# Projet Info804 Deep learning

Samy Amarouche – Raphaëlle Huynh

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

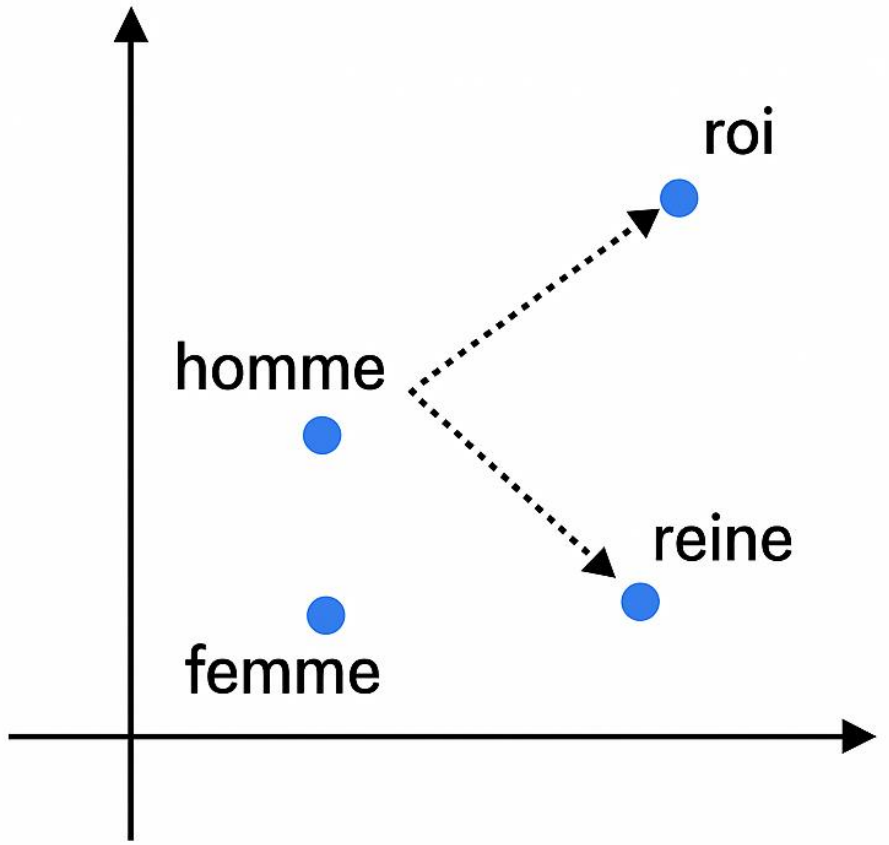
# SOMMAIRE

# La méthode word2vec - Introduction

- Word2Vec : une méthode clé dans le NLP.
- Objectif : représenter les mots dans un espace vectoriel pour capturer leurs relations sémantiques.
- Approche distribuée du sens des mots.
- Famille de modèles de langage prédictifs développée par Google (2013).
- Produit des embeddings de mots continus à faible dimension.

# La méthode word2vec

- Représentation d'un mot (ou d'un objet) sous forme d'un vecteur de nombres réels.
- Permet de capturer :
  - La sémantique des mots : Les mots proches en sens sont proches dans l'espace vectoriel.
  - La syntaxe : certaines relations grammaticales aussi (genre, conjugaison, etc.).
- Exemple :
  - roi - homme + femme ≈ reine



# La méthode word2vec - Architectures

- CBOW (Continuous Bag of Words) :
  - prédit un mot à partir de son contexte
  - Exemple :
    *Contexte :* "Le ___ dort sur le canapé"
    *CBOW prédit :* "chat »

  Avantages :
  - Plus rapide à entraîner
  - Performant avec des corpus de grandes tailles

- Skip-gram : prédit le contexte à partir d'un mot cible.
  - Exemple :
    *Mot cible :* "chat"
    *Skip-gram prédit :* "Le", "dort", "sur", "le", "canapé »

  Avantages :
  - Meilleure performance sur les mots rares
  - Plus précis sur des petits corpus

# La méthode word2vec - Entrainement

## Un réseau de neurones très simple avec

Une **couche d'entrée** : le mot (ou les mots du contexte), encodé en one-hot.

Une **couche cachée** sans activation : cette couche apprend les **vecteurs de mots** (embeddings).

Une **couche de sortie** : prédit le mot cible ou le contexte via une fonction de probabilité.

## Objectif d'apprentissage et entrainement

Optimiser la probabilité que le modèle prédise correctement un mot à partir de son contexte (CBOW) ou l'inverse (Skip-gram).

Utilise des milliers à des milliards de paires mot-contexte.

Chaque itération ajuste les vecteurs pour que :
- Les mots non liés soient éloignés.
- Les mots proches en contexte soient proches en vecteur.

# La méthode word2vec – Les problèmes

- Le vocabulaire peut contenir des millions de mots.
- Calculer la probabilité de chaque mot : coûteux.
- D'où l'intérêt des **méthodes d'optimisation** :
  - Subsampling
  - Negative Sampling

# Les méthodes d'optimisations - Subsampling des mots fréquents

- But : réduire la fréquence des mots très courants ("le", "et", "de", etc.).
- Ces mots n'apportent souvent pas d'information sémantique utile.
- Formule :

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

- avec $f(w_i)$ la fréquence du mot, et $t$ un seuil (ex: $10^{-5}$)

- Si un mot est très fréquent → $f(w_i)$ est grand → $\frac{t}{f(w_i)}$ est petit → $\sqrt{t/f(w_i)}$ est petit → probabilité de suppression élevée.
- Si un mot est rare → $f(w_i)$ est petit → probabilité de suppression proche de zéro.
- Exemple
  - Si "le" apparaît dans 5% des mots (f = 0.05)

$$P(le) = 1 - \sqrt{\frac{10^{-5}}{0.05}} \approx 1 - 0.0144 \approx 0.986$$

  - Il a de très forte probabilité d'être ignoré pendant l'entraînement.

# Les méthodes d’optimisation – Negative sampling – Le principe

Objectif :

- Remplacer le coût énorme du softmax sur tout le vocabulaire (des dizaines de milliers de mots).
- On ne classe pas parmi tous les mots → on discrimine juste entre :
  - Un vrai couple (mot, contexte) ➕
  - Des faux couples (mot, faux contexte choisi au hasard) ➖

C’est une tâche de classification binaire :

- "Ce couple mot-contexte est-il réel ou faux ?"

