Natural Language Processing
Exercise 1: skip-Gram

Philibert de Broglie, Sai Deepesh Pokala, Ouassim Hammouch, Mohamed El Hajji

22 February 2020

1 Introduction

In this exercise an attempt was made to reproduce the skip-Gram model using negative sampling, in order to generate our own word embedding. The model architecture would have to be realised with Numpy library only. Skip-Gram architecture is simple, two weights matrices that actually are the words embedding, which are updated throughout the training. In this exercise, we will present the different architecture and experiments that were made in order to generate the best skip-Gram implementation we could come up with.

2 Implementing the model

The skip-Gram model network updates it weights so it learns to predict the most probable context words with respect to the inputted target word. Made of two weights matrices, the target word vector is multiplied with the first matrix while, positive (context) and negative ones are multiplied with the second weight matrix. The network outputs the probability for a word to be a context word of the target word or not to be one. The input context words must be outputted with a probability of one, and the negative words which were sampled randomly in the vocab must have a probability of appearing next to this word of 0. However, the output of the skip-Gram network is never used, only the first weight matrix is finally used as the weights corresponds to the word embedding. The model we chose for skip-Gram using negative sampling, is to have the input word as a one hot encoded vector (same for the context and negative words) and only one hidden layer and a softmax in order to have probabilities of ones or zeros, see figure 1.

In order to do so, a loss measuring the log-likelihood of two words being similar is computed as follow:

$$L(\boldsymbol{\theta}) = \sum_{(t,p)\in+} -\log \frac{1}{1+\exp\left(-\mathbf{w}_t^\top \mathbf{c}_p\right)} + \sum_{(t,n)\in-} -\log \frac{1}{1+\exp\left(\mathbf{w}_t^\top \mathbf{c}_n\right)}$$

where t, c, p, n are used for a target word, a context word, a positive word and a negative word respectively.

The positive words (context words) are denoted by the '+', and the negatives ones by the '-'. The higher the result of the dot product between two words vectors, the more similar they are to each other. Hence, the target word and i is, the more similar or closer the two vectors are together. The use of the logistic function transforms this dot product into a probability. Hence for positive words the goal is to maximise this the fraction on the left hand side so having a large dot product between w_t^T and c_p which is done by minimising the negative log of this. Additionally the goal is to minimise the similarity of target and negative words hence to minimise w_t^T and c_n hence maximising $\frac{1}{1+exp(w_t^T c_n)}$ and therefore minimising the negative log of this. This is why this loss is adapted to the skip-gram model using negative sampling as we try to reduce the loss L by varying the matrix W and C corresponding to the embedding.

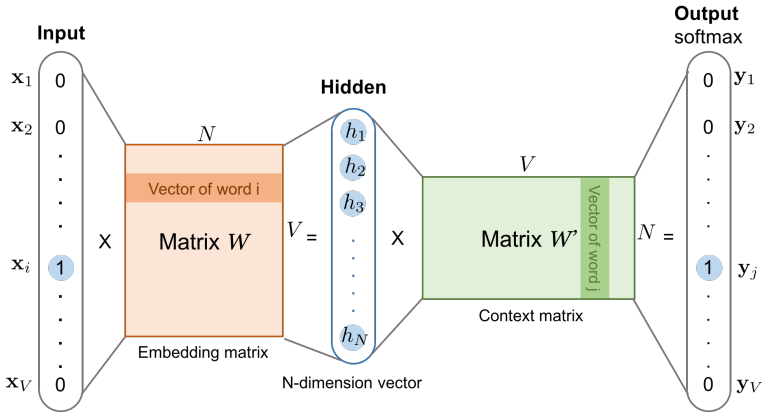

Figure 1: Skip-Gram network Model

Stochastic gradient descent is then used to update the matrices, while in the end, only the first matrix will be used for its embedding. And as the output doesn't really matter, there is no need to actually use a softmax as it won't impact the weight matrices.

3 Training and Testing

In our implementation, we started by writing a "forward" function that did the propagation , but in the end this function was not useful since the loss equation shows us that we can afford to select the columns/lines we need to do the back-propagation, and this without doing the predictions itself! This allowed us to considerably improve the training time since the softmax on such a large vector took some time. The training parameters are the following:

1. Window Size= 2
2. Embedding size = 100
3. Number of Negative Samples per Positive Sample = 4
4. Learning rate = 5e-4

We have tried several learning rates. As soon as the learning rate exceeds 10e-2, we have problems with exploding gradients and therefore very large weight matrices. We have also introduced

gradient clipping to try to counter this phenomenon.

Concerning the embedding size, we tried to increase it to 200 but it has considerably increased the training time. This one being already quite long, we went back to 100.

We have chosen to use a window size of 2. the consequence of this is that the model will learn simpler relationships, but the training time is greatly reduced. also, we have lowered the number of negative samples from 5 to 4 to speed up the training as well. Compared to the initial parameters, only 20 words are chosen (4 context and 16 negative) for each target word, compared to the 48 start words(8 context and 40 negative). Thanks to the negative sampling technique, this means that the training will be more than 2 times faster with these parameters. Maybe that the performances will be lower, but since with 24 hour we only managed to train our model on 4 files out of 100 of the billion word corpus, we think that it will be effective.

4 Conclusion

Building a skip-Gram model without using any libraries such as pytorch or tensorflow was an intense exercise which required a great knowledge of this model architecture, in order to be able to only use numpy. Indeed, after implementing the model in a classic way, it was realised that the loss did not seemed appropriated to the negative sampling. Furthermore, we decided to improve our training time and hence remove the softmax as is is useless for the use we want to make of skip-Gram, only the weight matrix will be used in the future. From there, several features were left behind, such as softmax and sigmoid function which were not really required to produce word embedding and were too complicated to do properly. It was found that with a simpler model, only multiplying or vectors together, and implementing a log-likelihood loss while doing a stochastic gradient descent, all the necessary steps were realised, producing some results. Because the training was very long and time consuming, not much architecture changes and tests could be performed. But, one thing was that for large number of words, a small number of negative sampling should be enough