Spring 2018 CS 5740: Assignment 2

GITHUB GROUP: THREE-EYED-CROW https://github.com/cornell-cs5740-18sp/assignment2-three-eyed-crow

Yubin Xie
yx443

Disheng Zheng
dz336

Xiang Niu
xn22

1 Introduction

Task: Our project objectivity is to build vector representations word embedding given a large corpus of training text. Our aim is to represent each word as a vector in a way that similar words are close to each other. Formally, we define the dot product similarity of these vectors accurately represents the semantic similarity of the words they represent.
Data: We mainly used the top 1 million sentences in 1 Billion Word Language Model Benchmark as our development training data. Top 1.8, 2.7 million sentences are used for comparison on training size and 2.7 million is used for error analysis. The data contain news sentences in raw text and the 1-M data also analyzed with a part-of-speech tagger and dependency parser. The parsed data is in CONLL format.
Evaluation: The evaluation code calculates the similarity of a word pair by taking the dot product similarity of the two words vectors. The embedding will be evaluated on the development and test dataset. The entire dataset is then scored using the Spearmans rank correlation coefficient between these predicted similarities and the human annotations in the dataset.

2 Approach

We took the approach of skip-gram with negative sampling model as our word embedding method (Mikolov et al. 2013 a,b). Not only does it produce useful word representations, but it is also very efficient to train, works in an online fashion, and scales well to huge copora as well as very large word and context vocabularies. We experiment with linear bag-of-words (BoW) contexts and dependency-based syntactic contexts. In particular, the bag-of-words nature of the contexts in the skip-gram model yield broad topical similarities, while the dependency-based contexts yield more functional similarities of a cohyponym nature.

2.1 Linear Bag-of-Words Contexts

In this model, using a window of size k around the target word w, 2k contexts will be produced: the k words before and the k words after w. When k = 1, the contexts of the target word w are w_1, w_{+1}. Let's take a sentence in our data *"What a World Cup"*. We generate the context pair {(what a), (a what), (a world), (world a), (world cup), (cup world)} and the vocabulary of {what, a, world, cup}.

2.2 Dependency-based Contexts

We utilized the structured context dependency parse tree model to capture arguments and modifiers. Therefore, similar functional words would have similar scores. Here is an example sentence Australian scientist discovers star with telescope., the scientist is a subject noun (nsubj). Australian is its abbreviation modifier (amod). Star is its dependent object (dobj). Telescope is its preposition (prep). Scientist could be similar to astronomer. This information is represented in the CONLL data file, where the 8th column is the DEPREL: Universal dependency relation to the ROOT or a defined language-specific subtype of one.

2.3 Skip-gram model

Skip-gram model is used to find similar words that appeared in similar context. The goal of skip-gram model is to maximize the corpus probability given corpus D of word pairs (w, c) where w is the word and c is the context.

$$argmax_{v_w, v_c} \sum_{w,c \in D} (\log e^{v_w * v_c} - \log \sum_{c\prime} e^{v_w * v_{c\prime}})$$

Here c' is all the context. In the skip-gram model, each word w ∈ W is associated with a vector $v_w \in R^d$ and similarly each context $c \in C$ is represented as a vector $v_c \in R^d$, where W is the words vocabulary, C is the contexts vocabulary, and d is the embedding dimensionality. Following the example in section 2.1, for a given input word of {"what"}, the Skip-gram model tries to optimize according to the cost function described earlier, and gives an output of its most likely context in this case is {"a"}.