# Les méthodes d’optimisations - Negative Sampling — Comment ça marche

Pour chaque vrai couple (w<sub>t</sub>, w<sub>c</sub>) :

- On le garde comme exemple positif.
- On génère k mots aléatoires (w<sub>i</sub>) comme contextes négatifs.
- On apprend :
  - À rapprocher les vrais couples (vecteurs proches)
  - À éloigner les faux (vecteurs opposés)

Fonction objectif :

- $\log \sigma\,(v_c . v_t) + \sum_{i=1}^{k} \log \sigma(- v_i . v_t)$

  $v_t$ = vecteur du mot cible

  $v_c$ = vecteur du vrai contexte

  $v_i$ = vecteurs des contextes négatifs

  $\sigma(x) = \frac{1}{1 + e^{-x}} = sigmoïd$

Intuition :

- On renforce la similarité entre vrais couples.
- On réduit celle avec les faux.

Avantages :

- Beaucoup plus rapide que softmax.
- Fonctionne très bien, même avec peu de négatifs (ex: k = 5 à 20).

# Les bases de données utilisées - Présentations

- Pour entrainer nos Embeddings nous avons utilisé la base de données 20 Newgroups
  - La base de données contient 20.000 documents de 20 groupes de discussion différents
- Pour réalisation la classification pour tester la robustesse de nos Embeddings nous avons utilisé la base de données IMDB Movie Reviews
  - Une base de données de 50,000 critiques de films avec des étiquettes de sentiment (positif/négatif).

# Les bases de données utilisées – 20 New Groups

Pour utiliser la base de données nous avons du nettoyer les données dans un premier temps en réalisant un pipeline.

Class ***TextPreprocessor*** dédiée à cette tâche

**Normalisation du texte :**

- Passage en minuscules pour uniformiser le texte.
- Suppression de tous les caractères spéciaux (ponctuation, chiffres, symboles) à l'aide d'expressions régulières

**Tokenisation :**

- Découpage du texte en mots (tokens) via str.split().
- Suppression des stopwords (mots très fréquents mais peu informatifs comme "the", "and", etc.), chargés une seule fois depuis nltk.corpus.stopwords.,

**Gestion des mots rares :**

- Calcul des fréquences d'apparition des mots avec collections.Counter.
- Filtrage des mots ayant une fréquence inférieure à un seuil (min_freq=5), afin de réduire le bruit et la taille du vocabulaire.

**Gestion des stopwords**

- Utilisation de la liste des stopwords de nltk pour supprimer les stopwords

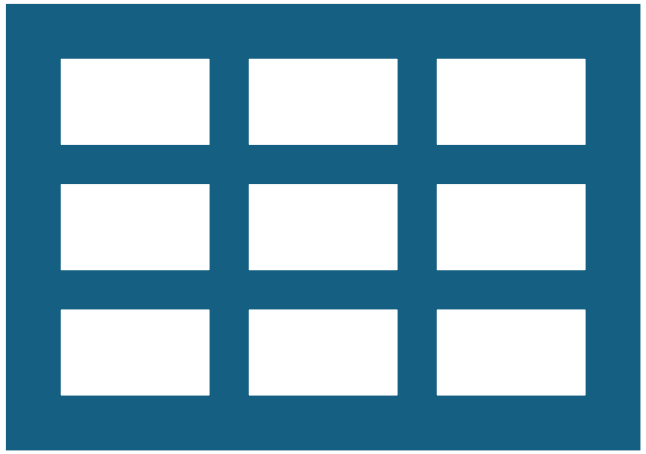
# Les bases de données utilisées – Construction du vocabulaire

Après avoir nettoyé et filtré, nous avons généré deux dictionnaires pour gérer les représentations numériques :

- word2idx : associe un mot à un identifiant unique

- idx2word : associe un identifiant à son mot d'origine

- Cela permet d'encoder et décoder les textes facilement pour les phases d'apprentissage et d'évaluation.



# Les bases de données utilisées - IMDB

Dans le cadre de la classification de sentiments sur le jeu de données IMDB :

- Les textes ont été nettoyés par la fonction clean_text (similaire à TextPreprocessor mais adaptée au format CSV).
- Chaque texte est transformé en liste de mots.
- Les étiquettes ("positive" / "negative") sont converties en labels binaires (1 / 0).
- Les stopwords ont aussi été retiré à l'aide de la liste des stopwords de la librairie nltk

# Les tests réalisés

- Tests sur les Embeddings
- Tests sur la classification de notre class ANNclassifier
- Tests avec l’Embedding de Google

