Rapport Projet Sciences des données : Exploiting word embeddings for machine translation

CôngMinh Dinh, Louis Monier, Maxence Philbert, Vincent Gouteux

Novembre 2019

Contents

1 Présentation du Problème et des donées **1**

2 Méthode supervisée **2**
2.1 Entraînement du modèle . . . 2
2.1.1 Descente de Gradient (GD) . . . 3
2.1.2 Descente de Gradient Stochastique (SGD) et mini-batch (BGD) . . . 3
2.1.3 Contrainte d'orthogonalité . . . 3
2.1.4 Résolution du système . . . 4
2.1.5 Résultats phase d'entraînement : . . . 4
2.2 Test du modèle et résultats . . . 5
2.3 Test sur d'autres langues . . . 7
2.3.1 Italien - Anglais . . . 7
2.3.2 Vietnamien - Anglais . . . 7

3 Méthode non supervisée **9**
3.1 Discriminant . . . 9
3.2 Générateur . . . 9
3.3 Algorithme et fonction de perte . . . 10
3.4 Similarité transversale mise à l'échelle locale (CSLS) . . . 10
3.5 Résultats obtenus par la méthode non supervisée . . . 10

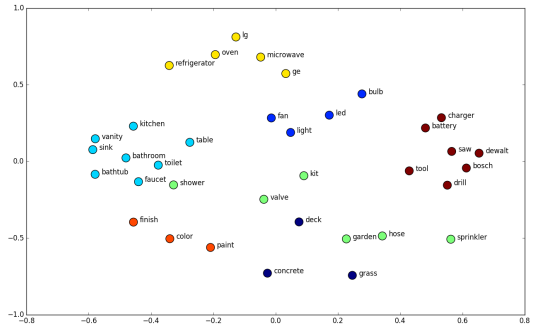
1 Présentation du Problème et des donées

Dans ce projet, nous nous intéressons à la construction d'un traducteur mot à mot fondé sur des embeddings. Nous allons voir deux méthodes qui permettent de résoudre ce problème : une méthode supervisée et une non-supervisée.

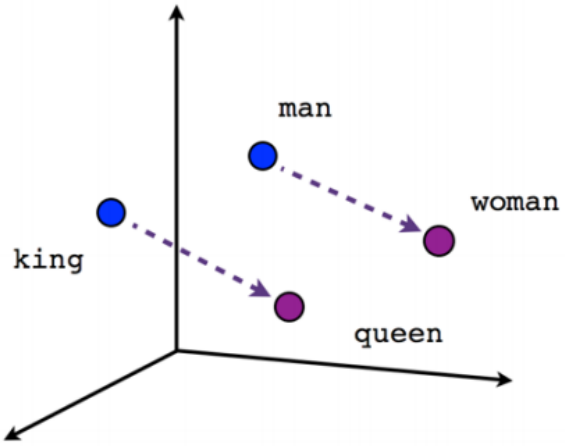
Nous utiliserons les jeux de données de la librairie MUSE mis à disposition par Facebook, plus précisément les embeddings FastText issus de Wikipedia (lien : Monolingual embeddings).

Les *Word Embeddings*, ou "plongement des mots" en français, est une méthode d'apprentissage très utilisée dans le domaine du traitement automatique du language (NLP). Elle consiste à représenter un mot par un vecteur de nombre réels, en prenant en compte son analyse sémantique (c'est-à-dire les contextes dans lesquels les mots apparaissent). Autrement dit, les "Word Embeddings" fournissent des représentations vectorielles de telle sorte à ce que les mots ayant un sens similaire, ou apparaissant dans des contextes similaires, soient relativement "proches" dans l'espace vectoriel. La forme vectorielle permet d'utiliser l'arsenal mathématique approprié afin d'étudier et résoudre de nombreuses problématiques du NLP.

Les mots sont reliés vectoriellement par des relations très simples. Ce fait marquant a été mis en évidence par T. Mikolov, Quoc V. Le et Ilya Sutskever dans "Exploiting Similarities among Languages for Machine Translation" (2013). L'exemple le plus couramment cité est : "King" + "Femme" - "Homme" = "Queen" (voir Fig 1.a et 1.b).



(a) Représentation vectorielle 2D de mots



(b) Transformations algébriques

Figure 1

2 Méthode supervisée

Dans un premier temps, nous avons implémenté l'approche supervisée fondée sur la création d'une matrice de translation W reliant un langage source à un langage cible (français vers l'anglais par exemple). Supposons que nous disposions d'un échantillon de mots à traduire et de leur traduction, ainsi que de leurs vecteurs représentatifs associés $\{(x_i, z_i), \forall i = 1, \ldots, n\}$ avec $x_i \in \mathbb{R}^{d1}$ l'embedding du mot i dans le langage source, et $z_i \in \mathbb{R}^{d2}$ l'embedding de sa traduction (dans le langage cible).

