Projet Science des données : Traducteur de mots non supervisé

Théo Delemazure, Jean Dupin, Oskar Rynkiewicz

Novembre 2019

Il s'agit dans ce deuxième projet d'exploiter les représentations vectorielles d'un vocabulaire (*word embeddings*) pour construire des traducteurs automatiques, limités à la traduction de mots. Deux approches ont été explorées dans le cadre de ce projet : l'apprentissage supervisé via l'utilisation d'un dictionnaire du langage source vers le langage cible pour un nombre réduit de mots (cf. [2]) ; l'approche non-supervisée où aucune information reliant le langage source au langage cible n'est disponible (cf. [1]). On présentera tout d'abord brièvement le principe permettant la création des words embeddings puis on résumera les méthodologies pour chacune de approches ci-dessus avant de présenter nos résultats.

1 Introduction aux *words embeddings*

Les systèmes de traitement d'image ou de signaux audio utilisent des jeux de données riches, à grande-dimension, que l'on peut stocker dans le cas des images par exemple sous la forme de vecteurs contenant le niveau d'intensité de chaque pixel. Ainsi, toute l'information est directement contenue dans la structure de données adoptée et les relations entre différentes entités d'un même domaine (exemple typique de la distinction entre *chat* et *chien*) peuvent être directement inférées.

Traditionnellement, dans le domaine du traitement automatique du langage naturel, on représente chaque mot par des atomes discrets. Mais un tel encodage est arbitraire et ne donne aucune information sur les relations qui peuvent exister entre atomes individuels. Pour reprendre l'exemple précédent, le système qu'on aura créé pourra difficilement exploiter ce qu'il a appris du traitement du mot *chat* pour établir des relations avec le mot *chien* (par exemple que ce sont tous les deux des animaux, qu'ils possèdent chacun quatre pattes, etc).

Une représentation plus riche est celle proposée avec *word2vec* par Google. Il s'agit de représenter les mots dans un espace vectoriel où les mots qui possèdent une similarité sémantique sont assignés à des points proches dans cet espace. Cette approche repose sur une hypothèse de distribution fréquentielle qui présuppose que des mots qui apparaissent dans un même contexte partagent une similarité sémantique.

1.1 CBOW/Skip-Gram and FastText

1.1.1 CBOW/Skip-Gram

Deux architectures sont utilisées dans le cadre de word2vec. Elles reposent chacune sur l'utilisation d'un réseau de neurones à deux couches pour apprendre des représentations vectorielles des mots.

— **CBOW** : il s'agit de prédire le mot cible (ex. "litière") en utilisant les mots de contexte ("le chat est assis sur"). C'est une architecture adaptée à des jeux de données de taille raisonnablement petite.

— **Skip-Gram** : il s'agit de prédire les mots de contexte en utilisant le mot cible (inverse de **CBOW**).

Dans le cadre de ce projet, on utilisera des *word embeddings* pré-entraînés avec l'approche **Skip-Gram**. Chaque mot en entrée du réseau est d'abord encodé par le biais d'une approche *one-hot encoding*. Le vecteur d'entrée du réseau est donc aussi large que le nombre total de mots dans le vocabulaire utilisé. On réduit la dimension de cette représentation initiale en utilisant une couche cachée avec un nombre limité de neurones (par ex. 300). La Figure 1 illustre le réseau de neurones avec un couple input/ouput ("Deep","Learning").

