Rapport projet TLNL

Leader : Romain Durand Follower : Thomas Allouche

18 septembre 2023

1 Introduction

Pour de nombreuse tâche en informatique, il est nécessaire de représenter un objet du monde concret ou abstrait. Cette représentation peut servir d'entrer à un modèle mathématique et/ou nous permettre d'analyser, comparer, visualiser et parfois comprendre certain objet complexe.
En TAL, nous nous intéressons à la représentation de mot. Une bonne représentation de mots doit être numérique, de dimension raisonnable, permettre de calculer une distance entre les mots qui ont du sens et prendre en compte la polysémie.Une représentation intuitive des mots comme l'encodage one-hot ou bien fondée sur les cooccurrences ne suffit pas à respect ces exigences.

Dans ce TP, nous allons étudier l'algorithme Word2veq, plus précisément Skip-gram with Negative Sampling (SGNS) qui nous permet d'avoir une représentation satisfaisante des mots.
Dans un premier temps, nous présenterons une version simplifiée de l'algorithme ainsi que ces résultats, puis nous étudierons une version améliorée de l'algorithme.

Les algorithmes seront entrainés sur le corpus "Le comte de Monte-Cristo" d'Alexandre Dumas qui comporte 211443 mots et 13021 mots différents.

2 Modèle Initial

2.1 Description :

L'algorithme SGNS utilise la notion de plongements d'un mot, qui est une représentation vectorielle de son contexte. On définit la probabilité que le mot c apparait dans le contexte du mot m comme $\sigma(m.c)$ avec σ la fonction sigmoïd.

En négligeant les dépendances entre mots du contexte et les positions relatifs des mots dans le contexte, on définit une fonction log vraisemblance négative :

$$L(m) = -\left[\log \sigma(c_{pos} \cdot m) + \sum_{i=1}^{k} \log \sigma(-c_{negi} \cdot m)\right] \tag{1}$$

où

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

- m est le plongement du mot cible
- cpos est le plongement d'un mot du contexte du mot cible
- $(c_{negi})_{i\in[0,k]}$ est un ensemble de k plongements de mot qui n'appartiennent pas au contexte du mot cible

L'objectif est de maximiser la log-vraisemblance et donc de minimiser notre fonction de cout.
Cette fonction de cout est optimisée par descente stochastique du gradient. La mise à jour des plongements se fait à l'aide des expressions des gradients de la fonction de cout en fonction des paramètres du modèle.
Après calcul, nous trouvons les équations suivantes :

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

L'algorithme est testé sur 100 triplets de mot. Le premier mot est similaire au deuxième et éloigné du troisième. On calcule la similarité cosinus des couples c1=(mot1,mot2) et c2=(mot1,mot3), si c1>c2 alors le plongement est construit correctement, sinon non. À l'aide de ces 100 mots, nous calculons un taux de réussite. On définit la similarité cosinus comme :

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

2.2 Mise en œuvre :

Afin de réaliser l'apprentissage de nos plongements, nous devons pré-traiter nos données d'entrainements. Pour cela, nous utilisons plusieurs fonctions :

clean-text : Cette fonction prend en paramètre le chemin du fichier et texte et renvoie le texte nettoyé, c'est-à-dire qu'on lui a enlevé les caractères "<s>" et "<\s>" qui sont considérés comme du bruit pour la mise à jour de nos plongements.

build-Index : Cette fonction prend en paramètre le texte nettoyé et renvoie un dictionnaire.Chaque mot unique du texte est indexé par un nombre. C'est avec ces indexations que nous travaillerons, en particulier pour un souci de mémoire.

build-Occ : Cette fonction prend en paramètre le texte nettoyé et son Index et renvoie un dictionnaire Occ. Le dictionnaire associe à chaque index son occurrence dans le texte.

build-Occ-freq : Cette fonction prend en paramètre Occ, Index et un paramètre min-count. Elle renvoie un dictionnaire Occ-freq qui associe à chaque mot dont l'occurrence dans le texte est au moins min-count, sa probabilité définit comme $\frac{occurence}{|V|_{occurence \geq min\_count}}$. Les mots de ce dictionnaire sont les mots dont nous voulons construire les plongements (mots cibles).C'est également le dictionnaire qui définira la distribution de proportionnalité utilisée pour l'échantillonnage des exemples négatifs, dans la fonction Exemple-apprentissage.