L'objectif est de trouver une matrice de traduction W de telle sorte que Wx_i approxime z_i, ce qui revient à résoudre le problème d'optimisation suivant :

$$\min_W C(W) = \min_W \sum_{i=1}^{n} \|Wx_i - z_i\|^2$$

Dans la prochaine section, nous détaillons plusieurs méthodes que nous avons développé pour résoudre ce problème. Pour chaque méthode implémentée, nous donnons un aperçu du principe général. Nous discuterons également certaines modifications apportées au modèle pour améliorer les performances.

2.1 Entraînement du modèle

Comme prescrit dans l'article de T. Mikolov, nous avons d'abord implémenté une résolution par descente de gradient. Par curiosité, et puisque c'était un bon exercice, nous avons implémenté les différences déclinaisons de la descente de gradient : classique (GD), stochastique (SGD) et mini-batch (BGD). Nous proposons également de calculer directement la solution en résolvant le système $W = {x_i}^{-1} z_i$.

2.1.1 Descente de Gradient (GD)

Nous avons premièrement calculé le gradient de la fonction de coût à minimiser, puis implémenté la méthode de descente de gradient grâce les formules détaillées ci-dessous (pour k étapes) :

- $W_{k+1} = W_k - \eta \frac{\partial C(W)}{\partial W}$
- $\frac{\partial C(W)}{\partial W} = 2 \times \sum_{i=1}^{n} (Wx_i - z_i)x_i^T$

2.1.2 Descente de Gradient Stochastique (SGD) et mini-batch (BGD)

En général, le calcul du gradient est très coûteux quand on travaille sur des matrice de très grande taille. Il est donc possible d'effectuer des méthodes assez similaires mais moins coûteuses. Notre descente de gradient marchait efficacement mais nous avons quand même décidé d'implémenter ces deux autres méthodes pour voir leur performances.

2.1.3 Contrainte d'orthogonalité

Nous avons également étudié une variante de notre modèle, en ajoutant une contrainte d'orthogonalité à notre matrice W, ce qui revient à résoudre le problème d'optimisation suivant :

$$\underset{W \in O_d(\mathbb{R})}{argmin} \sum_{i=1}^{n} \|Wx_i - z_i\|^2$$

W est cette fois une matrice orthogonale O. On peut donc réécrire le problème comme :

$$\underset{O}{max} \sum_{i=1}^{n} z_i^T O x_i$$

avec la contrainte d'orthogonalité suivante :

$$O^T O = Id$$

La fonction de coût définie ci-dessus est minimisée par :

$$O = UV^T$$

où U and V sont obtenus en effectuant la SVD (Singular Value Decomposition) de $Y^T X$.

Les fondements théoriques à l'origine de contrainte d'orthogonalité sur W sont donnés par : Chao Xing, Dong Wang, Chao Liu et Yiye Lin dans leur article "Normalized Word Embedding and Orthogonal Transform for Bilingual Word Translation" (2015). Intuitivement, cette contrainte est intéressante puisqu'elle permet d'assurer que la distance entre 2 points n'est pas changée par la transformation linéaire. En effet, la norme euclidienne issue de notre fonction objective et la similarité (en cosinus) sont deux mesures fondamentalement différentes mathématiquement. L'orthogonalité de W permet d'assurer que les distances sont conservées, autrement dit que la transformation est une isométrie. Les deux espaces (source et cible) sont donc alignés alors qu'une rotation aléatoire de l'un par rapport à l'autre était possible. En définitive, l'orthogonalité résout cette incohérence et permet d'obtenir une meilleure accuracy.

2.1.4 Résolution du système

Nous avons implémenté ces descentes de gradient pour ne pas sur-apprendre notre échantillon d'entraînement. Cependant nous avons par curiosité calculé la matrice W qui apprenait par coeur les données test et résolvait directement le système $Wx_i = Z_i$ pour tout mot de notre ensemble d'entraînement. Nous avons pensé dans un premier temps que les résultats avec cette matrice seraient moins bons qu'avec une descente de gradient cependant pendant la phase de test il s'est avéré que cette méthode était plus efficace.

