TP2 - Plongements de mots statiques

IAAA - TLNL

15 septembre 2023

1 Objectif

L'objectif de ce projet est de programmer un modèle de plongement de mots statique inspiré du modèle word2vec, plus spécifiquement de la méthode *skip-gram with negative sampling*.

Le plongement d'un mot m est un vecteur de réels $\mathbf{m}$ qui représente de manière abstraite les contextes possibles d'occurrence de m. Deux mots m et m' véhiculant le même sens, tel que les mots *vélo* et *bicyclette*, peuvent apparaître dans les même contexte. Leurs plongements doivent alors être très proches.

2 Le classifieur

Le cœur du modèle de calcul des plongements est constitué d'un classifieur binaire C, prenant en entrée deux mots c et m. m est appelé le mot cible et c un mot du contexte d'occurrence de m. Le classifieur calcule la probabilité $P(+|m, c)$, qui est la probabilité que c appartienne au contexte de m. Cette probabilité est d'autant plus élevée que les plongements $\mathbf{m}$ et $\mathbf{c}$ sont proches. Cette proximité peut être évaluée en calculant le produit scalaire de ces deux vecteurs : $\mathbf{m} \cdot \mathbf{c}$.

On transforme ce produit scalaire en probabilité à l'aide de la fonction sigmoïde, définie de la façon suivante :

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

La probabilité $P(+|m, c)$ se calcule donc ainsi :

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

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

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

Dans le modèle décrit ci-dessus, les mots possèdent deux plongements distincts : un plongement en tant que mot cible (le vecteur **m**) et un plongement en tant que mot de contexte (le vecteur **c**). Ces vecteurs sont stockés dans deux matrices distinctes, notées **M** et **C**. A l'issue de l'apprentissage, ce sont les vecteurs contenus dans la matrice **M** qui sont conservés en tant qu'embeddings pour chaque mot m. Les valeurs dans la matrice **C** servent uniquement à apprendre les embeddings dans **M**, mais ils sont jetés à l'issue du processus.

2.1 Données d'apprentissage

Le calcul des plongements de mots est réalisé à l'aide d'exemples positifs et négatifs. Pour générer de tels exemples, il suffit de parcourir un texte. Pour tout mot m, on sélectionne des mots c_{pos} apparaîssant dans une fenêtre de taille $2L + 1$ centrée sur le mot m (les L mots précédant m et les L mots suivant m). Chacun de ces mots, notés c_{pos} permet de construire un exemple positif : $(+, m, c_{pos})$. Pour construire un exemple négatif, il suffit de prendre des mots au hasard dans le lexique. Pour chaque exemple positif, on construit k exemples négatifs : $(-, m, c_{neg_1}), \ldots (-, m, c_{neg_k})$.

Les données d'apprentissage sont donc constituées d'exemples positifs et d'exemples négatifs (k fois plus d'exemples négatifs que d'exemples positifs). Un exemple positif et les k exemples négatifs lui correspondant constituent un *mini-batch* qui est l'unité de mise à jour du classifieur.

2.2 Fonction de perte

Pour chaque mini-batch de l'ensemble d'apprentissage, on souhaite maximiser la probabilité de l'exemple positif : $P(+|m, c_{pos})$ et minimiser la probabilité des exemples négatifs : $P(+|m, c_{neg_i})$. Cela revient à maximiser l'expression suivante :

$$P(+, m, c_{pos}) \prod_{i=1}^{k} P(-, m, c_{neg_i})$$

Pour des raisons de simplification du calcul du gradient, on préfère minimiser l'opposé du logarithme d'un tel produit, c'est cette dernière expression qui constitue la fonction de perte du classifieur :

$$\begin{aligned} L &= -\log \left[P(+, m, c_{pos}) \prod_{i=1}^{k} P(-, m, c_{neg_i}) \right] \\ &= -\left[\log P(+, m, c_{pos}) + \sum_{i=1}^{k} \log P(-, m, c_{neg_i}) \right] \\ &= -\left[\log \sigma(\mathbf{m} \cdot \mathbf{c}_{pos}) + \sum_{i=1}^{k} \log \sigma(-\mathbf{m} \cdot \mathbf{c}_{neg_i}) \right] \end{aligned} \tag{1}$$

3 Gradient

L'introduction de la fonction logarithme dans la fonction de perte permet d'obtenir une expression simple du gradient de cette dernière, comme nous l'avons vu en TD.

