Word2Vec

Implémentation et fondements théoriques

Modèle Skip-gram avec *Negative Sampling* en NumPy

Rachid Ghodbane

Encadré par le Professeur **François Jacquenet**

L3 informatique – Université Jean Monnet, Saint-Étienne – 2025/2026

Table des matières

1 Introduction **2**

2 L'hypothèse distributionnelle **2**

3 Le modèle Skip-gram **3**

3.1 Formulation de l'objectif . . . 3

3.2 La probabilité softmax et son coût . . . 3

4 Negative Sampling **4**

4.1 Idée centrale . . . 4

4.2 La distribution de bruit . . . 4

4.3 Sous-échantillonnage des mots fréquents (optionnel) . . . 4

5 Fonction de perte et gradients **5**

5.1 Une factorisation élégante . . . 5

5.2 Gradients vectoriels . . . 5

5.3 Mise à jour SGD . . . 5

6 Algorithme d'entraînement **6**

7 De la théorie au code **6**

8 Évaluation des embeddings **7**

8.1 Similarité cosinus . . . 7

8.2 Analogies vectorielles . . . 7

8.3 Visualisation par PCA . . . 8

9 Résultats et limites **8**

10 Conclusion **9**

Ressources **10**

1 Introduction

Le traitement automatique des langues (TAL) repose sur une difficulté fondamentale : le langage est symbolique et discret, alors que les modèles d'apprentissage statistique manipulent des vecteurs réels. La question centrale est donc de savoir comment *représenter un mot par un vecteur* qui capture une part de son sens.

Les approches historiques utilisaient des représentations dites *one-hot*, où chaque mot d'un vocabulaire de taille V est encodé par un vecteur de $\{0,1\}^V$ ne contenant qu'un seul 1. Ces vecteurs sont creux, gigantesques, et surtout *orthogonaux deux à deux* : la distance entre « chat » et « chien » y est identique à celle entre « chat » et « avion ». Aucune information sémantique n'est encodée.

Word2Vec, introduit par Mikolov et al. (2013), répond à ce problème en apprenant des représentations *denses* et de faible dimension (typiquement 100 à 300), appelées *word embeddings*. Ces vecteurs ont la propriété remarquable de placer les mots sémantiquement proches dans des régions voisines de l'espace, et d'encoder certaines relations sous forme linéaire (la fameuse analogie $\text{roi} - \text{homme} + \text{femme} \approx \text{reine}$).

Ce rapport présente les fondements théoriques du modèle **Skip-gram avec Negative Sampling** (SGNS) et décrit son implémentation complète en NumPy, sans recourir à aucune bibliothèque d'*embedding* préexistante. L'objectif est double : comprendre précisément les mathématiques sous-jacentes et démontrer leur traduction directe en code.

2 L'hypothèse distributionnelle

Définition 1 (Hypothèse distributionnelle)**.** *Des mots qui apparaissent dans des contextes similaires tendent à avoir des sens similaires. Selon la formule attribuée à Firth (1957) :* « You shall know a word by the company it keeps »*.*

Word2Vec exploite directement ce principe. Plutôt que de définir le sens d'un mot, on apprend à *prédire son voisinage*. Le sens émerge alors comme un sous-produit de la tâche de prédiction : deux mots qui prédisent les mêmes contextes finiront par recevoir des vecteurs proches.

On distingue deux architectures dans le papier original :

- **CBOW** (*Continuous Bag-of-Words*) : prédire le mot central à partir de son contexte.
- **Skip-gram** : prédire le contexte à partir du mot central. C'est l'architecture retenue ici, plus performante sur les mots rares.

3 Le modèle Skip-gram

3.1 Formulation de l'objectif

Soit un corpus représenté par une suite de mots $w_1, w_2, \ldots, w_T$. Pour une *fenêtre* de demi-largeur c, le modèle Skip-gram cherche à maximiser la log-vraisemblance moyenne des mots de contexte sachant le mot central :

$$\mathcal{J}(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-c \leq j \leq c \\ j \neq 0}} \log p\big(w_{t+j} \mid w_t\big). \tag{1}$$

Chaque mot w possède *deux* représentations vectorielles dans $\mathbb{R}^d$:

- v_w lorsqu'il joue le rôle de **mot central** (lignes de la matrice W_{in}) ;
- u_w lorsqu'il joue le rôle de **mot de contexte** (lignes de la matrice W_{out}).

