Word Embedding
Les mots et le Machine Learning

Culture Sciences de l'Ingénieur

Édité le 07/11/2022

école normale supérieure paris—saclay

Saša RADOSAVLJEVIC - Solal NATHAN

Cette ressource est issue d'une co-publication avec le numéro 110 de La Revue 3EI d'octobre 2022 et fait partie du « Dossier Intelligence Artificielle » [6] sur Culture Sciences de l'Ingénieur. *Saša Radosavljevic est élève du DER Nikola Tesla de l'ENS Paris-Saclay en M2 SETI, Solal Nathan est élève du Département d'Enseignement et de Recherche Nikola Tesla de l'ENS Paris-Saclay et doctorant au Laboratoire Interdisciplinaire des Sciences du Numérique (LISN).*

Cette ressource introduit des méthodes de représentation des mots dans l'informatique au travers d'un premier exemple simple utilisant l'occurrence des mots dans un corpus de textes puis d'un exemple utilisant Word2Vec qui regroupe un ensemble de réseaux de neurones pour l'apprentissage de vectorisation des mots afin d'opérer sur ceux-ci. Une attention particulière sera faite quant aux biais introduits par rapport aux données d'apprentissage.

1 – Introduction

Il existe de nombreux moyen de représenter les données. Le Machine Learning s'intéresse notamment à exploiter et analyser toutes les données possibles. Les données tabulaires sont les plus simples à représenter et à analyser informatiquement. Les images peuvent être traitées presque directement par réseau de neurones [1]. Certaines données sont cependant plus difficiles à traiter : les données temporelles [2] ou le texte par exemple. Les données textuelles ont pourtant une importance majeure car elles sont disponibles en très grandes quantités et sont une manière très compacte de transmettre de l'information, permettant de créer des bases de données beaucoup plus denses. Nous avons également accès à des données historiques beaucoup plus anciennes et variées que pour d'autres médias comme la vidéo.

Pour traiter les données textuelles il nous faut donc une représentation mathématique plus simple. Le word embedding (ou plongement lexical en français) est un ensemble de techniques permettant de transformer un mot ou ensemble de mot sous forme de vecteur.

Il s'agit en réalité d'une forme intelligente de réduction de dimensionnalité. En effet, gérer des données de grandes dimensions est très difficile à cause du « fléau de la dimension » (ou curse of dimensionality en anglais).

Comment créer et définir ces vecteurs de nombres réels à partir de simples mots ?

Une façon de faire est la représentation en fonction du contexte. Effectivement, les mots chat et chien seront souvent utilisés dans le même contexte, c'est-à-dire les mots qui entoure le mot cible. On parlera souvent d'un chat et d'un chien comme d'un animal de compagnie, une boule de poils. Placés dans la phrase : « Tu as beaucoup de chance d'avoir ce ____ comme animal de compagnie. » les deux mots pouvant prendre place à l'emplacement vide, ils seront alors tout deux représentés par des valeurs proches. Quid des autres animaux ? Un perroquet ne ressemble pas du tout à un chat. C'est tout à fait viable pour d'autres animaux de compagnie. Cependant, la fréquence d'apparition de ceux-ci dans ce contexte est moins importante, diminuant la ressemblance des

mots dans cette représentation vectorielle. C'est ainsi qu'avec un corpus de texte et de contexte très étoffé que les représentations seront plus ou moins précises après l'apprentissage du modèle.

Pourquoi est-ce important ?

Imaginez-vous opérer avec des nombres. Nous avons tous appris à l'école à faire des additions et des soustractions pour des tâches quotidiennes. Plus tard, vous avez appris à modifier l'amplitude d'un signal ou à soustraire des niveaux de couleurs à une image. Essayez de faire pareil avec des mots et cela devient tout de suite plus compliqué, surtout informatiquement. Avec une représentation vectorielle bien construite, il devient possible d'effectuer ces opérations. Par exemple, $\text{papa} - \text{homme} + \text{femme} = \text{maman}$, semble plutôt logique.

