Natural Language Processing
Assignment 1: *Word2Vec*

Marien Chenaud, Maxence Gélard

1. Introduction and goals

The goal of the project is to implement a language model using *Word2Vec*, which corresponds to a **Skim-Gram model with Negative Sampling**. In the report, we will be first deriving the model, especially the training step and link it with our implementation. Then, some experiment, including hyperparameters tuning (embedding dimension, training parameters, ...) will be discussed. Finally, we will address one particular issue that word embeddings can encounter, which is **gender bias**, and we'll see a way to tackle the problem.

2. Architecture of *Word2Vec* model

In this section, we will be presenting the theoretical analysis behind the *Word2Vec* model, with especially the manual derivation of the loss' gradients. This part, namely for the presentation of the model, mostly relies on the NLP Lecture by M. Galle and on [3].

2.1. Probabilistic model

Let's consider a corpus of N words. We aim at finding words representations, *a.k.a* embeddings, in a vector space of dimension k (k will refer to the *embedding dimension*), such as for all pair of (word, context word) represented as (w, c), our goal would be to maximize the probability $p(w|c)$. Given a word, one defines a context word as another word which is located at $+/-$ a certain *window size* of the current word. The model thus defines two embedding matrices, $W \in \mathbb{R}^{N\times k}$ and $C \in \mathbb{R}^{N\times k}$ which are respectively the words embedding and the context words embedding.

To achieve the task of learning W and C, we will use the following modelling. Let's denote I the random variable "(x, y) is a valid pair" (valid meaning y is indeed co-occurring with x). By denoting x_W the embedding of x as a word and y_C the embedding of y as a context word (line vectors), we will define, noting $\theta = [W, C]$ all the model's parameters:

$$p(I = 1|x, y, \theta) = \frac{1}{1 + e^{-x_W y_C^T}} \equiv \sigma(x_W y_C^T)\ (sigmoid)$$

We thus aiming at maximizing $\prod_{(x,y)\in\mathcal{W}} \sigma(x_W y_C^T)$, with $\mathcal{W}$ the corpus of (word, context word) pairs. This problem suffers from the issue of having a trivial solution by taking constant (*e.g* all equal to vector of 1's) embeddings. Thus we add the notion of negative sampling, i.e for each word / context word pair (x, y), we will pick a fixed number of words z that are context words which don't co-occur with x. Let's denote this set $\mathcal{NS}(x)$. Thus, our goal is now:

$$Argmax_\theta \prod_{(x,y)\in\mathcal{W}} \left(p(I = 1|x, y, \theta) \prod_{z\in\mathcal{NS}(x)} p(I = 0|x, z, \theta) \right).$$

2.2. Negative sampling details

Before diving into the derivation of the optimization of the previously stated objective, we have been using the sampling method described in [3], which corresponds in assigning a probability distribution to all words that is slightly different than just a word frequency. So for a given word x, we will need to build its negative samples set $NS(x)$ that will be formed by sampling any words (except x) to which we would have assigned the following weights:

$$p(z) = \frac{frequency(z)^{\frac{3}{4}}}{\sum_{w\in vocabulary\setminus\{x\}} frequency(w)^{\frac{3}{4}}}$$

where the frequency of a word represents the number of times it occurs in the whole corpus divided by the total number of words in the vocabulary. This transformation can be seen as a Box-Cox transformation, the goal being to rescale the probabilities in order not to rule out some less frequent words.

2.3. Derivation of training step

In order to learn word representation, we consider the following loss function (negative log-likelihood), which is obtained by taking the negative log-loss of the objective we defined previously:

$$L(\theta) = -\sum_{(x,y)\in\mathcal{W}} log(\sigma(x_W y_C^T)) - \sum_{z\in\mathcal{NS}(x)} log(\sigma(-x_W z_C^T))$$

with x_W corresponding to the word embedding (i.e using matrix W) of x and y_C and z_C respectively corresponding to context embedding of words y and z. $\mathcal{W}$ corresponds to word pairs and $\mathcal{NS}(x)$ negative samples for word x, and $\theta = (W, C)$ the embedding matrices. σ denotes the sigmoid function.