Remarque 1. *Cette dualité v/u n'est pas un détail d'implémentation : elle simplifie considérablement les dérivées et évite des pathologies (comme le produit scalaire d'un mot avec lui-même). Les* embeddings *finalement utilisés sont, par convention, les v_w (matrice W_{in}).*

3.2 La probabilité softmax et son coût

La probabilité conditionnelle d'un mot de contexte o sachant le mot central c est classiquement modélisée par un *softmax* sur le vocabulaire entier :

$$p(o \mid c) = \frac{\exp\big(u_o^\top v_c\big)}{\displaystyle\sum_{w=1}^{V} \exp\big(u_w^\top v_c\big)}. \tag{2}$$

Le produit scalaire $u_o^\top v_c$ mesure la compatibilité entre les deux mots ; le softmax la normalise en distribution de probabilité.

Le problème est le **dénominateur** : il faut sommer sur les V mots du vocabulaire à chaque calcul, et son gradient implique tous les vecteurs u_w. Pour un vocabulaire de plusieurs centaines de milliers de mots, ce coût en $\mathcal{O}(V)$ rend l'entraînement impraticable. C'est précisément le problème que résout le *negative sampling*.

4 Negative Sampling

4.1 Idée centrale

Plutôt que de prédire la distribution exacte sur tout le vocabulaire, on reformule la tâche en un problème de **classification binaire**. On distingue :

- les vrais couples (w, c) observés dans le corpus (exemples *positifs*, étiquette $D = 1$) ;
- des couples (w, n) tirés au hasard (exemples *négatifs*, étiquette $D = 0$).

On entraîne le modèle à distinguer les deux. Pour un couple positif (w, c) et k négatifs $n_1, \ldots, n_k$, l'objectif à **maximiser** pour cet exemple devient :

$$\ell(w, c) = \log \sigma\big(u_c^\top v_w\big) + \sum_{i=1}^{k} \mathbb{E}_{n_i \sim P_n(w)} \big[\log \sigma\big(-u_{n_i}^\top v_w\big)\big], \tag{3}$$

où σ est la fonction sigmoïde :

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \qquad \sigma(-x) = 1 - \sigma(x). \tag{4}$$

Interprétation. Le premier terme pousse le produit scalaire $u_c^\top v_w$ vers $+\infty$ (donc $\sigma \to 1$) : le modèle doit reconnaître les vrais couples. Le second terme pousse $u_{n_i}^\top v_w$ vers $-\infty$ ($\sigma(-x) \to 1$) : le modèle doit rejeter les couples aléatoires. On a ainsi remplacé une somme sur V termes par une somme sur $k + 1$ termes seulement, avec $k \in [5, 20]$.

4.2 La distribution de bruit

Les mots négatifs ne sont pas tirés uniformément. Mikolov et al. montrent empiriquement que la meilleure distribution est la distribution *unigramme élevée à la puissance* $3/4$:

$$P_n(w) = \frac{f(w)^{3/4}}{\sum_{w'=1}^{V} f(w')^{3/4}}, \tag{5}$$

où $f(w)$ est la fréquence brute du mot w. L'exposant $3/4$ atténue le poids des mots très fréquents (articles, prépositions) tout en laissant aux mots rares une chance d'être tirés comme négatifs. C'est exactement cette distribution qui est implémentée dans la méthode `build_vocab`.

4.3 Sous-échantillonnage des mots fréquents (optionnel)

Pour réduire l'influence des mots ultra-fréquents, on peut écarter aléatoirement un mot w_i avec une probabilité :

$$P(\text{retirer } w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}, \tag{6}$$

où $t \approx 10^{-5}$ est un seuil. Cette technique accélère l'entraînement et améliore la qualité des vecteurs des mots rares.

5 Fonction de perte et gradients

L'entraînement minimise l'opposé de l'objectif (3). Pour un unique couple positif (w, c) et ses négatifs, la **perte** s'écrit :

$$\mathcal{L} = -\log \sigma\big(u_c^\top v_w\big) - \sum_{i=1}^{k} \log \sigma\big(-u_{n_i}^\top v_w\big). \tag{7}$$

5.1 Une factorisation élégante

Introduisons une étiquette $t_j \in \{0, 1\}$ pour chaque mot de sortie j considéré ($t_j = 1$ pour le vrai contexte c, $t_j = 0$ pour les négatifs), et notons le score $s_j = \sigma(u_j^\top v_w)$. En utilisant la dérivée de la sigmoïde $\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$, on obtient pour *tous* les termes une même forme remarquablement simple :

$$\boxed{\frac{\partial \mathcal{L}}{\partial (u_j^\top v_w)} = \sigma(u_j^\top v_w) - t_j = s_j - t_j.} \tag{8}$$

Démonstration pour le terme positif. On a $\frac{\partial}{\partial x}\big[-\log \sigma(x)\big] = -\frac{\sigma'(x)}{\sigma(x)} = -\big(1 - \sigma(x)\big) = \sigma(x) - 1 = s_c - 1$, ce qui correspond bien à $s_j - t_j$ avec $t_c = 1$.

