جامعة ابن زهر
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵣⵀⵔ
UNIVERSITÉ IBN ZOHR

الكلية متعددة التخصصات - ورزازات
ⵜⴰⵖⵣⵓⵍⵜ ⵜⴰⵎⵙⴰⵡⵉⵜⵜ – ⵡⵓⵔⵣⴰⵣⴰⵜ
FACULTÉ POLYDISCIPLINAIRE DE OUARZAZATE

Master "Ingénierie mathématiques et sciences des données"

Rapport de TP — Word Représentation

Réalisé par :

LAGHFIRI Abdelaziz

Encadré par :

Mohamed Benaddy
&
Aissam Hadri

Année universitaire : 2025 – 2026

Abstract

Ce travail pratique est un tutoriel basé sur le modèle **Word2Vec**, un modèle fondamental du Deep Learning appliqué au traitement automatique du langage (NLP). L'objectif principal est de comprendre comment les mots peuvent être représentés sous forme de vecteurs numériques tout en conservant une partie de leur signification sémantique. Grâce à Word2Vec, chaque mot du vocabulaire est associé à un vecteur dans un espace continu, où les distances géométriques reflètent la similarité de sens entre les mots. Dans ce TP, nous avons exploré plusieurs concepts essentiels liés à Word2Vec : la **similarité cosinus**, qui permet de mesurer la proximité entre deux mots, la **fonction d'analogie**, qui exprime des relations linguistiques du type

$$\text{roi} - \text{homme} + \text{femme} \approx \text{reine},$$

ainsi que des fonctions de **neutralisation** et d'**égalisation**, qui visent à corriger les biais présents dans les représentations vectorielles. Ce tutoriel nous a permis de manipuler concrètement les vecteurs de mots et d'observer comment des relations sémantiques complexes peuvent être traduites par de simples opérations mathématiques. Ainsi, nous avons acquis une compréhension pratique du fonctionnement et de l'importance de Word2Vec dans le domaine du Deep Learning pour le langage.

`https://github.com/abdel-af/TP/blob/main/mon_code.ipynb`

Table des matières

Chapitre 1

Introduction

Le traitement automatique du langage naturel (NLP – *Natural Language Processing*) est une branche importante de l'intelligence artificielle. Son objectif est de permettre aux machines de comprendre, interpréter et générer le langage humain. Pour qu'un ordinateur puisse travailler sur du texte, il doit d'abord convertir les mots en nombres. Cependant, les méthodes simples comme le codage *one-hot* ne tiennent pas compte du sens ni des relations entre les mots : chaque mot est représenté par un vecteur indépendant, sans lien avec les autres.

Pour résoudre ce problème, des approches plus avancées ont été développées, notamment le modèle **Word2Vec**, introduit par Mikolov *et al.* (2013). Word2Vec repose sur une idée simple mais puissante : deux mots ayant des contextes similaires dans un texte ont également des significations proches. Le modèle apprend à chaque mot un *vecteur dense*, appelé *embedding*, qui encode sa signification linguistique. Ainsi, les mots ayant des sens proches sont représentés par des vecteurs proches dans l'espace géométrique.

Ce travail pratique s'inscrit dans un cadre **tutoriel**. Il vise à comprendre les principes fondamentaux de Word2Vec et à manipuler les opérations vectorielles associées. Nous mettons en œuvre plusieurs fonctions de base :

- la **similarité cosinus**, pour mesurer la proximité sémantique entre deux mots ;
- la **fonction d'analogie**, qui permet de retrouver des relations du type roi $-$ homme $+$ femme $\approx$ reine ;
- les fonctions de **neutralisation** et d'**égalisation**, qui servent à corriger les biais présents dans les représentations de mots.

Chapitre 2

Problématique

Le langage humain est complexe, riche en nuances et en relations sémantiques. Lorsqu'on souhaite utiliser le langage naturel dans des applications d'intelligence artificielle – comme la traduction automatique, la recherche d'information, ou la génération de texte –, il devient nécessaire de représenter les mots sous une forme numérique compréhensible par la machine.