Exemple-apprentissage : Cette fonction prend en paramètre la liste des phrases du texte, l'Index, L, k, min-count, Occ-freq. Elle renvoie le dictionnaire Data-train qui associe à chaque mot cible, les mots de son contexte (exemples positifs) et k fois plus de mot hors de son contexte (exemples négatifs). Les exemples positifs et négatifs sont déterminés en utilisant une liste glissante de taille L fixe. On parcourt le texte et dès qu'un mot cible apparait dans la fenêtre cible, nous lui associons les mots de son contexte dont l'occurrence est supérieure à min-count. Pour chaque mot sélectionné, nous tirons selon la distribution de probabilité définit par Occ-freq, k exemples négatif.Ces exemples sont tirés à l'aide de la fonction Distrib.

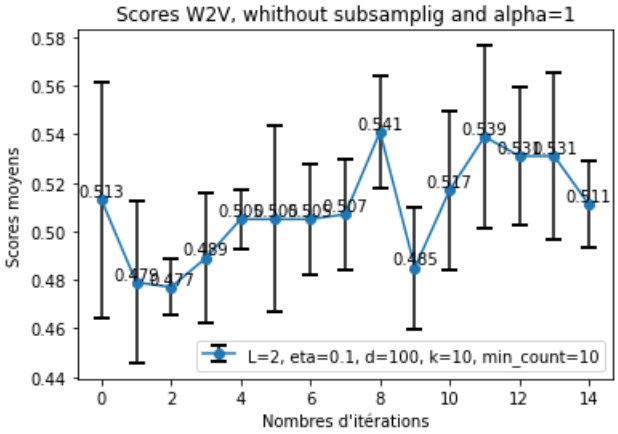
Distrib : Cette fonction prend en paramètre Occ-freq. On prend un nombre aléatoire n entre 0 et 1. Pour chaque mot dans Occ-freq, on calcule le somme cumulé de leur fréquence. Dès que n est plus petit que cette somme, on retourne le mot actuel de la boucle. Cette méthode est une méthode simplifiée d'inversion de la fonction de répartition. Elle est en O(n), avec n le nombre d'événements (nombre de mots dans Occ-freq).

Une fois le pré-traitement effectué, nous passons à la phase d'apprentissage :
ApprentissageW2v : Cette fonction prend en paramètre Index, d, L, k, eta, mincount, nbr-iter et Data-train et Occ. Nous commençons par initialiser les deux matrices de plongements (cible et contexte) avec des échantillons aléatoires d'une distribution uniforme dont les bornes sont inversement proportionnelles à la dimension d des plongements. Cette initialisation permet d'avoir d'empêcher l'explosion des gradients même si le problème est complexe. Pour chaque exemple d'apprentissage dans Data-train, on met à jour les plongements selon les équations décrites dans la partie description. Cet algorithme d'apprentissage est utilisé nbr-iter fois.À la fin de chaque itération, on calcule le score des plongements à l'aide de la fonction eval. La fonction renvoie les plongements des mots cible et les scores en fonction des itérations.

eval : Cette fonction prend en paramètre le chemin du fichier test, les plongements cibles et Index et renvoient le score des plongements sur les 100 triplets de mots du fichier test.

W2v : Cette fonction renvoie un graphique. Nous calculons les scores moyens des modèles sur 5 itérations de l'algorithme de Word2Vec, entrainé sur 5-fold (avec 1 fold est égale à 90% du texte) avec les intervalles à 95%. Nous avons observé ces résultats sur 15 itérations de l'algorithme de mise à jour de plongement. Nous utilisons 5-fold pour limité le surapprentissage. Pour la baseline, nous utilisons les hyper-paramètres fournit dans le TP, soit L=2, k=10, min-count=10 et d=100.

