. The python nltk lemmatization method is based on WordNet's built-in morphy function, which tries to find the base forms of words.

(2) lemma to Part Of Speech or POS. We experimented with different POS-taggers and opted for the simple yet quite accurate nltk "Perceptron Tagger" [3], which is by default trained on 130,000 words of text from the Wall Street Journal and reaches an accuracy of 97.1%. We also tried Spacy, a more robust NLP python library which includes also a parser. However, the end-results in terms of accuracy were not significantly better.

(3) Tuples (lemma, POS) into synset. We tried different Words Sense Disambiguation algorithms starting with the well-known Lesk algorithm (1986), which chooses the sense whose dictionary definition shares the most words with the target words neighborhood. The results were not satisfactory so we switched to more recent variants of Lesk:
   - Cosine Lesk (use cosines to calculate overlaps instead of using raw counts)
   - Extended Lesk which not only looks for overlap between their glosses (word definition), but also between the glosses of the senses that are hypernyms, hyponyms, meronyms, and other relations of the two concepts. We also experimented with the SVM method used in IMS (It Makes Sense [4]), but the model seems more geared towards solving several SensEval and SemEval 2003 tasks than open for general usage. We settled for DKPro WSD [5]), a Java framework which is no longer active but seemed to provide out-of-the box good sense disambiguation

[2] http://babelfy.org/guide
[3] https://explosion.ai/blog/part-of-speech-pos-tagger-in-python
[4] www.comp.nus.edu.sg/~nlp/software.html
[5] https://github.com/dkpro/dkpro-wsd

by using WordNet connectivity graph and returning for senses the WordNet sensekeys. More specifically, from DKPro, we used the "Simplified ExtendedLesk" algorithm with a cosine similarity.

(4) This last step consists of using the synsets obtained from the previous transformation to train word2vec . We initially created our own version of word2vec inspired by the basic tensorflow example [6]. In this version of word2vec, we experimented with different optimizers and loss functions:
   – optimizers: Adagrad, Adam and SGD
   – loss functions: nce_loss and sampled_softmax_loss

During our testing, Adam optimizer was unstable and we did not find a large difference for the final accuracy numbers to favor Adagrad [7] over SGD, and the same was also observed regarding the loss function. In the mean time, we faced performance issues, the optimized version of word2vec [8] with the default optimizer and loss function, then seemed to be a reasonable choice.

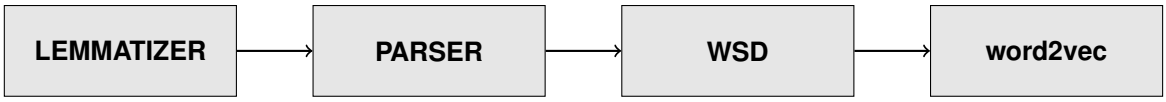

FIGURE 2. Processing Platform

ALGEBRAIC PROPERTIES

Technically the word embedding vectors do not constitute a vector space: commutativity, associativity and inverse element properties are often not verified ($v_{\text{dog rescue}} \neq v_{\text{rescue dog}}$). And synset embedding vectors in the same way fail to verify vector space properties. Nevertheless, looking at the t-SNE representation, the synset vectors have the same cluster shape. And discounting the model mistakes, they also verify the same vector difference: a* - a + b = b*, for vectors having similar senses.

[6] https://github.com/tensorflow/tensorflow/blob/r0.12/tensorflow/models/embedding/word2vec.py

[7] http://sebastianruder.com/optimizing-gradient-descent

[8] https://github.com/tensorflow/tensorflow/blob/r0.12/tensorflow/models/embedding/word2vec_optimized.py

EVALUATION

We initially aimed to generate two models, one based on the corpus text8 [9] and wiki5M, which is the 500Mb of a 2016 Wiki dump and a larger corpus (500Mb versus 96Mb). However due to memory requirements of non-optimized WSD library, our attempts ended up with out-of-memory errors and we were only able to generate text8 based models.

We perform our tests on the same Google's dataset (Mikolov et al.) which contains 19,558 questions expressing two relations: semantic and syntactic. The semantic question contains pairs of tuples of word relations like for example *"Bejing China Paris France"*, there are four kinds of relations: capital-common-countries, capital-world, currency, city and family. A syntactic relation is, for example *big* and *biggest*. There are nine kinds of such relations: adjective-adverb, opposites, comparative, superlative, present-participle, nationality, past -tense, plural nouns and plural verbs. The task is to predict the word y2 that best fits the relationship: (x 1, y1) :: (x2, y2). The question dataset is also disambiguated, the disambiguation step is performed using the same methods as the ones used for the corpus. However, we have to correct the mistakes made by the tagger.

