Translittération automatisée du néo-égyptien à l'aide de réseaux de neurones

Une expérience de *Deep Learning*

Serge Rosmorduc[1]
2020[2]

Résumé

Nous appliquons des techniques de Deep Learning à une tâche de translittération automatisée du néo-égyptien. Après une brève présentation de la technologie utilisée, nous examinons les résultats pour mettre en évidence les capacités du système, qui est capable de traiter un large éventail de problèmes, y compris grammaticaux et phraséologiques. Nous procédons ensuite à l'extraction des valeurs des signes à partir de ce que le système a automatiquement appris.

1. Introduction

Cet article présente un système de translittération automatisée. Comme il existe de nombreux styles de translittération en égyptologie, nous définirons ce que le système est censé faire. Notre objectif est de reproduire le type de translittération que l'on trouve dans la plupart des ouvrages philologiques, et illustré, par exemple, dans la grammaire de François Neveu (Neveu 1996). Notre système ne produira pas directement une analyse des valeurs de signe. Comme nous utilisons des techniques d'apprentissage automatique, le système sera dépendant de son corpus d'entraînement, extrait du Projet Ramsès, et reproduira le style de translittération qui y est utilisé. En particulier, la translittération sera fortement « normalisante ». Il esquissera l'analyse grammaticale de la phrase, et fournira occasionnellement des terminaisons de mots ou des éléments grammaticaux tombés en désuétude à la fin de la XXe dynastie, même lorsqu'ils ne sont pas écrits.

Depuis un certain temps déjà, nous nous intéressons à la translittération automatique de l'égyptien par ordinateur ; en 2005, nous avons mis en place un système de preuve de concept (Rosmorduc 2008), mais il fallait un grand nombre de règles créées par des experts pour être efficace. Ces règles étaient souvent contradictoires : les signes pouvaient avoir plusieurs valeurs, plusieurs segmentations étaient possibles… Résoudre ces conflits à grande échelle de manière fiable était extrêmement difficile.

---

[1] Laboratoire CEDRIC, Conservatoire National des Arts et Métiers, Paris. (serge.rosmorduc[at] qenherkhopeshef.org).
Ce travail n'aurait pas été possible sans le corpus de Ramsès. Je tiens tout particulièrement à remercier pour leurs excellents commentaires Jean Winand et Stéphane Polis, Mark-Jan Nederhof, ainsi que les relecteurs anonymes de cet article. Les données utilisées pour produire ce système, ainsi qu'une implémentation entraînée et fonctionnelle en Python du translittérateur actuel, sont disponibles sur https://gitlab.cnam.fr/gitlab/rosmorse/ramses-trl. Les données devraient s'améliorer, à la fois en raison des révisions du Corpus Ramsès lui-même et du code que nous utilisons pour extraire le corpus translittéré. Ces améliorations seront également publiées sous forme de nouvelles versions du corpus, avec un numéro de version différent. Un article orienté informatique a été écrit et sera éventuellement disponible via le site gitlab.

[2] Texte original publié en anglais dans le *LingAeg 28*, pp. 233 à 257.

Grâce aux progrès de l'apprentissage automatique, nous présentons un nouveau système capable de gérer assez efficacement les textes du néo-égyptien. L'une de ses caractéristiques les plus intéressantes est qu'il est capable d'« expliquer » d'une manière ou d'une autre son comportement, en utilisant ce qu'on appelle un *mécanisme d'attention*. Cet article évaluera non seulement le système mais mettra également en évidence certaines des caractéristiques linguistiques les plus intéressantes que la machine a apprises.