2.3 Résulat :



Nous remarquons que le score est faible, quasiment une prédiction aléatoire. La variance est très importante et les résultats ne converge pas. De plus, le temps de calcul est particulièrement long pour un corpus aussi court, en moyenne 6.3 min pour effectuer l'échantillonnage total des exemples négatifs.
L'objet de la partie 2 de ce rapport est d'apporter des solutions à ces problèmes.

2.4 Piste à creuser 1 : Améliorations des performances :

2.4.1 Description :

L'objectif de cette partie est d'atteindre un score pour W2V aux alentours de 0.8 avec un écart type raisonnable et un temps de calcul plus faible. Nous implémenterons une solution pour améliorer le temps de calcul et deux solutions décrites dans [1] qui sont le sous-échantillonnage et une distribution d'échantillonnage ajusté.

Changement des hyper-paramètres : Les hyper-paramètres utilisés dans la Baseline étaient pas adapté à notre corpus et à nos données de test. Nous changeons L et eta.

Méthode Alias : La méthode Alias, décrite et découverte dans [2], est une technique efficace pour générer des variables aléatoires discrètes avec une complexité de temps constante pour la génération, après une phase de prétraitement. Nous ne la détaillerons pas en détail dans ce TP. Nous l'utiliserons pour la suite à la place de la méthode simplifiée d'inversion de la fonction de répartition qui est en O(n), avec n le nombre d'événements (nombre de mots dans Occ-freq). Son rôle est donc, comme pour la précédente méthode, de générer, selon la distribution de probabilité définit par Occ-freq, k exemples négatif pour un exemple positif pour le mot cible.

Changement Sous-échantillonnage : L'objectif du sous-échantillonnage est de réduire les mots fréquence des mots les plus courants lors de l'entraînement.Nous faisons l'hypothèse que cela améliorerait les performances du modèle et accélérerait son entrainement. Les mots très fréquents dans un corpus, comme "le", "est", ou "à", apportent souvent peu d'informations sur le contexte d'un mot donné parce qu'ils apparaissent dans de nombreux contextes différents. En sous-échantillonnant ces mots, on réduit leur influence sur l'apprentissage des vecteurs de mots, permettant au modèle de se concentrer sur des associations plus significatives. Ainsi, on fournit au modèle des exemples positif plus informatif et des exemples négatifs moins corrélé au mot cible. De plus, on réduit le nombre total d'opérations d'entrainement et le modèle passe moins de temps à équilibre les plongements de mots qui sont déjà sur-représentés.

Échantillonnage négatif par distribution ajustée : La technique consiste à ajouter un exposant alpha à la distribution de probabilité des mots lors de l'échantillonnage négatif. Nous supposons que cela pourrait avoir plusieurs avantages sur notre modèle : L'exposant aide à "aplatir" la distribution, permettant une variété plus large de mots à être représentée, c'est-à-dire qu'on réduit l'effet de la fréquence sur la probabilité de sélection. Par la même idée que le sous-échantillonnage, les mots moins fréquents peuvent apporter plus d'informations contextuelles spécifiques et donc les favoriser par rapport aux mots très fréquents, qui pourraient ne pas être très informatifs, pourrait améliorer la qualité des plongements. Également, l'exposant alpha est un

hyper-paramètre de plus et peut permettre de peaufiner le modèle en fonction des caractéristiques spécifiques du corpus ou de la tâche.

2.4.2 Mise en œuvre :

Changement des hyper-paramètres : Ici la tâche est de capter les similarités entre les mots, ainsi augment L permet de saisir plus d'information de contexte. On passe donc de L=2 à L=3. Augmenter encore L pourrait induire du suraprentissage sur notre corpus d'entrainement.De plus, après quelques essais, nous nous rendons compte que eta=0.0005 semble plus adapté à notre problème que eta=0.1.L'hyper-paramètre d et min-count semble avoir une taille raisonnable. La valeur d'occurrence minimale des mots dans le fichier test est de 12, min-count=10 nous permet de conserver des mots rares dans l'ensemble d'échantillons d'exemple négatif et également d'accélérer l'apprentissage.

