Rapport projet TLNL
Plongements de mots statistiques

4 novembre 2023

Leader : Yanis Labeyrie | Follower : Cléa Han

1 Introduction

L'objectif de ce projet est d'implémenter en Python, d'entraîner et d'évaluer un modèle de plongement de mots statique semblable au célèbre algorithme word2vec. Le modèle de base que nous avons implémenté ici est alors le modèle *skip gram with negative sampling*. Nous démarrons par une implémentation basique de la méthode susmentionnée, puis nous proposons une implémentation des améliorations de ce modèle comme proposé dans le papier de Mikolov [1].

2 Modèle Initial

Le modèle initial à réaliser est sous la forme d'un fichier w2v.py qui prend en entrée un texte tokenisé, ainsi que les valeurs des paramètres suivants : la dimension des embeddings n, la taille des contextes L, le nombre de contextes négatif k pour un contexte positif, *eta* le taux d'apprentissage, e le nombre d'itérations et *minc* le nombre minimal d'occurrence d'un mot pour que son plongement soit produit.

En effet, le plongement ou autrement appelé "embedding" permet la vectorisation des mots. Les mots sont alors représentés sous forme de vecteurs de nombres réels.

Notre modèle initial retourne un fichier d'embeddings qui indique en première ligne le nombre de plongements contenus dans le fichier et la dimension d de chaque plongement utilisé, puis chaque ligne suivante représente un mot sous forme de chaîne de caractères suivi de son plongement. Ainsi, dans la suite du rapport, plongement ou embeddings désigneront les vecteurs de nombres réels représentants les mots manipulés.

2.1 Le classifieur

Le modèle initial se base sur la classification binaire de plongements. En effet, le classifieur va calculer la probabilité entre deux mots m et c en utilisant leur plongement, où m est le mot cible et c un mot du contexte d'occurrence de m : $P(+|m, c)$. C'est la probabilité que c appartienne au contexte de m. Plus cette probabilité est élevée, plus les plongements m et c sont proches.

La proximité entre deux plongements est évaluée avec le produit scalaire des deux vecteurs : $m \cdot c$.

Afin d'en obtenir des probabilités sur la proximité de deux mots m et c, la fonction sigmoïde est utilisée, on a alors le calcul suivant de la probabilité $P(+|m, c)$:

$$P(+|m, c) = \frac{1}{1 + \exp(-m \cdot c)}$$

La probabilité que c n'appartienne pas au contexte de m, notée $P(-|m, c)$ se calcule alors de la manière suivante :

$$P(-|m, c) = 1 - P(+|m, c) = \sigma(-m \cdot c) = \frac{1}{1 + \exp(m \cdot c)}$$

De cette manière, chaque mot du corpus étudié devra posséder un plongement en tant que mot cible et un plongement en tant que mot de contexte. Les plongements des mots considérés comme mots cibles seront stockés dans une matrice M conservée, tandis que les plongements des mots considérés comme mots de contexte sont contenus dans une matrice C temporaire qui servira à l'apprentissage notamment.

De plus, dans la matrice M, seuls les plongements des mots dont le nombre d'occurrence est supérieur à *minc* seront considérés, contrairement à la matrice C.

2.2 Apprentissage du modèle initial

L'apprentissage du modèle initial w2v consiste à calculer les plongements des mots considérés (dont le nombre d'occurrence est supérieur à *minc*). Pour se faire, nous générons des plongements de manière aléatoire en créant des vecteurs de dimension n composé de nombres réels aléatoires. Puis à partir de ces vecteurs, nous allons les mettre à jour avec la méthode de descente de gradient selon un certain taux d'apprentissage à l'aide d'exemples positifs et négatifs.

En effet, nous générons alors en amont des exemples positifs et négatifs en parcourant le texte étudié. Pour tout mot cible m, on sélectionne un mot dans le contexte de m notés cpos et des mots aléatoires hors du contexte de m notés cneg. Cela permet de construire un exemple positif de type : (+, m, cpos) et k exemples négatifs : (-, m, cneg1), ...(-, m, cnegk).

