Le Jeu de GEENSON
Une introduction aux *word embeddings*

Vincent Segonne
vs_teaching@hotmail.com

1 Contexte

Les plongements lexicaux, ou *word embeddings*, sont des représentations de mots sous la forme de vecteurs de nombres réels. Intuitivement, chaque mot correspond à un point dans un espace à d dimensions, où d est typiquement de l'ordre de 50 à 1000. On parle alors d'espace de mots. On donne en Figure 1 une représentation schématique d'un tel espace à deux dimensions. Précisons dès à présent que ces vecteurs ne sont pas créés à la main, mais estimés automatiquement à partir de gros corpus et à l'aide de méthodes statistiques (Mikolov et al., 2013a; Bengio et al., 2003; Turney and Pantel, 2010).

On peut voir ces vecteurs de mots comme des représentations complémentaires aux représentations classiques par chaînes de caractères. Habituellement, pour représenter *pomme* et *fruit*, on utilise des constantes symboliques (par exemple des chaînes de caractères comme `"pomme"` et `"fruit"`). Lorsqu'on utilise ces représentations discrètes, la notion de similarité entre mots n'est pas naturelle. Du point de vue du sens, *pomme* est plus similaire à *fruit* qu'à *mardi*. Les représentations symboliques ne montrent pas *a priori* cette similarité.

Au contraire, dans un espace de mots, on peut utiliser des outils mathématiques, notamment géométriques, pour mesurer des distances ou des similarités entre les mots. Dans un bon espace de mots, on s'attend à ce que $\overrightarrow{v_{pomme}}$ (la représentation de *pomme* sous la forme d'un vecteur) soit très proche de $\overrightarrow{v_{poire}}$ et de $\overrightarrow{v_{fruit}}$, mais éloigné de $\overrightarrow{v_{mardi}}$. En quelque sorte, ces modèles permettent de regrouper des mots qui ont quelque chose en commun.

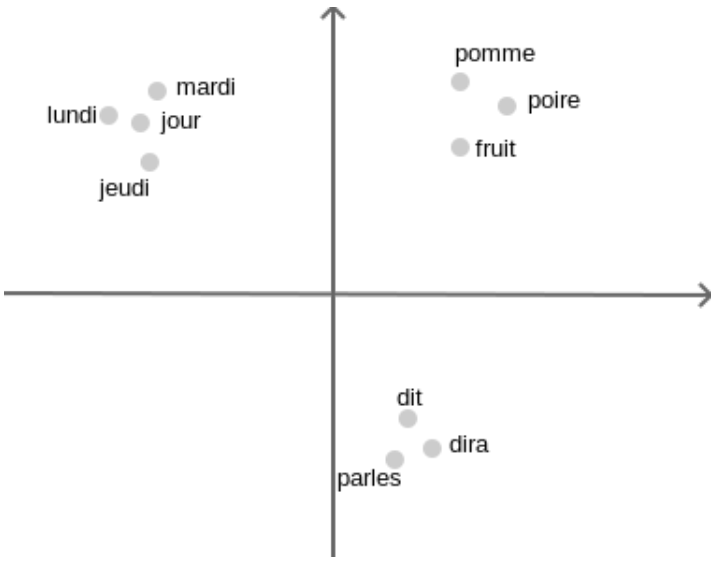

FIGURE 1 – Représentation schématique d'un espace de plongements lexicaux.

Récemment, Mikolov et al. (2013a) ont proposé une méthode efficace pour apprendre des plongements lexicaux, et donné une implémentation de cette méthode appelée `word2vec`.[1] Les plongements estimés à l'aide de ce modèle manifestent certaines régularités sémantiques et syntaxiques (Mikolov et al., 2013b). Par exemple, on observe certaines relations ou analogies telles que :

$$\overrightarrow{v_{king}} - \overrightarrow{v_{man}} \approx \overrightarrow{v_{queen}} - \overrightarrow{v_{woman}}$$

1. `https://code.google.com/p/word2vec/` La page du projet `word2vec` contient des liens vers les articles référencés ici.