2.4 Negative sampling

To train the skip-gram neural network, it needs to update a large amount of weights for each of our 10-million-level context pairs. Negative sampling addresses this issue by only updating small amount of weights rather than all of them for each training context pair. Thus, negative sampling is an efficient way of deriving word embeddings. For each observed word-context pair, negative samples D' are generated based on trigram distribution. $argmax_{v_w, v_c} \sum_{w,c \in D} \log \frac{1}{1 + e^{-v_w * v_c}}$

This objective admits a trivial solution in which P(D = 1—w, c) = 1 for every pair (w, c). This can be easily achieved by setting $v_c = v_w$. In order to prevent the trivial solution, we generated the set D' of random incorrect (w, c) pairs, yielding the negative-sampling training objective:

$$argmax_{v_w, v_c} (\sum_{w,c \in D} \log \frac{1}{1 + e^{v_w * v_c}} + \sum_{w,c \in D\prime} \log \frac{1}{1 + e^{-v_w * v_c}})$$

The probability of selecting a word as negative sample is proportional to its frequency in the corpus. More frequent words such as "said", "the" will have bigger chances of being selected as negative samples. The original word2vec implementation, the author empirically found that to model this probability as: $P_n(w) = \frac{U(w)^{3/4}}{Z}$ outperform the trigram *U(w)* and uniform distribution.

3 Experimental Setup

In this section we described our empirical setup. And all the details necessary to replicate our experiment, including details about the data (preprocessing) and the model architectures (hyperparameters tunning, ablations). Finally, we present a final model used for testing, including hyperparameters and features used.

3.1 Data processing

3.1.1 Feature preprocessing: Upon inspection of the data:
1. Case-folding (reducing all letters to lower case) was applied to avoid distinguish between words simply on case.
2. Remove Number and Punctuation since they are not relevant and not informative to our analysis.
3. Remove English stop words that are too common to be informative.
4. Stemming to transform words to its root form.
5. Remove sparse terms by a threshold. Limited training examples can hardly produce reliable results and the vocabulary size will be significantly reduced after removing infrequent words (Table 1). However, a large threshold removes words with reasonable number of training examples. We considered this trade-off by increasing the value of threshold and choose 10 as our threshold for 1-M data, 15 for 1.8-M data, 30 for 2.7M data. Here we take the 1-M data as an example: threshold 10 helps to reduce vocabulary size from 214K to 33K. Considering the form of contexts in dependency-based model is different from the bag-of-word model, after remove sparse words, we also remove sparse context with threshold 10, it helps reduce the context size from 480K to 126K.

Table 1 Relationship between threshold and vocabulary size based on 1-M data

Threshold	0	5	10	20	50
Vocabulary size	214K	50K	33K	22K	13K

The pre-processing was done before generating word-context pair, this enlarges the context windows size for many tokens in the bow model, because they can now reach words that were not in their original K-sized windows.

3.1.2 Feature representation: We sought a representation that captures semantic and syntactic similarities between words, where words in similar contexts have similar meanings. By utilizing skip-gram with negative sampling embedding model, we represent words as dense vectors that are derived by various training methods inspired from neural-network language modeling. We have utilized dyNets lookup parameter function to quickly lookup and update parts of the embedding.

3.1.3 Sub-sampling: Frequent words are less informative and sub-sampling frequent words can decrease the number of training examples and speed up computation. Here we experimented two sub-sampling methods.
Method1: Sub-sampling on the high frequent words by applying a threshold (**Sub-high**). we used threshold 2000, which means we randomly pick 2000 samples of the words whose occurrence is above 2000.This reduced training sample in bow model with window size 2 from 44M to 13M and reduced 15M to 7.2M.
Method2: Sub-sampling on all the words weighted on its frequency (**Sub-weight**). This method was used by Mikolov. For each word w_i in the training set, well discard it with probability given by $P(w_i) = 1 - \sqrt{t/f(w_i)}$, where f is the frequency of word w_i, and t is the self-defined threshold equals to 1e-5. This reduced training sample in bow model with window size 2 from 44M to 12M and reduced 15M to 3.9M.
We experimented these two methods on BoW model with windows size 2, dimension 50, batch 300, negative sample size 5. After 6 epochs, we got 0.356 predicted-actual correlation with method 2 and 0.488 with method 1. Thus, we used method 1 for the future development.

