Natural Language Processing
Implementation of Word2Vec with negative Sampling

Chenene Mohamed,
Aouad Elias,
Qabel Aymen,
Florez de la Colina Inès

1. Introduction

Word embedding has become one of the most popular representations of a document's vocabulary. A given word can be represented by a vector. Our objective is to create embeddings that give us information about how close two words of a given vocabulary are, how similar they are. For example, the words light and bright should be more similar than light and president. To compute those embeddings we can use the Word2Vec method obtained either by using Skip Gram or Common Bag of Words. In this report we will focus on the implementation of Skip Gram from scratch. We will train our algorithm on the Billion word corpus and test it on SimLex-999. In a nutshell, the Skip Gram's algorithm is based on finding two embeddings: one as word and one as context. Finally, we will save the word embeddings to compute the similarity between the words of the test set.

2. Skip Gram Implementation

2.1. Preparing the data

Our training set is composed of a thousand sentences written in a text file. To use this data, we first have to apply a tokenisation process. This process consists of first, separating each sentence, then in each sentence separating each word. We have decided to remove punctuation as well as digits as they don't give us any information about the similarity between two words and we do not want our alogirthm to predict that the most similar word is a comma for example. Keeping the punctuation or digits can only decrease the performance of the algorithm. To do so, we have used the string library that enables us to get rid of the following symbols:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~0123456789'
```

Now that the data has been pre-processed, we can use it as training data for our algorithm.

2.2. Negative Sampling

First of all, we created two dictionaries containing the words of the vocabulary, as keys and the number of occurrences of each word as values. The first dictionary contains all the words of the vocabulary, whereas the second dictionary only contains the words that appear more than minCount number of times. This reduced vocabulary will allow us to proceed with negative sampling.

Negative sampling is used to ensure that we are training our model correctly. We cannot only rely on positive samples and we have to add 'false labels' in order to not reach the trivial solution. Indeed, let's consider a pair (w, c) of a word and a context. We denote by $p(I = 1|w, c; \theta)$ the probability that this pair came from the corpus data. Here θ represents the two embeddings, W and C. Our goal is to maximize the following product:

$$\arg\max_{\theta} \prod_{(w,c)\in I} p(I = 1|w, c; \theta) \tag{1}$$

By taking the log, and defining the probability $p(I = 1|w, c; \theta)$ as the softmax function we get the following optimization problem:

$$\arg\max_{\theta} \sum_{(w,c)\in I} log\frac{1}{1 + e^{-x_w \cdot y_c}} \tag{2}$$

This objective function has a trivial solution by setting θ such that the probability for every pair, of coming from the corpus data is equal to 1. Hence, the need for negative samples.

To select negative samples, we use a unigram distribution. For every word contained in the frequent word vocabulary we compute the following probability:

$$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}} \tag{3}$$

This probability function is used to increase the probability of less frequent words and reduce the probability of the

more frequent ones. To choose negative samples in python while respecting this probability distribution, we can create a list of indices where the number of times an index appears is proportional to its probability. To do so, we use the length of the frequent vocabulary as the coefficient of proportionality. We will choose randomly from this list to get our negativeRate number of negative samples.

2.3. Pairs training

2.4. Training the model

Now that we have implemented the negative sampling we can focus on training the model. To do so, we go through all the sentences of our training set and for a given word, in a given sentence, we compute its positive pairs and apply the stochastic gradient descent to this word, its context words and the negative samples. We denote by σ the sigmoid function. We obtain the following loss:

$$L = \sum_{(w,c)\in I} \log \sigma(x_w.y_c) + \sum_{(w,z)\in I'} \log \sigma(-x_w.z_c) \quad (4)$$