Sous-échantillonnage : Lors du prétraitement des données, on ajoute la fonction sumbsampled-text qui prend un argument, notre corpus d'entrainement sous la forme d'une liste de phrase, un paramètre t qui est le seuil de sous-échantillonnage, et l'Index des mots du corpus. On commence par créer une liste vide, appelé sub-text, puis nous parcourons chaque phrase du corpus d'entrainement et pour chaque mot m des phrases, nous calculons la probabilité de sous-échantillonner le mot, donné par $1-\sqrt{\frac{t}{Occ[Index[m]]}}$, avec $Occ[Index[m]]$ l'occurrence du mot m dans le corpus. En comparant cette probabilité avec un nombre généré entre 0 et 1, nous formons une nouvelle phrase avec les mots conservés et l'ajoutons à la liste su-text. On retourne sub-text. Finalement, nous changeons l'argument de la méthode Exemple-apprentissage de text-phrase à sub-text nos données d'entrainements soit bien tiré du texte sous-échantillonné.
La formule de conservation de mot est adapté à notre objectif : Pour un mot m fréquent, $Occ[Index[m]]$ sera élevé , donc $\sqrt{\frac{t}{Occ[Index[m]]}}$ sera faible et $1-\sqrt{\frac{t}{Occ[Index[m]]}}$, avec $Occ[Index[m]]$ sera important et donc le mot m aura plus de chance d'être sous échantillonné et inversement pour les mots mois fréquent. Le seuil t est un hyper-paramètre, En ajustant t, on peut contrôler l'agressivité du sous-échantillonnage.

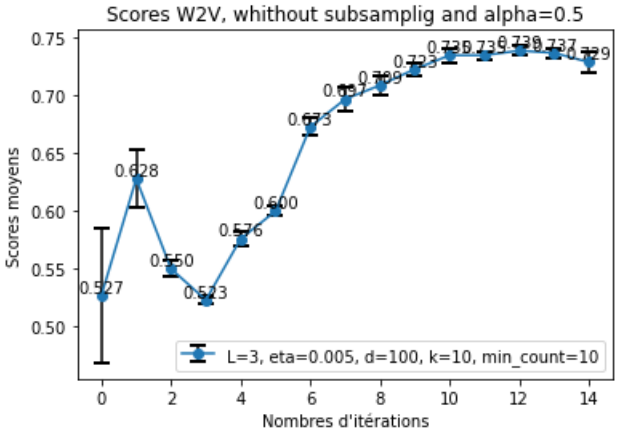
Échantillonnage négatif par distribution ajustée : Nous modifions simplement la méthode build-Occ-freq qui prend désormais en argument le dictionnaire d'occurrence Occ , min-count et alpha. Pour chaque mot m qui passe le filtre de min-count, on calcule son occurrence élevée à la puissance alpha et on met à jour le somme cumulé des occurrences des mots à la puissance alpha. $Occ[m]$ contient finalement $p_\alpha(m) = \frac{count(m)^\alpha}{\sum_{i=1}^{|V|} count(m_i)^\alpha}$.
Dans le reste du programme, on utilise Occ-freq de la même manière qu'avant.