Un exemple positif associé à ses k exemples négatifs permet de constituer un mini-batch pour chaque mot cible m. Chaque mini-batch permet alors de mettre à jour le classifieur et donc les valeurs des plongements des mots appris, aussi bien les mots cibles que les mots de contextes, qu'ils constituent des exemples positifs ou des exemples négatifs.

Dans l'apprentissage, pour chaque mini-batch, nous avons en terme de calculs le problème d'optimisation suivant : on souhaite maximiser la probabilité de l'exemple positif : $P(+|m, cpos)$ et minimiser la probabilité des exemples négatifs : $P(+|m, cnegi)$. Cela revient dont à maximiser le rapport de $P(+|m, cpos)$ avec le produit des probabilités des exemples négatifs. Pour simplifier le calcul du gradient, on minimise alors l'opposé du logarithme d'un tel produit, ce qui constitue la fonction de perte du classifieur :

$$L = -\log\left(\frac{P(+, m, cpos)}{\prod_{i=1}^{k} P(-, m, cnegi)}\right)$$

$$L = -\log \sigma(m \cdot cpos) + \sum_{i=1}^{k} \log \sigma(-m \cdot cnegi)$$

La fonction logarithme dans la fonction de perte permet d'obtenir une expression simple du gradient :

$$\frac{\partial L}{\partial cpos} = [\sigma(m \cdot cpos) - 1]m$$

$$\frac{\partial L}{\partial cneg} = [\sigma(m \cdot cneg)]m$$

$$\frac{\partial L}{\partial m} = [\sigma(m \cdot cpos) - 1]cpos + \sum_{i=1}^{k} [\sigma(m \cdot cnegi)]cnegi$$

Pour chaque mini-batch, le gradient de la fonction de perte est calculé et les paramètres du modèle sont mis à jour, c'est-à-dire les plongements, en prenant en compte un taux d'apprentissage η :

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(m \cdot c_{pos}^{t}) - 1]m$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(m \cdot c_{neg}^{t})]m$$

et

$$m^{t+1} = m^{t} - \eta([\sigma(m^{t} \cdot c_{pos}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(m^{t} \cdot c_{neg_i})]c_{neg_i})$$

2.3 Résultats

Pour entraîner le modèle, nous avons repris les hyperparamètres proposés par l'énoncé à titre de comparaison, puis nous avons pu faire varier différentes valeurs pour chaque hyperparamètre considéré par notre modèle initial Word2Vec, tout en veillant à ne pas surcharger significativement la charge et le temps de calcul des ressources à nos dispositions. Nous allons présenter les meilleurs résultats que nous avons pu obtenir selon les hyperparamètres optimaux suivants :

Hyperparamètres	Valeur
Taille des embeddings n	100
Nombre d'itérations e	1
Taille de la fenêtre centrée L	5
Taux d'apprentissage eta	0.1
Nombre d'exemples négatifs k	10
Nombre minimal d'occurrence pour avoir un plongement $minc$	2

TABLE 1 – Hyperparamètres d'entraînement des embeddings

Après un entraînement assez long (3h) on obtient la courbe d'entraînement suivante :

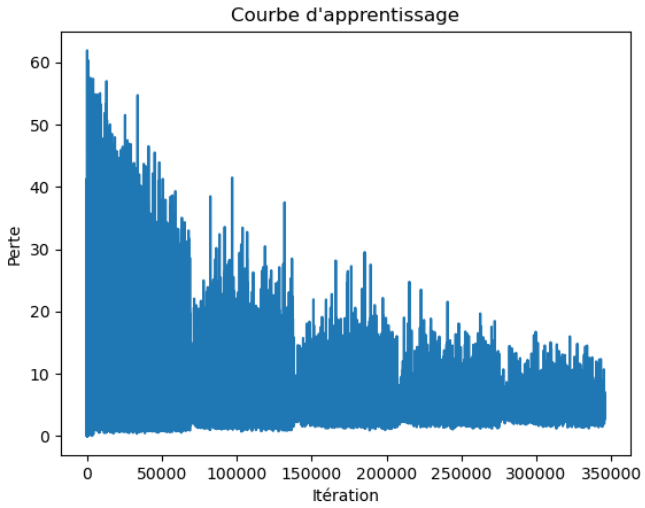

FIGURE 1 – Valeur de la fonction de coût en fonction du nombre d'itérations

On constate d'après cette courbe d'apprentissage que le modèle effectue un apprentissage qui semble efficace car la loss diminue bien au fur et à mesure des itérations.