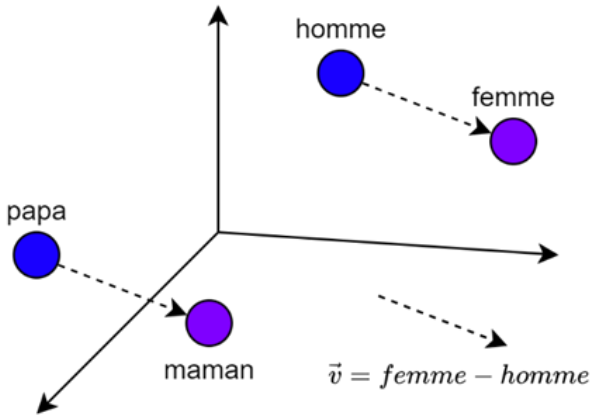

Figure 1 : Représentation vectorielle simplifiée

Cela reste cependant très complexe à mettre en place dans un algorithme. La capacité d'apprentissage automatique de la machine à l'aide de réseaux de neurones et la quantité importante de données présentes en ligne permet de faciliter cette représentation vectorielle.

2 – Différentes méthodes

Afin de réaliser le word embedding, différentes méthodes et approches existent. Avant de voir ces différentes méthodes, il faut parler du Bag of words (BOW) ou Sac de mots. La représentation par BOW permet de décrire un document, dans notre cas un texte. Le modèle BOW vient compter l'occurrence de mots dans un texte et ainsi donner son histogramme. De cette manière, le résultat de la vectorisation sera impacté par l'histogramme des mots présents dans le corpus d'apprentissage.

Question 1 :

À l'aide d'internet, décrivez une méthode de prédiction de mot utilisant l'histogramme d'occurrence.

Réponse 1 [3][4] :

Dans le modèle Continuous Bag of Words (CBOW), l'objectif de l'apprentissage est de combiner la représentation vectorielle des mots du contexte (mots entourant le mot à prédire) afin de prédire le mot manquant.

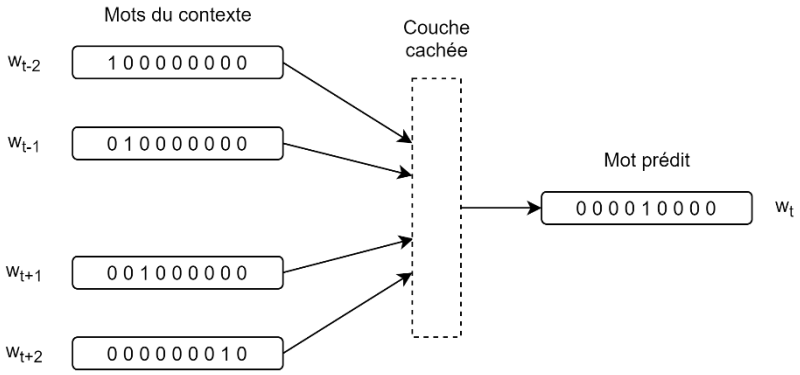

Figure 2 : Modèle CBOW

Dans le modèle Skip-gram à l'inverse, on va chercher à partir d'un mot observé, les mots de son contexte.

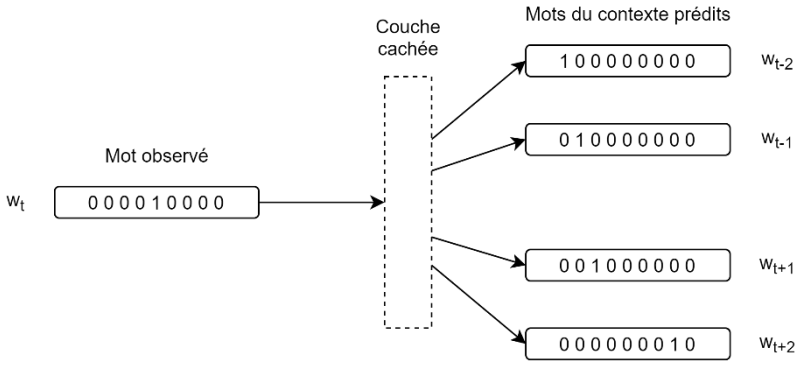

Figure 3 : Modèle Skip-gram

3 – Première approche

Commençons par prendre plusieurs phrases (1) qui vont composer notre corpus et construire notre dictionnaire de mots.

```
document_1 = "le chat mange la souris"
document_2 = "le chien regarde le canard"
document_3 = "le canard regarde le chat"
corpus = (document_1, document_2, document_3)
```