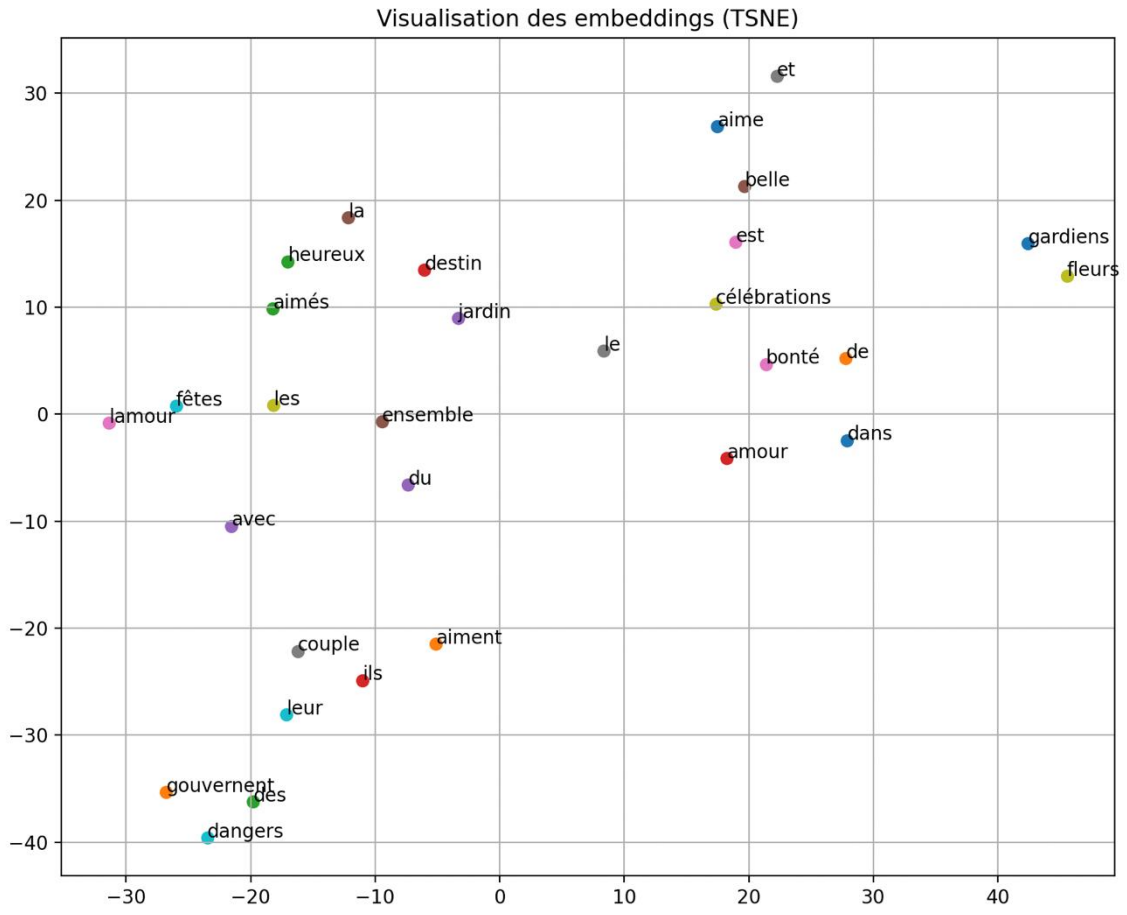
# Les tests réalisés – Les embeddings

- Dans un premier temps nous avons réalisés des tests sur nos embeddings avec une base de données simple générés à l'aide de copilot pour vérifier si nos embeddings fonctionnaient.

```
corpus = (
        "Le roi aime la reine et la reine aime le roi. Le roi est puissant et la reine est belle. "
        "Le roi et la reine se promènent dans le jardin. Ils aiment les fleurs et les oiseaux. "
        "Le roi et la reine sont heureux ensemble. Ils aiment passer du temps ensemble. "
        "Le roi et la reine sont les souverains du royaume. Ils gouvernent avec sagesse et bonté. "
        "Le roi et la reine sont aimés de leur peuple. Ils organisent des fêtes et des célébrations. "
        "Le roi et la reine sont un couple royal. Ils sont le symbole de l'amour et de l'harmonie. "
        "Le roi et la reine sont unis par le destin. Ils sont liés par un amour éternel. "
        "Le roi et la reine sont les gardiens de la paix. Ils protègent leur royaume des dangers. "
)
```

# Les tests réalisés – Les embeddings

- Cela nous a permis de vérifier si notre pipeline de prétraitement fonctionnait mais aussi de vérifier l'efficacité de nos Embeddings.
- Nous avons commencé par tester avec cbow sans optimisations
- Nous avons ensuite vérifié la logique des mots les plus proches à l'aide du cosinus de similarité



# Les tests réalisés – Cosinus similarity cbow

**Lien plutôt logique**

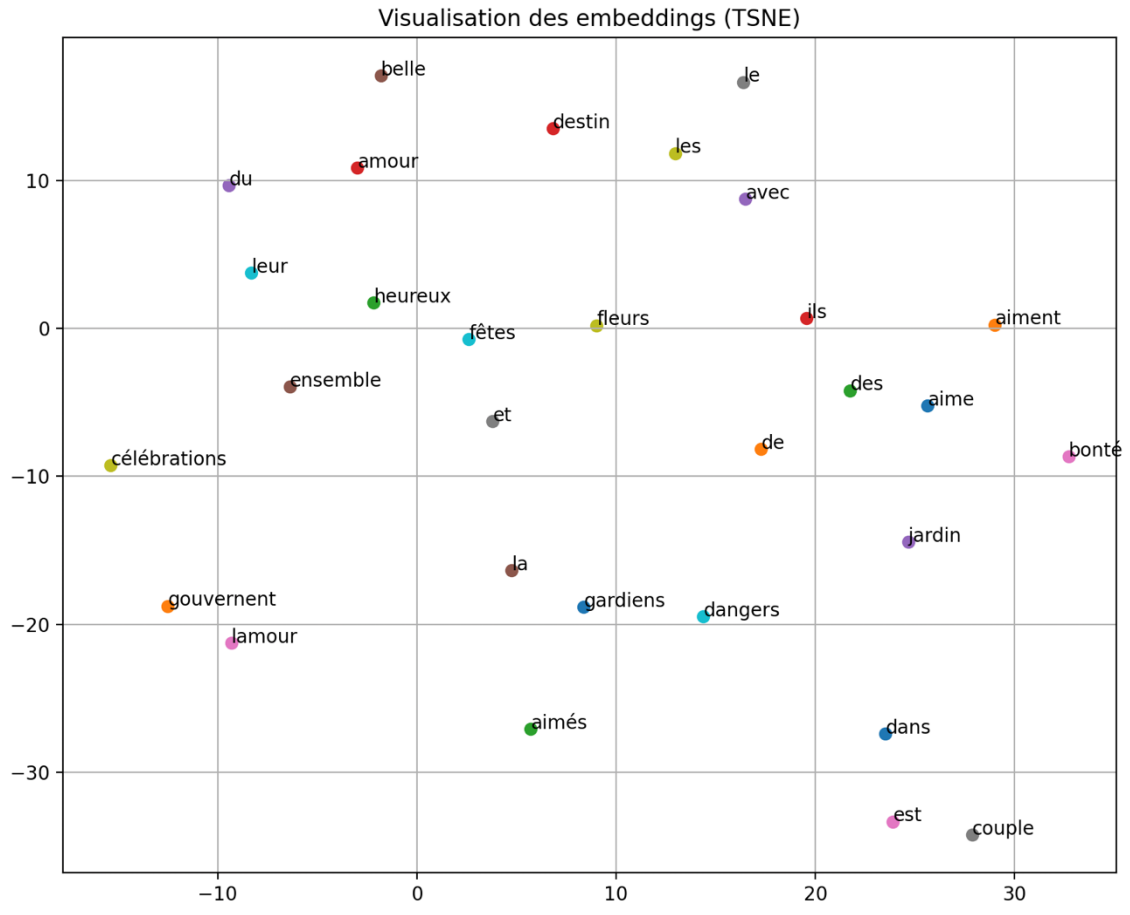
- Mot: royaume, Mots similaires: ['protègent', 'peuple', 'des', 'ils', 'royaume’]
- Mot: éternel, Mots similaires: ['un', 'liés', 'par', 'amour', 'éternel’]
- Mot: liés, Mots similaires: ['heureux', 'par', 'royal', 'couple', 'liés’]
- Mot: lamour, Mots similaires: ['paix', 'puissant', 'sagesse', 'le', 'lamour’]
- Mot: fêtes, Mots similaires: ['organisent', 'sagesse', 'célébrations', 'fleurs', 'fêtes’]
- Mot: fleurs, Mots similaires: ['sont', 'célébrations', 'oiseaux', 'fêtes', 'fleurs']