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{c}_{pos}} &= [\sigma(\mathbf{m}\cdot\mathbf{c}_{pos})-1]\mathbf{m}\\ \frac{\partial L}{\partial \mathbf{c}_{neg}} &= [\sigma(\mathbf{m}\cdot\mathbf{c}_{neg})]\mathbf{m}\\ \frac{\partial L}{\partial \mathbf{m}} &= [\sigma(\mathbf{m}\cdot\mathbf{c}_{pos})-1]\mathbf{c}_{pos}+\sum_{i=1}^{k}[\sigma(\mathbf{m}\cdot\mathbf{c}_{neg_i})]\mathbf{c}_{neg_i}\end{aligned} \tag{2}$$

4 Mise à jour des plongements par descente du gradient

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

$$\begin{aligned}\mathbf{c}_{pos}^{t+1} &= \mathbf{c}_{pos}^{t}-\eta[\sigma(\mathbf{m}\cdot\mathbf{c}_{pos}^{t})-1]\mathbf{m}\\ \mathbf{c}_{neg}^{t+1} &= \mathbf{c}_{neg}^{t}-\eta[\sigma(\mathbf{m}\cdot\mathbf{c}_{neg}^{t})]\mathbf{m}\\ \mathbf{m}^{t+1} &= \mathbf{m}^{t}-\eta([\sigma(\mathbf{m}^{t}\cdot\mathbf{c}_{pos})-1]\mathbf{c}_{pos}+\sum_{i=1}^{k}[\sigma(\mathbf{m}^{t}\cdot\mathbf{c}_{neg_i})]\mathbf{c}_{neg_i})\end{aligned} \tag{3}$$

5 Fichier des plongements

A l'issue de l'apprentissage des plongements, le programme qui les a calculé produit un fichier textuel qui comporte un plongement par ligne, comme dans l'exemple ci-dessous :

```
100 50
vélo 0.45 0.34 ... 0.12
bicyclette 0.46 0.29 ... 0.09
...
chat 0.12 0.84 ... 0.001
```

La première ligne du fichier comporte deux nombres entiers, le premier indique le nombre de plongements contenus dans le fichier et le second, la dimension d de chaque plongement. Les lignes suivantes ont toutes le même format : un mot représenté sous la forme d'une chaîne de caractères, suivi de son plongement (suite de d nombres réels), avec des espaces pour séparer les mots et les valeurs.

6 Évaluation

Afin d'évaluer la qualité des embeddings produits à l'issue du processus d'apprentissage, nous allons utiliser un ensemble de triplets de mots : (m, m_+, m_-). Ces triplets ont été produits manuellement de sorte que le sens de m_+ soit plus proche m que l'est celui de m_-, comme dans l'exemple *(vélo, bicyclette, chat)*.

Etant donné les plongements $\mathbf{m}$, $\mathbf{m}_+$ et $\mathbf{m}_-$, on aimerait que :

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

où $sim(\mathbf{a}, \mathbf{b})$ est une mesure de similarité qui est d'autant plus élevée que les deux vecteurs $\mathbf{a}$ et $\mathbf{b}$ sont proches. On utilisera le cosinus de l'angle de deux vecteurs comme mesure de similarité :

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

Le format du fichier d'évaluation est très simple, chaque ligne comporte trois mots, qui sont respectivement les mots m, m_+ et m_-, comme dans les exemples suivants :

```
vélo bicyclette chat
mangue goyave balais
...
```

Un jeu d'évaluation vous est fourni avec les données (le fichier `Le_comte_de_Monte_Cristo.100.sim`).