Cependant, les représentations classiques, comme le codage *one-hot*, présentent plusieurs limites : chaque mot est représenté par un vecteur unique et isolé, sans lien avec les autres. Ainsi, deux mots ayant des sens proches, comme *roi* et *reine*, ou *homme* et *femme*, sont perçus comme totalement différents par la machine. Cette représentation empêche toute forme de raisonnement sémantique ou analogique.

Problème principal :
Comment représenter les mots de manière à préserver leurs relations de sens et leurs similitudes linguistiques ?

Le modèle **Word2Vec** répond à cette question en apprenant des représentations vectorielles continues appelées *embeddings*. Ces vecteurs permettent non seulement de mesurer la similarité entre les mots, mais aussi d'effectuer des opérations géométriques qui traduisent des relations linguistiques, comme dans l'analogie

$$\text{roi} - \text{homme} + \text{femme} \approx \text{reine}.$$

Chapitre 3

Méthodologie et implémentation

Ce travail pratique est organisé sous forme de **tutoriel guidé**, dont l'objectif est de manipuler et de comprendre le fonctionnement interne du modèle Word2Vec. L'approche adoptée repose sur une expérimentation progressive : après avoir chargé les vecteurs de mots préentraînés, nous implémentons différentes fonctions permettant de réaliser des opérations sémantiques simples et de vérifier les propriétés géométriques des *word embeddings*.

3.1 Chargement et préparation des données

La première étape consiste à charger les vecteurs de mots préentraînés à partir du fichier **GloVe (Global Vectors for Word Representation)**. Ce modèle fournit pour chaque mot un vecteur de dimension fixe (par exemple 50 ou 100). Les vecteurs sont ensuite stockés dans un dictionnaire Python, où chaque mot correspond à son vecteur associé.

3.2 Calcul de la similarité cosinus

Nous avons ensuite implémenté la fonction `cosine_similarity(u, v)`, qui calcule la similarité entre deux vecteurs selon la formule :

$$\text{similarité}(u, v) = \frac{u \cdot v}{\|u\|\|v\|}$$

Cette mesure indique dans quelle mesure deux mots ont un sens proche. Par exemple, la similarité entre `man` et `woman` est élevée, tandis que celle entre `table` et `sky` est faible.

3.3 Résolution d'analogies sémantiques

La fonction `complete_analogy(word_a, word_b, word_c, word_to_vec_map)` permet de résoudre des relations du type :

$$A \text{ est à } B \text{ comme } C \text{ est à ?}$$

Par exemple : `king - man + woman = queen`.

Cette opération démontre la capacité du modèle à représenter la **logique sémantique** sous forme géométrique.

3.4 Réduction des biais : neutralisation et égalisation

Enfin, deux fonctions supplémentaires ont été mises en œuvre :

— `neutralize(word, gender, word_to_vec_map)` : pour rendre un mot neutre vis-à-vis d'un axe de biais (ex. genre) ;

— `equalize((word1, word2), gender, word_to_vec_map)` : pour équilibrer deux mots opposés (comme `man`/`woman`) sur l'axe de biais.

Ces fonctions illustrent comment les *embeddings* peuvent être corrigés pour limiter les biais sociolinguistiques, une problématique moderne dans le domaine du **NLP**.

Chapitre 4

Résultats & discussion

Ce chapitre présente les **résultats obtenus** à partir du dictionnaire de vecteurs `d` fourni par `generateTestCases.py`, en utilisant les fonctions `cosine_similarity`, `complete_analogy`, `neutralize` et `equalize`. Nous indiquons pour chaque expérience le **code Python exécuté** et la **sortie obtenue**, puis discutons brièvement l'interprétation.

Contexte d'exécution

- Vocabulaire minimal : dictionnaire de vecteurs `d` (fichier `generateTestCases.py`).
- Axe de biais `gender` : vecteur 50D fourni (genre).
- Fonctions : `cosine_similarity`, `complete_analogy`, `neutralize`, `equalize`.

