**Implementation of Skip-Gram with Negative Sampling**

***Project Report***

|  |  |  |
| --- | --- | --- |
| Gabriel Drai  MSC AI at CentraleSupelec  gabriel.drai@student-cs.fr | Charles Boy de la Tour  MSC AI at CentraleSupelec  charles.boy-de-la-tour@student-cs.fr | Zakariae El Asri  MSC AI at CentraleSupelec  zakariae.elasri@student-cs.fr |

Our code is available at <https://github.com/CharlesBoydelaTour/NLP---Skip-gram-implementation>

# **Abstract**

One of the most important questions in Natural Language Processing is how to provide a great representation of words and their meaning. Those types of representations are called word embedding, they have an array shape and can be used as features for NLP machine learning tasks. One of the most famous and used word embedding algorithms is called word2vec and has two modalities: Bag-Of-Words and Skip-Gram. The first one acquires a word representation by predicting a word with its context, vice versa for the second one.

We are focusing on the last one, and how we can improve its performances with different methods such as negative sampling or filtering words depending on their frequency in the training corpus.

# **Introduction**

Traditionally, Count-based language modeling was the major technique for NLP tasks. These models try to capture meaningful relationships among words by computing a co-occurence matrix, this technique was not performant in capturing complex relationships.

To compute high-quality representations of word vectors, *Mikolov et al.* [4] have introduced a prediction-based approach, called Word2Vec, where the proposed model uses neural networks to learn intelligent representation of words in a vector space. This model tries to maximize a word's classification based on another word in the same sentence. There are two variants of Word2Vec model, Continuous bag of words and Skip-gram. We focus here in the last one, where each word is used as an input to a Log-linear classifier with a linear projection hidden layer to predict words within a specific range around the input word (see Figure 1).