Thus the goal is to find $\hat{\theta}$, such that:

$$\hat{\theta} = Argmin_\theta L(\theta)$$

This optimization problem has been solved using Stochastic Gradient Descent (SGD). We thus require the gradient of $L(\theta)$ with respect to x_W ad y_C for the SGD updates. Let's call $l(\theta) = -log(\sigma(x_W y_C^T)) - \sum_{z\in NS(x)} log(\sigma(-x_W z_C^T))$. Using the following result:

$$\frac{\partial log(\sigma(xy^T))}{\partial x} = y\sigma(-xy^T)$$

One gets:

$$\frac{\partial l(\theta)}{\partial x_W} = -y_C\sigma(-x_W y_C^T) + \sum_{z \in NS(x)} z_C \sigma(x_W z_C^T)$$
$$\frac{\partial l(\theta)}{\partial y_C} = -x_W\sigma(-x_W y_C^T)$$
$$\frac{\partial l(\theta)}{\partial z_C} = x_W\sigma(x_W z_C^T)$$

(the third equation being computed for all $z \in NS(x)$).

Thus, by denoting α the SGD's learning rate, one get the following updates rules for x_W and y_C (which are respectively rows of W and C):

$$x_W = x_W - \alpha * \frac{\partial l(\theta)}{\partial x_W}$$
$$y_C = y_C - \alpha * \frac{\partial l(\theta)}{\partial y_C}$$
$$z_C = z_C - \alpha * \frac{\partial l(\theta)}{\partial z_C}, \forall z \in NS(x)$$

3. Methodology

This sections will present the different steps that are used in order to train our *Word2Vec* model.

3.1. Text preprocessing

We were given raw texts divided in different lines of texts. After separation of each line, we tokenized the text using *Spacy* module (*en_core_web_sm* model), to get separated words. The tokenisation process thus allowed to retrieve individual words to which we applied 3 rules to refine our preprocessing

- Removing **stop-words** (*e.g* "about", "every", etc).
- Removing **punctuation** (".", "?", etc.) and digits
- **Lemmatize** tokens (*e.g* "women" is transformed to "woman").

Then we build a vocabulary by keeping only words that appear a sufficient number of times, this being determined by a parameter called *min_count* (set to 2 in our final model).

3.2. Model definition and initialisation

Our skip-gram model with negative sampling, defined in the $skipGram.py$ main file, needs to be initialized along with the different hyperparameters (number of epochs, learning rate, negative sample rate, window size, minimum count, embedding dimension), but we also need to initialize several arrays:

- The **word embedding** and the **context embedding** matrices: these are arrays of shape (length of vocabulary, embedding dimension) and are initialized randomly (standard normal or uniform initialization).
- **Negative samples probabilities**: in order to speed up the training step, we pre-compute for each word in the vocabulary the probabilities that will be used for negative sampling.

4. Training step

In order to train our model, we will follow the formulation in terms of probabilistic model and use the gradients derivation done in the second part of this report to optimize our loss function using Stochastic Gradient Descent. The training loop will look at each sentence and for each of these sentences we will be using a word, along with its context words ($+/-$ a given *window_size*) and its negatives samples (we use negative sampling for each (word, context word) pair), to update the embedding matrices. This leads to the following algorithm (for a given sentence, and we will loop over all sentences):

Algorithm 1: *Word2Vec* training step, 1 sentence

```
Data: sentence: list of words
Initialize: $W, C$
for x in sentence do
    for y in window_around_word do
        $NS(x) \leftarrow neg\_sampling(x)$;
        $x_W \leftarrow x_W - \alpha\frac{\partial l(\theta)}{\partial x_W}$;
        $y_C \leftarrow y_C - \alpha\frac{\partial l(\theta)}{\partial y_C}$;
        for z in NS(x) do
            $z_C \leftarrow z_C - \alpha\frac{\partial l(\theta)}{\partial z_C}$
        end
    end
end
```

This training loop is implementing in the *SkipGram* class (namely *train_word* function) which computes the manually derived gradients.

5. Experiments and Parameters tuning

This section will deal with the study of parameters influence and tuning. The tuning of hyperparameters has been done in two ways, depending on the chosen parameters: influence on the loss or influence of a score computed on the test set. For the latter, we have been using the provided test set and we rescaled all the reference similarities to compare them with the one returned by our *SkipGram* model (that are between 0 and 1). The comparison score is the mean of the squared difference between prediction and reference similarities. Another more classical way to compare two list of values, x and y is to use the Pearson Correlation coefficient which is defined by:

$$r = \frac{(\sum_{i=1}^{n} x_i - \bar{x})(\sum_{i=1}^{n} y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ corresponds to the mean of x. We thus will also be using the correlation between the true and predicted similarties (by maximizing it) to tune our parameters.

In order to have fair experiments, we fixed default parameters and only moved one parameters for each experiment: *learning rate* 0.001, *window size* 10, *embedding dimensions* 100, and 1000 training lines. Below is given the different experiments to understand the effect of different parameters and tune them.

- **Learning rate**

The learning rate controls the size of the steps that are taken during training (gradient descent). In Figure 1 is given the result of the evolution of the loss through the different epochs, for different learning rates.