Using the default settings for the hyperparameters (embedding_size=200, learning_rate=0.025, window_size=5, min_count=5, subsample=10-3) and varying only the number of epochs (iterations), in every test case, our model on the synset corpus performs better than the one based on the word corpus. We also can observe that increasing the number of iterations does not significantly improve the accuracy.

Epoch	15	150	300
word text8	36.5% (6511/17827)	39.3% (7005/17827)	39.6% (6888/17827)
synset text8	57.4% (9459/16479)	61.5% (10075/16479)	60.6% (9988/16479)

TABLE 2. Accuracy for word and synset models

[9] http://mattmahoney.net/dc/text8.zip

The results are promising but the breakdown per category shows that, overall, they are not satisfying.

	word text8	word text8	word text8	synset text8	synset text8	synset text8
Epoch	15	150	300	15	150	300
capital-common-countries	82.4 (417/506)	85.6 (433/506)	82.6 (418/506)	47.8 (242/506)	55.1 (279/506)	58.1 (294/506)
capital-world	39.2 (1396/3564)	46.3 (1651/3564)	46.8 (1668/3564)	24.8 (767/3091)	31.5	32.4 (1000/3091)
currency	11.9 (71/596)	13.1 (78/596)	9.7 (58/596)	5.7 (26/460)	2.6 (12/460)	1.5 (7/460)
city	39.1 (912/2330)	48.1 (1120/2330)	46.7 (1088/2330)	18.9 (386/2046)	28.7 (588/2046)	26.1 (533/2046)
family	46.7 (196/420)	44.0 (185/420)	48.1 (202/420)	28.3 (107/378)	35.2 (133/378)	33.3 (126/378)
gram1-adj-adv	7.7 (76/992)	7.2 (71/992)	6.1 (61/992)	2.8 (26/928)	4.1 (38/928)	4.1 (38/928)
gram2-opposite	6.6 (50/756)	6.2 (47/756)	7.9 (60/756)	3.7 (17/454)	3.1 (14/454)	1.1 (5/454)
gram3-comparative	48.6 (648/1332)	40.8 (543/1332)	41.3 (550/1332)	100 (1322/1322)	100 (1322/1322)	100 (1322/1322)
gram4-superlative	20 (198/992)	18.2 (181/992)	15.6 (155/992)	97.1 (1086/1118)	97.1 (1086/1118)	97.1 (1086/1118)
gram5-present-participle	21.5 (227/1056)	26.9 (284/1056)	22.9 (242/1056)	96.9 (1016/1048)	96.9 (1016/1048)	96.9 (1016/1048)
gram6-nationality-adj	74.2 (1129/1521)	77.1 (1172/1521)	76.4 (1162/1521)	61.9 (895/1445)	72.2 (1043/1445)	68.7 (992/1445)
gram7-past-tense	26.9 (419/1560)	24.9 (388/1560)	25.6 (400/1560)	95.1 (1472/1548)	95.1 (1472/1548)	95.1 (1472/1548)
gram8-plural	42.5 (566/1332)	48.3 (644/1332)	46.8 (624/1332)	97.0 (1231/1269)	97.0 (1231/1269)	97.0 (1231/1269)
gram9-plural-verbs	23.7 (206/870)	23.9 (208/870)	23.0 (200/870)	100 (866/866)	100 (866/866)	100 (866/866)

TABLE 3. Accuracy for word and synset models per question category

The errors of the synset models, are word disambiguation errors. They are of different a nature, but can be grouped into three categories:

- wrong prediction, the prediction has no relation to the question
- error due to unknown word, within the context the word was not properly disambiguated like in the question:

Correct Answer	**Predicted**
'malaysia%1:15:00' 'ringit%1:23:00::' 'canada%1:15:00::' 'dollar%1:21:01::'	canada

- confusion of the real meaning of the word in the context of the question. Few examples are given here:

Correct Answer	**Predicted**
Soviet Union (soviet_union%1:15:00::)	Russia, Russian Federation (russia%1:15:04::)
Greece the country (greece%1:15:00::)	Greece, ancient Greece (greece%1:15:01::)
China, People's Republic of China (china%1:15:00::)	China, Taiwan (china%1:15:01::)
England (england%1:15:00::)	UK (uk%1:15:00::)