L'approche adoptée ici consiste à considérer la translittération comme une sorte de tâche de réécriture. Un texte d'entrée, qui est l'encodage au format du Manuel de Codage du document égyptien original, est réécrit comme un texte de sortie, la translittération. Il existe maintenant de nombreux outils et algorithmes d'apprentissage automatique pour les tâches de réécriture. Ces outils ont été développés à l'origine pour la traduction automatique, une tâche bien plus complexe que la translittération. Les informaticiens les ont appliqués à une grande variété de problèmes où la sortie est un texte qui est en quelque sorte une réécriture de l'original, notamment dans des tâches de résumé (Nallapati et al. 2016) et analyse syntaxique (Vinyals et al. 2015).

La disponibilité de grands corpus est une condition *sine qua non* pour la plupart des algorithmes d'apprentissage automatique modernes. A cet égard, le *Corpus Ramsès* (Winand et al. 2015, http://ramses.ulg.ac.be) est une base solide sur laquelle construire un tel système.

Après un rapide état de l'art sur la translittération automatisée, nous donnons un bref aperçu des réseaux de neurones, leur utilisation dans le traitement automatique du langage, nous présentons notre corpus et nous discutons nos résultats.

2 Travaux précédents

Les premières tentatives de translittération automatisée remontent aux années 1990, avec la thèse de Sophie Billet[3] (1995 ; Billet et al. 1994). Ces premières tentatives, ainsi que celles du présent auteur (Rosmorduc 2008 ; Barthélemy et al. 2011), étaient pour la plupart basées sur des règles écrites à la main et testées sur des corpus relativement petits.

Ces approches artisanales souffrent de la difficulté à gérer des règles contradictoires de manière fiable, par exemple pour choisir entre différentes valeurs pour un signe donné. Notre propre système reposait sur des priorités données aux règles et aux valeurs des signes. Mais le choix ultime d'une translittération implique de nombreux niveaux différents : les valeurs des signes, les combinaisons de signes possibles, le vocabulaire proprement dit, la syntaxe du texte, sa sémantique et même la phraséologie. Les équilibrer tous par une approche par essais et erreurs peut conduire à de bons résultats sur un corpus limité mais elle est en quelque sorte vouée à l'échec pour des textes aléatoires.

[3] Apports à l'acquisition interactive de connaissances contextuelles : SAPIENS, un système pour la translittération de textes hiéroglyphiques. Thèse de doctorat en Sciences appliquées, Informatique, sous la direction de Danièle Hérin-Aimé, soutenue en 1995 à Montpellier 2.

Il était donc raisonnable d'essayer d'utiliser des techniques d'apprentissage automatique qui permettent à un système de calculer ses paramètres à partir d'un corpus. Mark-Jan Nederhof et Fahrurrozi Rahman (2017) ont fait les premiers pas dans cette direction en proposant un formalisme de translittération basé sur des règles statistiques. Cependant, leur système nécessitait un corpus spécifiquement annoté pour s'entraîner.

À ce moment-là, nous envisagions l'utilisation de techniques de traduction automatique pour la translittération. Nous avions le Corpus Ramsès sur lequel travailler, et très vite, la publication d'algorithmes très performants nous a décidés à essayer cette approche.

3 Le Corpus

Les algorithmes sophistiqués de l'apprentissage automatique seraient de peu d'utilité sans données pour les alimenter. La disponibilité des corpus est primordiale. Dans notre cas, notre participation au Projet Ramsès (Winand et al. 2015, http://ramses.ulg.ac.be) de l'Université de Liège nous a fourni ce qui est probablement la plus grande collection de textes hiéroglyphiques annotés disponible sur ordinateur aujourd'hui.

Le *Corpus Ramsès* est un vaste corpus annoté de néo-égyptien, développé depuis 2006. Il fournit, pour chaque texte, la transcription hiéroglyphique de ses mots, et leur analyse en flexions de lemmes spécifiques. Contenant plus de 500 000 mots, le corpus est suffisamment volumineux pour les méthodes d'apprentissage en profondeur.

Évidemment, comme les textes sont en néo-égyptien, un système entraîné sur celui-ci serait influencé par cette évolution de la langue et ses particularités graphiques. Cependant, il contient également un certain nombre de textes monumentaux, dont certains de la XVIIIe dynastie, qui couvrent un peu la langue et l'orthographe du moyen égyptien.