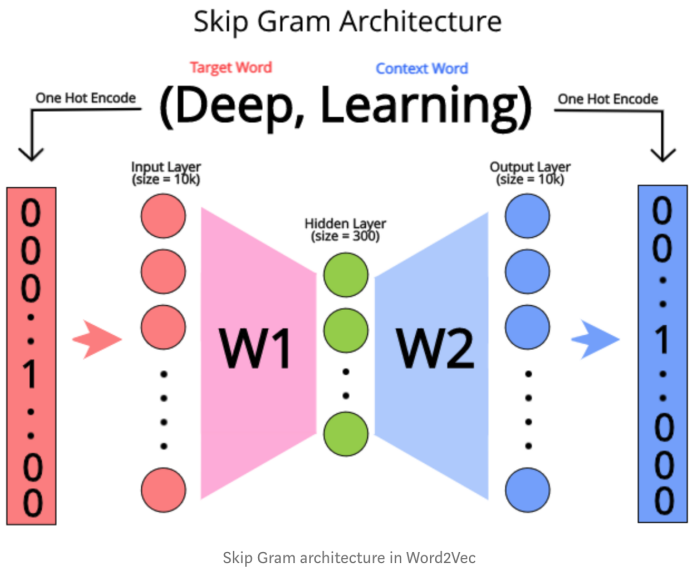

FIGURE 1 – Skip-gram architecture (*Harsha Bommana/Word Vectors with Word2vec*)

Après optimisation, chacune des lignes de la matrice W_1 représente l'encodage de chaque mot d'entrée grâce au réseau de neurones.

1.1.2 FastText

Les vecteurs utilisés sont déjà pré-entraînés et proviennent de la librairie FastText. Chacun de ces vecteurs a la taille de $\mathbb{R}^{300}$. Précisons qu'il s'agit là de *word embeddings* monolingues et donc qu'aucun alignement préalable n'a été réalisé entre les espaces vectoriels correspondants à chaque langue. Précisons aussi

que les mots sont triés par ordre de fréquence (en particulier dans l'approche non supervisée, on utilisera uniquement les 50,000 mots les plus fréquents).

1.2 Dictionnaires

Dans l'approche supervisé, on utilise un dictionnaire pour entraîner notre transformation linéaire. Un dictionnaire est également utilisé pour évaluer la transformation linéaire.

- On utilise les dictionnaires fourni par Facebook Research dans le cadre du projet MUSE (notamment le dictionnaire Anglais-Français). Pour chaque mot du dictionnaire, on récupère les vecteurs correspondants dans les embeddings de FastText. Pour lire les vecteurs plus vite, on ne gardera ici que les 99% mots plus fréquents (les vecteurs sont triés par ordre de fréquence)
- La polysémie de certains mots signifie qu'il y a parfois plusieurs traductions possibles, toutes satisfaisantes. Par exemple ('talk', 'parlez'), ('talk', 'parler'), ('talk', 'parle')
- Certaines paires dans un dictionnaire ne sont pas corrects, comme ('talk', 'talk'). Les paires de ce genre sont préjudiciables aux traductions, cependant ils ne peuvent pas être facilement repérés à cause des mots qui ont la même orthographe en français et en anglais, e.g. ('table', 'table').

1.3 Critère d'évaluation

Rappelons que pour deux vecteurs a et b, on a $\langle a, b\rangle = \|a\|\|b\|\cos(\alpha)$, où α est l'angle entre les vecteurs a et b. À normes égales, le cosinus de l'angle α indique la similarité cosine. Pour mesurer la qualité du modèle, dans une approche comme dans l'autre, on cherchera les 5 (ou premier) plus proches voisins de la sortie de notre modèle selon la similarité cosine que l'on comparera à la traduction ground-truth. Cela permet de définir une précision@5 ou une précision@1.

Enfin, on considérera qu'un mot est traduit correctement si la sortie du modèle est parmi les K plus proches voisins au sens de la similarité cosine. On appelera cela l'*accuracy@K*. Les valeurs classique de K sont 1, 5 et 10.

2 Apprentissage supervisée

Nos deux premières semaines de travail se sont essentiellement concentré sur l'approche supervisée de la traduction de mots, qui est résumée dans cette partie.

Dans le cadre de l'approche supervisée, on notera x_i le vecteur correspondant au mot du langage source et y_i le vecteur correspondant au mot du langage cible. Formellement, en utilisant un dictionnaire d'entraînement contenant n mots, on a donc les exemples labelisés $\{x_i, y_i\}_{i=1}^{n}$, $x_i, y_i \in \mathbb{R}^d$, où $d = 300$. Grâce aux similarités géométriques entre les langues, il est possible d'appliquer la transformation linéaire $W \in \mathbb{R}^{d\times d}$ sur x_i pour obtenir la traduction : $Wx_i \approx y_i$[2]. Le but de cette approche est d'apprendre la transformation W en utilisant un dictionnaire.