**Lien parfois moins logique**

- Mot: jardin, Mots similaires: ['avec', 'passer', 'gouvernent', 'ensemble', 'jardin’]
- Mot: oiseaux, Mots similaires: ['bonté', 'unis', 'belle', 'célébrations', 'oiseaux’]
- Mot: promènent, Mots similaires: ['de', 'unis', 'bonté', 'oiseaux', 'promènent']

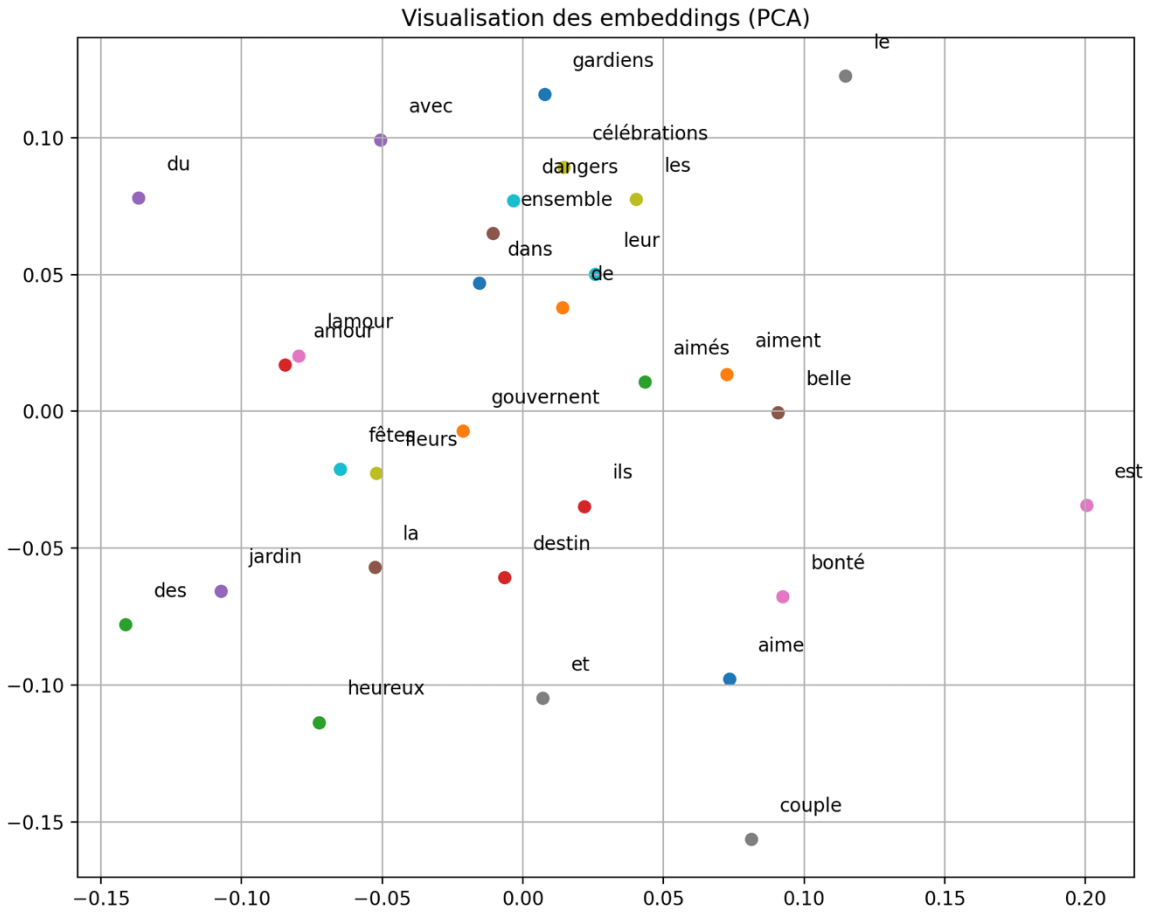
# Les tests réalisés – Les embeddings

- Ensuite nous avons réalisé un test sur skipgramp sans optimisation
- Dans un premier temps nous avons visualisé avec TSNE pour voir la différence avec cbow
- Puis nous avons testé la visualisation avec PCA pour voir les différences



# Les tests réalisés – Les embeddings

- Ensuite nous avons vérifié la différence des résultats entre cbow et skipgram en regardant les résultats de notre cosinus de similarité