It is interesting to note new kinds of relation exhibited by the synset model:

Question	**Predicted**
apparent apparently calm **calmly**	stay
apparent apparently infrequent **infrequently**	occasionally
calm calmly slow **slowly**	incremental
certain%3:00:01:: uncertain%3:00:01:: fortunate%5:00:00: blessed:00 **unfortunate%3:00:00::**	waterloo%1:11:00

The synset model has very good accuracy for categories gram3,4,5,7,8, and gram9. Although the difficulty of the synset version of the questions is less than the "word" version, the concept or sense is the same between the question and expected answer. For example, there is no conceptual difference between *decrease* or *decreases*. So we cannot exactly compare the accuracy between models, yet the synset model predictions are better.

NEAREST NEIGHBORS

We show that nearest neighbors in the synset embedding space are semantically related by presenting in tables 5 to 7, the nearest neighbors for the words *race, net, bank, proton, elephant, Maxwell,* and *dream*. When querying the synset model, we have to first select the exact meaning of the target word: race vs. race%1:11:00::, net vs. internet%1:0
6:00::, banks vs bank%1:06:00::, Maxwell vs. maxwell%1:23:00:: (the unit of measure), dream vs dream%1:09:01:: (the noun) or dream%2:36:00:: (the verb). The neighbors in both models are similar but the synset model results are more specific. They don't include plural forms and the similarities are stronger. It is particularly true for *internet*, *bank* and *Maxwell*. In addition, the results are more granular for the synset model, and richer. For example the model captures new relations like: *elephant* is a mammal, the *dream* as a verb and *psychoanalysis*, *proton* and *boson* and *fermion*. But it has also wrong associations like the ones between *bank*, the financial institution, and the other meanings of bank, or *dream* and *something*.

SENTENCE CLASSIFICATION TASK

After training the word embeddings on the small text8 corpus, we decided to run some experiments with the new embeddings. Due to the time constraints we pick a task which has code available online and [10] we decided to perform sentence classification tasks[7]. The idea explained in the paper consists of representing sentences as concatenated word embeddings and feeding this input to a convolutional neural network. There are seven tasks to be performed and they are explained in [7] extensively. 3 of 7 tasks(MR,SST-2,CR) are basically require the model to classify sentences either as positive or negative. SST-1 expands this idea and require 5 classes between positive and negative. Other 3 tasks(SUBJ, TREC, MPQA) involves similar sentence classification tasks.

Four different model are trained for each task by following four different strategy: updating or not updating word embeddings, doing both and randomly initializing the embeddings. As a decent unsupervised method for learning word embeddings word2vec improved the overall performance significantly, around 4-5%. We aimed to evaluate syn2vec embeddings by replacing it with GoogleNews-word2vec pretrained vectors.

To be able to perform the tasks we first converted each word in test sentences into their synset-ids using the algorithm explained in the previous sections. We modified the code base available online to perform the preprocessing. Then we trained the models using our text8-trained word embeddings on NYU's HPC cluster using the default parameters same as the paper and we got the results in Table 4

Task	random	word2vec(static)	syn2vec(static)	syn2vec(non-static)
MR	75.9	80.5	69.2	72.5
SST1	42.2	44.8	35.9	41.9
SST2	83.5	85.6	73.1	80.2
SUBJ	89.2	93.0	88.2	89.0
CR	78.3	83.3	75.0	77.4
MPQA	84.6	89.6	84.7	84.3
TREC	88.2	91.8	90.0	88.8

TABLE 4. Accuracy of different models on sentence classification tasks

[10] https://github.com/harvardnlp/sent-conv-torch

Even though comparing the accuracy of syn2vec embeddings trained on a relatively small corpus text8 wouldn't be fair, we will do so and investigate the possible reasons causing this decrease in performance.

The results with the new word-embeddings are not as good as the word2vec embeddings trained on the 100 billion word Google-News corpus, in some tasks they are slightly better then randomly initilized embeddings. Firstly, the reason for this performance gap might due to the size of the corpus. Even Word2vec trained on text8 gives significantly worst results on analogy tasks compare to its on-GoogleNews-trained counterpart. Secondly the conversion from words to synset-ids is expected to be less accurate when there are not many context words available in the sentence, which is the case for the given data set. In other words, there are sentences with one word or few words, which causes the word disambiguation task to be inaccurate.