3.2 Embedding architecture

Embedding dimensions: Here we tested 50, 200 and 300 dimensions and their analysis is in results section. **Negative sample size:** The original word2vec paper suggested that selecting 5-20 negative samples for smaller dataset while only 2-5 were needed for large dataset. Here we implemented with 5 negative samples per context pair. The frequency for a context being sampled is weighted by 3/4 power of its frequency in vocabulary. **Batch size:** 300. **Optimizer:** AdamTrainer. **Loss:** $-(\log \frac{1}{1+e^{v_w * v_c}} + \sum_{c' \in C'} \log \frac{1}{1+e^{-v_w * v'_c}})$ where C' is the sampled context.

3.3 Unknown words inferring

During development, we noticed that in 1-M data, there are 3 unknown words in development set and 696 unknown words in test data given the described preprocessing. We developed 3 different approaches for handling unknown words in sequential order. If a unknown word is not handled by the previous approach, it will be passed onto the following approach.
3.3.1 Synonyms-based inferring: Many of the unknown words may have more common synonyms. We used synsets function in the nltk wordnet library to replace unknown words with their synonyms that exist in our vocabulary to create a more robust words embedding. Some words have multiple senses and their respective synonym. We have averaged the embedding for each synonym found.
3.3.2 Morphological inferring: Words have root form that could reveal essential information. Some complex words are a combination of different root words and they have similarity meaning to the combination of individual words, though sometimes more close to one of them. For example, *grandfather* is a combination of grand and father and its meaning is actually close to putting them together. We have preprocessed all words to their stem form. Then we would represent a unknown word if the word is contained in another word in the training data, or if the unknown word contains some word in the training data. We limited the word to be at least 4 characters long to proceed this morphological inferring since short root words are more ambiguous.
3.3.3 Low frequency words inferring: If there are still unknown words left from the previous processing steps, we

average the embedding of the words appeared less than 35 times in the training dataset and use that embedding for the left unknown words. In end, the unknown token behaves like other infrequent words.

3.4 Final model

Parameters: First 2.7M sentences in the 1-B Word Language Model Benchmark as training data, preprocessing as described at section 3.1.1 (threshold being 30 for low frequency words), sub-sampling method as Sub-high with threshold 2000, embedding dimension 300, negative sample size 5, batch size 300, unknown words inferring as described at section 3.3, 7 epochs.
Results: 0.576 in development data and 0.353 in test data.

4 Results and Analysis

4.1 Amounts of training data

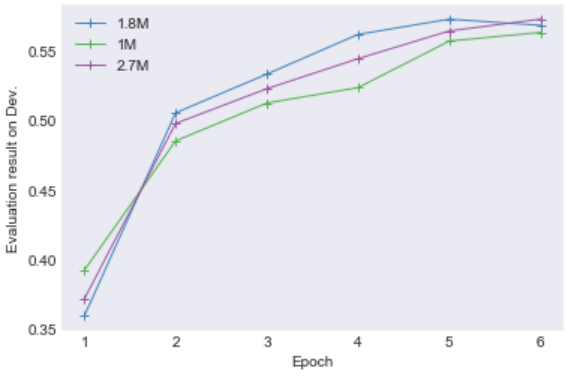

Figure 1: Compare Different Training Data Size

We first tested our model on the 1 million, 1.8 million and 2.7 million data with BoW contexts window size 2, 300 dimension, 5 negative sample (Figure 4.1). We found that by using larger datasets, we were able to achieve better similarity performance on the development set. Though the improvement is small.