3 Évaluation - Erreurs et limitations

Afin d'effectuer l'évaluation de la qualité des embeddings produits à l'issue de l'apprentissage de notre modèle initial, nous allons sélectionner manuellement des triplets de mots (m, m+, m-). Ces triplets sont produits de sorte que le sens de du mot m+ soit plus proche de m que ne l'est celui de m-, comme dans l'exemple (vélo, bicyclette, chat). Cela permettra alors de vérifier la capacité du modèle à prendre en compte la sémantique linguistique des mots étudiés à travers l'utilisation de plongements.

On définit alors une mesure de similarité *sim* qui prend en argument deux vecteurs a et b. Cette mesure de similarité est défini par le cosinus de l'angle des deux vecteurs :

$$\text{sim}(a, b) = \cos(a, b) = \frac{a \cdot b}{\|a\| \times \|b\|}$$

Étant donné les plongements m, $m+$, et $m-$, on souhaite donc que :

$$\text{sim}(m, m+) > \text{sim}(m, m-)$$

afin de témoigner de la bonne implémentation des plongements des mots du corpus considéré.

Nous disposons d'un fichier d'évaluation disponible pour le texte *Le Comte de Monte Cristo* d'Alexandre Dumas (le fichier Le_comte_de_Monte_Cristo.100.sim).

Lorsque l'on applique notre modèle initial à ce jeu de données, on obtient seulement 49% d'accuracy ce qui est un résultat plutôt décevant, qui peut s'expliquer par la simplicité du modèle mais aussi par plusieurs problèmes théoriques :

- On choisit des mots négatifs au hasard, alors qu'il existe peut-être une manière plus pertinente de les choisir vis-à-vis de notre tâche. En effet, la langue française possède des nuances plus ou moins subtils en ce qui concerne la signification des mots dans leur contexte, il faudrait alors nuancer à quel point un exemple est plus négatif qu'un autre exemple. Choisir les mots les "plus négatifs" lors de l'entraînement afin d'obtenir un modèle plus précis en terme de sémantique.
- Le texte contient énormément de mots qui reviennent trop souvent, ce qui peut diminuer la capacité de notre modèle à produire des embeddings pertinents. En effet, cela diminue l'importance des embeddings des mots qui apportent réellement de l'intérêt au corpus étudié en les diluant en quelque sorte parmi les embeddings d'autres mots qui n'apportent en réalité pas de sens sémantique au texte. Dans ce cas, en plus de la variable minc prise en compte dans notre modèle initial, il faudrait envisager de mettre de côté les embeddings de particules et de mots servants uniquement à la syntaxe et non au sens des mots, comme les pronoms par exemple.
- Il y a un compromis à réaliser entre charge et temps de calculs et complexité du modèle. En effet, nous pourrions prendre une taille de dimension d'embeddings n plus importante ce qui augmenterait naturellement la complexité de notre modèle et améliorerait relativement ses performances, cependant, cela devrait se faire de manière adaptée et ajustée à l'ampleur du projet et des ressources allouées. Les ressources à notre disposition ne nous permettait pas d'augmenter la complexité du modèle sans augmenter significativement notre temps et charge de calculs pour l'apprentissage du modèle.

4 Piste à creuser : Améliorations de l'algorithme

4.1 Description

D'après l'article de référence sur le modèle [1] : il existe des solutions pour contourner les différents problèmes mentionnés précédemment. Par exemple concernant le fait que certains mots comme "le","la", ect.. reviennent trop souvent dans le corpus, il existe pour lutter contre ce problème une méthode appelée "subsampling of frequent word" : *"In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., "in", "the", and "a"). Such words usually provide less information value than the rare words"* [1].