2.1.5 Résultats phase d'entraînement :

Choix de parametres

D'abord, nous avons effectués quelques tests sur le pas de la descente (η) de gradient afin de trouver celui qui rendrait l'apprentissage optimal (voir Figures 2 - 3) :

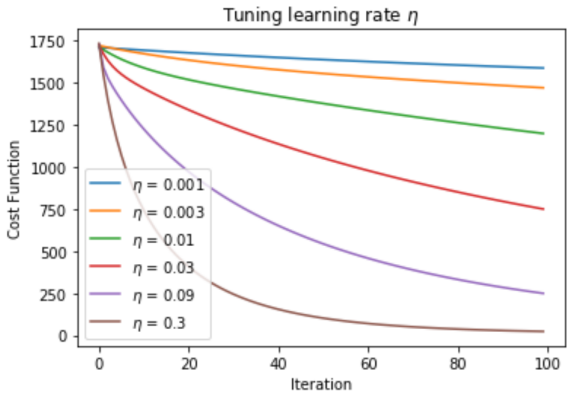
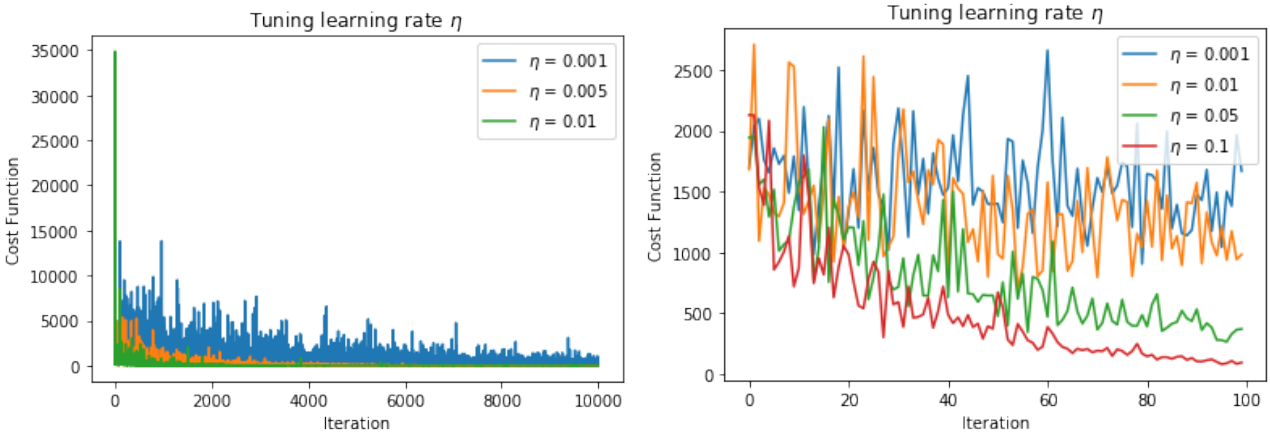


Figure 2: GD

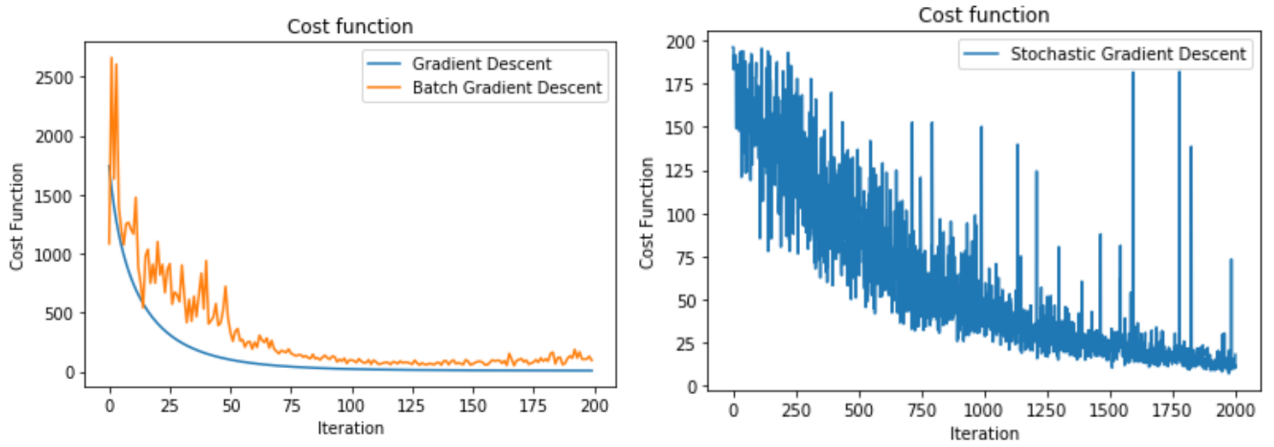


(a) SGD (b) BGD

Figure 3

Fonction de coût

Afin de tester la cohérence de ces méthodes, nous avons tracé la fonction de coût en fonction du nombre d'itérations pour ces trois méthodes (voir Figure 4).