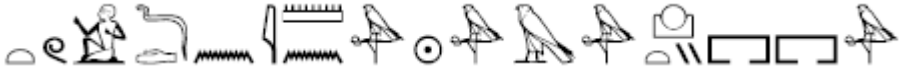
L'apprentissage automatique nécessitant un grand volume de textes, nous avons choisi d'utiliser l'ensemble du Corpus Ramsès. Il comprend à la fois les textes qui ont été mis à disposition sur le site Web du Projet Ramsès, qui ont été recoupés et relus attentivement, et les textes qui doivent encore être validés. La plupart d'entre eux sont de très haute qualité, mais quelques erreurs ont pu se produire ici ou là. Cela conduira à des cas intéressants plus tard, où le système de translittération automatisé peut en fait améliorer une partie de l'analyse.

La translittération du texte n'a pas été enregistrée dans Ramsès. Heureusement, chaque mot est normalement annoté avec des références à une orthographe, un lemme, et généralement à une inflexion spécifique de ce lemme. Dans le lexique de la base de données, des translittérations « canoniques » sont fournies pour les orthographes, les lemmes et les inflexions. Par conséquent, nous pouvons générer une translittération *artificielle* qui est celle que nous utilisons plus tard dans le processus de formation. Pour chaque mot :

– si une inflexion est précisée, on utilise sa translittération
– si un lemme est spécifié mais pas d'inflexion, la translittération du lemme est utilisée
– si seule une orthographe est spécifiée, la translittération de l'orthographe est utilisée.

En conséquence, notre translittération est hautement normalisée. Un mot sera translittéré de la manière dont il « devrait » être écrit plutôt que de la manière dont il est réellement écrit. Par exemple, l'infinitif du verbe *jrj* sera translittéré *jrj.t* même s'il s'écrit simplement  .

Cette approche donne de mauvais résultats dans quelques cas. Par exemple, comme toutes les occurrences de la préposition *m* sont regroupées (même lemme, même inflexion), les orthographes comme translittérées sont en *m* et non en *jm*.

Le corpus de Ramsès fournit également des prépositions lorsqu'elles sont omises au premier présent, séquentiel ou troisième futur. Comme nous précisons qu'il s'agit d'« ajouts d'éditeur », ils n'apparaîtront pas dans la transcription hiéroglyphique du texte, mais ils seront présents dans la translittération, entre des marques ecdotiques. Par exemple  , sera translittéré *twj (ḥr) ḏd n jmn-rˁ-ḥr-ȝḫty*.

Au fur et à mesure que le système sera entraîné à produire des translittérations en utilisant le corpus comme échantillon, il apprendra à fournir les prépositions manquantes, fournissant ainsi une analyse morpho-syntaxique grossière des textes.

Lorsque nous créons le corpus de translittération, les mots sont conservés dans le même ordre que dans le texte hiéroglyphique. Cela signifie que les transpositions honorifiques sont ignorées, sauf dans les noms propres et autres où la translittération est extraite telle quelle du lexique. Cette lacune est liée à la construction de notre corpus particulier et non au processus d'apprentissage automatique.

3.1 Organisation du corpus

Le premier problème à résoudre était de choisir comment le corpus serait préparé pour la tâche d'apprentissage automatique. Nous avons décidé de travailler au niveau de la phrase qui est suffisamment grande pour être utile et suffisamment petite pour pouvoir être traitée sur des ordinateurs relativement peu puissants.

Lorsque l'on travaille sur des tâches d'apprentissage automatique, l'approche standard consiste à couper le corpus en trois sous-corpus :

– un *corpus d'apprentissage* sur lequel se fait l'apprentissage proprement dit ;
– un *corpus de validation* que nous expliquons ci-dessous ;
– un *corpus de test*, qui servira à évaluer les résultats.