To apply the SGD, we have to compute the gradients:

$$\begin{aligned} \frac{\partial L}{\partial x_w} &= \sum_{(w,c)\in I} \sigma(-x_w.y_c)y_c - \sum_{(w,z)\in I'} \sigma(x_w.z_c)z_c \\ \frac{\partial L}{\partial y_c} &= \sum_{(w,c)\in I} \sigma(-x_w.y_c)x_w \\ \frac{\partial L}{\partial y_c} &= \sum_{(w,z)\in I'} -\sigma(x_w.z_c)x_w \end{aligned} \quad (5)$$

The parameters of the SGD are the number of epochs and the learning rate. We initialized the embeddings W and C by taking random values between -0.5 and 0.5. We will discuss the choice of hyperparameters in the next section.

3. Additionnal Experiments

3.1. Our results

First of all, let us present our results. We have chosen as a baseline for our SkipGram the following parameters: no use of subsampling, no use of vector normalization, a window size between 1 and 3 (random choice), a negative rate of 5, a learning rate of 10^{-1} and finally a number of epochs of 50.

We have chosen a list of very frequent words (president, company, country, oil) and some less frequent words (medals, students, climate, chairman). Here are the results of our baseline model:

- president: parisian, limit, welcome, john, kennedy
- company: depository, listed, spread, caliber, howard
- country: trailing, motivation, stifles, horror, parties
- oil: economics, coastal, ask, delivery, pressured
- medals: winter, olympics, silver, awards, arrangements
- students: baffling, rate, hundred, audience, viability
- climate: acknowledgement, pact, effects, forge, global
- chairman: faraj, bernanke, closure, nonexecutive, nation

3.2. Impact of hyperparameters

3.2.1 Learning rate

In this section, we will discuss the effect of the learning rate on the learning process. Below we can see the evolution of the objective function with respect to the number of epochs for three different values of the learning rate.

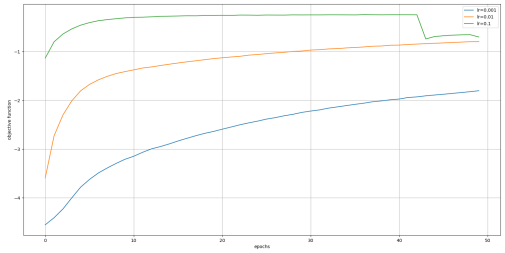

Figure 1. Evolution of the objective function with respect to the number of epochs for three different values of the learning rate

As we can see on figure 1, the best learning rate applicable here is 0.1, as it outputs the best values in terms of objective function. Recall that the objective function is the log-probability that we are trying to maximize. With no doubt, the training process with learning rate of 0.1 outperforms the other.

3.2.2 Negative rate

In this section, we will see the influence of the negative rate on the learning process. Recall that the negative rate is the number of words that are sampled to form negative pairs with the target in the training process. The objective function is the average log-similarity function that we are trying to maximize, defined by the following equation :

$$L = \sum_{(w,c)\in I} \log \sigma(x_w.y_c) + \sum_{(w,c')\in \mathcal{N}} \log \sigma(-x_w.z_{c'})$$

The results are shown in the figure below :

As we can see on figure 2, the result shows that the lowest value for negative sampling outperforms the others. However, recall that the more we sample negative pairs, the more we add terms to our precedent equation, the harder it gets to optimize.

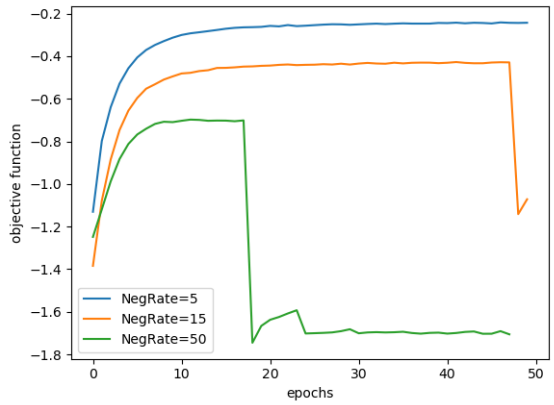

Figure 2. Evolution of the objective function with respect to the number of epochs for three different values of the negative rate

As we can also see, they all reach a plateau at the same number of iterations. The learning is performed at the same speed. After 15 epochs, the training becomes useless. The same sudden decrease as before is observe, which is probably due to an exploding gradient.

3.2.3 Window Size

As we can see on the below figure 3, the window size has very little impact on the objective function. The same convergence is reached for all window sizes.

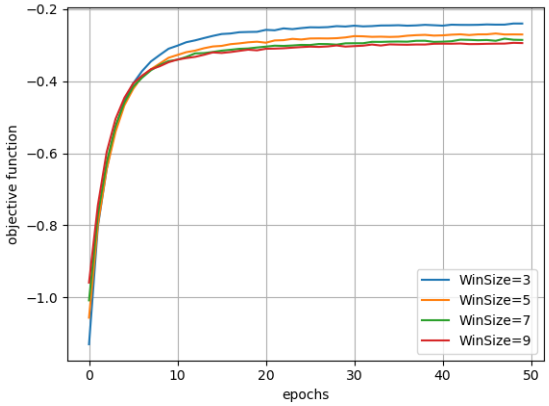

Figure 3. Evolution of the objective function with respect to the number of epochs for three different values of the window size

However, in terms of computation time, it increases linearly as we take bigger values for the window size. For simpler and more accurate result, we will stick with a window size of 5, because 3 is just too small for capturing real context.

Here again, this parameter has an effect on the objective function : the higher we take our window size, the more positive pairs we add to our computation of the objective function, and the more terms we have to optimize.

3.2.4 Number of epochs

Here, we will choose the best number of epochs for the learning process. We chose the best parameters for learning rate, window size, and negative rate, according to what we have seen before.