2.1 Norme 2

Il s'agit de résoudre le problème d'optimisation suivant :

$$\min_{W \in \mathbb{R}^{d \times d}} \sum_{i=1}^{n} \|W x_i - y_i\|_2^2$$

Le problème peut se réécrire de la façon suivante, en posant $X, Y \in \mathbb{R}^{n \times d}$

$$\min_{W} \|XW - Y\|_2^2$$

Nous pouvons calculer la meilleure estimation de la matrice W grâce à la solution explicite (cf. cours d'optimisation) :

$$\hat{W} = \underset{W}{\operatorname{argmin}} \|XW - Y\|_2^2 = (X^\intercal X)^{-1} X^\intercal Y$$

En utilisant numpy, cette solution est facilement calculée avec :

```
np.linalg.lstsq(X, Y)
```

Notons que nous avons aussi employé une méthode de Stochastic Gradient Descent pour obtenir le résultat mais l'expérimentation indique qu'un très grand nombre d'itérations est nécessaire.

Xing et Al ont également montré dans [3] que le résultats était meilleurs si l'on orthogonalisait la matrice obtenu. Pour cela, nous avons donc calculé la décomposition SVD de la matrice $\hat{W} = (X^\intercal X)^{-1} X^\intercal Y$:

$$U \Sigma V_T = SVD(\hat{W})$$

Et nous avons pris la matrice UV^T, ce qui nous permet d'augmenter de quelques pourcent notre accuracy (5% pour le dictionnaire de validation de l'accuracy@1 par exemple).

2.1.1 La qualité de traductions par rapport au nombre des vecteurs

MUSE fourni un dictionnaire Anglais-Français avec environ $100,000$ traductions. Nous ne pouvons pas utiliser toutes les paires du dictionnaire, car le calcul des plus proches voisins serait trop long. Pourtant, l'évaluation de la qualité des traductions en fonction de nombre d'exemples est importante.

Nous étudions la qualité de traducteurs entraînés sur des ensembles des données contenant entre $1,000$ à $20,000$ paires de mots Anglais-Français. Dans l'expérience, le dictionnaire de validation contient 2000 paires. L'ensemble des $20,000$ vecteurs du langage cible est un espace de recherche pour trouver les vecteurs les plus proches de Wx_i. Les résultats sont présentés sur Figure 2.

Nous pouvons constater qu'après environ $5000 - 7500$ paires x_i, y_i utilisés pour le calcul de W, l'ajout d'autres exemples a peu d'effet sur le résultat obtenu. La qualité d'un traducteur reste au même niveau avec une accuracy d'environ 79% @5.

Pour la suite de la partie supervisé, nous allons utiliser uniquement un petit dictionnaire de 5000 paires pour l'entraînement et 1000 paires pour la validation. Nous avons montré que l'impact de nombre de données s'arrête assez vite dans

le cas supervisé et donc, il est plus pratique de garder petit nombre d'exemples pour construire un traducteur supervisé.