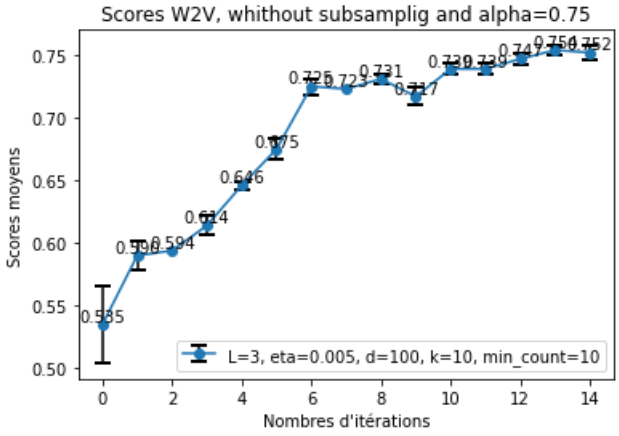
Nous avons finalement deux hyper-paramètres en plus pour notre modèle. Nous allons désormais effectuer un grid-search avec cet hyper-paramètre afin de déterminer s'ils ont effectivement un impact positif sur notre modèle, et si oui, quels couples d'hyper-paramètres sont optimaux pour notre modèle. Dans l'article [1], l'auteur explique les valeurs empiriques optimales sont $t = 10^{-4}$ et $alpha = 0.75$. Nous prendrons comme plages de valeur autour de ces valeurs pour tester notre hypothèse. $T = [10^{-2}, 10^{-3}, 10^{-4}]$ et $A = [0.5, 0.75, 1]$ respectivement pour les valeurs de t et d'alpha.

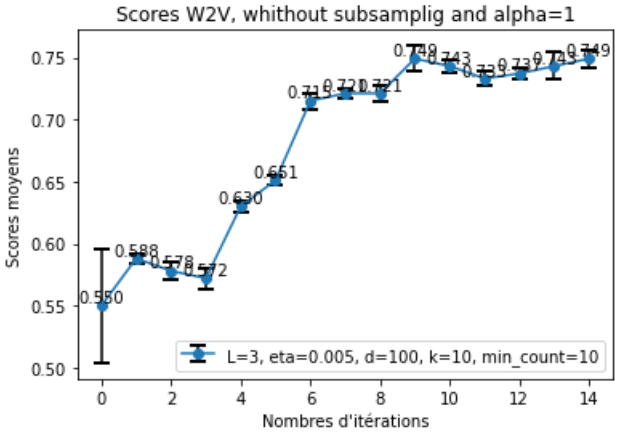
2.5 Résultats :

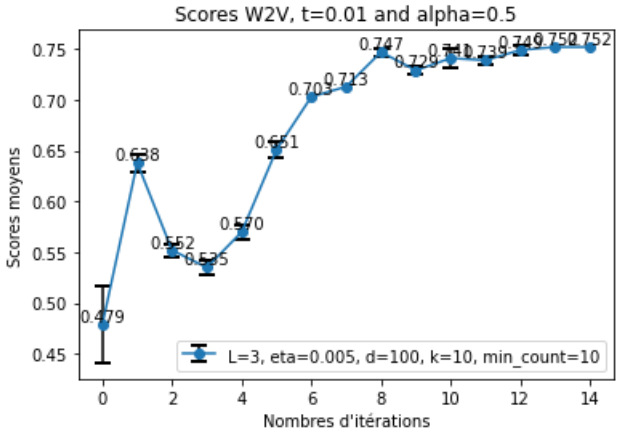
La méthode Alias réduit fortement le temps d'échantillonnage des exemples négatifs, nous passons de 6.3 min en moyenne à 1.4 min pour traiter tout le corpus.

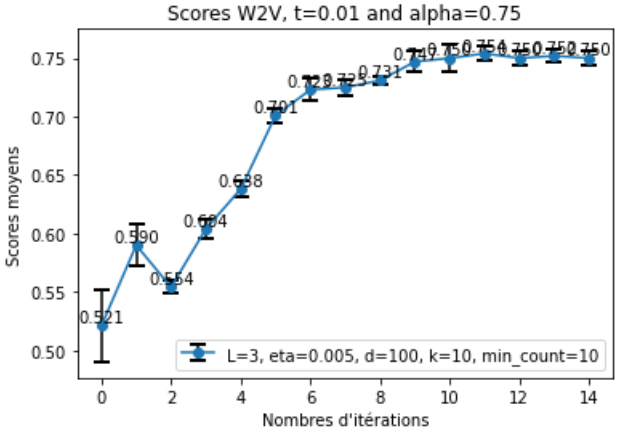
De la même manière que pour les résultats de la baseline, pour chaque couple d'hyper-paramètres, nous avons calculé les scores moyens des modèles sur 5 itérations de l'algorithme de Word2Vec entrainé sur 5-fold. Nous avons observé ces résultats sur 15 itérations de l'algorithme de mise à jour de plongements. Les scores ont été calculés pour le même jeu de test.