Démonstration pour un terme négatif. On a $\frac{\partial}{\partial x}\big[-\log \sigma(-x)\big] = \frac{\sigma'(-x)}{\sigma(-x)} = 1 - \sigma(-x) = \sigma(x) = s_{n_i}$, ce qui correspond à $s_j - t_j$ avec $t_{n_i} = 0$. □

5.2 Gradients vectoriels

En appliquant la règle de la chaîne au résultat (8), les gradients par rapport aux vecteurs s'écrivent :

$$\frac{\partial \mathcal{L}}{\partial u_j} = \big(s_j - t_j\big)\, v_w, \qquad \text{pour chaque } j \in \{c\} \cup \{n_1, \dots, n_k\}, \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial v_w} = \sum_j \big(s_j - t_j\big)\, u_j. \tag{10}$$

5.3 Mise à jour SGD

La descente de gradient stochastique applique, pour chaque couple positif et son taux d'apprentissage η :

$$u_j \leftarrow u_j - \eta\,(s_j - t_j)\, v_w, \tag{11}$$

$$v_w \leftarrow v_w - \eta \sum_j (s_j - t_j)\, u_j. \tag{12}$$

Ces deux lignes constituent le cœur de l'algorithme. Tout le reste n'est qu'orchestration : parcours du corpus, tirage des négatifs et suivi de la perte.

6 Algorithme d'entraînement

L'enchaînement complet est résumé ci-dessous :

1. **Tokenisation** : découper le corpus en mots (minuscules, séparation par espaces).
2. **Vocabulaire** : compter les occurrences, attribuer un indice à chaque mot, calculer $P_n(w)$ selon (5).
3. **Initialisation** : W_{in} tirée selon une loi uniforme centrée et réduite par $1/d$, W_{out} initialisée à zéro.
4. **Génération des couples** : pour chaque mot central, produire un couple avec chaque mot de la fenêtre.
5. **Boucle d'entraînement** : pour chaque époque, mélanger les couples, puis pour chaque couple appliquer les équations (9)–(10) et accumuler la perte (7).

7 De la théorie au code

Le passage des formules au code NumPy est direct. Le cœur de la boucle d'entraînement traduit littéralement les équations (9) et (10) :

```
1  v = self.W_in[target]                # v_w          (d,)
2  idxs = np.concatenate(([context], negatives))  # {c} U negatifs
3  labels = np.zeros(len(idxs)); labels[0] = 1.0   # t_j
4
5  u = self.W_out[idxs]                 # u_j          (1+k, d)
6  scores = self.sigmoid(u @ v)         # s_j = sigma(u_j . v)
7
8  grad   = scores - labels             # (s_j - t_j)   eq. (8)
9  grad_v = grad @ u                    # somme_j (s_j-t_j) u_j  -> eq.(10)
10 grad_u = np.outer(grad, v)           # (s_j - t_j) v_w        -> eq.(9)
11
12 self.W_out[idxs] -= self.lr * grad_u # mise a jour u_j
13 self.W_in[target] -= self.lr * grad_v # mise a jour v_w
```

On reconnaît le vecteur `grad` $= s_j - t_j$ de l'équation (8), le produit `grad @ u` qui réalise la somme (10), et le produit extérieur `np.outer(grad, v)` qui réalise (9). La sigmoïde est implémentée de façon numériquement stable par troncature de l'argument dans $[-50, 50]$ avant l'exponentielle.

8 Évaluation des embeddings

8.1 Similarité cosinus

Pour comparer deux vecteurs, on utilise la *similarité cosinus*, insensible à la norme :

$$\cos(a, b) = \frac{a^\top b}{\|a\| \, \|b\|} \in [-1, 1]. \tag{13}$$

Une valeur proche de 1 indique des mots sémantiquement proches. La méthode `most_similar` calcule ce score entre un mot cible et tout le vocabulaire, puis renvoie les plus proches voisins.

8.2 Analogies vectorielles

La propriété la plus célèbre de Word2Vec est l'encodage *linéaire* de relations sémantiques. La relation « masculin → féminin » se traduit par une translation approximativement constante :

$$v_{\text{roi}} - v_{\text{homme}} + v_{\text{femme}} \approx v_{\text{reine}}. \tag{14}$$

On résout l'analogie « a est à b ce que c est à ? » en cherchant le mot dont le vecteur maximise la similarité cosinus avec $v_a - v_b + v_c$, en excluant a, b et c.