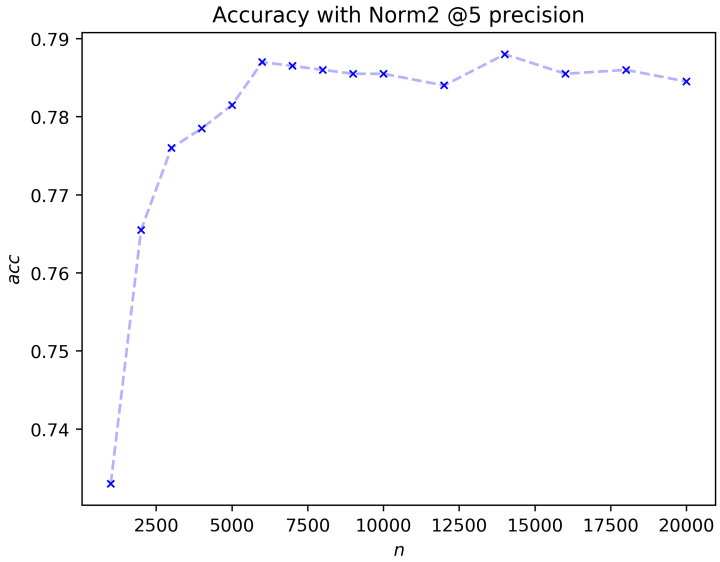

FIGURE 2 – L'impact de nombre des vecteurs sur l'accuracy

```
Nearest neighbors of eat (ground truth manger):

cos_sim=0.783   manger
cos_sim=0.733   consommer
cos_sim=0.700   nourrir
cos_sim=0.665   cuisiner
cos_sim=0.664   avaler
```

FIGURE 3 – Exemple des meilleures prédictions pour la traduction de *eat*

2.2 Similarité cosine

Comme l'ont montré Xing et Al dans [3], la norme euclidienne n'est pas forcément la plus adéquate si l'on considère que la mesure de similarité que l'on utilise pour évaluer la performance du modèle est la similarité cosine (i.e. minimiser l'angle entre les deux vecteurs donc maximiser le cosinus de l'angle entre les deux vecteurs).

Plutôt que d'optimiser selon la norme l^2, on peut optimiser directement selon la similarité cosine, en résolvant le problème suivant

$$\max_{W \in \mathbb{R}^{d \times d}} \sum_{i=1}^{n} (Wx_i)^T y_i$$

Nous avons utilisée la méthode de descente de gradient. On remarque tout

d'abord que le gradient ne dépend pas de W : $grad = \sum_{i=1}^{n} xi^T y_i$. Ainsi, chaque itération de la descente de gradient est composée de 2 étapes :

1. Descente de gradient : $W \leftarrow W + \alpha grad$ avec α le pas de la descente. On utilise un $+$ au lieu d'un $-$ car on veut maximiser la similarité cosine.
2. Orthonormalisation de la matrice : $W \leftarrow UV_T$ avec $U\Sigma V_T = SVD(W)$.

L'évolution de la similarité cosine avec cette méthode est présentée en figure 4. Encore une fois, il est possible de trouver la forme close qui résout ce problème

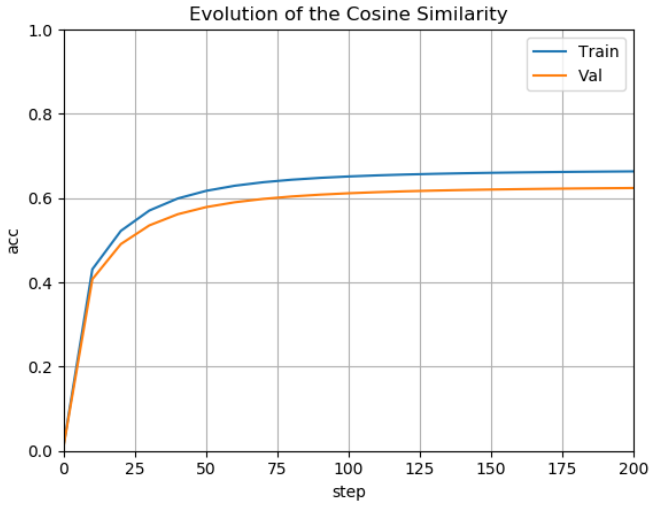

FIGURE 4 – Évolution de la Similarité cosine

d'optimisation. En effet, dans le papier [1], une référence à la forme close de la norme de Frobenius est évoquée, et la matrice obtenue est celle qui maximise la similarité cosine. Pour rappel, la norme de Frobenius est définie comme :

$$||A||_F = \sqrt{Tr(AA^H)}$$

où A^H est la matrice transposée conjuguée de A.

On peut alors calculer le minimum de l'erreur avec comme contrainte que la matrice doit être orthogonale grâce à une forme close :

$$\hat{W} = argmin_{W \in O_d(\mathbb{R})} ||WX - Y||_F = UV_T$$

tel que

$$U\Sigma V_T = SVD(YX^T)$$

Comme on peut le voir Table 1, les résultats sont comparables à ceux obtenus avec la norme euclidienne, et avec la descente de gradient sur la similarité cosine.

