Department of Computer Science and Technology

Natural Language Processing

Assignment 2

Sahand Sabour

2020280401

1 Gradient Calculation

Assuming that we are given a predicted word vector v_c for the center word c in skip-gram, and word prediction is made with the following Softmax function:

$$y_o = p(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w=1}^{W} exp(u_w^T v_c)} \tag{1}$$

where w denotes the w-th word and $u_w = (w = 1, ..., W)$ are the "output" word vectors for all words in the vocabulary. In addition, assuming that the cross entropy loss is used, we would have:

$$\mathcal{L} = -\sum_{w=1}^{W} t_i log(y_i) = -log(p(o|c)) = -log(exp(u_o^T v_c)) + log(\sum_{w=1}^{W} exp(u_w^T v_c)) \tag{2}$$

Where t is the label and is either 1 or 0 since the input is one-hot encoded. Therefore, there would only be one element of the sum that is non-zero (for the expected word o). Further simplifying the loss function gives:

$$\mathcal{L} = -u_o^T v_c + log(\sum_{w=1}^{W} exp(u_w^T v_c)) \tag{3}$$

Accordingly, we would derive the gradient from this loss as follows:

$$\begin{aligned}\frac{\delta\mathcal{L}}{\delta v_c} &= -\frac{\delta u_o^T v_c}{\delta v_c} + \frac{\delta log(\sum_{w=1}^{W} exp(u_w^T v_c))}{\delta v_c} \\ &= -u_o^T + (\frac{1}{\sum_{w=1}^{W} exp(u_w^T v_c)})(exp(u_w^T v_c))(u_w^T) \\ &= \sum_{w=1}^{W} p(o|c) u_w^T - u_o^T\end{aligned} \tag{4}$$

2 Word2vec Implementation

For this assignment, Pytorch was chosen as the deep learning framework for implementing this model. This implementation consists of the following steps:

2.1 Data Processing

2.1.1 Reducing Corpus Size

We are provided with an unprocessed Wikipedia corpus, which is approximately 10 GB large. Due to the limited computing resources, a smaller version of this file, which included 1/10 of the data and was around 1GB large was created. The newly created corpus was used for the rest of this assignment.

2.1.2 Creating Word Count Dictionary

For each line in the corpus, there are a number of filters that are used to process the line: first, the line is tokenized using simple processing methods from the gensim library; then, the nltk library is utilized to remove any stop-words; lastly, words that are a set of repeated letters (such as 'aaa', 'aaaa'), which were observed to be fairly frequent in the corpus, are removed. Accordingly, words that pass these set of filters are saved in a dictionary and their frequency is recorded.

2.1.3 Creating Vocabulary Dictionary

Initially, based on the word count dictionary, words that are comparatively less frequent are removed. Accordingly, two dictionaries that give an index to each word and map each word to an index (namely word_to_id and id_to_word) respectively are created. These will be referred to as vocabulary.

2.1.4 Sub-Sampling

According to the source code of the original word2vec implementation, the probability of a word w_i staying in the vocabulary is calculated as

$$P(w_i) = (\sqrt{\frac{count(w_i)}{0.001}} + 1) \times \frac{0.001}{count(w_i)} \tag{5}$$

where count is the number of times a word has been seen in the corpus. According to this equation, words that occur frequently have a higher chance of being kept in the vocabulary and less frequent words are more likely to be removed. The same equation is used in the implementation for this assignment to perform sub-sampling on the corpus.

2.1.5 Negative Sampling

In a skip-gram architecture, for each word in the corpus, referred to as a center word, there are a number of words, referred to as context words, that are shown near this word. The task of this architecture is to find context words given center words. However, it is not satisfactory if all the samples presented to the model are positive: meaning they are all actual context words of the given center word. In order to have meaningful training, we need to present negative samples in which the selected words are not context words for the given center word. For this task, a list of words in the vocabulary is initially created, where each word is repeated as many times as its number of occurrences in the word count dictionary. Accordingly, this list is randomly shuffled and each time a constant number of elements is read from this list as to provide negative samples.

2.2 Training

First, we need two embeddings: namely word and context embeddings. These embeddings are similar in shape (vocabulary size $\times$ embedding dimension) and are both initialized with random values. Upon receiving a new batch of inputs, which consists of center words and their corresponding positive and negative samples, the word embedding is used to look up the center word's word vector while the context is used to collect the vector of the other words. Accordingly, the goal is to maximize the similarity between a center word's vector with the vectors of its context words (positive samples) and minimize the similarity between the word and its negative samples. Since the softmax log likelihood loss is computationally expensive, sigmoid log likelihood loss is utilized in this implementation instead.
As required three models with embedding dimensions 100, 200, and 300 were respectively trained. The data was provided to the model in batches of 128, with window size of 5 and 5 negative samples. The model was trained for 5 epochs with the learning rate of 0.001.

