MODÈLE 'WORD2VEC' POUR LA CONSTRUCTION DE VECTEURS DE MOTS

Projet final de master 1 linguistique-informatique Université de Paris-cité

Eléonora Khachaturova Armand Garrigou Léo Rongieras

Table of contents

1 Introduction **1**

2 Approche théorique **2**

2.1 Utilisation de Batch 4

References **5**

List of Figures

List of Tables

1 Introduction

Le modèle Word2Vec a marqué une étape significative dans le domaine du traitement du langage naturel, en révolutionnant la manière d'obtenir des représentation du sens des mots. Depuis son introduction par Mikolov et al. (2013), Word2Vec est devenu l'un des modèles les plus influents et largement adoptés pour la représentation des mots dans les tâches de traitement automatique du langage.

L'objectif de ce rapport est de présenter en détail notre implémentation du modèle Word2Vec et de discuter des résultats obtenus lors de nos expérimentations. Nous aborderons en premier lieu les fondements théoriques du modèle.

Le modèle Word2Vec se base sur l'hypothèse distributionnelle, selon laquelle les mots ayant des contextes similaires ont tendance à partager des significations similaires. En exploitant de vastes corpus de textes, Word2Vec apprend des représentations vectorielles denses pour chaque mot, capturant ainsi les relations sémantiques et syntaxiques entre les mots. Ces représentations vectorielles, souvent appelées embeddings, ont été utilisées avec succès dans de nombreuses applications telles que la traduction automatique, la recherche d'informations et la classification de documents.

Dans notre projet, nous avons souhaité développer notre propre implémentation du modèle Word2Vec, afin de mieux comprendre son fonctionnement interne. Cette approche "from scratch" nous a permis d'explorer en profondeur les mécanismes de Word2Vec, depuis la création des fenêtres contextuelles jusqu'à l'entraînement du réseau neuronal.

Nous avons défini les objectifs suivants pour notre projet : (1) implémenter l'algorithme Word2Vec en utilisant le modèle CBOW, (2) entraîner notre modèle sur un corpus de texte de grande envergure, et (3) évaluer la qualité des embeddings appris, en utilisant différentes mesures de similarité sémantique et en effectuant des tâches d'analogie de mots.

2 Approche théorique

Le modèle Word2Vec propose deux architectures principales : Skip-gram et CBOW (Continuous Bag-of-Words). Dans notre implémentation, nous avons choisi de nous concentrer sur l'architecture CBOW. Contrairement au modèle Skip-gram qui prédit les mots environnants à partir d'un mot central, CBOW utilise le contexte environnant pour prédire le mot central. Cette approche nous permet d'apprendre des embeddings de mots en exploitant les relations contextuelles.

Par exemple en supposant une taille de fenetre de deux : pour la phrase "le chat dort les souris chantent", si notre mot central est "dort" notre contexte sera défini par [le, chat, les, souris]. Comme le montre le schéma suivant :

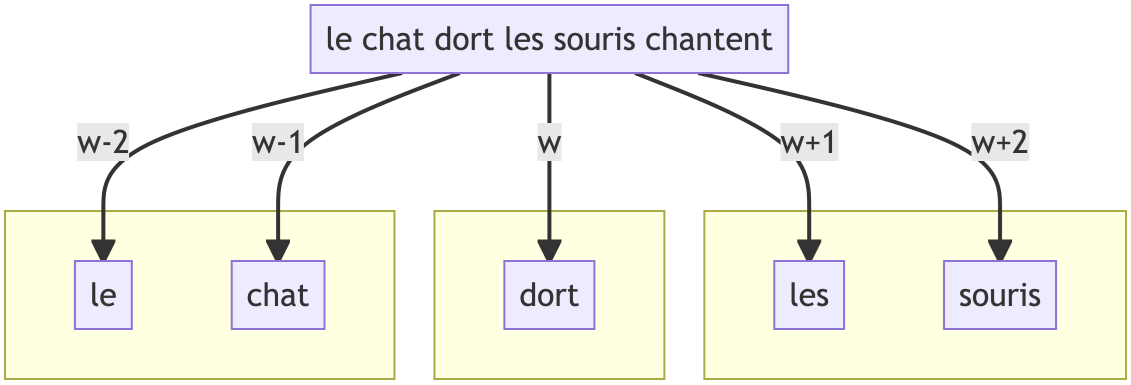


Schéma de la décomposition mot-cible/contexte

L'idée qu'un mot est déterminé par son contexte est parfaitement explicité dans les modèles de langue comme les modèles n-grams, on l'on assume que la probabilité de la présence d'un mot dans une phrase est déterminé par les mots qui précèdent, soit pour un modèle bi-gram :

$$P(w_1, w_2, ..., w_n) = \prod_{i=2}^{n} P(w_i|w_{i-1})$$

.

L'intérêt de l'architecture CBOW est qu'elle permet d'exploiter aussi les mots qui apparaissent *après* notre mot cible. Comme le montre notre schéma, contrairement à un modele n-grams traditionnel, le contexte *global* est pris en compte comme information. Un des apports majeur

du modèle word2vec est donc se passage à la *bi-directionalité*, prendre le contexte de gauche et de droite.

Nous allons donc voir comment CBOW peut apprendre ces probabilités.