2.3 Analyse des erreurs

En analysant les erreurs faites par nos transformations linéaire, on constate que la plupart du temps on obtient un synonyme du mot attendu, ou un

mot proche (par exemple *"comprendre"* au lieu *"compris"* pour *"understood"*). Parfois même, notre traduction était meilleure que celle du dictionnaire (par exemple nous traduisions *"war"* en *"guerre"* et le dictionnaire attendait *"war"*). Enfin, il y avait des cas ou notre traducteur ne renvoyait pas le bon mot du tout (par exemple *"Malheureusement"* au lieu de *"parti"* pour *"gone"*). Il y a également quelques cas amusant, comme le fait que *"background"* soit traduit en *"#aaa"* (les adeptes de CSS verront que *#aaa* est en fait la couleur de fond des encadrés *Wikipedia* qui a servi de base de donnée pour entraîner les embeddings FastText), ou encore le cas suivant pour la traduction de *"Windows"*, qui devrait être traduit en *"Fenêtre"* :

```
Nearest neighbors of windows :

cos_sim=0.785   windows
cos_sim=0.684   microsoft
cos_sim=0.649   linux
cos_sim=0.620   ordinateurs
cos_sim=0.596   interface
```

2.4 Comparaison

Tous les modèles supervisés ont été entraînés sur 80% du dictionnaire d'entraînement fourni par Facebook Research et évalués sur les 20% restants, avec la mesure de l'accuracy@K pour $K \in [1, 5, 10]$. Pour rappel, l'accuracy@K est le pourcentage de cas pour lequel la bonne traduction se trouve dans les K premières prédictions du modèle, parmi tous les mots présents dans le dictionnaire.

Nous avons introduit une autre mesure, l'*accuracy1@K*. Pour définir cette valeur, on considère le dictionnaire $\mathcal{D} = \{(x_i, S_i) | 1 \leq i \leq n\}$ tel que $\forall i$, x_i est un mot du langage source (ici, l'anglais) et S_i est l'ensemble des traductions possibles de ce mot (à cause de la polysémie, on peut avoir $|S_i| > 1$). On a alors l'*accuracy1@K* la pourcentage de mots x_i tel qu'une des traductions correctes (i.e. $\exists y_{i,j} \in S_i$) apparaisse parmi les K premières prédictions du modèle. Dans le cas de l'*accuracy@K*, on considère chaque paire $(x_i, y_{i,j})$ avec $y_{i,j} \in S_i$ séparément. L'*accuracy1@K* permet donc de ne pas prendre en compte les erreurs de traduction du dictionnaire de test (par exemple "talk" traduit en "talk").

Mesure	acc@1		accy@5		acc1@5
Données	Train	Val	Train	Val	Train+Val
$\|\|MX - Y\|\|_2$ (forme close)	61%	53%	88%	81%	96%
$\|\|MX - Y\|\|_F$ (forme close)	60%	51%	87%	82%	95%
$cos(MX, Y)$ (GD)	60%	51%	87%	82%	95%

TABLE 1 – Résulats de l'efficacité des traducteurs obtenus avec les différents modèles

3 Apprentissage non supervisé (semaine 3)

3.1 Les GAN

Enfin, nous avons tenté de reproduire le travail de Connneau dans [1] en créant un traducteur de mots Anglais-Français non supervisé. Il est donc interdit d'utiliser des informations reliant les deux langages pour l'entraînement, comme par exemple un mini-dictionnaire. Pour répondre à cette problématique, Conneau utilise un *Generative Adversarial Network (GAN)* que nous avons schématisé en Figure 5.

Dans ce genre de réseau, deux réseaux de neurones sont en concurrence :

— Le Générateur G qui doit apprendre à envoyer des mots du langage source vers le langage cible et à faire croire au Discriminateur qu'il s'agit de mots du langage cible.

— Le Discriminateur D qui doit apprendre à distinguer les mots du langage cible et les "faux" mots qui ont été transformé par le Générateur.