3.1 - Création du dictionnaire

On vient créer un dictionnaire de mots uniques qui va nous permettre de compter l'occurrence des mots.

```
# construction du vocabulaire
vocabulary = []
for d in corpus:
   for w in d.split(" "):
   if w not in vocabulary:
      vocabulary.append(w)
```

3.2 - Calcul de l'histogramme

La manière la plus simple de visualiser l'occurrence ou la fréquence d'informations est l'histogramme. Pour ce faire, nous utiliserons les modules **matplotlib.pyplot** et **pandas :**

```
import pandas as pd
import matplotlib.pyplot as plt
…
# initialisation de l'histogramme
freq = dict()
for v in vocabulary:
   freq[v] = 0
```

Question 2 :

Compléter le code précédent pour afficher l'histogramme des mots du corpus.

Réponse 2 :

```
# comptage des occurrences
for d in corpus:
for w in d.split(" "):
   freq[w] += 1

print(freq)
df = pd.DataFrame({'freq':freq.values()}, index=freq.keys())
ax = df.plot.bar(rot=0)
plt.show()
```

Qui nous donne le résultat suivant :

{'le': 5, 'chat': 2, 'mange': 1, 'la': 1, 'souris': 1, 'chien': 1, 'regarde': 2, 'canard': 2}

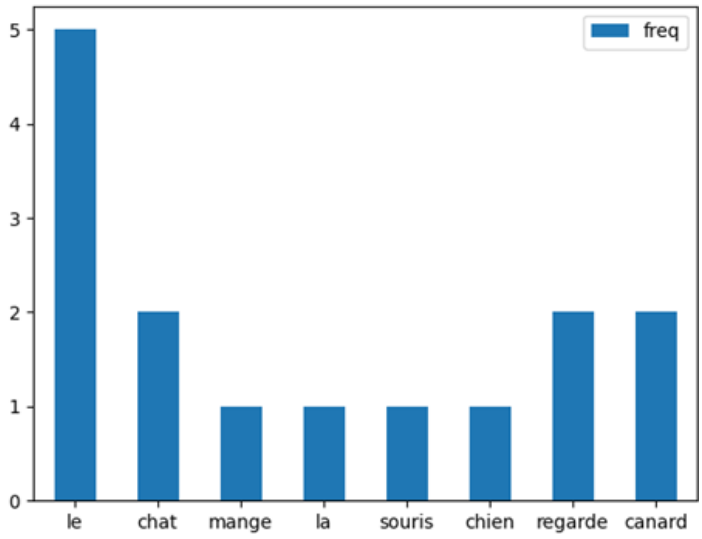

Figure 4 : Histogramme du corpus, source {1}

Plusieurs problèmes se posent avec une telle représentation. D'une part, on perd l'information sur l'ordre des mots et d'autre part, les mots tels que « le » et « la » vont être poussés vers le dessus alors qu'ils ne possèdent pas d'information sémantique.

3.3 - Bag-of-words

Pour pallier ces premiers problèmes, on vectorise les phrases avec un histogramme par phrase (ou par texte) tout en gardant le vocabulaire entier présent dans le corpus.

```
# calcul d'un histogramme par document
import numpy as np

V = len(vocabulary)
D = len(corpus)
tf_idf = np.zeros([D, V])

for i, d in enumerate(corpus):
for w in d.split(" "):
   j = vocabulary.index(w)
        tf_idf[i,j] += 1
print(tf_idf)
```

Résultat :

[[1. 1. 1. 1. 1. 0. 0. 0.]
[2. 0. 0. 0. 0. 1. 1. 1.]
[2. 1. 0. 0. 0. 0. 1. 1.]]