Notre modèle prend en entré un vecteur contexte x et retourne un vecteur mot y qui correspond au mot au centre de notre contexte. On définit deux matrices $\mathcal{A}^{(c)} \in \mathbb{R}^{|V| \times n}$ et $\mathcal{A}^{(w)} \in \mathbb{R}^{n \times |V|}$ On note :

- w_i le mot en position i du vocabulaire V
- $\mathcal{A}^{(c)} \in \mathbb{R}^{|V| \times n}$ la matrice des mots en entré où a_i correspond à la i-ème ligne de $\mathcal{A}^{(c)}$ c'est à dire le vecteur qui représente le mot en entré w_i
- $\mathcal{A}^{(w)} \in \mathbb{R}^{n \times |V|}$ la matrice en sortie où y_i correspond à la i-ème colone de $\mathcal{A}^{(w)}$ c'est à dire le vecteur qui représente le mot en sortie w_i
- n correspond à la dimension arbitraire des embeddings

Les étapes du fonctionnement du modèle peuvent être décrites de telle sorte:

1. On crée un vecteur d'entré composé des indices i des mots $\in V$ en contexte avec une fenètre de taille N soit
   $v^c = (x^{c-N}, ..., x^{c-1}, x^{c+1}, ..., x^{c+N})$
2. On obtient nos embeddings pour ce contexte. Soit $\mathcal{A}^{(c)} v^c$, on obtient une matrice de taille $N \times n$ ou chaque ligne i correspond à l'embeddings du mot en contexte. Chaque vecteur dense est de dimension n
3. On veut récupérer la somme des vecteurs appartenant au contexte C, c'est à dire la somme des éléments **par colone** de notre matrices $\mathcal{A}^{(c)} v^c$ soit
   $$\hat{x} = \sum_{i=1}^{N \times 2} a_i$$
4. Notre vecteur score z est obtenu par
   $$z = \mathcal{A}^{(w)} \sum_{i=1}^{N \times 2} a_i = \mathcal{A}^{(w)} \hat{x}$$
5. On retourne ce score transformé en log probabilité soit $\hat{y} = \text{log\_softmax}(z)$

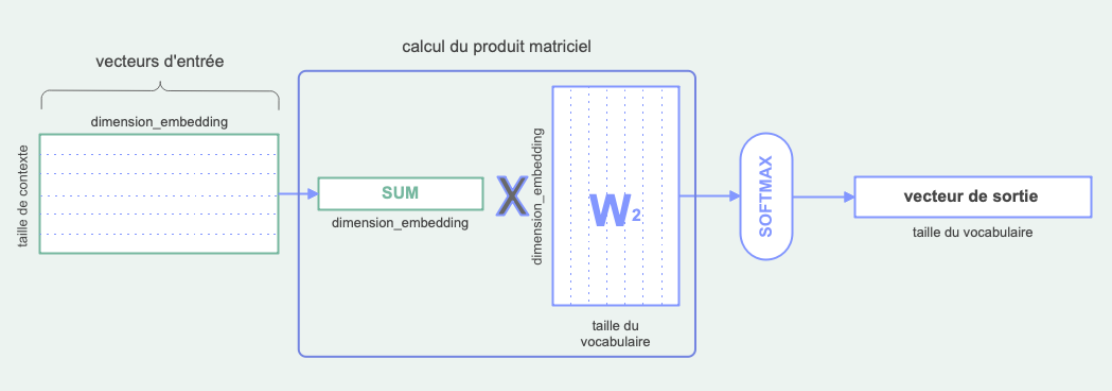
On peut donc remarquer que après l'application du $softmax$ on obtient un vecteur $\hat{y}$ de la taille du vocabulaire V, où chaque $\widehat{y_i}$ correspond à la *log-probabilité* que $\widehat{y_i}$ soit le mot cible du contexte en entrée.

On peut le vérifier algébriquement : en effet notre score est obtenu par le scalaire entre la matrice $\mathcal{A}^{(w)} \in \mathbb{R}^{n \times |V|}$ et le vecteur $\hat{x}$ de taille n on a donc :

$$\begin{bmatrix} y_{1,1} & y_{2,1} & \cdots & y_{|V|,1} \\ y_{1,2} & \cdots & \cdots & \cdots \\ y_{1,3} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ y_{1,n} & \cdots & \cdots & y_{|V|,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} \widehat{y_1} & \widehat{y_2} & \widehat{y_3} & . & . & . & \widehat{y_{|V|}} \end{bmatrix}$$

On obtient donc un vecteur $\hat{y}$ de taille $|V|$

On peut résumer l'ensemble avec le schéma suivant :



[1]

en dernier lieu, le *softmax* prend en entré le vecteur de score z de taille $|V|$ et renvoie un vecteur de même dimension, où chaques composants i est défini comme :

$$(\text{softmax}(z))_i = \frac{e^{z_i}}{\sum_{j=1}^{|V|} e^{z_i}}$$

2.1 Utilisation de Batch

[1] On peut noter qu'il n'y pas dans l'architecture CBOW de fonction d'activation, la seule couche cachée du réseau correspond justement à la somme des embeddings des mots en contexte que nous avons décrit, le nombre de neurone correspond simplement au nombre de mot dans le vocabulaire.

References

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient Estimation of Word Representations in Vector Space." https://arxiv.org/abs/1301.3781.