4.2 Dimensions for the embeddings

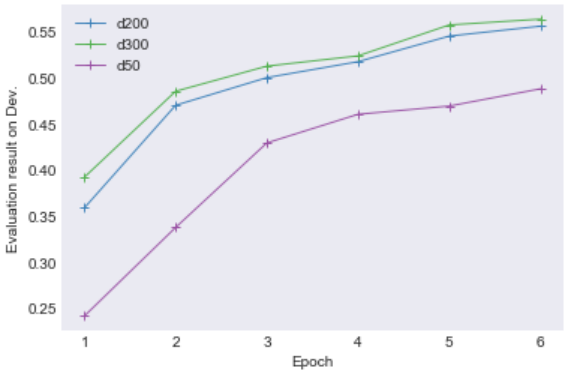

Figure 2: Compare Different Embedding Dimensions

We tested 50, 200 and 300 dimensions of word embeddings all under the 1-M data, BoW contexts with window size of 2 and 5 negative samples (Figure 4.2). We found that with only 50 dimensions, the model word vector similarity performance on the development set is around 0.488 which is much worse comparing to 200 and 300 dimensions, which all reached around 0.56 similarity. It suggested that the intrinsic dimensionality of our data is better represented by higher word vector dimensions around 200 to 300. Lower word vector dimensions like 50 may not be optimal to represented the similarity of the words in our vocabulary.

4.3 Contexts generation

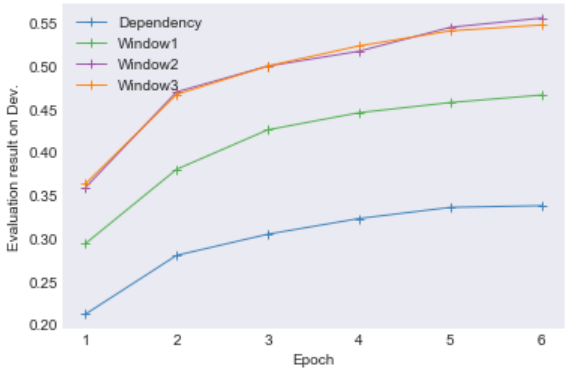

Figure 3: Compare Different Context Embeddings

We experimented with BoW contexts with window size 1,2,3 and dependency-based context described above (Figure 4.3). The model was implemented using 1-M data, 200 dimensions and 5 negative samples. For BoW model, window size 2,3 show similar results: around 55% (size 2 out-performs size 3 at the 6th epoch though) while window size 1 shows a 10% lower performance, perhaps due to lack of ability to capture relatively long-distance but essential contexts. Interestingly, we observed lower performance of word similarity: 33% in dependency-based context model. We noticed that after parsing the dependency relationship, our context pair increased from 33906 to 126793 which resulted in a great increase in sparsity. Therefore the limited data size could be a potential explanation that we didn't achieve similar performance as the authors stated in Omer and Goldberg 2014.

4.4 Embedding Visualization (tSNE)

We visualized our word embeddings (Figure 4.4) from the final model using nonlinear dimensional reduction of tSNE (Maaten and Hinton 2008). In short, the words shared similar meanings or semantic functions should be closed to each other on the tSNE plot. We picked 3311 words from dev and test data with 300 dimensions from our outputed embeddings. Here we selected four different words {*'large','weapon','church','hazard'*} together with their top 5 closest words based on the 300-dimension word vectors, and visualized on the tSNE plot. We know that these words were pretty different by their functions and meanings, and in the

plot they are largely separated. The closest words to *large* included its noun form *largeness* and its synonym *spread, widen, grow*. *Portioned* is usually used together with *large*. The closest words to 'weapon' included its verb form *weaponize* and situation that using weapons like 'contraband' and adjective for weapons, like *indiscriminate* . Hazard is close to its synonym 'dangerous' and words related to the events with hazard like *unforeseen, distressful, distress, disturbances*. For *church*, here we have its purls *churchs* (maybe a typo of churches) or it is actually a name but after being lowercase and stem, it was mistakenly regarded as church. As for the rest words, *baptistic, cemetery, preordained, ordian* are highly related to church and religions. To summarize, the tSNE visualization confirmed and verified the quality of our word vector embeddings.