On suit ainsi l'évolution de la loss des deux réseaux et on les optimise tour à tour pour que chacun d'entre eux devienne de plus en plus performant. En faisant cela, on espère qu'à terme, le Générateur trouve un Mapping linéaire entre les mots source et les mots cible.

La loss utilisée est la Binary Cross-Entropy Loss (BCELoss). Si on considère que $x_1, ..., x_n$ sont des mots du langage source et $y_1, ..., y_m$ des mots du langages cible :

$$\mathcal{L}_D(x_1, ..., x_n, y_1, ...y_m) = -\frac{1}{n}\sum_i^n \mathbb{P}_D(\text{cible}|G(x_i)) - \frac{1}{m}\sum_i^m \mathbb{P}_D(\text{source}|y_i)$$

$$\mathcal{L}_G(x_1, ..., x_n, y_1, ...y_m) = -\frac{1}{n}\sum_i^n \mathbb{P}_D(\text{source}|G(x_i)) - \frac{1}{m}\sum_i^m \mathbb{P}_D(\text{cible}|y_i)$$

Où $\mathbb{P}_D(\text{cible}|x)$ est la probabilité que le mot x soit classé comme venant du langage cible par le discriminateur. Un schéma simplifié de *GAN* est présenté en Figure 5

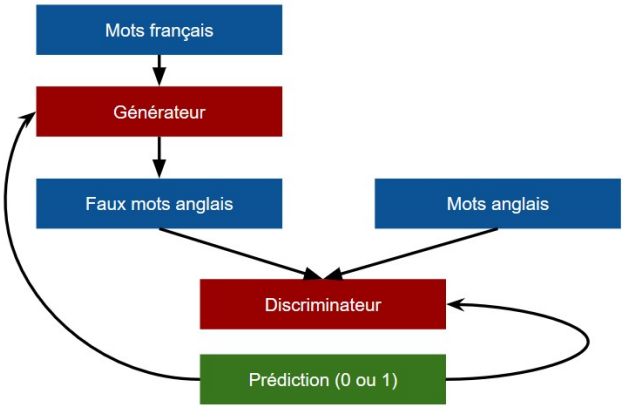

FIGURE 5 – Schéma simplifié d'un *Generative adversarial network*

Nous avons donc implémenter l'algorithme suivant :

Algorithm 1 Unsupervised learning

for $i = 1$ to N_{iters} **do**
  for $j = 1$ to N'_{iters} **do**
    Optimization step for the Discriminator D
  end for
  for $j = 1$ to N'_{iters} **do**
    Optimization step for the Generator G
  end for
  Orthogonalisation of the Generator's matrix of weights W
end for

Ainsi, chaque réseau est entraîné N'_{iters} fois avant que l'on entraîne l'autre. Cela permet de faire des plus grands pas à chaque étape et cela semblait être plus efficace empiriquement. Nous avons pris $N'_{iters} = 3$.

Une étape d'optimisation pour le discriminateur consiste a prendre un batch de 32 mots du langage source choisit au hasard parmi les 50000 mots les plus fréquent, et les transformer avec le générateur G pour créer des *faux* mots du langage cible. On prend un deuxième batch de 32 mots du langage cible, et on regarde l'erreur du discriminateur sur ces deux batch grâce à la loss $\mathcal{L}_D$. On utilise une descente de gradient pour améliorer le discriminateur. De même, pour le générateur, on utilise encore deux batchs (un pour le langage source et un pour le langage cible) et on calcule la loss $\mathcal{L}_G$ pour l'améliorer. On remarquera qu'il n'y a pas besoin d'introduire le langage cible dans la loss, puisque l'on prends le gradient par rapport au générateur G, qui n'influe pas dans le cas du langage cible et présentera donc un gradient nul.

Enfin, à la fin de chaque itération, on orthogonalise la matrice utilisée dans le réseau du générateur, avec la formule proposée dans [1] :

$$W \leftarrow (1 + \beta)W - \beta(WW^T)W$$

avec $\beta = 0.01$ qui semblait bien marcher. Cette formule est proposée en remplacement à celle utilisée en supervised learning (ou on divise par les valeurs propres) car c'est beaucoup plus rapide de faire ça que de faire la décomposition *SVD*. On perds malheureusement un petit peu en efficacité.