# Les test réalisés - Cosinus simarity skipgramp

**cbow**

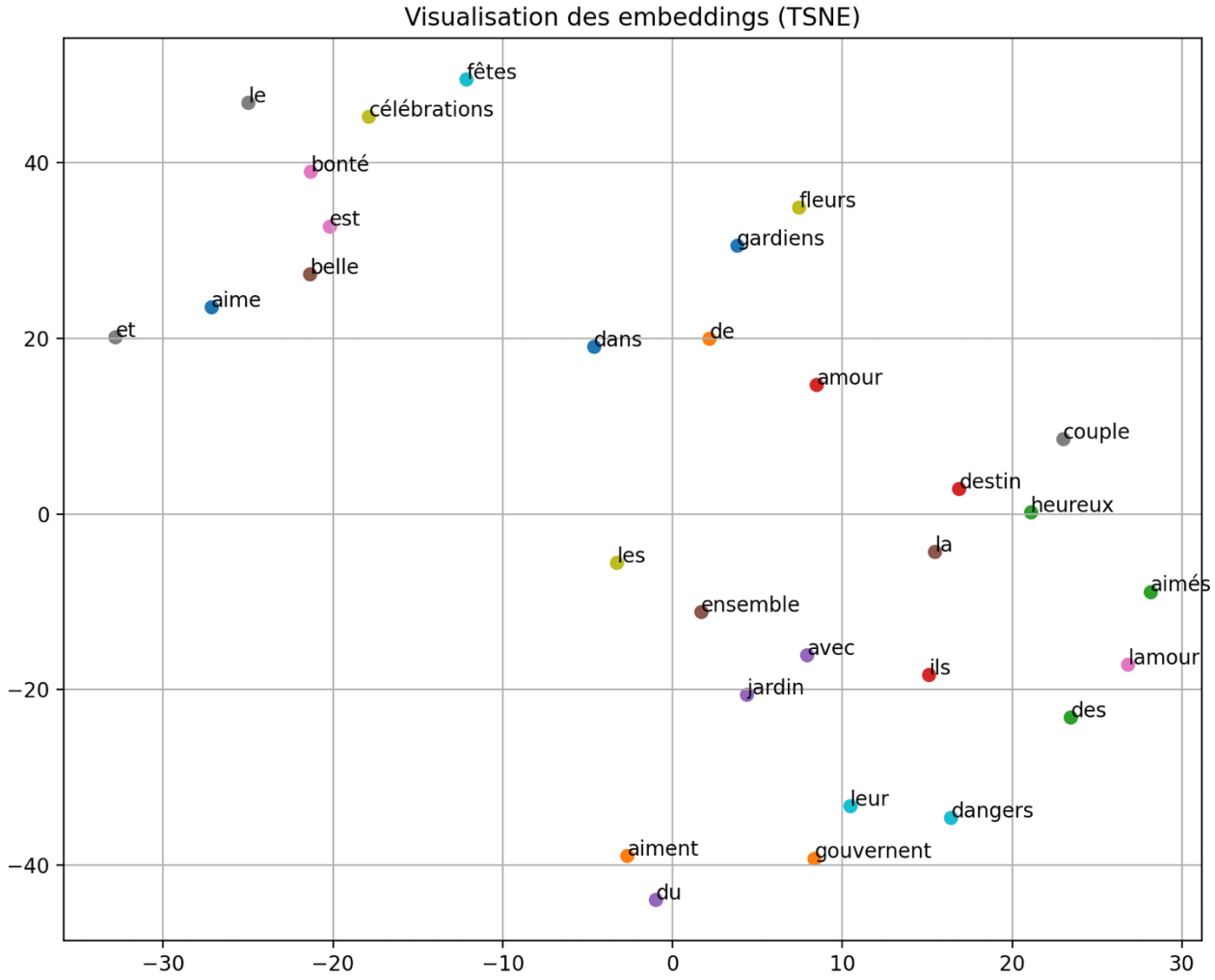
- Mot: royaume, Mots similaires: ['protègent', 'peuple', 'des', 'ils', 'royaume’]
- Mot: éternel, Mots similaires: ['un', 'liés', 'par', 'amour', 'éternel’]
- Mot: liés, Mots similaires: ['heureux', 'par', 'royal', 'couple', 'liés’]
- Mot: lamour, Mots similaires: ['paix', 'puissant', 'sagesse', 'le', 'lamour’]
- Mot: fêtes, Mots similaires: ['organisent', 'sagesse', 'célébrations', 'fleurs', 'fêtes’]
- Mot: fleurs, Mots similaires: ['sont', 'célébrations', 'oiseaux', 'fêtes', 'fleurs']

**skipgramp**

- Mot: royaume, Mots similaires: ['unis', 'paix', 'les', 'aimés', 'roya
- Mot: éternel, Mots similaires: ['royal', 'les', 'de', 'lharmonie', 'éternel’]
- Mot: liés, Mots similaires: ['royal', 'et', 'jardin', 'fêtes', 'liés’]
- Mot: lamour, Mots similaires: ['avec', 'protègent', 'ensemble', 'jardin', 'lamour’]
- Mot: fêtes, Mots similaires: ['du', 'et', 'liés', 'roi', 'fêtes’]
- Mot: fleurs, Mots similaires: ['un', 'reine', 'se', 'peuple', 'fleurs']

# Les tests réalisés - Embeddings

- Dans cette partie nous avons ajouté l'optimisation subsampling pour différencier nos résultats avec cbow



# Les tests réalisés – Cosinus similarity cbow

**Résultat sans subsampling**

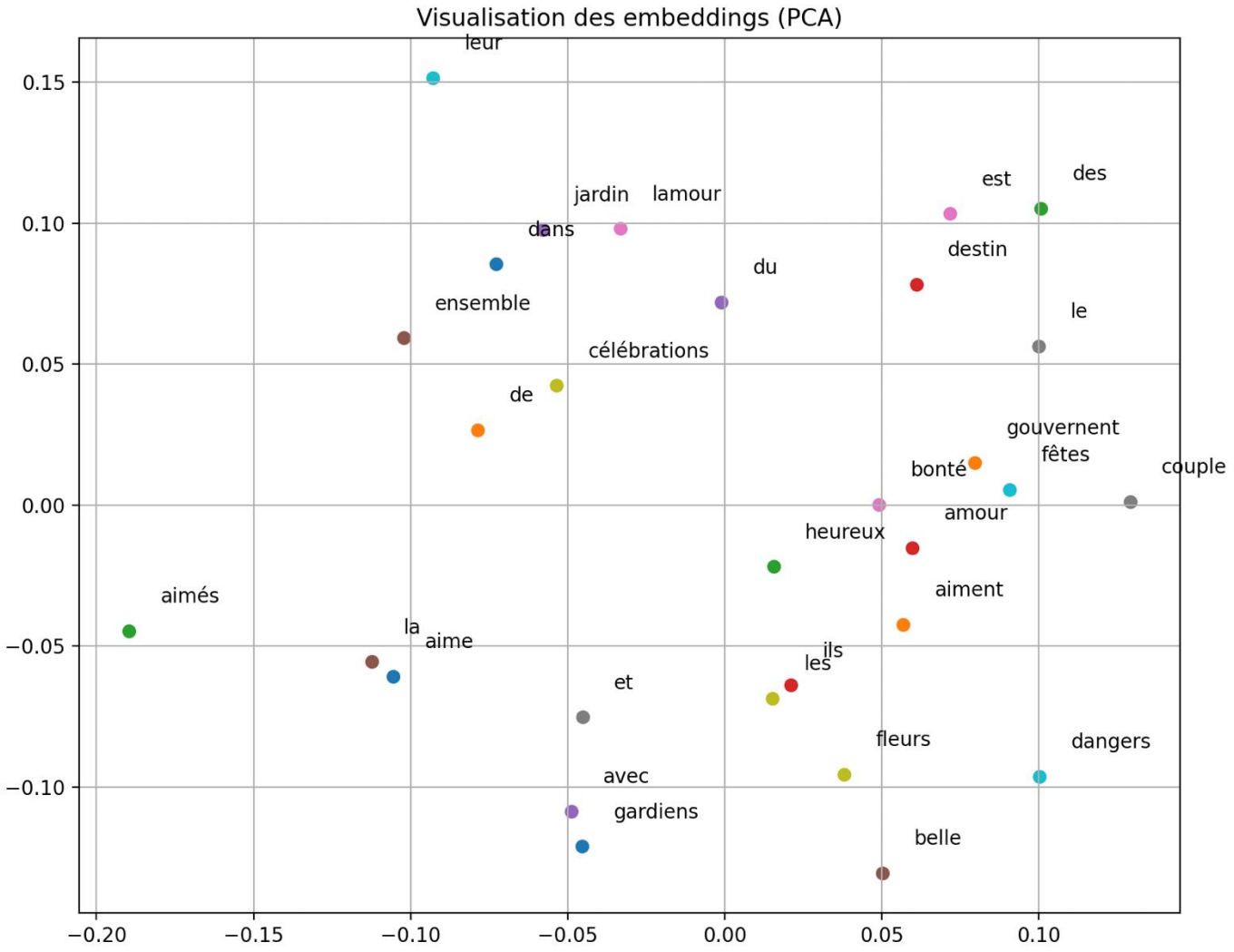
- Mot: royaume, Mots similaires: ['protègent', 'peuple', 'des', 'ils', 'royaume’]
- Mot: éternel, Mots similaires: ['un', 'liés', 'par', 'amour', 'éternel’]
- Mot: liés, Mots similaires: ['heureux', 'par', 'royal', 'couple', 'liés’]
- Mot: lamour, Mots similaires: ['paix', 'puissant', 'sagesse', 'le', 'lamour’]
- Mot: fêtes, Mots similaires: ['organisent', 'sagesse', 'célébrations', 'fleurs', 'fêtes’]
- Mot: fleurs, Mots similaires: ['sont', 'célébrations', 'oiseaux', 'fêtes', 'fleurs']