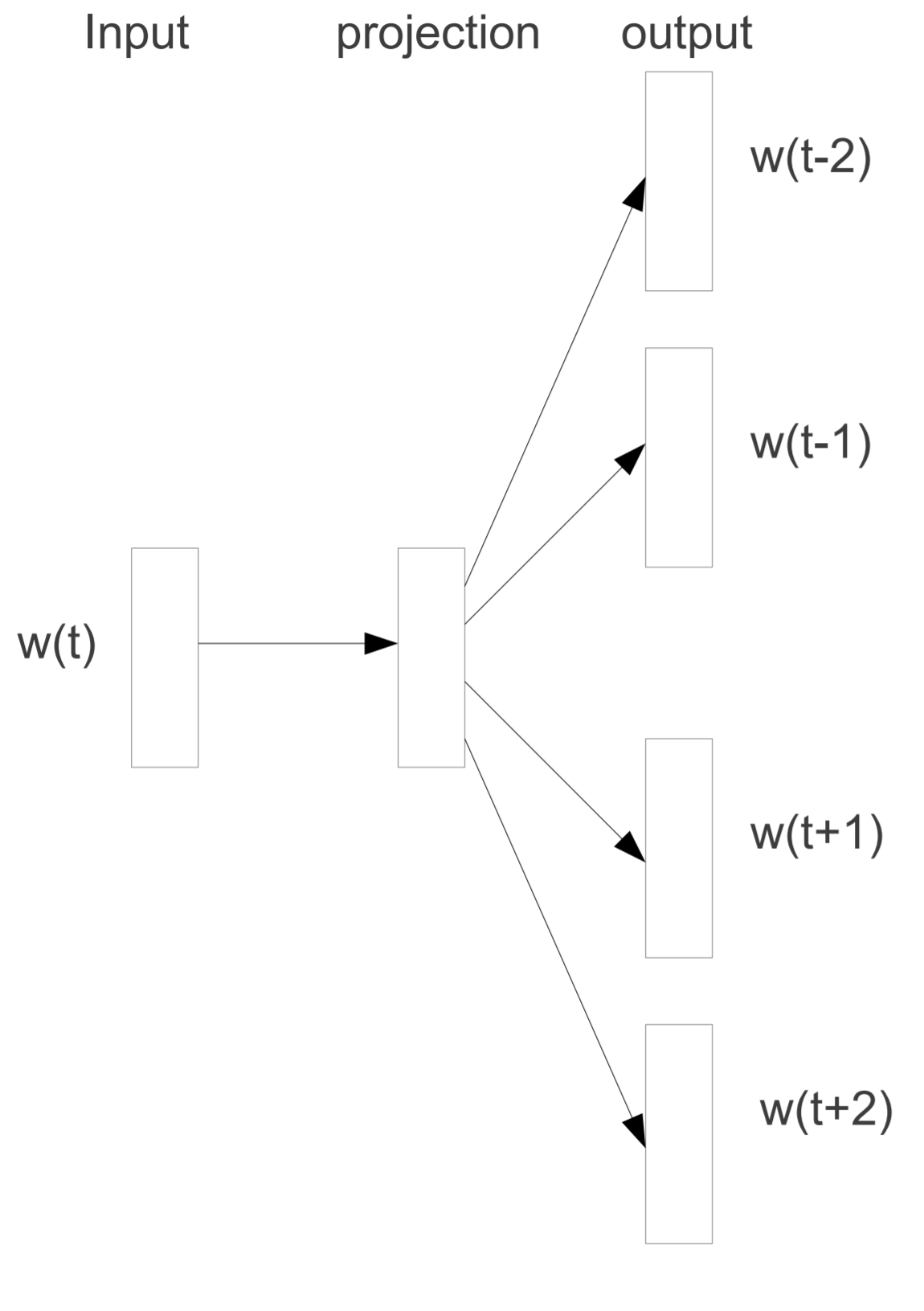


Figure1 : Original Skip-gram model architecture

This model had great enhancement compared to previous models, thanks to avoiding dense matrix multiplications, but its training step is still considered slow. This issue was addressed, in a later work, by *Mikolov et al.* [3] by introducing three major enhancements:

* Negative Sampling: a concept where each training sample updates a small percentage of weights rather than all.
* Subsampling frequent words to decrease the number of training examples, since frequent words
* Treating common word pairs or phrases as single “words” in their model, to decrease the number of training examples.

This reduces the computation time of the training process and improves the quality of results as well.

# **Model implementation:**

In this exercise, we built a skip-gram model, we trained it on a fake task to learn the representations of the input words (word embeddings), then we used the learned representations in the real task to compute the similarity between two words.

We used the One Billion Word Benchmark dataset [1] for training our model, then we evaluated it on the SimLex-999 [2] as a testing dataset for word similarity tasks.

## **Training :**

To train our Skip-Gram implementation we applied the negative sampling method. The latter allows us to simplify the problem into a classification task. The cost function we used is the same in the original Word2Vec paper [4]:

Une image contenant texte, périphérique, jauge

Description générée automatiquement

Where *wI* is the input word, *w0* is the word present in the context of *wI*,

Wneg represents the set of negative samples and *wJ* is one of the negative words,

h is the hidden layer

v’w0 is the word vector in the output weight matrix corresponding to w0. The same for wj.

After derivation for each layer of our neural network of this cost function, SGD is applied to train our model.

* Hyper parameters:

As the original skip-gram model, the hyper parameters of our model are:

the embedding dimension: the size of the word embedding, it is also the size of the hidden layer

the negative samples rate: it represents the number of negative samples in the negative sampling approach.

the window size: it consists of the number of context words around the input word.

In addition to classic hyper parameters for neural networks such as Learning rate.

* Preprocessing step:

To handle this exercise, the first step we start with is tokenizing our training set into sentences, then we create our vocabulary and corpus that we used for sampling the negative words. We create also a dictionary mapping each word in the vocabulary to an index (W2Id)

* Initialization step:

After the preprocessing step, we go further for the training step. We start first by initializing our hyper-parameters listed above. We notice that we did the training for different initialization values (listed in the evaluation part).

We also initialize the weights of our neural networks, w1 represents the weights of the hidden layer and w2 represents those of the output layer. So, the weights W1 and W2 are initialized from a uniform distribution in [-0.001,0.001].

For the stochastic gradient descent, we initialize the value of learning rate at 0.01

* Training step:

Once the preprocessing and initialization are done, we can start training our model.