À partir de cette observation, on peut formaliser une tâche qui vise à donner une réponse à la question suivante : *Qu'est-ce qui est à A ce que B est à C ?* Par exemple : *Qu'est-ce qui est à* man *ce que* queen *est à* woman *?* Une méthode simple pour tenter de résoudre ce problème est de chercher quel est le vecteur de mot le plus proche de $\overrightarrow{v_{queen}} - \overrightarrow{v_{woman}} + \overrightarrow{v_{man}}$, c'est-à-dire :

$$\hat{x} = \underset{\overrightarrow{v_x}, x \in Vocabulaire}{\operatorname{argmax}} \text{Similarity}(\overrightarrow{v_{queen}} - \overrightarrow{v_{woman}} + \overrightarrow{v_{man}}, \overrightarrow{v_x})$$

Pour calculer la similarité entre vecteurs, la fonction la plus utilisée est la similarité cosinus, définie comme suit :

$$\cos(\overrightarrow{a}, \overrightarrow{b}) = \frac{\overrightarrow{a} \cdot \overrightarrow{b}}{||\overrightarrow{a}|| \cdot ||\overrightarrow{b}||} = \frac{\sum_{i=1}^{d} a_i b_i}{\sqrt{\sum_{i=1}^{d} a_i^2} \sqrt{\sum_{i=1}^{d} b_i^2}}$$

2 Le Projet

Le projet consiste à réaliser un jeu qui sert de support à l'utilisation de word embeddings. Le principe du jeu est simple, les joueurs (nombre non limité) lancent un dé à tour de rôle et avancent sur un plateau. Le but du jeu est d'arriver en premier à la case finale. À chaque tour, le système propose au joueur un mot *m* à lui faire deviner. Le joueur doit alors donner un certain nombre d'indices et le sytème en fonction de ceux-ci retournera les *k*-réponses qui lui semblent les plus pertinentes. La notion de pertinence des réponses sera rendue par la similarité des vecteurs de mots, le système devra retourner les *k*-mots les plus similaires aux indices proposés. Si le mot *m* à deviner est dans les *k*-réponses, alors le joueur à réussi son tour et relance le dé. Le projet se divise en deux tâches spécifiées dans les prochaines sections.

2.1 L'utilitaire de vecteurs de mots

Une première partie du projet consiste à créer un utilitaire qui manipule les vecteurs de mots. Celui-ci sera utilisé par le jeu pour permettre au système d'élaborer des réponses. Il devra être capable de représenter et réaliser les opérations vectorielles de base : addition, soustraction, moyenne, multiplication par un scalaire, norme, distance et similarité entre vecteurs. La similarité cosinus devra obligatoirement être implémentée, d'autres mesures pourront être implémentées optionnellement. Un inventaire de vecteurs de mots pré-entrainés vous sera fourni.

2.2 Le jeu

La deuxième partie consistera à implémenter le jeu. Ce sera ici l'occasion de montrer que vous avez compris les notions fondamentales de POO, (encapsulation des données, héritage etc..). Il est demandé une version 'de base' qui implémente au moins les éléments suivants :

- un plateau de jeu
- les cases du plateau de jeu
- les joueurs
- un dé
- la gestion du jeu (tours, cas de victoire etc..)

De plus, votre jeu devra intégrer ces paramètres de bases :

- Au moins deux joueurs
- Deux types de cases autres que les cases normales : « relancer le dé » et « reculer de trois cases »
- Deux types de dé : un dé normal et un dé avec une face piège « restez où vous êtes »
- Les joueurs devront donner trois indices et auront 3 essais pour faire deviner le mots
- Le système fera la moyenne ou la somme des vecteurs des indices donnés et retournera les 10 premiers mots les plus similaires à ce vecteur.

2.3 Exemple de tour de jeu

Le jeu commence, c'est au `joueur1` de jouer, il est sur la case initiale. Il lance le dé et fait un 3. Il avance de 3 cases. Il tombe sur une case 'relancer le dé', il relance et fait un 6. Il avance de 6 cases. Il tombe sur une case normale. Le système choisi aléatoirement un mot à lui faire deviner : ordinateur. Le joueur tape les indices en lignes de commande : `calcul portable écran`. Le système calcule une similarité entre les vecteurs des indices donnés et les vecteurs de mots de son vocabulaire et retourne les 10 mots les plus similaires :

```
Indices : écran / portable / calcul

ordinateur, 0.07550823729684752
enregistreur, 0.1040160614910286
téléviseur, 0.11193168677630472
tactile, 0.11558805775286252
caméscope, 0.12014727058902797
magnétoscope, 0.12227530149240717
processeur, 0.1297779586556489
baladeur, 0.13152212201465963
boitier, 0.13778913465843334
microprocesseur, 0.139831253471273
```

Ordinateur est bien dans la liste proposées par le système. Le joueur1 rejoue. S'il n'avait pas réussi, il aurait eu deux autres essais.