Ce modèle est appelé sac de mots ou Bag-of-words (BOW). Ce modèle ne corrige pas tous les problèmes, notamment car il n'y a toujours pas d'information sur l'ordre des mots. Les phrases « Le chat regarde le chien » et « Le chien regarde le chat » auront un vecteur identique. BOW n'apprend pas le sens des mots. En ce sens, la distance des vecteurs ne représente pas forcément la différence de sens.

4 – Word2Vec

Word2Vec est un groupe de réseaux de neurones avec relativement peu de couches cachées (~2) qui vectorise les mots dans un espace de « faible » dimension. Il implémente en particulier les modèles CBOW et Skip-grams vus en 2 - Réponse 1.

On utilisera Word2Vec à l'aide de la bibliothèque Python **Gensim** [5] :

```
pip install gensim
```

4.1 - Entraînement d'un model simple

En commençant par l'importation du modèle Word2vec, on peut, à partir d'un certain corpus de phrases et de textes, entraîner le modèle sur nos données.

```
from gensim.models import Word2Vec
```

Pour utiliser des données pour l'entraînement, il faut au préalable les mettre en forme correctement. Reprenons le corpus en 3 -, la mise en forme de la première phrase pour l'apprentissage ressemble à ceci :

```
p_1 = ["le", "chat", "mange", "la", "souris"]
```

Vient alors le paramétrage de l'entraînement :

```
model = Word2Vec(min_count=1, vector_size=5)
```

où de nombreux paramètres sont disponibles. Dans cet exemple, seuls les paramètres min_count (ignore les mots dont l'occurrence est moindre que le nombre) et vector_size (dimension des vecteurs) sont utilisés pour une utilisation simplifiée. Le détail des paramètres peut être trouvé ici.

La création du dictionnaire ne nécessite qu'une fonction :

```
model.build_vocab(corpus)
```

Puis l'entraînement,

```
model.train(corpus, total_examples=model.corpus_count, epochs=model.epochs)
```

On peut maintenant visualiser la représentation vectorielle du mot « chat » de dimension 5 avec l'objet wv qui contient la transposition entre mot et vecteur :

```
print(model.wv['chat'])
print(model.wv['souris'])
```

Résultat :

[-0.036320 0.057531 0.019837 -0.16570 -0.188976]
[-0.068111 -0.018929 0.115377 -0.150441 -0.078726]

et regarder les deux mots les plus proches.

```
print(model.wv.most_similar(positive=['chat'] ,topn=3))
```

Résultat :

[('souris', 0.76687), ('canard', 0.61786) , ('la', 0.54855)]

On remarque alors le premier souci d'avoir un faible jeu de données : la proximité des mots dans une phrases et l'occurrence des autres mots du corpus. De cette manière, le mot « chien » qui serait en réalité plus proche de « chat » que ne l'est « canard » ne se trouve pas dans les mots les plus similaires.

Il existe de nombreux modèles déjà entraînés avec différents datasets disponibles en ligne et qui sont présents dans l'API **gensim.downloader**.

4.2 - Utilisation d'un modèle pré-entrainé

Cette fois-ci, on utilise l'API qui propose un modèle entrainé à partir du dataset de Google News contenant environ 3 millions de mots en anglais très majoritairement.

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

Gensim propose quelques méthodes de traitement de mots. On peut regarder quelques mots présents dans le dictionnaire.

```
# Affichage de quelques mots du vocabulaire
for index, word in enumerate(wv.index_to_key):
    if index == 5:
        break

    print(f"word #{index}/{len(wv.index_to_key)} is {word}")
```

Résultat :

word #0/3000000 is </s>
word #1/3000000 is in
word #2/3000000 is for
word #3/3000000 is that
word #4/3000000 is is

Les méthodes proposées par Gensim utilisent directement la représentation vectorielle des mots. Pour cela, il faut convertir les mots.

```
# Conversion du mot en vecteur
vec_king = wv['king']
```

Le vecteur étant assez volumineux, il n'est pas important de regarder son contenu.