**Avec**

- Mot: royaume, Mots similaires: ['paix', 'peuple', 'protègent', 'ils', 'royaume’]
- Mot: éternel, Mots similaires: ['ensemble', 'est', 'unis', 'amour', 'éternel’]
- Mot: liés, Mots similaires: ['royal', 'heureux', 'destin', 'couple', 'liés’]
- Mot: lamour, Mots similaires: ['peuple', 'protègent', 'aimés', 'des', 'lamour’]
- Mot: fêtes, Mots similaires: ['roi', 'reine', 'puissant', 'célébrations', 'fêtes’]
- Mot: fleurs, Mots similaires: ['oiseaux', 'se', 'reine', 'gardiens', 'fleurs']

# Les tests réalisés - Embeddings

- Dans cette partie nous avons ajouté l'optimisation subsampling pour différencier nos résultats sur l'embedding skipgram



# Les test réalisés - Cosinus simarity skipgramp

**Skipgramp sans subsampling**

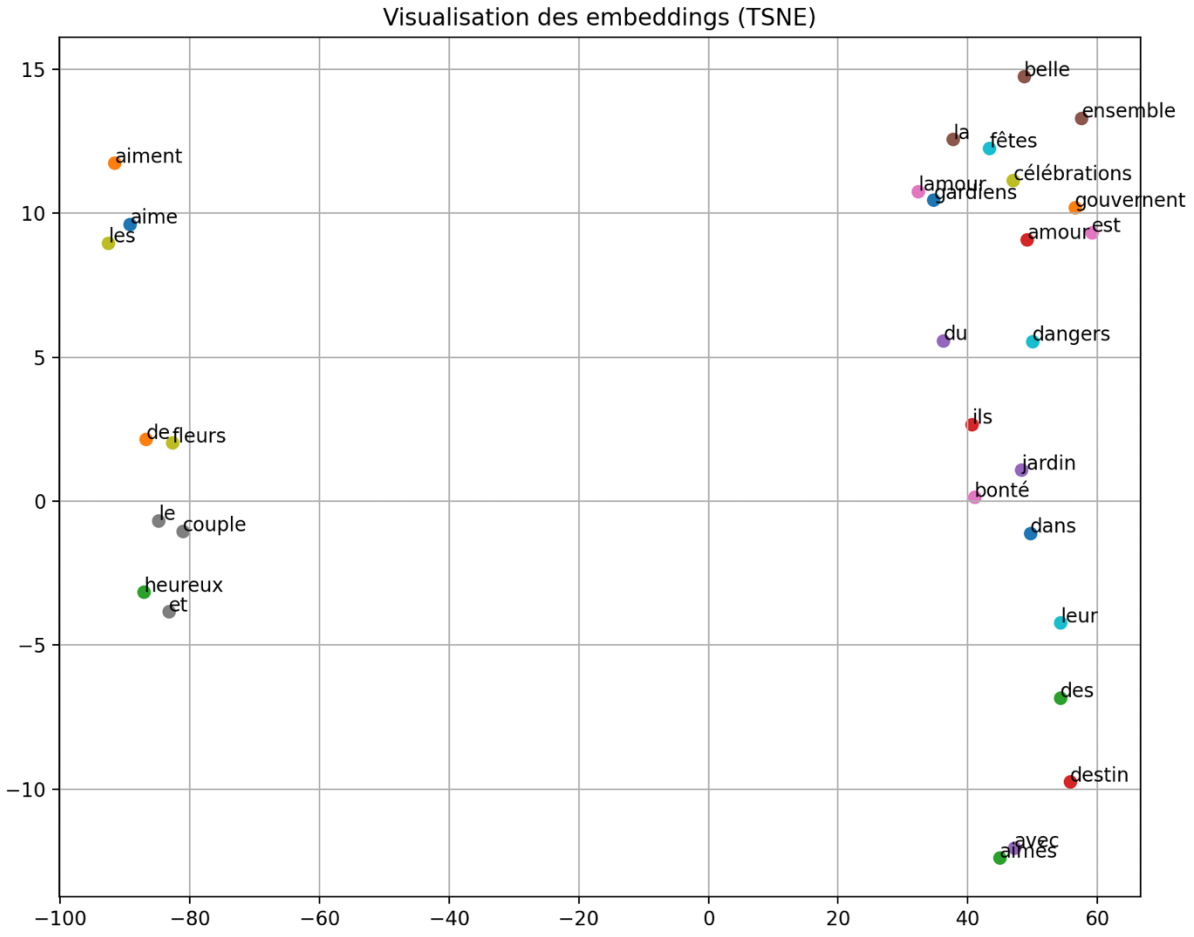
- Mot: royaume, Mots similaires: ['unis', 'paix', 'les', 'aimés', 'roya
- Mot: éternel, Mots similaires: ['royal', 'les', 'de', 'lharmonie', 'éternel’]
- Mot: liés, Mots similaires: ['royal', 'et', 'jardin', 'fêtes', 'liés’]
- Mot: lamour, Mots similaires: ['avec', 'protègent', 'ensemble', 'jardin', 'lamour’]
- Mot: fêtes, Mots similaires: ['du', 'et', 'liés', 'roi', 'fêtes’]
- Mot: fleurs, Mots similaires: ['un', 'reine', 'se', 'peuple', 'fleurs']