(a) Fonction de coût selon le nombre d'itérations (b) Fonction de coût selon le nombre d'itérations

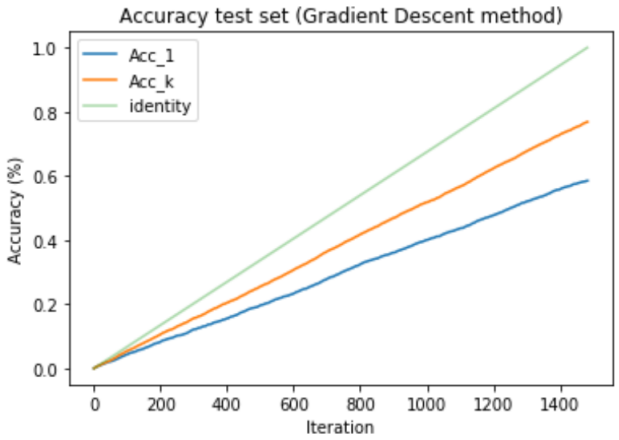
Figure 4

2.2 Test du modèle et résultats

Pour tester le modèle nous avons défini la formule de similarité entre deux embeddings qui indique si deux mots sont proches ou non. Si les mots sont proches on obtiendra un similarité proche de 1, à l'inverse, si il ne le sont pas on obtiendra une similarité très faible voire négative.

$$sim(x_i, z_i) = \frac{< x_i, z_i >}{||x_i|| * ||z_i||}$$

Pour mesurer la précision du modèle nous calculons Wx_i et nous regardons dans le dictionnaire cible si le mot avec la similarité la plus haute correspond bien a la traduction. Si c'est le cas on incrémente le nombre de "bonnes traductions" notre précision finale est donc le nombre de bonne prédictions sur le taille de notre dictionnaire de test. Nous obtenons des résultats satisfaisant et similaires a ceux des papiers dont nous nous sommes inspirés. Il s'avère que la technique ou on résoud tout simplement le système $Wx_i = z_i$ pour chaque mot de notre jeux d'entraînement ne fait pas de sur apprentissage et obtient des résultats très satisfaisants




```
Gradient descent method :
Final accuracy @1 = 58.46 %
Final accuracy @5 = 76.8 %

Analytical method :
Final accuracy @1 = 60.22 %
Final accuracy @5 = 77.14 %
```

(a) Résultats de la traduction Francais - Anglais

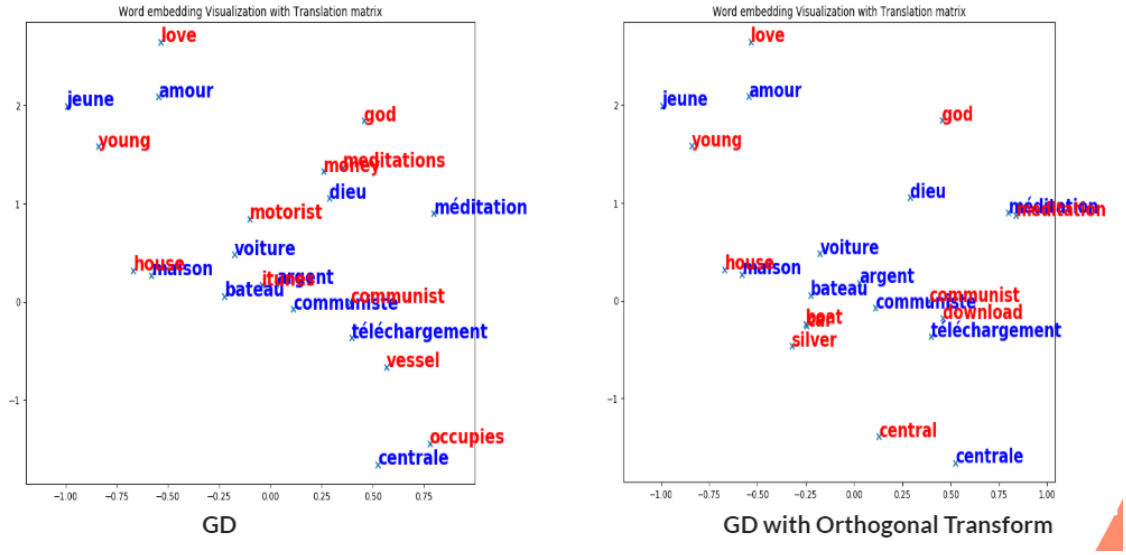
(b)