Le réseau utilisé pour le discriminateur est un *Multi Layer Percepetron (MLP)* avec 2 couches cachées et 2048 neurones sur chaque couche. Le réseau utilisée pour le générateur est un réseau linéaire (équivalent à une transformation linéaire W). Puisqu'il faut beaucoup de temps pour entraîner un réseau (et que nous n'en avions pas beaucoup), nous avons utilisé les paramètres proposés dans [1], comme un learning rate de $\alpha = 0.1$ pour les deux réseaux et un *smoothing coefficient* de 0.1 pour le discriminateur. C'est à dire que dans la *BCE Loss*, on considérait maintenant que pour les mots issus du langage source, le discriminateur devait retourner une proba de 0.9 et non de 1 et pour les mots du langage cible, une proba de 0.1.

3.2 Résultats

Pour la traduction Français-Anglais, on constate bien que la qualité de traduction du Générateur s'améliore puisque la similarité cosine augmente. Pour

rappel, la similarité cosine est calculée comme :

$$\frac{1}{n} \sum_{(x_i, y_i) \in \text{dico}} cos(G(x_i), y_i)$$

Ce qui signifie que plus la similarité cosine est élevée, plus les traductions du générateur sont proches des traductions réelle. Une similarité cosine de 1 indique un traducteur parfait.

Comme on le constate donc Figure 7, la similarité cosine pour la traduction anglais-français augmente bel et bien, lentement mais sûrement, atteignant 0.25 au bout de 10000 itérations. Après avoir fait tourner notre algorithme pendant un moment, nous avons obtenu une similarité cosine de 0.52, ce qui est plutôt correct pour un traducteur non supervisé. C'est avec ce réseau que nous avons calculé les *accuracy* Table 3.2.

L'évolution des loss mérité également notre intérêt : Comme nous le constatons Figure 6, les deux courbes subissent de grosse variations d'une itérations à l'autre mais restent globalement autour d'une valeur, avec la loss du générateur plus élevée que celle du discriminateur, ce qui se traduit par le fait qu'il est plus facile d'apprendre à séparer les deux ensemble que d'apprendre à les rendre similaire. Cette Loss est également plus faible à cause du *smoothing coefficient* qui permet au discriminateur de se tromper un peu moins à chaque itération.

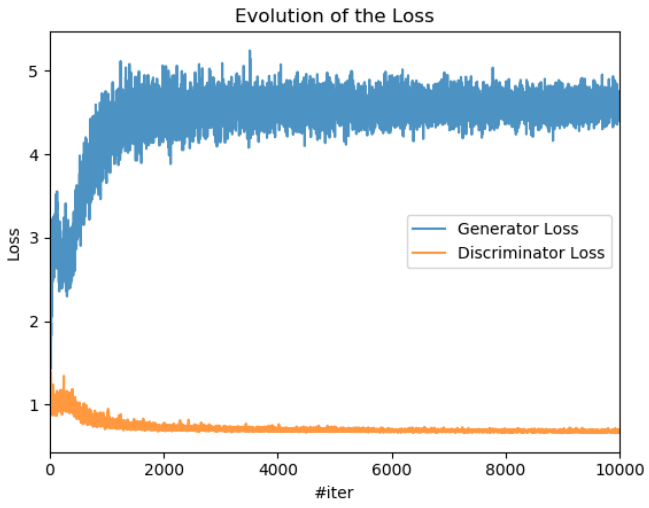

FIGURE 6 – Evolution des loss du générateur et du discriminateur

Après quelques dizaines de milliers d'itérations, nous avons donc obtenu un traducteur de mots Anglais-Français assez performant, comme on peut le constater Table 3.2. Les résultats sont moins bon que pour la version supervisée, mais ils ne sont pas décevant. Bien sûr, ces bons résultats sont "boostés" par le fait que l'on fait des prédictions uniquement sur les mots du dictionnaire, et que le dictionnaire de Facebook est adapté à ce genre de problème.