For a certain number of epochs, we loop through the sentences in our training text then we loop through words in these sentences.

For each word, we define its context in function of the window Size. Then for each pair (Word, Context) we use the function Sample(w,c) to generate k negative words for the given pair (w,c) wherek is the NegativeRate.

Now, we perform backpropagation for the positive word and negative ones:

We compute the dot-product between the input word weights and the context word weights (also between the input word weights and negative words weights) then we apply the sigmoid function to turn these results into probabilities.

We calculate the gradients then we update L1 et L2 by multiplying by the learning rate.

Finally, we calculate the loss function.

## **Evaluation**

To evaluate the performance of our model, we implemented a similarity function that determine how two words are close to each other in terms of their context or meaning. We used the cosine similarity defined by the expression below for two words A & B:

Une image contenant texte

Description générée automatiquement

To ensure that our similarity is included in [0,1], we apply a Min-Max normalization on A & B before the calculation.

# **Experiments:**

In order to find the best parameters of our model, we train it on the first 1000 lines of the corpus provided for the training dataset, then we evaluate it with a Mean Square Error between our cosine similarity scores and those provided in the simlex file. This score cannot be interpreted because we suppose the model which provided the results of the simlex file was certainly trained with the 300000 lines of the corpus on several epochs, in opposition to us. However, calculating and comparing the MSE at each epoch allows us to select the best parameters while not risking overfitting.

In this part, we trained several models with different values of hyperparameters over 40 epochs for each and we plot the results.

At first time, we initialized the model with different numbers of neurons (50, 100, 200, 500) for the embedding layer while keeping the negative rate and windows size at their default values (5, 5). The results in the figure 2 shows a better convergence for Embedding dimension = 50 and 100.

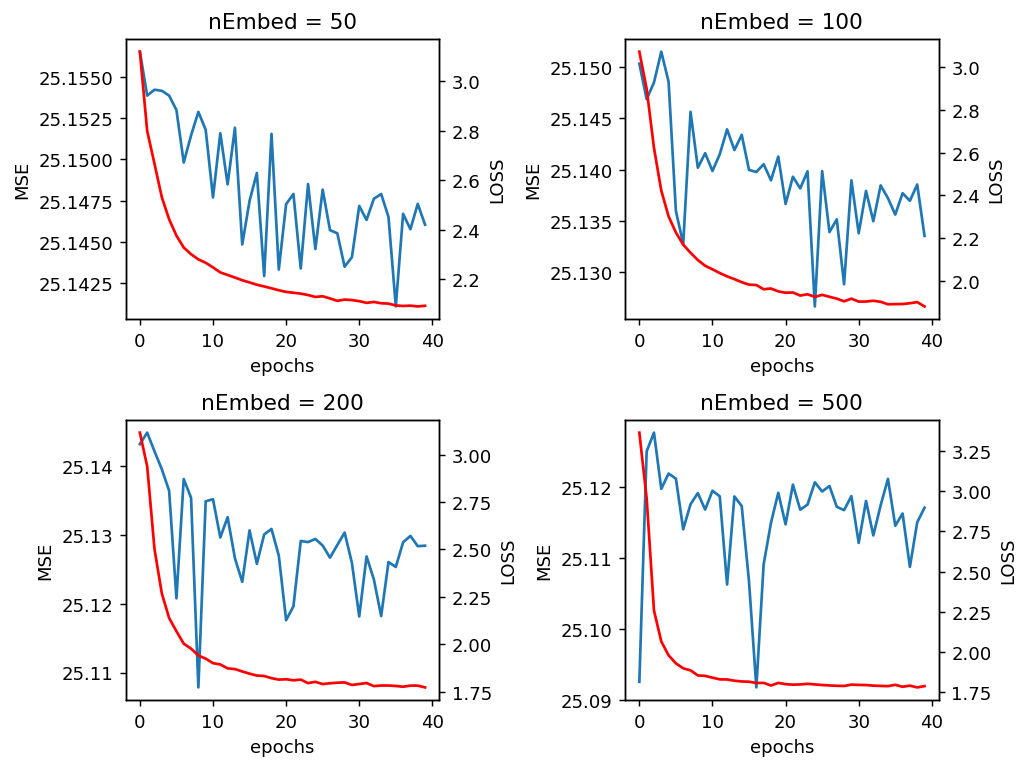


Figure 2: Training the SkipGram model with different sizes of embedding layers.

