Université de Genève

Information Retrieval
14x011

TP 4 : Word Embeddings

Author: Sajaendra Thevamanoharan

E-mail: Sajaendra.Thevamanoharan@etu.unige.ch

Author: Deniz Sungurtekin

E-mail: Deniz.Sungurtekin@etu.unige.ch

09 May 2021

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Introduction

In this exercise, we will study the concept of Word2Vec models. Word2vec can utilize either of two models architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. This lab will be concertrated on the SkipGram models. In the SkipGram model, the main goal is to predict the context words from its target word. This task require some data preparation, Machine Learning methods that we will go through in this lab.

Data preparation

We are given a set of French books. We select one among them as our corpus. In our case we selected the "20000 Lieues Sous Les Mers" as our corpus. In this part we will see how we prepare our data to have an efficient representation for the ML algorithms.

Given the corpus, we parse it sentence by sentence. Let's us consider the sentence "The quick brown fox jumps over the lazy dog." as an example of the dataset. We first form a dataset of words and the contexts in which they appear. With a window size of (m = 2), we define context by taking two words to the left of the target, two words to the right of the target.

As we use a Neural Network to train, we cannot feed the neural network words as a text string, we need to find a numerical representation of words. For this, we select a vocabulary of words V from our training documents, e.g. 1000 unique words ($|V| = 1000$).

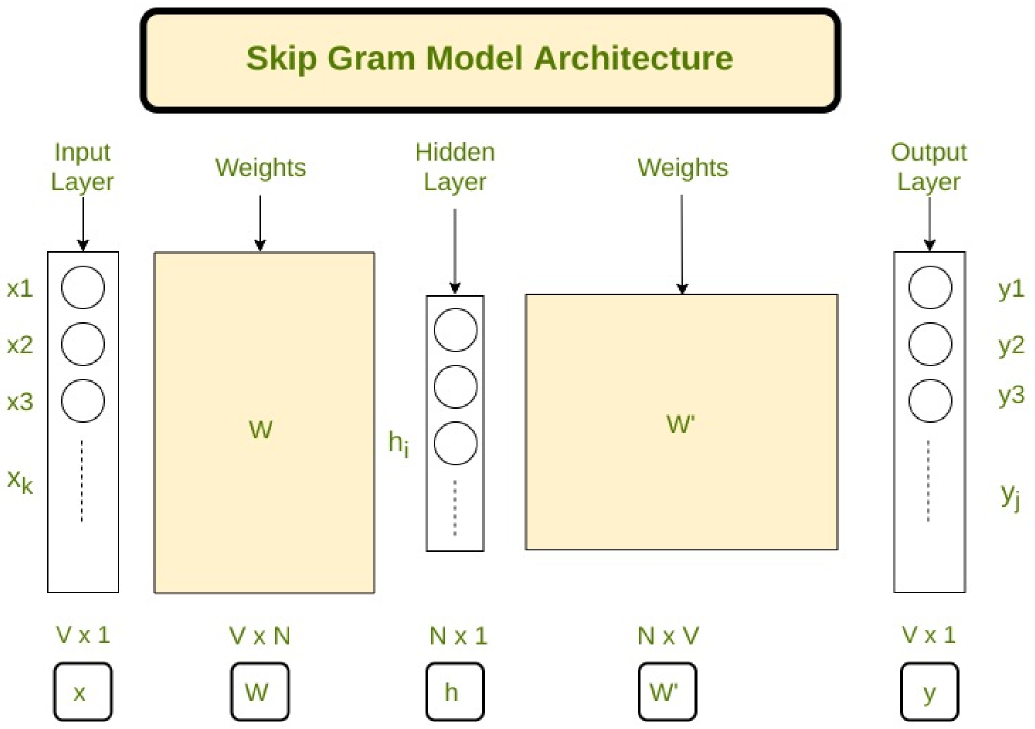


Figure 1: SkipGram model architecture

with N, the embedding size, which is a hyperparameter (features of our words). W and W' are matrices of dimension (Embedding size $\times$ Vocabulary size) and are the learnable parameters of the model. The input layer will be in fact the representation of our words as a one-hot vector. Each vector would have $|V|$ elements (one for every word in our vocabulary) in which "1" is placed in the position corresponding to the given word and and "0" in all of the other positions.