5. Conclusion

Every step of the pipeline from transforming a given corpus to a list of synsets and to synset embedding vectors, is critical. The weakest part in our experimentation is the word disambiguation step. It seems that unless tagging for which there are well-known solutions, WSD is still not easily available and really mature. Indeed there is an active, on-going research about it. At the time of our experiments, we did not have access to a GPU, so we could not run a lot of experiments. Even so, besides the imperfections of the synset model, we showed that synset embedding vectors provide more accurate and richer semantic representations than the traditional word vectors. They should be more appropriate than word embedding vectors when being used in NLP tasks such as topic modeling, sentiment analysis, relationship extraction and even Part-of-speech tagging or WSD itself. Next step will be to research a Neural Network based architecture in a supervised setting, performing tagging, WSD and minimizing the errors on the semantic relations of the synset embedding vectors. The source code of our implementation and all results are available at FML-FA16-Project.

References

[1] T Mikolov - 2013 *Efficient Estimation of Word Representations in Vector Space*.
[2] Xin Rong - 2013 *word2vec Parameter Learning Explained*
[3] Roberto Navigli and Simone Paolo Ponzetto - 2012 *Multilingual WSD with Just a Few Lines of Code: the BabelNet API*
[4] Adam Berger, Stephen Della Pietra, and Vincent Della Pietra - 1996 *A maximum entropy approach to natural language processing*
[5] Bartunov, Sergey, Kondrashkin, Dmitry, Osokin, Anton, and Vetrov, Dmitry - 2015 *Breaking sticks and ambiguities with adaptive skip-gram*

[6] Eric Nalisnick, Sachin Ravi - 2015 *Infinite Dimensional Word Embeddings*
[7] Kim, Yoon - 2014 *Convolutional Neural Networks for Sentence Classification*

6. APPENDIX

word model	*cosine similarity*	*synset model*	*cosine similarity*
race	1.0000	race%1:11:00::	1.0000
races	0.6672	race%1:14:00::	0.7133
racing	0.5809	driver%1:18:02::	0.6552
car	0.5795	car%1:06:01::	0.6349
driver	0.5520	racing%1:04:00::	0.5998
daytona	0.4885	race%2:33:00::	0.5809
pennant	0.4829	wimille	0.5215
prix	0.4816	race%1:06:00::	0.5109
hispanic	0.4749	motorcycle%1:06:00::	0.5094
wimille	0.4704	prix	0.5083
drivers	0.4656	pennant	0.5050
motorcycle	0.4616	daytona	0.4934
championship	0.4606	earnhardt	0.4924
cars	0.4521	pennant%1:06:00::	0.4821
horse	0.4501	championship%1:04:00::	0.4754
speedweeks	0.4459	car%1:06:02::	0.4717
ethnicity	0.4443	championship%1:26:00::	0.4705
ancestry	0.4441	bike%1:06:02::	0.4689
speedway	0.4370	nascar	0.4676
road	0.4318	race%2:38:10::	0.4652

TABLE 5. Nearest Neighbors of *race*

word model	*cosine similarity*	*synset model*	*cosine similarity*
net	1.0000	internet%1:06:00::	1.0000
http	0.5305	internet	0.7942
org	0.5211	ip%1:09:00::	0.6143
com	0.5189	protocol%1:10:01::	0.5777
www	0.5164	ip	0.5731
migration	0.4536	provider%1:18:00::	0.5720
https	0.4498	user%1:18:02::	0.5694
html	0.4440	web%1:05:01::	0.5597
website	0.4411	isps	0.5583
online	0.4411	network%1:06:03::	0.5483
htm	0.4395	server%1:18:01::	0.5450
irc	0.4380	arpanet	0.5388
migrant	0.4250	rfc	0.5373
wiki	0.4099	network%1:06:02::	0.5342
database	0.4078	mail%1:10:00::	0.5285
forum	0.4053	wireless%3:00:00::	0.5256
page	0.4016	browser%1:10:00::	0.5201
pdf	0.4015	email%1:10:00::	0.5199
site	0.4007	telnet	0.5159
index	0.4003	ipv%1:06:00::	0.5154