For the second experience, we kept the value 100 for embedding dimension, we fixed the window size at its default value 5, and we tried different values for the negative rate (5, 10, 20, 50). We can see in the figure3 that the MSE decreases when increasing the Negative rate, but the convergence is good only for values lower than 20.

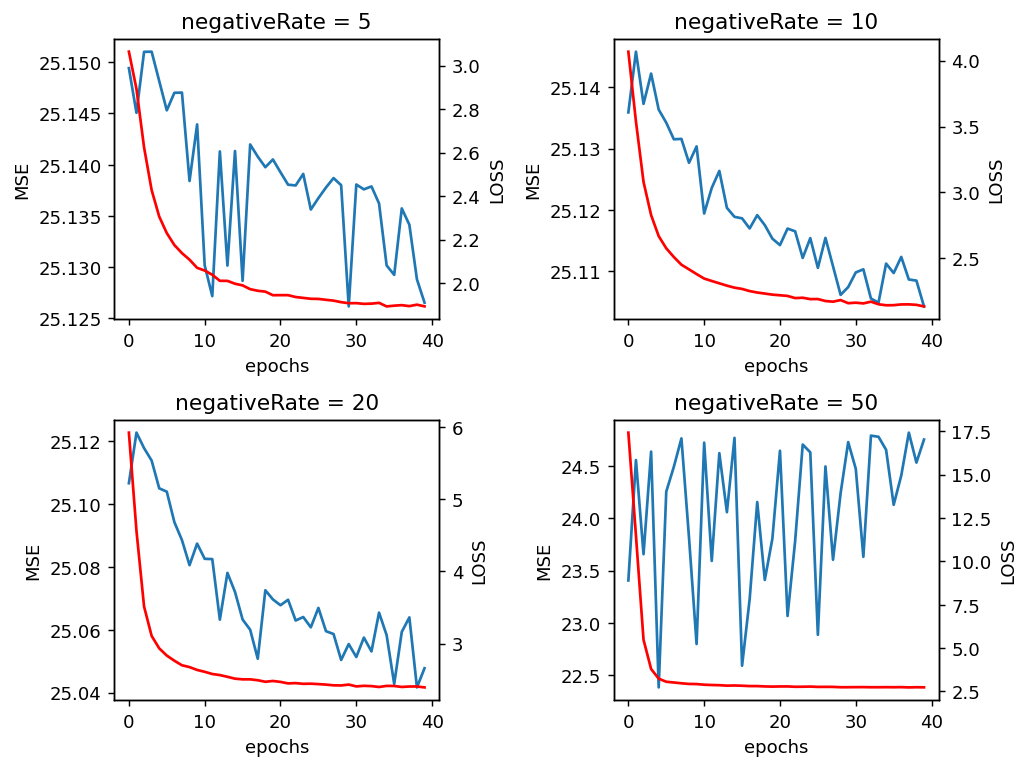


Figure 3: Training the SkipGram model with different numbers of negative samples per context words.

For the third experience, we kept values 100 for embedding dimension, 20 for negative rate, and we tried different values for the window size (5, 10, 20, 50).