Neural Network

Once the data is well prepared, one should train the model to get a good performance. With these parameteres described just before, the probability model is given by

$$P(c_j|x) = \frac{exp((W'^T h)_j)}{\sum_{i=i}^{|V|} exp((W'^T h)_j)} = y_j$$

To maximize our likelihood we will take the negative log likelihood to define our loss function, which we want to minimise:

$$Loss = -\log(\prod_{c=1}^{C} \frac{(W'^T h)_c)}{\sum_{i=i}^{|V|} exp((W'^T h)_i)}) = -\sum_{c=1}^{C}(W'^T h)_c + C * \log(\sum_{i=1}^{V} exp((W'^T h)_i)$$

Where c are the indexes of the contexts words and C the total number of context words.
The training use stochastic gradient descent (SGD).

Results and Analysis

In this part, we will analyse the results that we obtain from the SkipGram model. This section will fall into two parts : with raw data and with negative sampling.

Raw data

Our books contains more than 7500 sentences which takes a lot of computation times in the range of this two weeks project, so we decided to takes only a part of the entire dataset to test the parameters and analysis their impact without taking too much learning time and data preprocessing time. First, we will take a dataset made of 780 sentences:

We choose a learning step of 0.002, because if this value was bigger we had a divergence on the loss evolution. This value permited us to take only an Embedding size of 10 and 30 epochs without having an insane learning times. Of course, theses small value have a big impact on our model that is why we also did a model on a smaller dataset with only 10 sentences to compare the evolution of the loss with better parameters.

Here are our results for the first dataset:

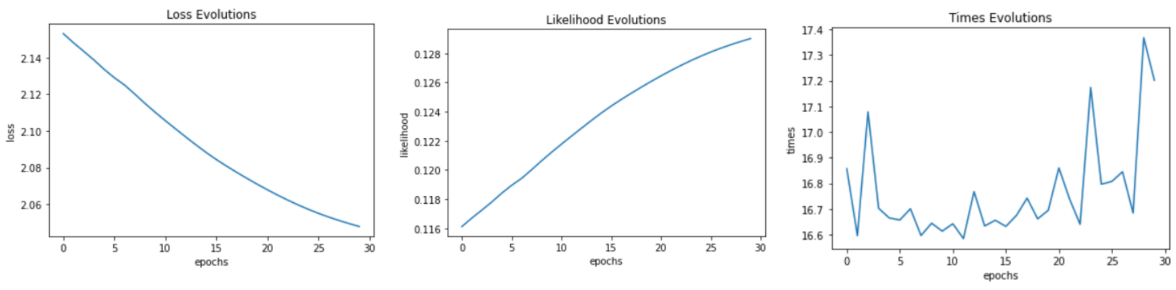


Figure 2: learning and times evolution per epoch

In regards of the training evolution, we can clearly see a linear evolution which in the end was slowing down and giving a logarithmic curve if we had more epochs (we tested it, and the improvement was negligible for a much bigger time). The times of each epoch are pretty similar as we have a varation between 16.5 to 17.5 seconds. By testing a lot of differents parameters, we oberved that changing the embedding size has a exponential effect on the times of each epochs because it add an entier column of size V in our weights matrix. Moreover, this V is our number of unique words in the dataset that's why adding more sentences have a good chance to add unique word which also increase our computation time exponentially.

To better understand the impact of the embedding size and have a better view on the evolution of the model, we decided to take a smaller dataset by reducing the number of sentences and so the number of unique word. For an embedding size of 1000 and 70 epochs (number at which we had a convergence using a 0.005 learning step), we had the following result:

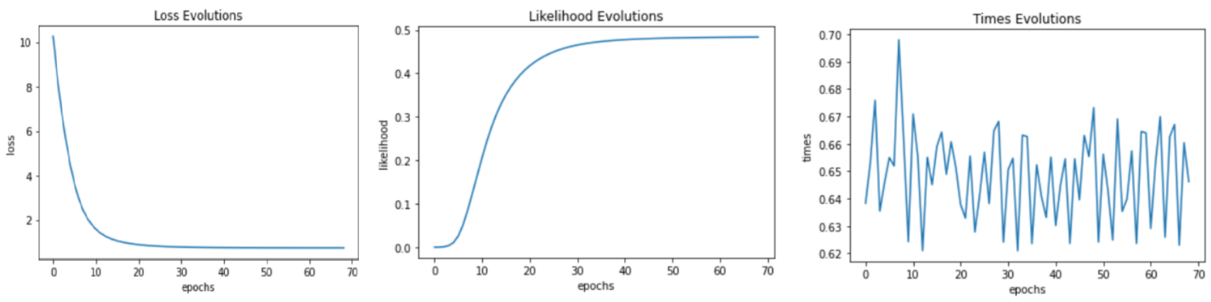
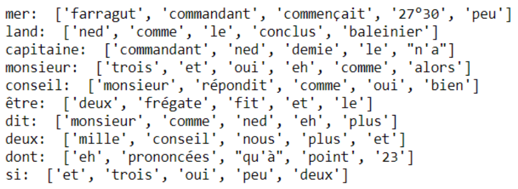