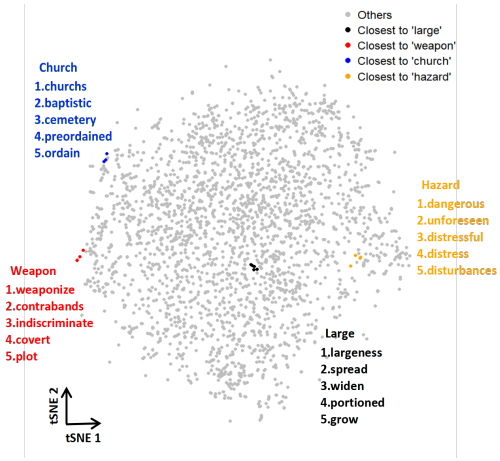

Figure 4: tSNE dimensional reduction of the embedding

4.5 Error analysis

Table 2 Top 5 error pairs in the dev set

Rank	Human	Prediction	Pair
1	8.87	1.25	(asylum, madhouse)
2	7.88	0.88	(life, death)
3	8.03	1.2	(hotel, reservation)
4	4.25	11.45	(benchmark, index)
5	7.61	0.90	(boxing, round)

We quantified the prediction error by measuring the increase of predicted-actual Spearman's rank correlation after removal of that comparison pair and listed the top 5 in the above table, which is based on the final model. Part of the error can be explained by lack of training examples. Take top 1 error as an example. The word *asylum* appeared more than 200 times in training data whereas *madhouse* is not included in our vocabulary. This problem might be overcome with larger training data and with different data source (*madhouse* is an informal word and hardly will it appear in news data). Another reason can be that words usually appear in different context because words can have different senses. In our secondly ranked error, *Life* and *death* are common words in our training data and should be close to each other given their meaning. In our results, the top similar words for *life* are *lesson, experience, learn*, and they are near this sense of life: a characteristic state or mode of living instead of this: the state of be alive. Nearest words for *death* are *victim, suspect, prisoners*, which uses the sense of 'the state of being not alive'. The source of this error is that one of the sense of life, which is not relevant to death, is used more often in real life, or at least in our training data. When we analyzed 3^{rd} and 5^{th} error pair, we noticed that the pre-processing can be the source of error. In our embedding, top similar words for *reservation* are *reserve, icelandic, unreserved*, this turns out that *reservation* after stemming becomes *reserv*, and what we actually captured from embedding is one of the *reserve*'s sense: a district that is reserved for particular purpose. This is far away from *hotel*, whose nearest words here are avenue, anterooms. After stemming, *boxing* becomes box, and naturally its closest words in our embedding is *box, tableware*. After some testing, we noticed that lemmatization will not make boxing to box, reservation to reserv. What's more, error could be generated by the bias of the training data, in the 4th error, we noticed *benchmark* and *index* are surrendered by words like *mercantile, dividend, currency*, which are about finance. Then we noticed that *benchmark* and *index* are common words in finance and stock market. On the one hand,it is because of one word has many senses. One the other hand, financial news is one of the biggest part in daily news, which introduces a bias in the context weighting and therefore in word similarity.

5 Conclusion

We observed that on average, embedding model window size 2 and 3 outperform window 1 by 10% with 200 dimensions. Because of great sparsity and small data size, structure dependency model does not perform well. In the future work, we need to increase the data size for the dependency-based context model. We observed that evaluation accuracy increases with embedding dimension. However, this increase slows down after dimension reaches 300. Also, we have seen that the bigger the size of our data pool, the better the performance, though the increase of accuracy can be small. What's more important, inferring unknown words by the known knowledges is essential in word embedding since there will always be unknown words in the wild. One more thing is that error analysis helps us understand the source of error and inspires us to come up with better solution to improve the model. For the future work, we will try to combine data from other source to overcome limitation of news sentences. And instead of stemming, lemmatization can be tried for the pre-processing. Since the error sources are more than training size, this can explain why merely increasing data did not help too much on the performance on our development.