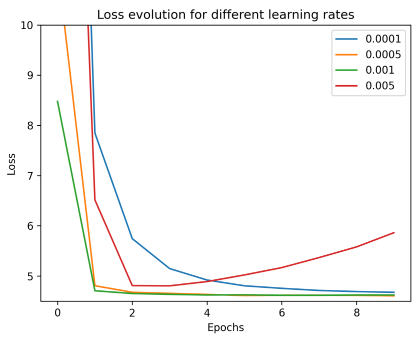


Figure 1. Influence of learning rate

One can observe that a too high learning rate (≥ 0.005) leads to a divergence of the loss while a too small learning rate converges too slowly (typically for 0.0001). The final choice that has been made is thus 0.001. Moreover, already with this experiment, we could observe that a total of **10 epochs** were enough to go until the convergence of the loss.

- **Windows size and embedding dimension**

The window size controls the number of context word that will be taken into account to form pairs (word, context word). A too small window size would lead to only consider very short-term dependencies between a word and its context, thus leading to poorer result. In our experiment, we tested mainly 3 values of window sizes (2, 10, 20). Another important point that need to be taken into account is the increasing time it takes to train a model as we increase the window size (factor of 3 in computational time between 2 and 20). Therefore, we chose to keep a window size of 10 as a trade-off between performance and computational time.

The embedding dimension corresponds to the "capacity of expression" of the model. We tested 4 values of embedding dimensions: 50, 100, 150 and 200. We didn't notice significant variation in the score we defined (difference less than 1%), so once again we chose the value that gives a satisfying trade-off between score and time performance and thus kept the value of 100.

The corresponding Pearson correlation (scaled by a factor of 10) are given below for different set of parameters:

Pearson correlation (embedding dimension)			
Embedding dim.	50	100	200
Correlation	0.6	**0.8**	-0.03

Pearson correlation (window size)			
Window size	2	10	20
Correlation	-0.4	**0.8**	-0.6

One can notice that we get quite negative values, which is namely due to the small number of lines used to tune the hyperparameters, that had the advantage of accelerating the computations. Nonetheless, this gives an intuition on the behavior of the different parameters.

- **Negative sample size**

The negative sample size is another hyperparameter of the model. It determines the number of words that, for a given tuple (x, y), are not in the context matrix of x.
Here are the corresponding Pearson correlations found by testing different values of the parameter (once again multiplied by 10):

Pearson correlation (negative sample size)				
Neg. sample	5	10	15	17
Correlation	-0.12	0.42	**0.6**	0.22

Hence, the selected value for this hyperparameter is 15. We did not investigate much bigger values, that could have been better, because of the increasing training time.

- **Embedding initialization**

The last experiment we carried out on parameters tuning was to look at the influence on embedding initialization, i.e how the matrices W and C are initialized. Two initialization have been considered: random standard normal and uniform between 0 and 1. We observed that when using the uniform initialization, the loss after 10 epochs is four times less than the loss after the same number of epochs when initializing with a standard normal distribution, hence our choice for the former.

- **Combination of embeddings**

Until now, when talking about word embedding, we always meant using the vector corresponding to a given word from the matrix W, i.e for a word x, we were using x_W. We also tried to use $\frac{x_W + x_C}{2}$ as an embedding, and we observed an decrease of almost 40% in the score defined previously, so we conclude that we will only be using only the word embedding matrix.

6. Gender bias issue

6.1. Presentation of the problem

As additional material to the implementation of the skip-gram models, we were interested in analyzing any gender bias that could appear in the learned representation of words. The analysis and the experiments carried on namely relies on [1]. The code mapping to what follows can be found in the *debiasing_embedding.py* file.

The main idea behind gender bias is that the representation of words that is learned by the *Word2Vec* model can reflect some biases that can be found in the training set. For example, it is likely that a name that sounds like a feminine name will have a larger similarity (*e.g* using cosine similarity on embeddings) with words like ”girl”, ”woman” or ”she” as it is more probable that these words co-occur frequently with this name, than with other words like ”boy”, ”he”. Even though for name this doesn't sound like an issue, if our training set carries strong gender biases that reflect gender stereotype (for example on jobs with scientific jobs only being referred to be ”male-jobs” and more literary jobs to be ”female-jobs”), this would generate biases in the similarity we get. To identify this bias and eventually try to correct it, we first need to define a vector embedding g that will encode the notion of gender. One simple way to do so would be to define $g = x_{woman} - x_{man}$. Thus for a word x, one can identify whether it is more ”female-oriented” or ”male-oriented” by looking at the similarity with its embedding and the vector g:

$$\begin{aligned} bias\_word(x) &= sign(cosine(x_w, g)) \\ &= \begin{cases} 1 & \text{corresponding to ”female-oriented”} \\ -1 & \text{corresponding to ”male-oriented”} \end{cases} \end{aligned}$$