Nous pouvons alors remarquer que les résultats obtenus avec la contrainte d'orthogonalité sur W sont meilleurs :

```
Gradient descent method :
Final accuracy @1 = 58.46 %
Final accuracy @5 = 76.8 %

Orthogonal method :
Final accuracy @1 = 62.58 %
Final accuracy @5 = 78.96 %
```

Nous avons représenté et comparé les embeddings obtenus dans le cas de W classique et W orthogonale à l'aide d'une ACP.
Sur l'image de visualisation nous avons confirme que les traductions sont plus précises, les embeddings d'un mot et de sa traduction sont plus "proches", par exemple pour les mots 'meditation', 'central'...



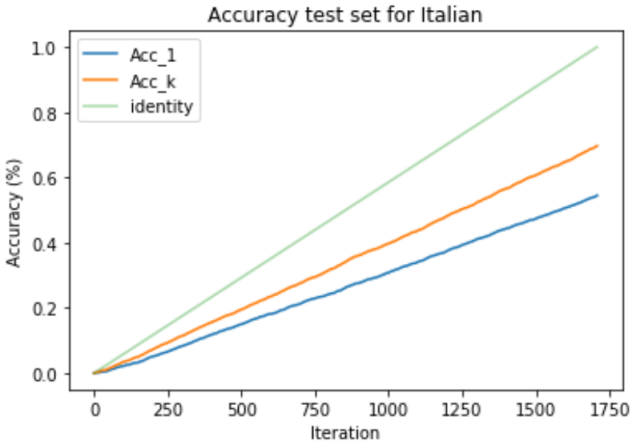
(a) Visualisation des embeddings : comparaison entre W et W_{ortho}

2.3 Test sur d'autres langues

Notre traducteur supervisé semble performant sur la traduction du français à l'anglais. Pour tester ses performances globales nous avons décidé de le tester sur d'autres langues : une autre langue européenne plutôt proche de l'anglais : l'italien et une langue plus éloignée : le vietnamien.

2.3.1 Italien - Anglais

Nous avons donc implémenté la même méthode avec cette fois les embeddings italiens comme langage source



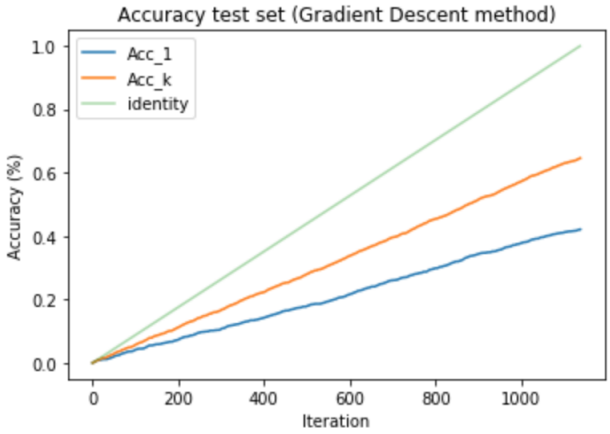

```
Analytical method :
Final accuracy @1 = 54.45 %
Final accuracy @5 = 69.67 %
```

(a) Résultats de la traduction Italien - Anglais

2.3.2 Vietnamien - Anglais

Après plusieurs tentatives pour obtenir la convergence, nous choisissons un η petit ($= 0.001$) dans le cas de la traduction vietnamienne. Nous avons atteint la convergence à la 3000ème itération. Le résultat obtenu ici n'est pas aussi bon que le français ou l'italien. Cela s'explique par le fait que la langue vietnamienne utilise souvent deux mots pour expliquer un mot en anglais. Notre jeu d'entraînement

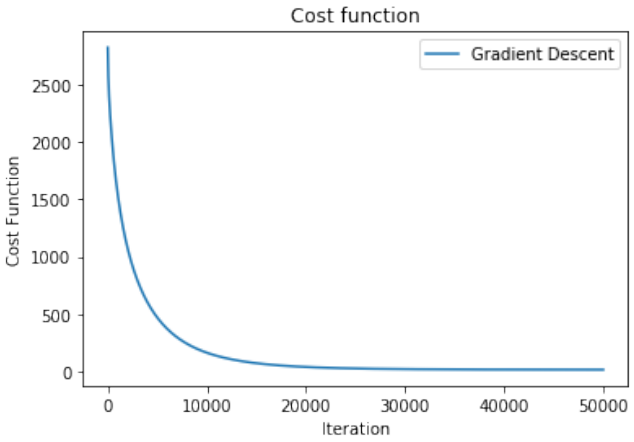
provient d'un jeu de données qui ne contient qu'un seul mot vietnamien, donc le résultat est assez limité. Cependant, nous obtenons tout de même un résultat acceptable.