2.4 Programme principal

Le code source ne doit contenir aucun nom de fichier en dur (hardcodé). L'invocation de l'exécutable donnera en argument du programme le(s) fichier(s) et informations nécessaire(s) à son exécution. Par exemple, si votre programme principal est dans une classe `Main` :

```
java Main --w2v ../vecs50 -nbJoueurs 2 --nbTry 3 --k 5 --de_magique
```

Ici le programme lance le jeu avec les options suivantes :

— `w2v` : indique le chemin vers le fichier de vecteurs de mots
— `nbJoueurs` : indique le nombre de joueurs
— `nbTry` : indique le nombre d'essais pour chaque tour
— `k` : le nombre de k-réponses à retourner par le système
— `de_magique` : le jeu fonctionnera avec un dé magique

2.5 Documentation

Vous rendrez la documentation de votre programme au format html (générée avec `javadoc`). Il est également demandé d'écrire un readme au format markdown (ou éventuellement `txt`) qui précise comment utiliser votre programme (ligne de commande à utiliser, commande pour compiler, syntaxe des requêtes).

Vous pouvez également utiliser le readme pour décrire l'architecture générale du programme, les choix d'implémentation, ou toute information que vous jugez utile / intéressante.

3 Format des données

Des représentations vectorielles pour les mots du français vous seront fournies dans un fichier nommé `vecs50` (sur Moodle). La première ligne du fichier contient deux entiers séparés par des espaces. Le premier entier N est le nombre de représentations contenues dans le fichier Le deuxième entier d est la dimension de chaque représentation. Chacune des N lignes suivantes contiendra un mot, puis une espace, et enfin d nombres réels séparés par des espaces. Ces nombres sont la représentation vectorielle du mot. Le programme que vous réaliserez devra être capable de lire ces données, et de les stocker dans les structures que vous jugerez appropriées.

4 Évaluation

Les critères de notations généraux comprendront les éléments suivants :
— bonnes pratiques de java (conventions de nommage, principes de l'orienté-objet, etc) ;
— cohérence de l'architecture de votre programme (organisation en classes, interfaces, classes abstraites) ;
— gestion des erreurs (est-ce que le programme plante si un utilisateur entre un mot inconnu ou une expression mal formée ?) ;
— facilité d'utilisation (est-ce qu'il suffit de lire le readme pour se servir du programme correctement ?) ;
— efficacité du programme, choix des structures de données et des algorithmes ;

5 Consignes pour le rendu du projet

— Le projet est mis en ligne le 05 mars, il est à rendre au plus tard le **17 mai 2019 à 23h59** (heure de Paris) ;
— Le projet est à faire en binôme ;
— L'ensemble des fichiers sources documentés devra être rendu dans une archive (tar.gz ou zip) qui portera le nom `projetNom1Nom2.tar.gz` ;
— Les données (`vecs50`) ne doivent pas être incluses dans l'archive ;
— Il est conseillé d'inclure dans l'archive un fichier texte contenant un exemple d'exécution (log).

6 Suggestion d'améliorations

On peut imaginer toutes sortes d'amélioration pour le jeu ou la manipulation des vecteurs. Pour la manipulation des vecteurs :
— Implémenter différentes mesure de similarité des vecteurs (distance euclidienne etc..)
— Faire un langage de gestion des indices qui ne code plus en dur mais permet de faire des opérations sur les vecteurs en ligne de commande : portable - mean(processeur, calcul)

Pour le jeu :
— Faire un plateau original avec des cases spéciales
— Mettre des contraintes sur les indices
— Règles spéciales etc..

References

Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3 :1137–1155.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751. The Association for Computational Linguistics.

Peter D. Turney and Patrick Pantel. 2010. From frequency to meaning : Vector space models of semantics. *CoRR*, abs/1003.1141.