2.3 Evaluation

2.3.1 Spearman's Correlation Coefficient

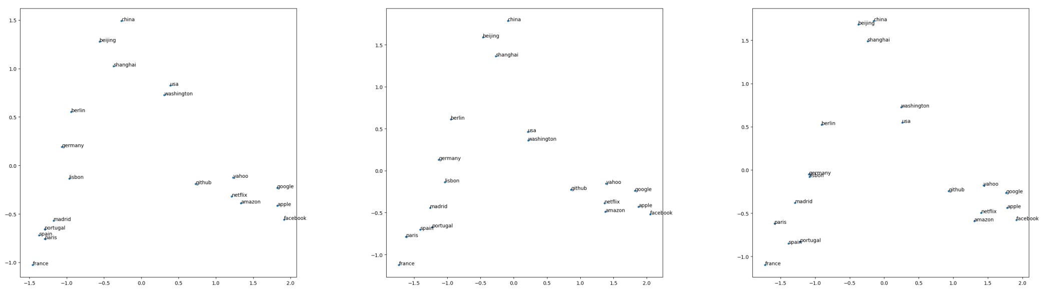
The trained word embeddings were evaluated on the wordsim dataset, which provides a pair of words as well as a mean human-annotated score of their similarity. In this section, the word embeddings of different dimensions were used to find the cosine similarities between the words in each pair and the resulting values were compared to the human annotation results via calculating the spearman's correlation coefficient. This implementation utilized the spearman function from the nltk library and the obtained results are provided respectively in the table below:

Embedding size	Correlation Coefficient (%)
100	52.14
200	56.27
300	59.49

Hence, it can be observed that increasing the embedding dimension is rather beneficial for creating more accurate word vectors. This is evident as more dimensions allow for more features to be captured and hence, increased details in modeling each word. However, increasing the dimensions considerably increases the training time and computation complexity. Hence, in practice, 100 is used as the embedding dimension for smaller datasets while 300 is used for large corpora.

2.3.2 Embedding Visualization

In addition, visualizing word embeddings could also be utilized to demonstrate the power of these embeddings since the similarities between different words are highlighted when illustrated. The below figure illustrates visualization of different word embeddings with a list of arbitrary words.



(a) $dim = 100$ (b) $dim = 200$ (c) $dim = 300$

Figure 1: Visualization of different embeddings

Based on the above figures, it can be observed that, although a slight improvement, in embeddings with higher dimensions the connection between different words of the same category is better realized. For instance, with higher dimensions words relating to big corporations such as apple and amazon move closer together. The same case stands for words such as shanghai, beijing, and china which evidently, are in the same region and therefore, highly related.

2.3.3 Word Analogy

The last task in the evaluation is word analogy. Word analogy can be simplified as the task of correctly finding a word that has the same connection with a given word that another pair of words have. For instance, if the pair of words (France and Paris) are given, the connection between these two words is (country and city). Hence, if we were to give a word that has a similar connection to a country name such as 'England', that word would be 'London': Paris is to France as London is to England. Due to the limited training of the word embeddings, due to limited resources as mentioned, this step of the evaluation was carried fairly briefly by manually testing the output of embeddings in response to different sets of arbitrary pairs. It was observed that all embeddings were able to perform this task to a level: for instance, all the embeddings correctly guessed 'paris' when words ['england', 'london', 'france'] were given. More thorough exploration can be expected from future work.

3 Word2vec Improvement

The approach proposed by [1] is implemented in this assignment to improve the word2vec architecture by adding sense embeddings. The methodology for this addition, as well as evaluation with the base model and discussion of the findings will be provided in this section.

3.1 Methodology

This method builds upon the skip-gram architecture, which was introduced, implemented and evaluated in the previous section. Hence, we are first required to train the word embeddings on the corpus and obtain trained word vectors for each word. Accordingly, we would initialize and update sense embeddings for words in the corpus based on their gloss. In work knowledge corpora such as WordNet, which are a combination of thesauruses and dictionaries, for each word, different meanings (referred to as gloss) as well as a group of its synonyms for this meaning (referred to as synsets) are provided.

3.2 Evaluation

3.3 Discussion

4 References

[1] Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. A unified model for word sense representation and disambiguation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pages 1025–1035, 2014