Cette méthode consiste à calculer la fréquence de chaque mot w_i du vocabulaire dans le corpus $f(w_i)$, et on définit la probabilité de supprimer le mot du corpus d'entraînement par $p(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$, où t est un seuil que l'article suggère de fixer à 10^{-5}.

Ainsi les mots qui reviennent trop souvent comme les pronoms et qui ne contiennent pas beaucoup d'information sont en partie éliminés du corpus.

On s'attend à ce que cette méthode ait deux avantages majeurs :
- L'accélération de l'apprentissage car on réduit la taille du dataset d'entrainement.
- L'amélioration des performances.

Une autre solution évoquée dans l'article est l'amélioration du "negative sampling". Notre implémentation initiale consistait à prendre au hasard un nombre k de mots négatifs (donc n'étant pas dans la fenêtre de contexte) avec une distribution uniforme dans le corpus. L'amélioration du négative sampling consiste à, au lieu de choisir des mots négatifs uniformément, les choisir relativement à leur distribution unigramme : on définit la probabilité de choix d'un mot négatif w par $P(w)$ où $P(w)$ est égal à la distribution unigramme (donc la fréquence du mot dans le texte) $U(w)$ mise à la puissance 3/4 (i.e.,$U(w)^{3/4}$). D'après l'article l'emploi de cette formule plutôt qu'une autre distribution uniforme ou unigramme mène à de significativement meilleurs performances car comme pour l'astuce précédente, les mots fréquents contiennent moins d'information.

4.2 Mise en œuvre

Par rapport à la précédente implémentation, nous avons rajouté avant entraînement une fonction de pré-traitement du corpus qui calcule les fréquences des différents mots et les supprime conformément à la règle susmentionnée. Par ailleurs, à chaque itérations quand on choisit un nombre k de mots négatifs, on réutilise les fréquences déjà calculées pour échantillonner les mots négatifs.

4.3 Résultats

En utilisant ces différentes améliorations de l'algorithme, on constate une amélioration des performances et on obtient une accuracy d'environ 58% après 5 epochs (donc 5 passages d'entraînement sur le corpus d'entraînement). En effet, on n'obtient pas de résultats satisfaisants en réalisant seulement un passage sur le corpus comme le montre la courbe d'entraînement suivante :

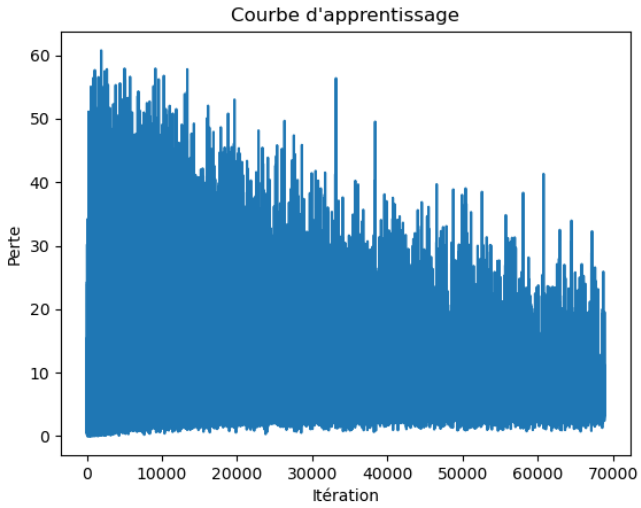

FIGURE 2 – Valeur de la fonction de coût en fonction du nombre d'itérations pour 1 epoch

On observe que la fonction de coût n'a pas atteint de plateau en fin d'entraînement, ce qui signifie que l'on peut espérer la faire baisser encore plus si on utilise plus d'itérations. Avec une epoch on obtient seulement 53% d'accuracy. Voici la courbe d'entraînement après 5 epochs :