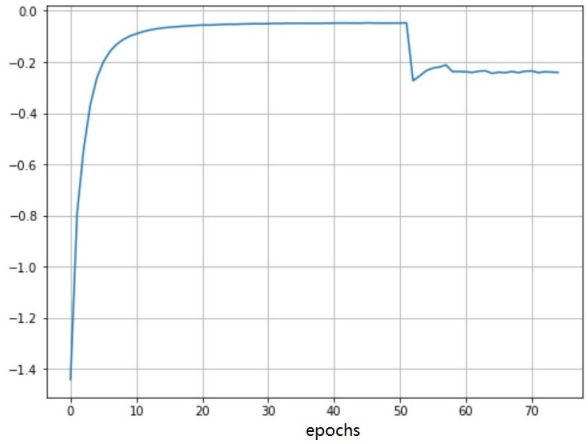

Figure 4. Evolution of the objective function with respect to the number of epochs

As we can see on figure 4, the best number of epochs seems to be around 20 and 30, since convergence is achieved at this rate. However, an interesting event happens after 50 epochs, as it seems that the objective function just drops suddenly to a new equilibrium value. This is probably due to exploding gradients. In the end, in order to avoid it, one should probably fix to 30 epochs.

3.3. Subsampling

In our additionnal experiments we also have decided to implement subsampling. Usually, in english, words like 'the', 'and', 'a'... don't give much information about the context of a given word. Less frequent words are the ones that actually carry the context associated to the studied word. Hence, our decision to use subsampling. This method is very similar to excluding the stop words of our vocabulary. Nevertheless, instead of excluding the usual english stop words found in librairies like NLTK, we find ourselves the stop words of our training vocabulary. To do so, we associate a sampling probability to each word:

$$p(x_w) = 1 - \sqrt{\frac{t}{f(x_w)}} \tag{6}$$

where t is a certain threshold and $f(x_w)$ is the frequency of the word x_w. Nevertheless, word2vec's code uses a different probability formula for the implementation of subsampling [1]:

$$p(x_w) = \frac{f(x_w) - t}{f(x_w)} - \sqrt{\frac{t}{f(x_w)}} \tag{7}$$

Subsampling consists of removing randomly the most frequent words. According to the litterature, t is set to 10^{-5}. After plotting the probability distribution, we have decided to remove all words with a frequency higher than 0.9 because they represented less than 100 words as we can see on figure 5.

We trained the SkipGram model after deleting the stopwords given by the subsampling method. The rest of the hyperparameters were kept as in the baseline model. To assess the performance of the model, we displayed the Top-5 similar words of the 8 reference words mentionned earlier:

- president: foreclosure, appropriate, options, responsibility, vice
- company: joined, partnership, assistance, stock, depository
- country: harnessing, suggesting, pushed, surged, appeal
- oil: gas, monthly, heavily, delivery, investing
- medals: silver, wore, olympics, won, athens
- students: audience, invited, uniforms, gaza, palestinian
- climate: pact, acknowledgement, effects, lindsey, graham
- chairman: irish, gleeson, ben, gs, stonewash

Globally, the model performs pretty well. The words company, oil and medals are well represented and have good similarities words. The advantage of subsampling is that it indirectly increases the window size. Moreover, there are more meaningful words in the neighbourhood of the center word. This explains why the model performs better.

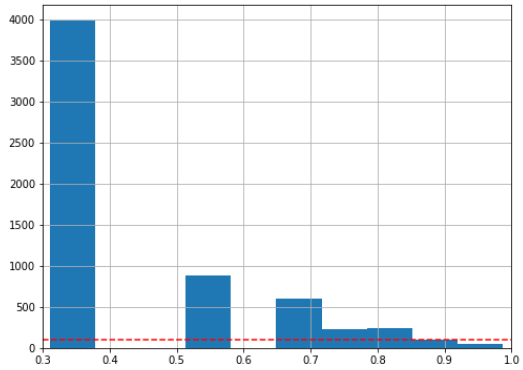

Figure 5. Distribution of words frequence for SubSampling

3.4. Vector Normalization

We also decided to implement vector normalization. We have chosen to normalise the columns of the word matrix W as used by Pennington et al. According to [1], the best normalization method is to use the standard L_2 normalization of W's rows which is equivalent to using the cosine similarity. That was used to predict similarity between two words, but we will add the column normalization as well. Here are the results for vector normalization:

- president: neighbours, parisian, john, abbas, welcome
- company: depository, listed, caliber, howard, botswana
- country: trailing, mirror, motivation, stifles, species
- oil: economics, coastal, ask, delivery, predicted
- medals: olympics, winter, silver, awards, six
- students: baffling, hundred, viability, rate, several
- climate: acknowledgement, forge, effects, pact, global
- chairman: faraj, bernanke, closure, nonexecutive, stepped

By comparing these results with the basline model's result, normalizing the columns doesn't seem to improve the performance of the model significantly. It seems that both models are very similar. Thus we will not keep this method in our final model.

4. Conclusion

According to all the experiments we have conveyed, we concluded that the best parameters for our skipGram are the following:

- Apply subsampling
- negativeRate = 5
- winSize = 5
- lr = 0.1, epochs = 30

References

[1] Omer Levy, Yoav Goldberg,Ido Dagan, 2015. *Improving Distributional Similarity with Lessons Learned from Word Embeddings* 3, 4