Figure 3: learning and times evolution per epoch

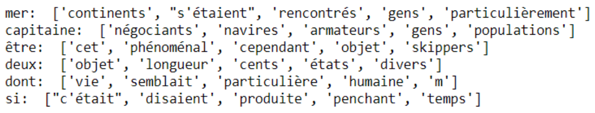
Clearly, the embedding size have a real impact on the convergence of the loss function as the convergence is done in a much smaller value. We could train the model more time to increase the likelihood value which is already better than the first model. As we can see, the embedding size is very important to have a better model but it has also an important impact on the computation time per epoch which is now very small because we considerably reduce the number of unique words.

Now we will compare the results of this two models taking the first five context words for 10 very frequent words in the big dataset:



```
mer:  ['farragut', 'commandant', 'commençait', '27°30', 'peu']
land:  ['ned', 'comme', 'le', 'conclus', 'baleinier']
capitaine:  ['commandant', 'ned', 'demie', 'le', "n'a"]
monsieur:  ['trois', 'et', 'oui', 'eh', 'comme', 'alors']
conseil:  ['monsieur', 'répondit', 'comme', 'oui', 'bien']
être:  ['deux', 'frégate', 'fit', 'et', 'le']
dit:  ['monsieur', 'comme', 'ned', 'eh', 'plus']
deux:  ['mille', 'conseil', 'nous', 'plus', 'et']
dont:  ['eh', 'prononcées', "qu'à", 'point', '23']
si:  ['et', 'trois', 'oui', 'peu', 'deux']
```

Figure 4: Result for big dataset



```
mer:  ['continents', "s'étaient", 'rencontrés', 'gens', 'particulièrement']
capitaine:  ['négociants', 'navires', 'armateurs', 'gens', 'populations']
être:  ['cet', 'phénoménal', 'cependant', 'objet', 'skippers']
deux:  ['objet', 'longueur', 'cents', 'états', 'divers']
dont:  ['vie', 'semblait', 'particulière', 'humaine', 'm']
si:  ["c'était", 'disaient', 'produite', 'penchant', 'temps']
```

Figure 5: Result for small dataset with common word

In the first response, we can clearly see that except for some words the context words make no sense but for the small dataset, we seem to have much better result as a lot of context words make sense. However, we know that this model is far from being perfect but the more important here is that we understood the impact of each parameters on the model and his learning time.

Negative Sampling

Negative sampling is simple solution to overcome the overfitting problem that we are having while performing the training part. The previous method is making the embedding for x closer to c and farther from all other terms $c' \neq c, c' \in V$ together. To solve this problem, the probability model is modified as :

$$P(c|x) = \sigma(h_c.h_x) = \frac{1}{1 + e^{-h_x.h_c}}$$

We generate a dataset with some negative sampling with a label 0 and 1 for positive samples.

$$D : ((quick, the), 1), ((quick, brown), 1), ((quick, jump), 0) \text{ and...}$$

We can rewrite the probability model to also include the labels as :

$$P(y = 1|(c, x)) = \sigma(h_c.h_x) = \frac{1}{1 + e^{-h_x.h_c}}$$

The likelihood can then be written as :

$$L = \lambda \sum_{((x,c),y) \in D} y \log P(y|(c, x)) + (1 - y) \log(1 - P(y|(c, x)))$$

which is the familiar cross entropy loss function used for binary classfication.