4.1 Similarité cosinus

Code.

```
u = d["pranked"]
v = d["notified"]

def cosine_similarity(u, v):
    import numpy as np
    dot = float(np.dot(u, v))
    return dot / (np.linalg.norm(u) * np.linalg.norm(v))

cos_val = cosine_similarity(u, v)
print(cos_val)
```

Résultat.

```
-0.014666333178310427
```

Discussion. La similarité cosinus entre `pranked` et `notified` est *très proche de zéro* (légèrement négative), indiquant une **faible proximité sémantique** dans cet espace vectoriel.

Ceci est cohérent linguistiquement : *to prank* (piéger/blaguer) et *to notify* (notifier/informer) apparaissent dans des contextes différents.

4.2 Analogie vectorielle

Code.

```
def complete_analogy(word_a, word_b, word_c, word_to_vec_map):
    import numpy as np
    def cosine_similarity(u, v):
        dot = float(np.dot(u, v))
        return dot / (np.linalg.norm(u) * np.linalg.norm(v))
    e_a = word_to_vec_map[word_a].astype(float)
    e_b = word_to_vec_map[word_b].astype(float)
    e_c = word_to_vec_map[word_c].astype(float)
    target = e_b - e_a
    best_word, best_sim = None, -1e9
    for w, vec in word_to_vec_map.items():
        if w in (word_a, word_b, word_c):
            continue
        sim = cosine_similarity(target, vec.astype(float) - e_c)
        if sim > best_sim:
            best_sim, best_word = sim, w
    return best_word, best_sim

best_word, sim_score = complete_analogy("notified","pranked","musburger", d)
print(best_word, sim_score)
```

Résultat.

```
buccinator 0.2847358451450477
```

Discussion. L'analogie `notified : pranked :: musburger : ?` renvoie `buccinator` avec un score d'alignement ≈ 0.285. Dans ce *mini-lexique*, le mot le plus cohérent géométriquement peut paraître peu intuitif. L'important ici est la **validité géométrique** de la relation $v_{\text{pranked}} - v_{\text{notified}} \approx v_{?} - v_{\text{musburger}}$. Sur un grand vocabulaire GloVe standard, on obtient en général des analogies plus naturelles.

4.3 Neutralisation d'un biais (axe `gender`)

Code.

```
import numpy as np

gender = np.array([
 -0.087144,  0.2182  , -0.40986 , -0.03922 , -0.1032  ,  0.94165,
 -0.06042 ,  0.32988 ,  0.46144 , -0.35962 ,  0.31102 , -0.86824,
  0.96006 ,  0.01073 ,  0.24337 ,  0.08193 , -1.02722 , -0.21122,
  0.695044, -0.00222 ,  0.29106 ,  0.5053  , -0.099454,  0.40445,
  0.30181 ,  0.1355  , -0.0606  , -0.07131 , -0.19245 , -0.06115,
 -0.3204  ,  0.07165 , -0.13337 , -0.25068714, -0.14293 , -0.224957,
 -0.149   ,  0.048882,  0.12191 , -0.27362 , -0.165476, -0.20426,
  0.54376 , -0.271425, -0.10245 , -0.32108 ,  0.2516  , -0.33455,
 -0.04371 ,  0.01258
], dtype=float)

def neutralize(word, g, word_to_vec_map):
    e = word_to_vec_map[word].astype(float)
    g = g.astype(float)
    e_bias = np.dot(e, g) / np.dot(g, g) * g
    return e - e_bias

neu_pranked = neutralize("pranked", gender, d)
print(np.round(neu_pranked[:10], 6))  # aper u 10 premi res composantes
```

Résultat (extrait).

```
[-0.273231 -1.146302  0.580177 -0.538272 -0.554813 -0.330141 -0.189922  0.44017
  0.163696  0.290657]
```

Discussion. La neutralisation retire la composante de `pranked` le long de l'axe `gender`. On obtient un vecteur **dé-biaisé** conservant l'information orthogonale à cet axe, utile pour atténuer des biais sociolinguistiques.