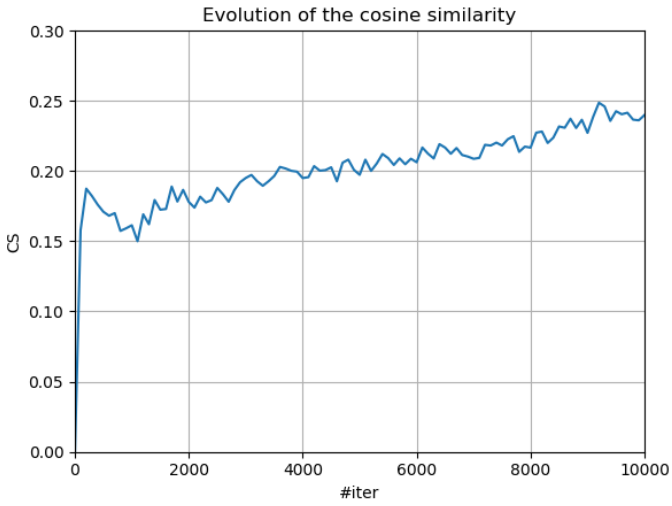

FIGURE 7 – Evolution de la Similarité Cosine

Mesure	acc@1		accy@5		acc1@5
Données	Train	Val	Train	Val	Train+Val
Meilleur supervisé	61%	53%	88%	82%	96%
Non supervisé	39%		69%		85%

TABLE 2 – Comparaison des accuracy pour les versions supervisées et non supervisées

3.3 Autres langages

Nous avons également brièvement tenté de construire un traducteur de mots non supervisé Anglais → Grec, qui possède un alphabet très différent. Mais à chacune de nos tentative, la similarité cosine stagnait autour de 0.2, même après 20,000 itérations. Nous avons donc laissé tombé le Grec pour le moment.

3.4 Autres remarques

Nous avons remarqué que le réseau apprenait plus ou moins vite un *mapping* à chaque tentative. Étant donné que nous n'avions pas de graine d'aléa et que nous ne changions aucun paramètre entre chaque tentative, nous en avons déduit que l'initialisation des poids du réseau, en particulier du réseau générateur, avait une très grosse influence sur la vitesse de convergence de celui-ci.

L'accuracy pourrait être amélioré si nous avions implémenté toutes les optimisations du papier [1], notamment l'utilisation de la mesure de similarité CSLS ou encore du critère de validation non supervisée.

Enfin, nous aurions besoin de mener une expérience similaire à celle dans la section 2.1.1 pour rechercher si l'accuracy est ici aussi bornée, même en augmentant le nombre de données d'entraînement. Après le papier original [1], nous pouvons soupçonner que dans le cas de ce traducteur non-supervisé, l'ajout

d'exemples pourrait augmenter la qualité, pouvant même dépasser celle d'un traducteur supervisé.

4 Conclusions

La traduction automatique est un problème complexe qui peut grandement bénéficier des progrès récents de l'apprentissage automatique. En particulier, les techniques de *Deep Learning* donnent de bons résultats pour l'apprentissage des représentations vectorielles de mots et l'alignement de ces vecteurs de manière non supervisée à l'aide de GAN. Nos expériences ont permis de voir que la méthode de pointe pour la traduction de mots non supervisée est un candidat puissant contre les méthodes supervisées. Cela ouvre également de nombreuses portes pour traduire des langages sur lesquelles nous avons pas assez d'informations mutlilingual pour construire un traducteur (par exemple, l'espéranto ou des langues anciennes)

Références

[1] Conneau, A., Lample, G., Ranzato, M., Denoyer, L., and Jégou, H. Word translation without parallel data. *CoRR abs/1710.04087* (2017).

[2] Mikolov, T., Le, Q. V., and Sutskever, I. Exploiting similarities among languages for machine translation. *CoRR abs/1309.4168* (2013).

[3] Xing, C., Wang, D., Liu, C., and Lin, Y. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies* (Denver, Colorado, May–June 2015), Association for Computational Linguistics, pp. 1006–1011.