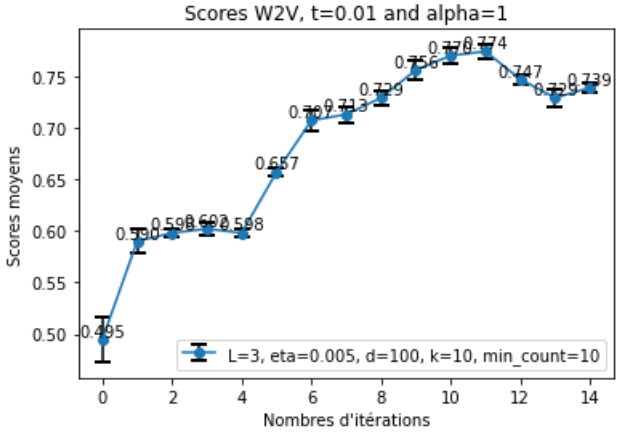

Scores W2V, whithout subsamplig and alpha=0.5
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10

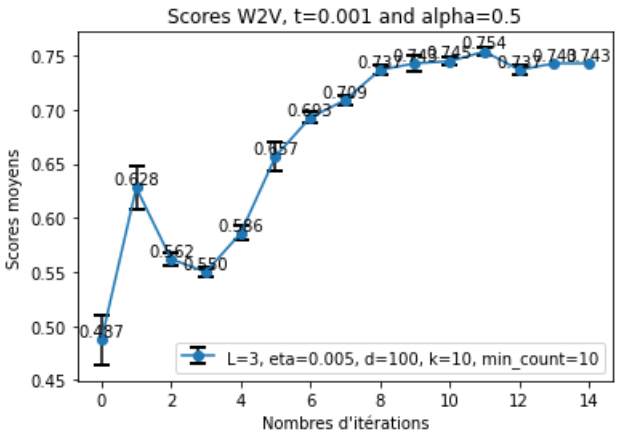

Scores W2V, whithout subsamplig and alpha=0.75
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10

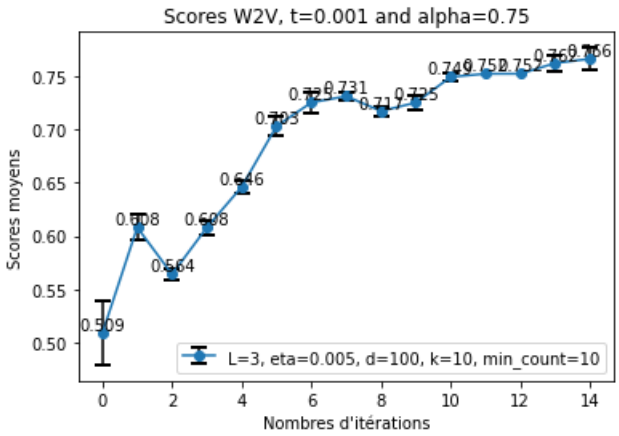

Scores W2V, whithout subsamplig and alpha=1
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10