L'outil le simple est l'évaluation de la similarité entre les mots. Pour cela, on utilise la fonction *similarity()*.

```
# Similarité entre des paires de mots
pairs = [
    ('wood', 'pine'),
    ('wood', 'leaf'),
    ('wood', 'plank')
]
for w1, w2 in pairs:
   print('%r\t%r\tSimilarité : %.2f' % (w1, w2, wv.similarity(w1, w2)))
```

Résultat :

'wood' 'pine' Similarité : 0.59
'wood' 'leaf' Similarité : 0.30
'wood' 'plank' Similarité : 0.26

Ou encore les mots les plus proches d'un mot cible avec *most_similar()*.

```
# N mots les plus proches du mot cible
print(wv.most_similar(positive=['wood'], topn=3))
print(wv.most_similar(positive=['bois'], topn=3))
```

Résultat :

[('lumber', 0.678), ('timber', 0.662), ('softwoods', 0.649)]
[('fleur', 0.682), ('jardin', 0.661), ("c'est_le", 0.651)]

La limite est alors d'une part dans le corpus de mots utilisés pour l'apprentissage du modèle et d'autre part la langue d'apprentissage. Le résultat pour les mots proches de 'bois' donne l'idée que le modèle peut fonctionner pour une autre langue mais connaissant les mots proches de 'bois', on se rend compte qu'il n'est pas très précis.

L'outil sur les similarités peut être poussé à l'utilisation pour trouver un mot intru dans un ensemble de mots avec *doesnt_match()*.

```
print(wv.doesnt_match(['wood', 'oak', 'tree', 'iron', 'leaf']))
```

Résultat :

Iron

Question 3 :

En utilisant la conversion de mots en vecteurs et la fonction de similarité *most_similar()*, proposer un code permettant de réaliser l'opération de l'introduction : papa – homme + femme = maman.

Réponse 3 :

```
# Opérations sur les mots
vec_father = wv['father']
vec_man = wv['man']
vec_woman = wv['woman']

result = wv.most_similar(positive=(vec_father - vec_man + vec_woman), topn=1)
print(result)
```

Résultat :

[('mother', 0.8671472072601318)]

Il existe également des petits jeux sur le web intégrant le word embedding tels que semantle.com ou word2vec.danielfrg.com.

5 – Conclusion

Nous avons exploré différentes méthodes simples pour effectuer des plongements de mots. Il en existe bien sûr des plus complexes, souvent à base de deep learning comme BERT qui est souvent utilisé pour cet usage. Le word embedding est souvent la première brique nécessaire pour le Natural Language Processing (NLP) ou Traitement Automatique du Language (TAL) en français. C'est une étape de préprocessing nécessaire et importante car elle pose les fondations sur lesquelles on va venir construire nos algorithmes.

Les applications du Word Embedding et du NLP sont nombreuses comme la traduction automatique, la détection d'émotion, la génération de texte, la réponse aux questions, les chatbots et bien d'autres.

Comme toutes les applications à grande échelle et surtout en machine learning, il faut cependant faire attention. On travaille souvent avec des données qui peuvent contenir des biais. Par exemple, après la sortie de Word2Vec on s'est rendu compte que l'algorithme reproduisait un biais sexiste qu'il avait appris dans les données. Si on lui pose la question « médecin – homme + femme » il nous répond « infirmière ». Heureusement, une partie importante du machine learning aujourd'hui s'intéresse à l'équité et l'explicabilité des algorithmes pour essayer d'améliorer la situation plutôt que de reproduire des biais appris.

Les différents programmes sont disponibles sur le dépôt Git :
https://gitea.auro.re/otthorn/TP_WordEmbedding

Références :

[1]: Valentin Noël, Séries temporelles et réseaux de neurones récurrents
https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14762/14762-series-temporelles-et-reseaux-de-neurones-recurrents-ensps.pdf

[2]: Olivier Tourvieille, Apprentissage supervisé – Comportement aérien d'un drone https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14545/14545-apprentissage-supervise-comportement-aerien-dun-drone-ensps.pdf

[3]: Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv https://arxiv.org/abs/1301.3781

[4]: Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J., "Distributed Representations of Words and Phrases and their Compositionality". arXiv https://arxiv.org/abs/1310.4546

[5]: Radim Řehůřek, Word2vec embeddings https://radimrehurek.com/gensim/models/word2vec.html

[6]: Dossier Intelligence Artificielle, juin 2022, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/dossier-intelligence-artificielle