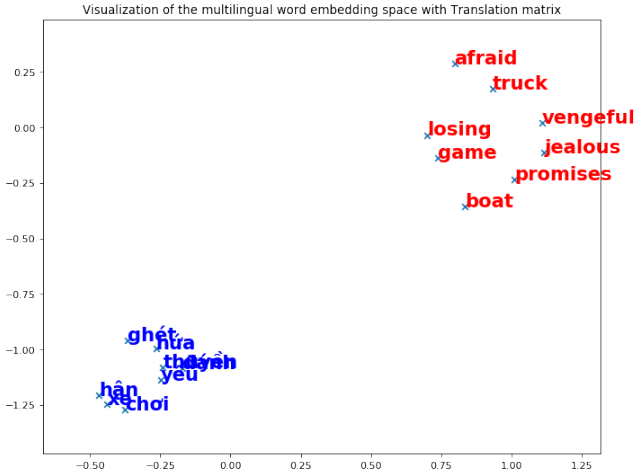

```
Gradient descent method :
Final accuracy @1 = 42.09 %
Final accuracy @5 = 64.59 %
```

(a) Résultats de la traduction Vietnamien - Anglais

Dans la figure de visualisation des mots ci-dessous, nous avons également remarqué une distance plus importante entre le vietnamien et l'anglais que dans le cas du français ou de l'italien. Par ailleurs, comme on pouvait le supposer, la différence sémantique entre l'anglais et le vietnamien peut également être un facteur justifiant le fait que notre modèle soit moins performant que pour les autres langues.



(a) Fonction de coût selon le nombre d'itérations



(b) La visualisation des mots (bleu: vietnamienne, rouge: anglais)

Être capable de déduire une matrice de traduction W sans connaître les traductions d'un langage vers un autre est un problème suscitant l'intérêt de la communauté du NLP. De nombreux papiers de recherche ont été publié récemment. L'objet de la prochaine partie est l'étude de l'un d'entre eux : "Word Translation Without Parallel Data" d'Alexis Conneau, Guillaume Lample, Marc Aurelio Ranzato, Ludovic Denoyer et Hervé Jégou (2017).

3 Méthode non supervisée

Pour cette partie, nous avons exploré la notion de réseaux génératifs d'adversaires (GAN) appliquée à la traduction de mots. Un GAN est un modèle génératif où deux réseaux sont placés l'un contre l'autre : ils sont adversaires. Le premier réseau est le **générateur** : il génère un échantillon "faux" (dans notre cas, il génère un embedding traduit, c'est-à-dire un Wx_i).
Tandis que l'autre réseau, le **discriminant** essaie de détecter si un échantillon est "réel" (= provenant du langage cible) ou s'il est le résultat du générateur (c'est-à-dire un Wx_i produit par le générateur). Le combat entre ces deux entités a pour objectif d'améliorer la qualité des échantillons produit par le générateur afin qu'ils soient suffisemment proche des "vrais" échantillons pour que le discriminant n'arrive plus à faire la différence.