7 Ce qu'il faut faire

1. Réaliser le programme `w2v.py` qui prend en entrée un texte segmenté, ainsi que les valeurs des paramètres suivants :
   - `n` dimension des embeddings
   - `L` taille des contextes gauche et droit (la largeur de la fenêtre vaut `2*L + 1`)
   - `k` nombre de contextes négatifs pour un contexte positif
   - `eta` taux d'apprentissage
   - `e` nombre d'itérations
   - `minc` nombre minimal d'occurrence d'un mot pour que son plongement soit produit

   `w2v.py` produit un fichier d'embeddings au format décrit dans la section 5.
2. Réaliser le programme `eval_w2v.py` qui prend en entrée un fichier de plongements (au format décrit dans la section 5) ainsi qu'un fichier d'évaluation au format décrit dans la section 6. Le programme calcule le rapport du nombre de triplets qui passent le test

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

sur le nombre de triplets.
A titre de comparaison, des plongements appris sur le corpus `Le_comte_de_Monte_Cristo.tok` avec le jeu de paramètres suivant : `n = 100, L = 2, k = 10, eta = 0.1, iterNb = 5, minCount = 5` permet d'obtenir un taux de réussite moyen de $0,532\%$ et un écart type de $0,039$ sur ce fichier d'évaluation. Il s'agit de la moyenne sur 10 expériences (les résultats varient du fait de l'initialisation aléatoire des matrices $\mathbf{C}$ et $\mathbf{M}$).

8 Quelques pistes à creuser

8.1 Amélioration des performances

Les performances affichées par l'implémentation décrite ci-dessus sont assez décevantes (53% de taux de réussite moyen). L'implémentation officielle de Word2vec atteint, sur les mêmes données avec les mêmes hyperparamètres un taux de réussite de 80%. Le but de ce projet est d'améliorer les performances obtenues, en implémentant en particulier une méthode d'échantillonnage des exemples négatifs et d'autres aspects que vous trouverez dans l'article original (*Distributed representations of words and phrases and their compositionality*) ou dans les nombreuses description du modèle que l'on trouve sur internet.

8.2 Analogies

Une légende dit qu'il est possible d'utiliser des plongements pour retrouver des analogies entre mots. Il serait donc possible de prendre les plongements des mots `roi`, `femme` et `homme` et en faisant le calcul `roi - homme + femme`, retrouver le plongement de ...`reine`. Le but de ce projet est de voir si cela est vrai. Pour cela, vous pourrez utiliser des plongements qui ont été calculés sur un nombre important de données (par exemple le modèle 43 de l'adresse suivante `http://vectors.nlpl.eu/repository`). Vous ferez l'évaluation sur un ensemble de 100 analogies de natures différentes, tel que (pommier, pomme, poirier, poire), (France, Paris, Somalie, Mogadiscio), (mangerai, mangeront, mangeai, mangèrent), (truie, cochon, chienne, chien) ...que vous aurez construit vous même.

Bien entendu, à l'issue du calcul évoqué ci-dessus, on ne tombera pas directement sur le plongement de `reine`, il faudra retrouver le mot du lexique dont le plongement est le plus proche de celui obtenu par le calcul. Pour cela il faudra implémenter un algorithme de recherche efficace du mot le plus proche.

Etant donné un ensemble de plongements $V = \{e_1, \ldots, e_N\}$ et un plongement particulier $x \notin V$, on aimerait trouver l'élément de V le plus proche de x, au sens de la distance euclidienne :

$$\hat{e} = \arg\min_{e \in V} d(e, x)$$

La méthode naïve consiste à parcourir l'ensemble V et à calculer la distance de x à chacun des éléments de V. Cette méthode est coûteuse dans la mesure

où le cardinal de V peut être élevé, ainsi que la dimension des plongements. Plusieurs méthodes existent pour résoudre ce problème de manière efficace, en particulier les arbres k-d. Choisissez un algorithme efficace parmi ceux proposés dans la littérature et implémentez le.

8.3 Fasttext

La méthode de construction de plongements implémentée ne permet pas de produire un plongement pour un mot qui n'a pas été vu dans le corpus d'apprentissage. Lorsque deux mots sont morphologiquement liés, comme `mangions` et `mangerions`, on pourrait essayer de construire le plongement de l'un à partir de l'autre. L'algorithme Fasttext propose une méthode pour faire cela en décomposant les mots en séquences de n-lettres, `mangerions`, par exemple, deviendrait `<ma man ang nge ger eri rio ion ons ns>`, si l'on choisit une décomposition en trigrammes. Pour chacun des trigrammes, un plongement est calculé et le plongement d'un mot est obtenu en faisant la somme des plongements des trigrammes qui le composent.

Après avoir lu l'article de fasttext *Enriching Word Vectors with Subword Information*, implémentez la méthode proposée et proposez une manière intelligente de l'évaluer.