4.4 Égalisation d'une paire de mots

Code.

```
def equalize(pair, bias_axis, word_to_vec_map):
    w1, w2 = pair
    e1 = word_to_vec_map[w1].astype(float)
    e2 = word_to_vec_map[w2].astype(float)
    g  = bias_axis.astype(float)
    mu = (e1 + e2) / 2.0
    mu_B = (np.dot(mu, g) / np.dot(g, g)) * g
```

```
    mu_orth = mu - mu_B
    e1_B = (np.dot(e1, g) / np.dot(g, g)) * g
    e2_B = (np.dot(e2, g) / np.dot(g, g)) * g
    denom1 = np.linalg.norm(e1_B - mu_B) + 1e-12
    denom2 = np.linalg.norm(e2_B - mu_B) + 1e-12
    scale  = np.sqrt(max(1e-12, 1.0 - np.linalg.norm(mu_orth)**2))
    e1B_corr = scale * (e1_B - mu_B) / denom1
    e2B_corr = scale * (e2_B - mu_B) / denom2
    return e1B_corr + mu_orth, e2B_corr + mu_orth

eq1, eq2 = equalize(("musburger","pranked"), gender, d)
print(np.round(eq1[:10], 6))
print(np.round(eq2[:10], 6))
```

Résultat (extraits).

```
eq(musburger)*: [-0.755557 -0.727754  0.502788  0.187333  0.320739 -0.190323 -0.894624 -
eq(pranked)*  : [-0.966538 -0.199479 -0.489507  0.092379  0.070886  0.732735  0.382439
```

Discussion. Après égalisation, les deux vecteurs deviennent **équidistants** du plan neutre associé à l'axe `gender`, ce qui **réduit l'asymétrie de biais** entre les deux mots tout en préservant leur composante sémantique partagée.

4.5 Tableau récapitulatif

Fonction	Entrée	Sortie	Interprétation
`cosine_similarity`	pranked / notified	-0.014666	Faible similarité sémantique (valeur proche de 0).
`complete_analogy`	(notified, pranked)	buccinator (0.285)	Relation géométrique la plus alignée dans le mini-lexique `d`.
`neutralize`	pranked	vecteur dé-biaisé	Composante le long de l'axe `gender` supprimée.
`equalize`	(musburger, pranked)	deux vecteurs symétrisés	Paires rendues équidistantes du plan neutre, biais atténué.

Remarque. Avec un *grand vocabulaire* (p. ex. GloVe 6B), on obtient généralement des analogies plus intuitives (*king - man + woman* $\approx$ *queen*). Les principes géométriques démontrés ici restent identiques.

Chapitre 5

Conclusion

Ce travail pratique nous a permis d'explorer les fondements de la représentation vectorielle des mots à travers le modèle **Word2Vec**, largement utilisé en *Deep Learning* pour le traitement automatique du langage (NLP). En partant d'un cadre tutoriel, nous avons relié des notions théoriques (représentations continues, géométrie des espaces vectoriels) à des expérimentations concrètes : *similarité cosinus* pour mesurer la proximité sémantique, *analogie vectorielle* pour raisonner sur des relations de type roi $-$ homme $+$ femme $\approx$ reine, et techniques de *neutralisation/égalisation* pour atténuer des biais (ex. liés au genre).

Les résultats montrent que des opérations mathématiques simples (produit scalaire, normalisation, projection) peuvent capturer des phénomènes linguistiques non triviaux. Ils soulignent également deux points essentiels : (i) la qualité des analogies dépend fortement du vocabulaire et des données d'entraînement ; (ii) la question des biais dans les embeddings est réelle et nécessite des traitements dédiés pour des usages responsables.

Au plan pédagogique, ce TP a consolidé la compréhension du lien entre **sémantique** et **géométrie** : la proximité de sens se manifeste par la proximité de vecteurs, et des directions particulières peuvent encoder des relations linguistiques. Sur le plan pratique, il constitue une base solide pour aborder des modèles plus avancés (GloVe, FastText) et des approches contextuelles modernes (BERT, GPT), qui étendent ces idées à des représentations dépendantes du contexte.

Perspectives. Des prolongements naturels incluent : (i) l'évaluation sur un vocabulaire large (GloVe 6B/300d) pour des analogies plus « naturelles » ; (ii) la visualisation des embeddings (PCA/t-SNE/UMAP) ; (iii) l'étude comparative Word2Vec vs. modèles contextuels ; (iv) l'analyse systématique et la mitigation des biais sur des paires et axes multiples.