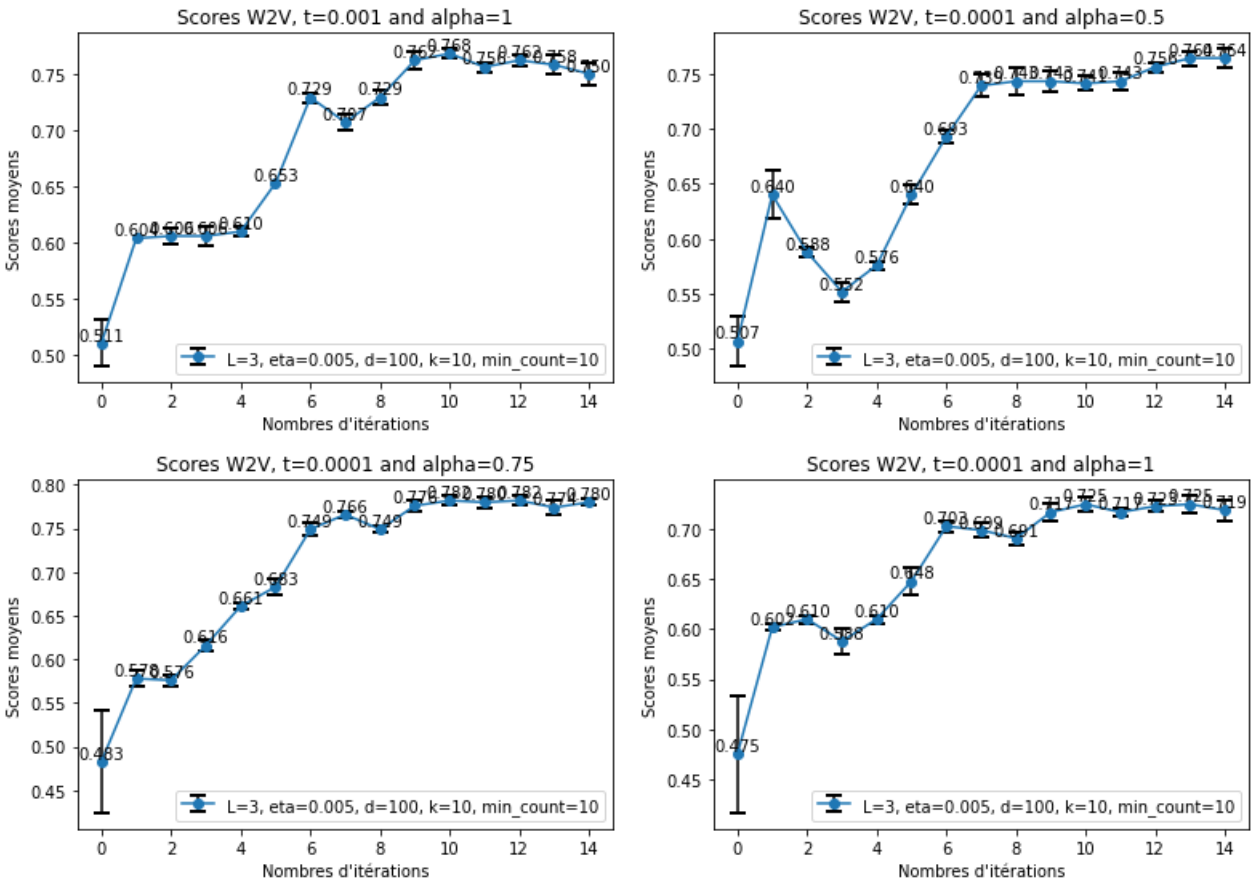

Scores W2V, t=0.01 and alpha=0.5
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10


Scores W2V, t=0.01 and alpha=0.75
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10


Scores W2V, t=0.01 and alpha=1
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10


Scores W2V, t=0.001 and alpha=0.5
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10


Scores W2V, t=0.001 and alpha=0.75
Scores moyens
Nombres d'itérations
L=3, eta=0.005, d=100, k=10, min_count=10



Pour interpréter nos résultats, nous allons nous concentrer sur trois axes : la convergence des scores, leur stabilité et leurs valeurs.