We can see in figure 3 similar results to those of the second experience

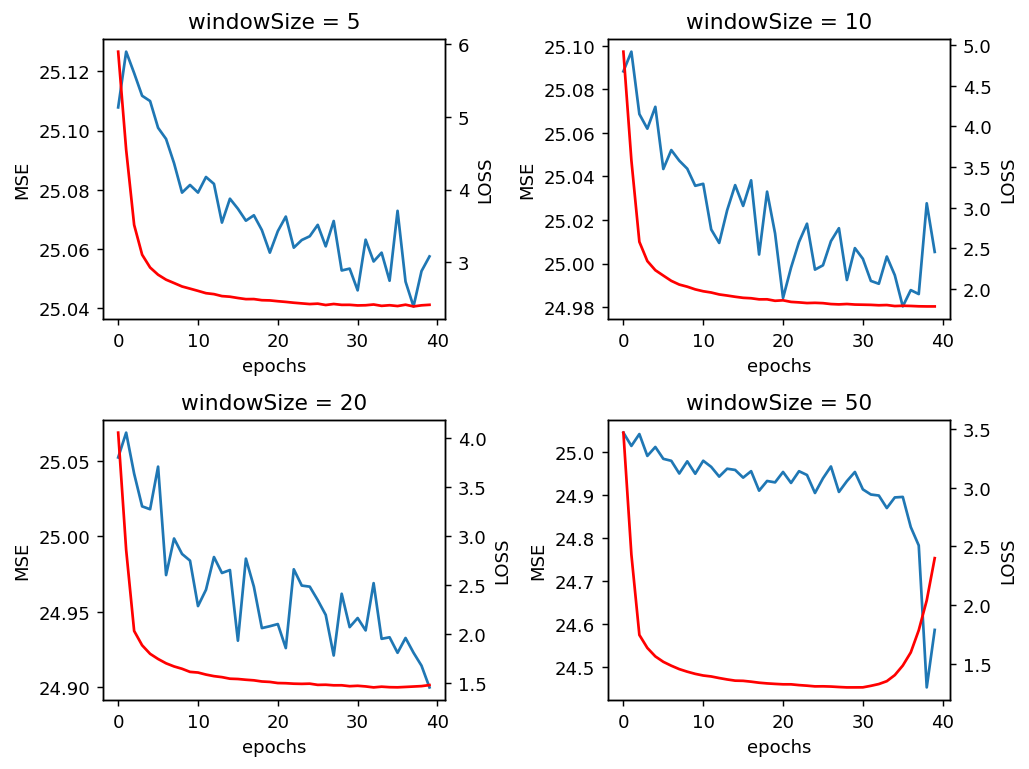


Figure 4: Training the SkipGram model with different window sizes.

All we can say is that increasing the size of each of these parameters causes each time a decrease of the MSE, while arriving faster at a form of convergence.

To find the best parameters, we would have had to train the model on more epochs and perform a grid search, which is too time consuming for our devices.

## **Additional Experimentations**

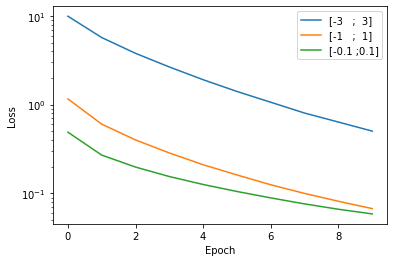


Figure 5: Evolution of the loss function through the epochs, regarding the range of the value of the weights during initialization

In this section, we tried several initializations for the range of the weight’s values, then we observed the evolution of loss function through the epochs.

We can see that increasing the range of weight’s values increases the loss.

We tried smaller ranges than [-0.1, 0,1] and we observed that the loss function remains the same as for [-0.1, 0,1].

# **Conclusion**

The original Skip-Gram paper stated that increasing window size and the negative rate increases the model's performance and they recommend a value between 5 and 20 for the negative rate when having a small training set. Our results adhere well to these recommendations, but they are still incomparable with to the state-of-the-art results due to resources and time constraints.

In this exercise, we focused more on understanding the model's architecture and how it works and the impact of the different hyper-parameters on a smaller scale.

# **References**

[1] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, **One billion word benchmark for measuring progress in statistical language modeling,** *arxiv.org/abs/1312.3005v3*, 2014.

[2] F. Hill, R. Reichart, and A. Korhonen, **Simlex-999: Evaluating semantic models with (genuine) similarity estimation**, *arXiv:1408.3456v1.* 2014.

[3] T. Mikolov, K. Chen, G. Corrado, and J. R. Dean, **Efficient estimation of word representations in vector space**, *arxiv.org/abs/1301.3781,* 2013.

[4] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. R. Dean, **Distributed representations of words and phrases and their compositionality,** *arxiv.org/abs/1310.4546,* 2013.