**Skipgramp avec subsampling**

- Mot: royaume, Mots similaires: ['la', 'sont', 'un', 'lharmonie', 'royaume']
- Mot: éternel, Mots similaires: ['fêtes', 'dans', 'lamour', 'roi', 'éternel’]
- Mot: liés, Mots similaires: ['heureux', 'bonté', 'aiment', 'éternel', 'liés’]
- Mot: lamour, Mots similaires: ['roi', 'jardin', 'éternel', 'des', 'lamour’]
- Mot: fêtes, Mots similaires: ['souverains', 'le', 'passer', 'couple', 'fêtes’]
- Mot: fleurs, Mots similaires: ['fêtes', 'belle', 'oiseaux', 'promènent', 'fleurs']

# Les tests réalisés - Embeddings

- Dans cette partie nous avons ajouté l'optimisation subsampling et negative pour différencier nos résultats avec cbow
- On peut constater une limitation du au peu de jeu de donnée que nous avons utilisé pour tester la fonction



# Les tests réalisés – Cosinus similarity cbow

**Résultat sans negative sampling**

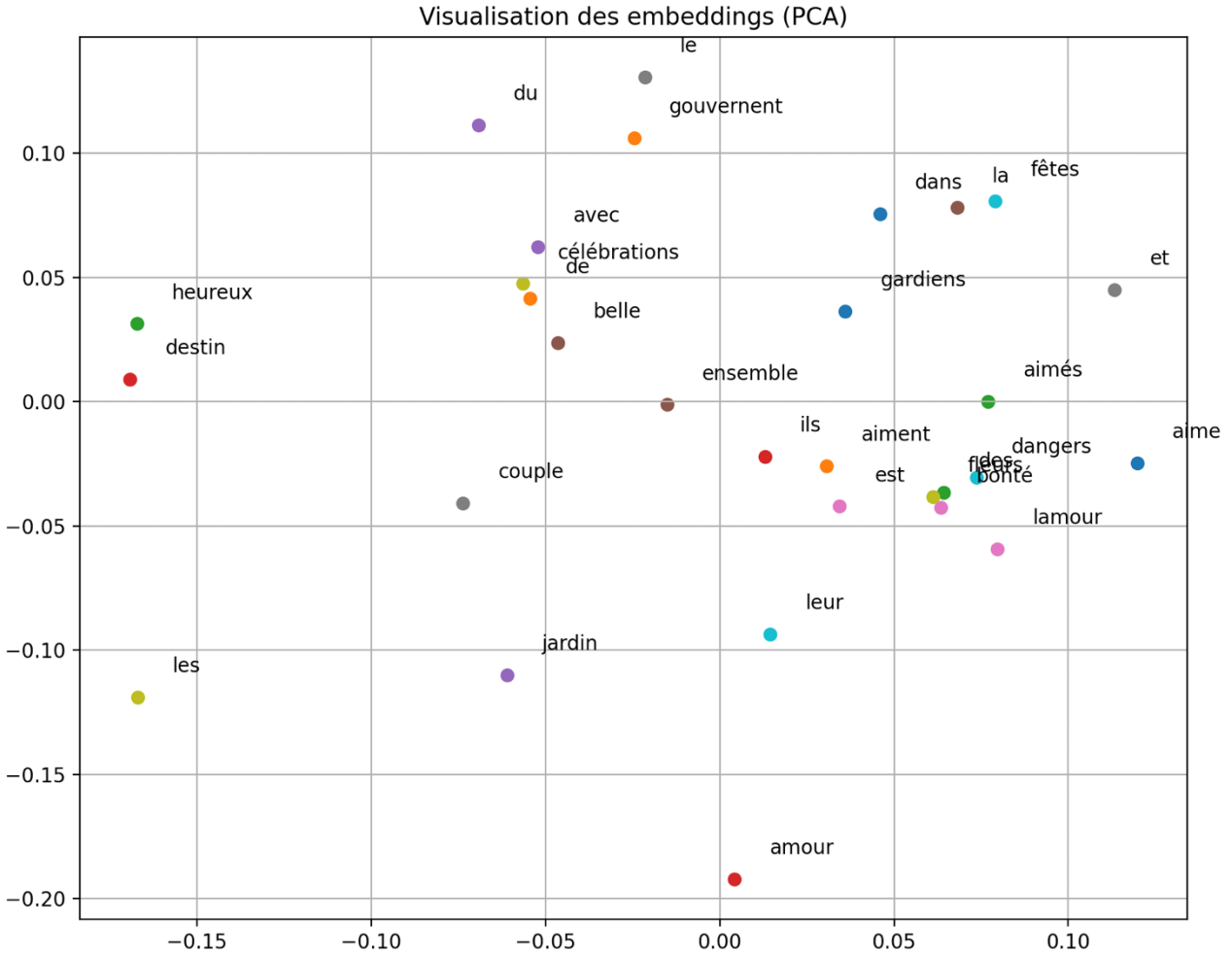
- Mot: royaume, Mots similaires: ['protègent', 'peuple', 'des', 'ils', 'royaume’]
- Mot: éternel, Mots similaires: ['un', 'liés', 'par', 'amour', 'éternel’]
- Mot: liés, Mots similaires: ['heureux', 'par', 'royal', 'couple', 'liés’]
- Mot: lamour, Mots similaires: ['paix', 'puissant', 'sagesse', 'le', 'lamour’]
- Mot: fêtes, Mots similaires: ['organisent', 'sagesse', 'célébrations', 'fleurs', 'fêtes’]
- Mot: fleurs, Mots similaires: ['sont', 'célébrations', 'oiseaux', 'fêtes', 'fleurs']

**Avec**

- Mot: royaume, Mots similaires: ['heureux', 'symbole', 'amour', 'ensemble', 'royaume’]
- Mot: éternel, Mots similaires: ['aimés', 'est', 'lamour', 'promènent', 'éternel']Mot: liés,
- Mot: liés, Mots similaires: ['gardiens', 'destin', 'la', 'bonté', 'liés’]
- Mot: lamour, Mots similaires: ['se', 'éternel', 'passer', 'gardiens', 'lamour’]
- Mot: fêtes, Mots similaires: ['ensemble', 'la', 'célébrations', 'oiseaux', 'fêtes’]
- Mot: fleurs, Mots similaires: ['et', 'peuple', 'sont', 'lharmonie', 'fleurs']

# Les tests réalisés - Embeddings

- Dans cette partie nous avons ajouté l'optimisation subsampling et negative pour différencier nos résultats avec skipgram
- On peut constater que l'embedding généré est deja plus interessant