La figure 1 illustre cette structure dans un espace d'*embedding* (réduit ici à 3 dimensions pour la visualisation) : la relation « masculin → féminin » correspond à une translation quasi constante, représentée par deux vecteurs parallèles.

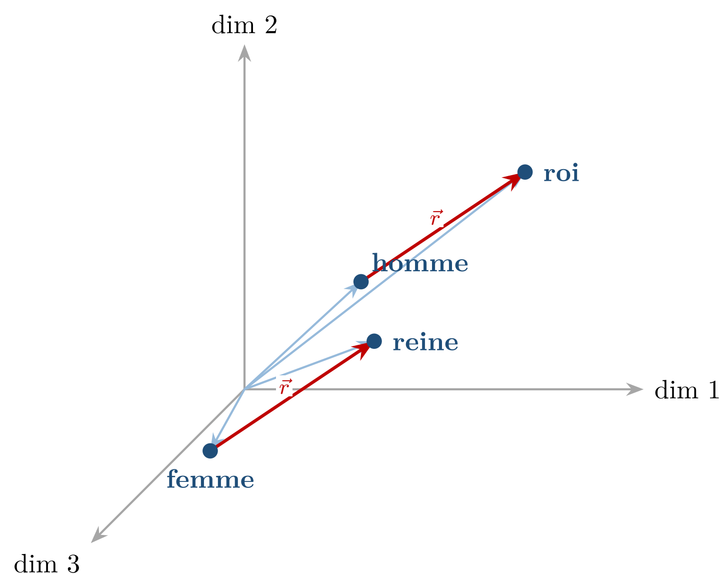

Fig. 1 : Représentation 3D de *word embeddings*. Les vecteurs bleus partent de l'origine vers chaque mot. Les deux vecteurs rouges $\vec{r} = v_{\text{roi}} - v_{\text{homme}} \approx v_{\text{reine}} - v_{\text{femme}}$ sont parallèles : la relation sémantique est encodée linéairement.

8.3 Visualisation par PCA

Pour visualiser des vecteurs de dimension d en deux dimensions, on applique une **Analyse en Composantes Principales**. Soit X la matrice des embeddings centrée (moyenne par colonne soustraite). Sa décomposition en valeurs singulières s'écrit :

$$X = U\Sigma V^{\top}. \tag{15}$$

La projection sur les deux premières composantes principales est $XV_{[:,1:2]}$, c'est-à-dire la projection sur les directions de plus grande variance. Cette projection est implémentée en NumPy pur via `np.linalg.svd`, sans aucune bibliothèque externe.

9 Résultats et limites

Sur le corpus jouet fourni (4 phrases, 16 mots), la perte décroît nettement au cours des époques, et les plus proches voisins reflètent les co-occurrences locales (par exemple « cat » se rapproche de « mat », « dog » de « ball »). Toutefois, un tel corpus est beaucoup trop réduit pour faire émerger une véritable sémantique : les régularités linéaires comme l'analogie roi/reine ne deviennent fiables qu'à partir de corpus de plusieurs millions de tokens.

Limites de l'implémentation pédagogique :

- Le tirage des négatifs s'effectue mot par mot (boucle Python), ce qui serait à vectoriser pour un grand corpus.
- Le sous-échantillonnage et la fenêtre dynamique ne sont pas activés par défaut.
- L'entraînement est mono-thread et en pleine précision ; une implémentation industrielle utiliserait du parallélisme et des tables de sigmoïde pré-calculées.

Ces choix sont assumés : la priorité a été donnée à la *lisibilité mathématique* du code plutôt qu'à la performance brute.

10 Conclusion

Word2Vec illustre une idée profonde : en transformant un problème sémantique flou en une simple tâche de classification binaire (distinguer les vrais couples des couples aléatoires), on obtient des représentations vectorielles riches et exploitables. La clé technique est le *negative sampling*, qui rend l'entraînement tractable en remplaçant un softmax coûteux par quelques produits scalaires. L'unité remarquable entre la théorie et le code — où le gradient se réduit à l'expression $s_j - t_j$ — montre que la rigueur mathématique et la simplicité d'implémentation peuvent aller de pair.

Ressources

- **Stanford CS224N — Natural Language Processing with Deep Learning**, support de cours et notes (notamment les conférences sur les *Word Vectors* et le *Negative Sampling*). `https://web.stanford.edu/class/cs224n/`
- **Cours de Traitement Automatique des Langages Naturels**, L3 Informatique, Université Jean Monnet de Saint-Étienne, Professeur François Jacquenet, 2025/2026.