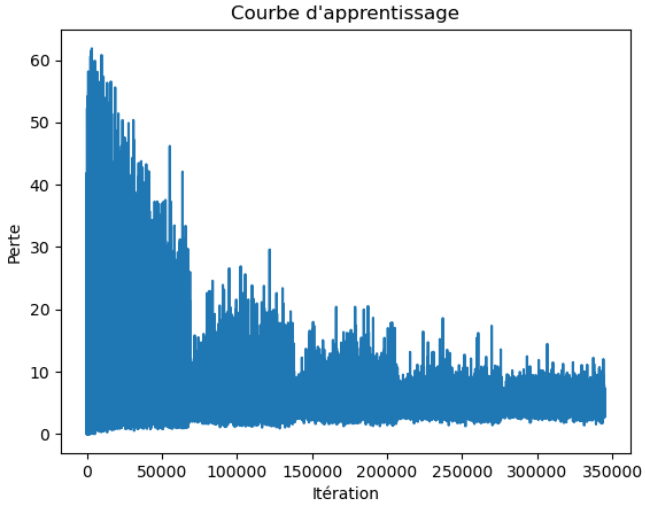

FIGURE 3 – Valeur de la fonction de coût en fonction du nombre d'itérations pour 5 epochs

On constate cette fois-ci que la fonction de coût atteint un plateau. Nous pouvons alors arrêter l'entraînement à ce stade là, car les améliorations appor-

tées pour plus d'itérations seront négligeables.

5 Conclusions et perspectives

Le modèle initial utilisant la méthode *skip-gram with negative sampling* fournit des résultats et des performances limitées malgré la prise en compte d'une fenêtre large de taille L tout autour du mot cible. En effet, l'analyse linéaire du corpus lors de l'entraînement ne permet pas au modèle de réellement prendre en compte la hiérarchie sémantique des mots traités, au vu de ses performances. L'utilisation d'embeddings reste une méthode pertinente afin de représenter les mots de manière numérique et à dimension raisonnable (n ici, correspondant à la dimension de chaque embedding), cependant elle représente difficilement la linguistique utilisée. Elle permet de prendre en compte la polysémie d'une certaine manière, car les embeddings prennent uniquement en compte les mots d'un même contexte. Cependant il y a une prise en compte uniquement partielle de la "distance" sémantique entre chaque mot : En comptant le nombre d'occurrences dans la fenêtre des mots de même contexte, le modèle va dans l'absolu compter qu'il y a un nombre largement significatif de mots courts non significatifs, comme les pronoms.

Ainsi ce modèle initial est sous l'hypothèse que chaque mot de la langue française porte un poids linguistique et sémantique égal d'une certaine manière, ce qui n'est pas le cas, notamment avec les mots de liaisons et les mots servant à la grammaire française. Ce qui explique un entraînement évoluant correctement, mais qui performe de manière décevante sur nos textes.

Néanmoins, lorsque cette hypothèse susmentionnée est compensée grâce à la méthode décrite dans la piste d'amélioration, il y a effectivement une relative amélioration des performances.

Ainsi, l'utilisation des plongements est une méthode pertinente pour une représentation mathématique des mots en traitement du langage, cependant, ils ne permettent pas une représentation totale de la linguistique, car la sémantique des mots n'est considérée que de manière partielle. Nous prenons en compte les mots et leur contexte qui leur sont "local", sans pour autant prendre en compte leur sémantique globale. La représentation pourrait être améliorée en prenant des plongements de dimension n plus conséquente, mais cela se fera au dépend du temps et de la charge d'entraînement.

Ainsi, à l'issue du travail sur les embeddings, une piste possible d'amélioration serait de chercher un moyen de prendre en compte la sémantique plus globale des mots, tout en ne surchargeant pas les ressources de nos systèmes.

Références

[1] Tomas Mikolov et al. « Distributed Representations of Words and Phrases and their Compositionality ». In : *arXiv* (oct. 2013). DOI : 10.48550/arXiv.1310.4546. URL : https://arxiv.org/abs/1310.4546.