# Les test réalisés - Cosinus simarity skipgramp

**Skipgramp sans negative sampling**

- Mot: royaume, Mots similaires: ['unis', 'paix', 'les', 'aimés', 'roya
- Mot: éternel, Mots similaires: ['royal', 'les', 'de', 'lharmonie', 'éternel’]
- Mot: liés, Mots similaires: ['royal', 'et', 'jardin', 'fêtes', 'liés’]
- Mot: lamour, Mots similaires: ['avec', 'protègent', 'ensemble', 'jardin', 'lamour’]
- Mot: fêtes, Mots similaires: ['du', 'et', 'liés', 'roi', 'fêtes’]
- Mot: fleurs, Mots similaires: ['un', 'reine', 'se', 'peuple', 'fleurs']

**Skipgramp avec negative sampling**

- Mot: royaume, Mots similaires: ['heureux', 'symbole', 'amour', 'ensemble', 'royaume’]
- Mot: éternel, Mots similaires: ['aimés', 'est', 'lamour', 'promènent', 'éternel']Mot: liés,
- Mot: liés, Mots similaires: ['gardiens', 'destin', 'la', 'bonté', 'liés’]
- Mot: lamour, Mots similaires: ['se', 'éternel', 'passer', 'gardiens', 'lamour’]
- Mot: fêtes, Mots similaires: ['ensemble', 'la', 'célébrations', 'oiseaux', 'fêtes’]
- Mot: fleurs, Mots similaires: ['et', 'peuple', 'sont', 'lharmonie', 'fleurs']

# Test réalisés – Embeddings avec 20 newsGroups

Entrainements avec 10% du dataset pour tester la pipeline complète avec la classification des sentiments.

Entrainements avec 30% du dataset pour réaliser un comparatif

Entrainements du modèle d'embedding de Google afin de faire une comparaison

# Test réalisés – Résultats attendus normalement

| Aspect | **CBOW** | **SkipGram** |
|---|---|---|
| **Vitesse d'entraînement** | Plus rapide (car moyenne du contexte) | Plus lent (un pair à la fois) |
| **Besoin en données** | Besoin de plus de données pour de bonnes représentations | Meilleur avec peu de données |
| **Performance avec peu de données** | Moins bon | Meilleur |
| **Qualité des embeddings pour mots rares** | Moins bon | Meilleur |

# Test réalisés – Embeddings avec 20 newGroups avec la classification et IMDB

- cbow

| Pourcentage de données utilisés | Accuracy | Loss | Val_accuracy | Val_loss | Test_accuracy |
|---|---|---|---|---|---|
| 10 % | 0.5945 | 0.6642 | 0.5764 | 0.6759 | 0.58 |
| 30 % | 0.7013 | 0.5683 | 0.6762 | 0.5974 | 0.67 |

- skipgram

| Pourcentage de données utilisés | Accuracy | Loss | Val_accuracy | Val_loss | Test_accuracy |
|---|---|---|---|---|---|
| 10 % | 0.6608 | 0.6152 | 0.6585 | 0.6129 | 0.65 |
| 30 % | 0.6727 | 0.6043 | 0.6732 | 0.6038 | 0.68 |

# Test réalisés – Embeddings avec 20 newGroups et IMDB

- Cbow a pris 4h pour 10% du dataset et environ 12h pour 30%
- Skipgram a pris 4h environ pour 10% et pour 30% 20h
- Les résultats obtenus lors de l'entrainement sont cohérants avec ce qui était attendu initialement d'après la théorie
- La qualité des embeddings à un impact significatif sur la qualité de la classification

# Test réalisés – Avec Embedding de Google

| Pourcentage de données utilisés | Accuracy | Loss | Val_accuracy | Val_loss | Test_accuracy |
|---|---|---|---|---|---|
| 10 % | 0.8289 | 0.3857 | 0.8399 | 0.3627 | 0.83 |
| 30 % | 0.8334 | 0.3794 | 0.8331 | 0.3709 | 0.84 |
| 100 % | 0.8330 | 0.3817 | 0.8439 | 0.3547 | 0.84 |

On peut donc conclure que la qualité de l'embeddings a un impact très important sur la qualité de la classification.
On peut aussi remarqué que le nombre de mot de vocabulaire n'a pas vraiment d'impact sur la qualité de la classification.

# Test réalisés – Avec Embedding de Google

- Réalisation de test sur l'importance des paramètres dans l'entrainement pour la classification
- Conclusion :
  - Batch_size entre 16 et 64 meilleurs que des grands batch_size
  - Input_lenght très important peut faire varier énormément les résultats de la classification, il faut privilégié un grand input_lenght, plus on met de mots dans notre ANN plus on peut récupérer facilement le contexte (variation jusqu'à 15% de performances en plus)

# Difficultés rencontrées et limitations

## Limitations rencontrées :

Problèmes de mémoire RAM pour des dataset avec 20 newReviews pour plus de 10% du dataset pour l'entrainement des embeddings

Problèmes de temps d'entrainements des embeddings

## Difficultés rencontrées :

Résultats décevant dans un premier temps avec l'embedding de google pour la classification

Problèmes de surapprentissage des le début de l'entrainement pour la classification

# Solutions appliquées - Problèmes de mémoire RAM

- Problème :
  - Mémoire RAM limité ne pouvant pas contenir tout le dataset à plus de 10%
- Solution appliquée :
  - Mise en place dans chaque class d'embedding d'une fonction génératrice permettant de **fournir en continu des mini-lots (batches)** de données pour entraîner le modèle

# Solutions appliquées - Problèmes de temps d'entrainements des embeddings

**Problèmes de temps d'entrainement :**

- Temps excrément long pour l'entrainement même avec un dataset diminué

**Solution appliquée :**

- Mise en place d'un système de multithearding pour des calculs parallèles sur notre CPU

# Solutions appliquées - Résultats décevant avec l'embedding de google pour la classification

## Problème :

- Résultats décevant avec l'embedding de google pour la classification dans un premier temps

## Solution appliquée :

- Fine-tunning du modèle de classification à l'aide des paramètres comme le batch_size et input_length

# Solution appliquées - surapprentissage des le début de l'entrainement pour la classification

- Problème :
  - Problèmes de surapprentissage des le début de l'entrainement pour la classification
- Solution appliquée :
  - Modification de la class ANNclassifier en ajoutant des DropOut
  - Ajout d'une méthode earlyStopping
  - Placement d'une couche Batch Normalization afin de stabilise les activations intermédiaires

# Fin

Avez-vous des questions ?