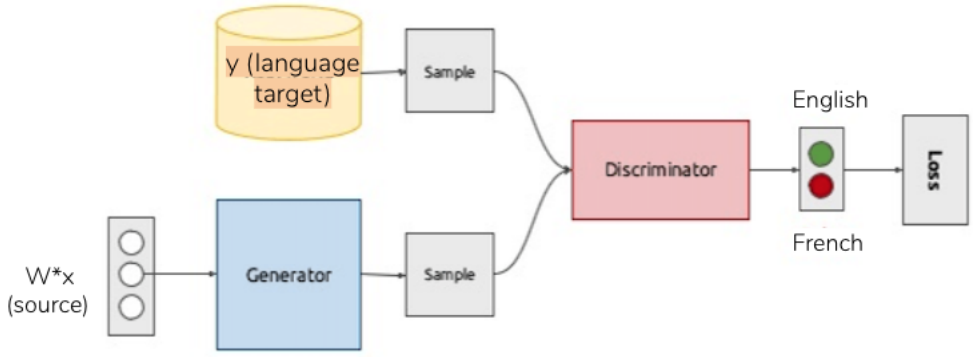


Figure 10: Architecture de notre GANs

3.1 Discriminant

L'objectif du modèle **discriminant** est de reconnaître si une donnée d'entrée est "réelle", c'est-à-dire appartient à l'ensemble de données d'origine - ou si elle est "fictive" = Wx (source et matrice de traduction entraînées) générée par le générateur. Pour nous, il s'agit donc d'un réseau de neurones à 2 couches cachées qui prend en entrée un embedding de taille 300 (dimension de l'espace vectoriel source) et qui renvoie la probabilité que cet embedding provienne du langage source. On doit donc minimiser la fonction de perte suivante :

$$\mathcal{L}_W(W|\theta_D) = -\frac{1}{n}\sum_{i=1}^{n} logP_{\theta_D}(source = 0|Wx_i) - \frac{1}{m}\sum_{i=1}^{m} P_{\theta_D}(source = 1|y_i)$$

3.2 Générateur

Dans notre cas, le générateur est, quant à lui, un "faux" réseau de neurones (avec 0 couche cachée) qui prend en entrée un vecteur x_i de taille 300 et renvoie un vecteur Wx_i de taille 300 également. Les poids du réseau sont en réalité les éléments de la matrice W recherchée. On doit donc minimiser la fonction de perte suivante :

$$\mathcal{L}_D(\theta_D|W) = -\frac{1}{n}\sum_{i=1}^{n} logP_{\theta_D}(source = 1|Wx_i) - \frac{1}{m}\sum_{i=1}^{m} P_{\theta_D}(source = 0|y_i)$$

3.3 Algorithme et fonction de perte

Nous utiliserons SGD comme algorithme d'optimisation pour les deux réseaux de neurones, avec un taux d'apprentissage de 0,1.
La fonction de perte que nous allons utiliser pour cette tâche est nommée Binary Cross-Entropy (ou perte BCE).

3.4 Similarité transversale mise à l'échelle locale (CSLS)

Nous voulons améliorer la métrique de comparaison de telle sorte que le voisin le plus proche d'un mot source ait plus de chances d'avoir comme voisin le plus proche, dans le langage target, ce mot source particulier. Les voisins les plus proches sont par nature asymétriques : y est un K-NN de x n'implique pas que x est un K-NN de y.
Nous considérons la similarité moyenne d'un source intégrant x_s à son voisinage cible :

$$r_T(Wx_s) = \frac{1}{K} \sum_{y_t \in \mathcal{N}_T(W,x_s)} cos(Wx_s, y_t)$$

Nous les utilisons pour définir une mesure de similarité CSLS entre les mots source "transformés" et les mots cibles :

$$CSLS(Wx_s, y_t) = 2cos(Wx_s, y_t) - r_T(Wx_s) - r_S(y_t)$$

Cette mesure, introduite dans la papier de Conneau, permet d'obtenir des accuracy plus élevées. Intuitivement, elle est assez intéressante, car elle permet de rapprocher les mots isolés et d'éloigner les mots très proches les uns des autres.

3.5 Résultats obtenus par la méthode non supervisée

Nous avons mis en place toute l'architecture du GAN en nous inspirant du tutoriel PyTorch sur les DCGAN. La structure à 2 réseaux de neurones adversaires fonctionne à partir d'un grand nombre d'itérations.