We train an another neural network on the same dataset with negative sampling. We plot then the training curve (likelihood/epoch) and plot the running time per epoch for the dataset with negative sampling

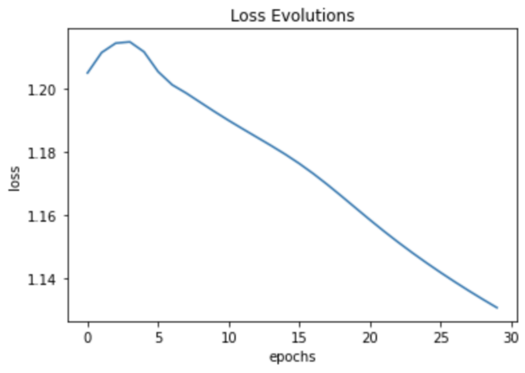


Figure 6: Loss per epoch

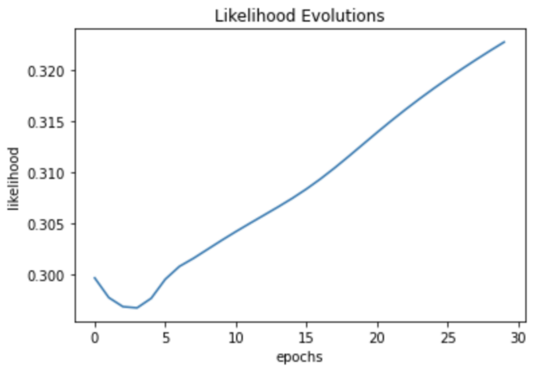


Figure 7: Likelihood per epoch

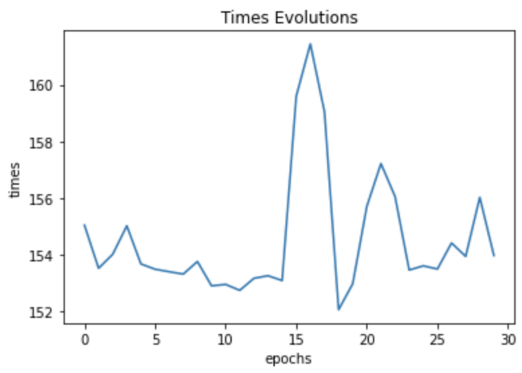
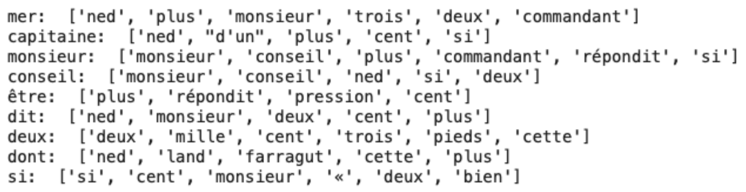


Figure 8: Time per epoch

We can notice that for the dataset with negative sampling, we obtain the same behaviour as the previous one. The negative sampling method is very slow because of the amount of data that we have in the dataset(positive sampling + negative sampling).

We notice also that the range of the loss is lower than the previous model. This may explain from the fact that we don't overfit the model to the exact corpus. This behaviour is also noticeable for the likelihood part.

We find next the 5 most likely words of 10 extremely frequent words, as before.



```
mer:  ['ned', 'plus', 'monsieur', 'trois', 'deux', 'commandant']
capitaine:  ['ned', "d'un", 'plus', 'cent', 'si']
monsieur:  ['monsieur', 'conseil', 'plus', 'commandant', 'répondit', 'si']
conseil:  ['monsieur', 'conseil', 'ned', 'si', 'deux']
être:  ['plus', 'répondit', 'pression', 'cent']
dit:  ['ned', 'monsieur', 'deux', 'cent', 'plus']
deux:  ['deux', 'mille', 'cent', 'trois', 'pieds', 'cette']
dont:  ['ned', 'land', 'farragut', 'cette', 'plus']
si:  ['si', 'cent', 'monsieur', '«', 'deux', 'bien']
```

Figure 9: 5 most likely words of 10 extremely frequent words

We noticed that we are getting different results for the five most likely words of 10 extremely frequent words. This is because of the negative sampling model that we have created.

Skip-gram Negative Sampling (SGNS) helps to speed up training time and improve quality of resulting word vectors. This is done by training the network to only modify a small percentage of the weights rather than all of them. Recall in our example above, we update the weights for every other word and this can take a very long time if the vocab size is large. With SGNS, we only need to update the weights for the target word and a small number (e.g. 5 to 20) of random 'negative' words.