Le système est entraîné à plusieurs reprises sur des phrases du *corpus d'apprentissage*. La différence entre le résultat calculé et la sortie attendue se termine par une erreur numérique. Cette erreur est utilisée pour modifier progressivement les paramètres du système, améliorant encore et encore ses résultats. Or, avec ce seul critère, un système qui apprendrait à translittérer exactement son corpus d'apprentissage, ni moins, ni plus, serait considéré comme parfait, alors qu'il est en fait inutile. Ainsi, les algorithmes de formation tentent d'éviter « l'apprentissage par cœur ».

Une façon de détecter l'apprentissage par coeur est d'exécuter le système sur un corpus

différent, le *corpus de validation*. Cette première évaluation montre les performances du système sur des textes extérieurs à son corpus d'apprentissage, mesurant sa capacité à généraliser ce qu'il a appris. Pour améliorer les performances sur le corpus de validation, l'informaticien peut modifier l'algorithme d'apprentissage. Son utilisation pour ajuster le processus d'apprentissage rend cependant le corpus de validation inadapté pour évaluer les performances du système sur des textes aléatoires, et, en particulier, inadapté pour comparer les performances de deux systèmes différents.

Le *corpus de test*, à son tour, résout ce problème. Bien qu'il ne soit pas du tout utilisé pendant le processus d'apprentissage, le seul but de ce corpus est de fournir une évaluation finale, en particulier lors de la comparaison de différentes approches d'apprentissage automatique. Il est techniquement judicieux d'analyser les résultats obtenus sur le corpus de validation pour modifier nos algorithmes d'entraînement ; mais en théorie, le spécialiste de l'apprentissage automatique ne devrait même pas regarder le contenu du corpus de test.

Pour construire notre corpus d'apprentissage, de validation et de test, nous avons dû décider comment répartir nos données. Il semblait peu judicieux de diviser les phrases d'un texte donné entre les trois corpus, car idéalement, ces corpus devraient être complètement séparés. Le même mot, apparaissant sur deux lignes d'un même texte, est susceptible d'avoir la même orthographe et la même translittération. Cela entraînerait probablement une évaluation trop optimiste de la qualité de l'apprentissage.

Ainsi, nous avons assigné chaque texte de la base de données à exactement un des sous-corpus. Nous avons assigné au hasard 200 textes à chacun des corpus de validation et de test, et 4 403 textes à l'apprentissage. Les tailles respectives ont été choisies sur une base relativement arbitraire. Par ailleurs, nous avons conservé la même proportion de textes hiéroglyphiques et de textes hiératiques dans chaque sous-corpus (7% de textes hiéroglyphiques et 93% de textes hiératiques, correspondant au ratio actuel dans le corpus de Ramsès). Le corpus d'apprentissage résultant est long de 1 426 499 signes. D'un point de vue statistique, cette approche est encore quelque peu problématique car elle suppose que nos textes soient un échantillon aléatoire représentatif du néo-égyptien, ce qui n'est pas vrai : le ratio des textes de *Deir el Medina*, par exemple, est très haut. Pour l'apprentissage automatique, nous n'avons pas de bonne solution à ce problème car le corpus original est encore relativement petit et que, de plus, des régions entières du pays en sont complètement absentes. Il serait cependant possible d'évaluer notre système sur des sous-corpus, tant sur le plan géographique que chronologique.

Chaque sous-corpus est composé de deux fichiers : un *fichier source*, qui contient les textes hiéroglyphiques, représentés sous forme de listes de codes Gardiner (plus quelques codes pour les lacunes), et un *fichier cible*, contenant les translittérations. Les phrases sont listées dans un ordre aléatoire, une phrase par ligne, une ligne du fichier source correspondant à une ligne du fichier cible.

Par exemple, ces deux lignes[4] du corpus de test source :

[4] Les lignes du corpus sont mélangées, donc ces deux lignes sont tirées de textes différents (KRI 5, 15, 7 et KRI 1, 325, 4 respectivement).