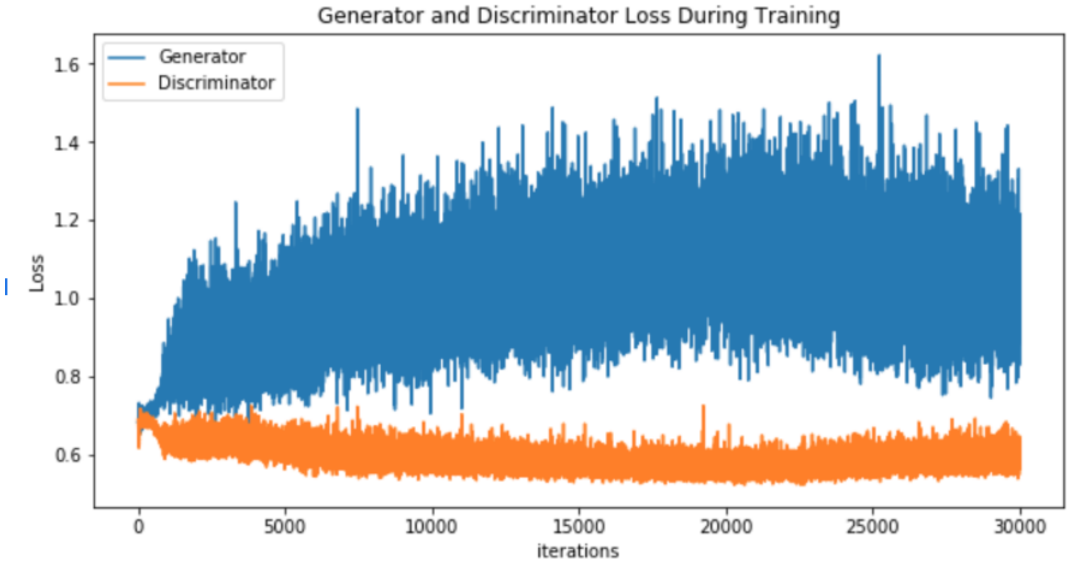
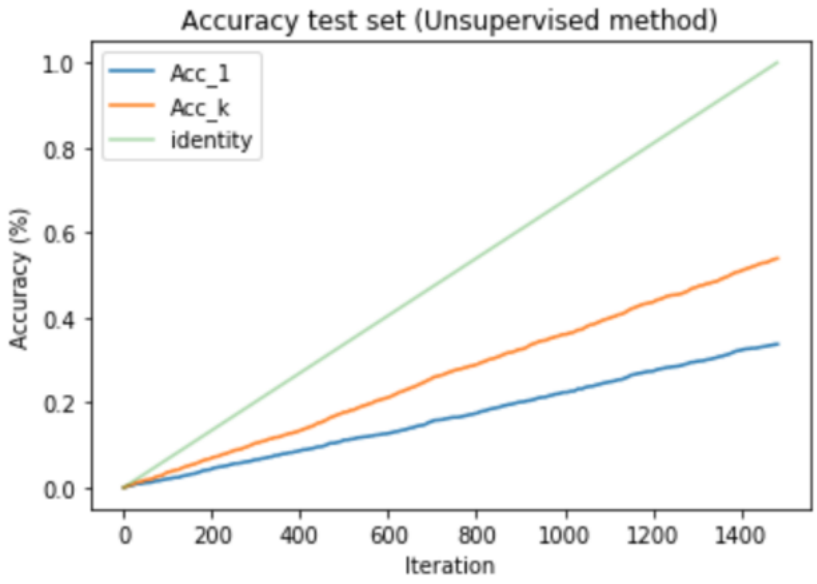


Figure 11: Fonction de pertes du générateur et du discriminant

Nous avons joué sur tous les paramètres du GAN, pour optimiser les résultats. Au bout de 30 000 itérations nous obtenons 34% de précision pour le top1 et 54% pour la top5. La méthode non supervisée ne donnait pas de résultats satisfaisant dans le premier notebook *Projet_NLP_Final.ipynb*, le code fonctionnel dont les résultats sont cités dans ce rapport a été ajouté dans un autre notebook

Unsupervised_correction.ipynb. Nous prions le lecteur du rapport de se référer au 2nd notebook pour visualiser le code de la méthode non-supervisée.




```
Unsupervised method :
Final accuracy @1 = 33.72 %
Final accuracy @5 = 53.88 %
```

Figure 12: Précision de notre modèle non supervisé

Enfin quelques tests sommaires sur des mots basiques montrent que les traductions obtenues sont satisfaisantes :

```
bateau --> traduction : ['sailboat', 'boat', 'boats', 'moored', 'mooring']
--
maison --> traduction : ['townhouse', 'cottage', 'upstairs', 'farmhouse', 'house']
--
argent --> traduction : ['gold', 'silver', 'bullion', 'bronze', 'valuables']
--
ordinateur --> traduction : ['computer', 'mainframe', 'workstation', 'virtualization', 'computers']
--
dieu --> traduction : ['god', 'divine', 'gods', 'almighty', 'deity']
--
```

Figure 13: Quelques tests sommaires pour vérifier les traductions obtenues par la méthode non-supervisée

Conclusion

Nous avons testé deux méthodes différentes une supervisée et une non supervisée. Dans les deux cas nous obtenons des résultats satisfaisants. L'implémentation PyTorch de la méthode non supervisée a été plus compliquée et plus longue mais donne des résultats intéressants. Cela a démontré un grand potentiel de l'apprentissage non supervisé pour la traduction automatique, bien que la méthode reste à améliorer.