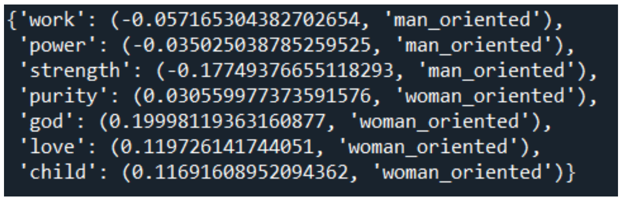
6.2. Debiasing embedding

A possible solution to remove this gender bias from the embedding would be to remove from the vectors the component corresponding to the encoding of gender. To do so, for a given word x, we will compute its orthogonal projection on g and the resulting gender unbiased embedding will be the subtraction of x and that orthogonal projection, which leads to the following formula:

$$x\_projection = \frac{x_W g^T}{||g||^2} g$$
$$x\_unbiased = x_W - x\_projection$$

6.3. Experimental results on gender bias

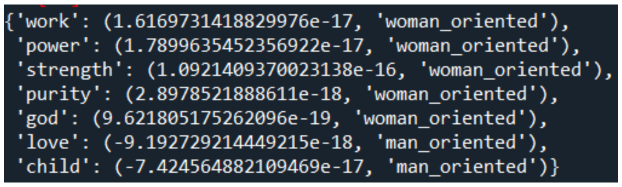
In order to test our approach, we had to find a text which would likely include gender biases. We decided to choose *Purity and truth : what a young woman ought to know*, written by Emma F. Angell Drake in 1901. This book aims at giving some codes of conduct for young women. We chose to focus on several words that could carry gender bias, namely : work, power, strength, purity, god, love and children. Here are the resulting biases for these words, as presented in section 6.1: As one could



```
{'work': (-0.057165304382702654, 'man_oriented'),
 'power': (-0.035025038785259525, 'man_oriented'),
 'strength': (-0.17749376655118293, 'man_oriented'),
 'purity': (0.030559977373591576, 'woman_oriented'),
 'god': (0.19998119363160877, 'woman_oriented'),
 'love': (0.119726141744051, 'woman_oriented'),
 'child': (0.11691608952094362, 'woman_oriented')}
```

Figure 2. Biased embeddings as generated by SkipGram.

expect, there is clearly a bias in the text. While work, strength or power are associated to men, purity, child and love are associated to women. The word god is interesting: it is associated to women, probably not because god is presented as a female figure, but because words related to women are close to the word god in the text, which could be interpreted the fact that women might be closer to religion.
We applied the debiasing procedure on the model, as described in the previous subsection: As expected, the biases have dis-



```
{'work': (1.6169731418829976e-17, 'woman_oriented'),
 'power': (1.7899635452356922e-17, 'woman_oriented'),
 'strength': (1.0921409370023138e-16, 'woman_oriented'),
 'purity': (2.8978521888611e-18, 'woman_oriented'),
 'god': (9.621805175262096e-19, 'woman_oriented'),
 'love': (-9.192729214449215e-18, 'man_oriented'),
 'child': (-7.424564882109469e-17, 'man_oriented')}
```

Figure 3. Debiased embeddings.

appeared, which proves that in some contexts, for well-known biases, these kind of procedures could be usefull. The limitation seems to be that we must have an idea of what kind of biases we could find in order to apply this procedure, and it is very sensitive to the words chosen (for instance, the biases associated to ”boy” or ”girl” have not been corrected here).

7. Conclusion

To sum up, we were able to implement the *Word2Vec* model which corresponds to a probabilistic model called Skip-Gram with negative sampling. In this report, we derived the corresponding training step which involves a gradient descent. The data processing as well as the training pipeline have been detailed, while some hyperparameters influence and tuning has been performed. The average time to train our model on 1000 lines for 10 epochs is around 15 minutes.

As an example of result, with the set of parameters defined in the hyperparameters tuning part, we show the top 4 best words that are the most similar to the word ”american”:

- ”mass”, score: 0.68
- ”international”, score: 0.67
- ”foreign”, score: 0.65
- ”cast”, score: 0.63

Although we didn't achieve very discriminative similarities (near 1 for example), we still can see that similar words are close to the lexical field of the reference word.

Further work on the topic of word embedding could consist in comparing our language model with more recent literature that involves Deep Learning such as transformer based models like Bert. [2].

References

[1] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, 29:4349–4357, 2016.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.