TABLE 6. Nearest Neighbors of *net*

synset	*cosine similarity*
bank%1:06:00::	
bank%1:06:00::	1.0000
bank%1:17:01::	0.5667
bank%1:17:00::	0.5239
proton%1:17:00::	
proton%1:17:00::	1.0000
electron%1:17:00::	0.7486
atom%1:27:01::	0.6992
elephant%1:05:00::	
elephant%1:05:00::	1.0000
elephant	0.5994
elephant%1:10:00::	0.5665
maxwell%1:23:00::	
maxwell%1:23:00::	1.0000
maxwell	0.6392
maxwell%1:18:00::	0.6180
dream%1:09:01::	
dream%1:09:01::	1.0000
dream%1:09:04::	0.5822
dream%1:26:00::	0.5057
dream%2:36:00::	
dream%2:36:00::	1.0000
lucid	0.5623
dream%1:26:00::	0.4833

TABLE 7. Examples of top three nearest neighbors

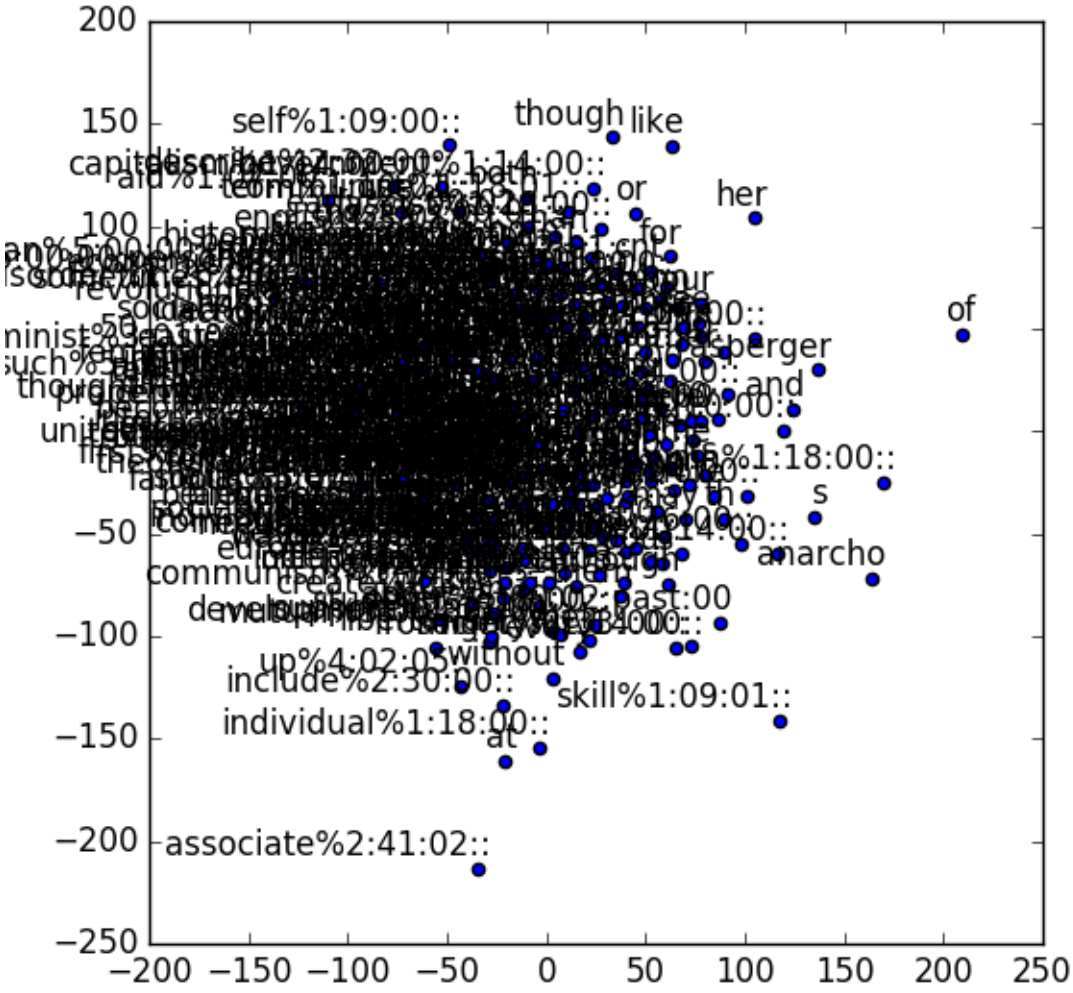

FIGURE 3. T-SNE synsets