Convergence : Dans chacun des cas, les graphiques montrent que, pour différentes valeurs de t et alpha, les scores moyens fluctuent au début avant de se stabiliser ou d'augmenter progressivement avec le nombre d'itérations. Les fluctuations lors des premières itérations sont attendues. Effectivement, les plongements sont initialisés aléatoirement et l'algorithme est en recherche de relation sémantique optimal. Ces fluctuations peuvent également être dues à l'hyper-paramètre eta qui dépasse le premier minimum local de la fonction de cout. Pour $t = 10^{-4}$ et $alpha = 0.75$, on observe une bonne stabilisation des valeurs à partir de l'itération 9. De même pour $t = 10^{-3}$ et $alpha = 0.75$, les scores augmentent progressivement. À l'inverse, pour $t = 10^{-2}$ et $alpha = 1$ ou $t = 10^{-3}$ et $alpha = 0.5$, les scores ont tendances à baisser dans les dernières itérations. Pour cet ensemble de test, cet ensemble d'entrainement et ces hyper-paramètres, il semble qu'une valeur de t faible et un alpha intermédiaire favorise une bonne convergence des scores moyens.

Variance : Pour chaque couple d'hyper-paramètres, les scores moyens semble avoir une variance faible, ce qui peut suggérer que le modèle n'a pas surapris sur l'ensemble d'apprentissage. Une tendance générale est que les scores moyens ont des variances qui diminuent avec le nombre d'itérations. Cela signifie que le modèle apprend bien et converge bien. Ces tendances sembelnt propres au modèle Word2Vec sur nos données et les paramètres t et alpha ne semblent pas influer sur la variance.

Valeur des Scores moyens : D'une manière générale, la valeur des scores moyens augmentent : l'algorithme apprend.L'augmentation progressive des scores indique que le modèle a mieux capturé les relations sémantiques entre les mots dans les triplets. Il est flagrant que pour $t = 10^{-4}$ et $alpha = 0.75$, les scores moyens sont les plus élevés à la fin des itérations. les hyper-paramètres $t = 10^{-4}$ et $alpha = 0.5$ donne également des scores satisfaisants. À l'inverse, pour $t = 10^{-4}$ et $alpha = 1$,$t = 10^{-3}$ et $alpha = 1$ ou bien $t = 10^{-3}$ et $alpha = 0.5$, les scores sont légèrement plus faibles. Les valeurs d'hyper-paramètre $t = 10^{-4}$ et $alpha = 0.75$ semblent particulièrement adaptés à notre problème. En général, un alpha intermédiaire donnera des meilleurs résultats en termes de valeurs.

3 Conclusions et perspectives

Notre objectif dans ce TP était d'implémenter un algorithme qui puisse construire des représentations de mots capables de correctement capter la similarité des mots, c'est-à-dire que les mots sémantiquement similaires sont placés à proximité les uns des autres dans l'espace vectoriel. Nous avons construit dans un premier temps une baseline qui ne donnait pas de résultat satisfaisant sur l'ensemble de test. Nous avons donc construit un nouveau modèle en y ajoutant le sous-échantillonnage et une distribution ajustée pour l'échantillonnage des exemples négatifs. Notre hypothèse est que pour les bons hyper-paramètres, ces ajouts nous permettrons d'avoir de meilleurs résultats.
Notre expérimentation nous permet de dire que pour un sous échantillonnage avec $t = 10^{-4}$ et une distribution ajusté avec $alpha = 0.75$, les résultats convergeaient mieux et étaient plus hauts. L'hypothèse est vérifiée.

Cependant, ces résultats sont valables uniquement pour les valeurs d'hyper-paramètres L=3, eta=0.005, d=100, k=10, min-count=10, sur cet ensemble de données et cet ensemble de test. Les résultats peuvent être sensibles la taille limitée de l'ensemble d'entrainement et de l'ensemble de test. Bien que les variances soient faibles, il est toujours probable que le modèle surapprenne. Il serait intéressant de procéder à des tests supplémentaires et d'augmenter la taille du corpus pour confirmer les tendances observées.

Également, pour améliorer les résultats, nous pourrions étudier en profondeur les autres hyper-paramètres comme L et d, et leur impact sur le surapprentissage. Nous pourrions étudier d'autre méthode d'optimisations que la SGD ou bien ajuster eta en fonction d'itérations.

Références

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013. Introduction of the Word2Vec (W2V) model.

[2] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM transaction on Mathematical software*, Volume 3(Numéro 3) :253–256, 1977.