N28 D36 D36 V31 S34 N35 Aa1 G24A Z2
M17 G17 G17 D36 D4 X1 Z7 V31A M17 M17 D21 T25 D58 Z7 Y1 I9

correspondent aux translittérations dans le corpus cible :

xaa =k anx rxy.t
imy iry.tw ky r DbA =f

et aux hiéroglyphes :

Dans le corpus, les grandes lacunes pour lesquelles aucun contenu n'est fourni sont signalées par le code « LACUNA » au lieu d'un code de glyphe Gardiner ; les mots en lacune qui ont été restitués par l'encodeur sont signalés par le code "MISSING".

Nous n'avons pas conservé les codes de position du Manuel de Codage, tels que '*', ':' et '-', dans le fichier source. Ils peuvent être utiles car les limites de mots ont tendance à se produire aux sauts de quadrant. Cependant, dans le Corpus Ramsès, chaque mot est encodé séparément. Par conséquent, les positions des glyphes au début et à la fin de chaque mot sont incertaines. Leur utilisation en formation conduirait le système à considérer systématiquement que les limites de mots sont alignées sur les limites de quadrants.

3.2 Critères d'évaluation

Pour évaluer la qualité du système, nous devons vérifier si les translittérations produites par le système sur le corpus de test sont correctes. Cependant, il est très peu probable que deux translittérations du même texte soient exactement identiques, même lorsqu'elles sont faites par deux experts en philologie.

Les spécialistes de la traduction automatique évaluent leur système en comparant chaque traduction à plusieurs traductions dites « or » écrites par un humain. Comme le corpus de test doit être suffisamment grand pour être raisonnablement significatif, il nécessite donc une quantité de travail considérable.

Comme la translittération est un peu plus simple que la traduction, nous avons choisi de considérer la translittération dans le fichier « cible » de test comme la translittération « or » correcte. Il s'agit d'une approximation grossière : certains résultats corrects différeront vraisemblablement du contenu du corpus « or ». Cependant, la taille des données utilisées dans le traitement du langage naturel est si grande que nous devons faire des compromis.

Il serait trop pessimiste d'évaluer la qualité du système en comptant le nombre de lignes identiques dans le standard « or » et le résultat généré. Au lieu de cela, nous avons recours au système utilisé dans les correcteurs orthographiques : la *distance de*

Levenshtein. C'est simplement le nombre de caractères qu'il faut modifier dans les translittérations calculées pour obtenir le standard « or ». Une correspondance parfaite donnera une distance de 0 ; si le résultat attendu était *sw ḥr stp* et que le résultat calculé était *sww ḥr stp*, nous aurions besoin de supprimer l'un des « w » pour obtenir le résultat correct, ce qui donnerait une distance de 1.

Pour obtenir un résultat cohérent sur le corpus, étant donné que les phrases ont des longueurs différentes, on divise cette distance par la longueur de la phrase d'or, pour obtenir un « nombre moyen d'éditions par caractères ». Pour l'exemple précédent, nous divisons alors la distance par 9, la longueur du résultat attendu (y compris les espaces) et obtenons 1/9.

La variante de la distance que nous utilisons est également approximative, car elle ne prend en compte que l'insertion et la suppression de caractères, ce qui signifie qu'un remplacement de caractère comme *jst* vs *jsṯ* se terminera par une distance de deux modifications.

Le résultat final sera une évaluation moyenne sur l'ensemble du corpus, que nous espérons obtenir bien en dessous de 1.

4 Apprentissage automatique et réseaux de neurones

Le traitement automatique du langage naturel a parcouru un long chemin depuis ses débuts, à la fin des années 50. En particulier, les années 1990 ont vu le passage d'un système formel sur mesure à une utilisation croissante de techniques à forte intensité de données, basées sur des modèles statistiques. Les dix dernières années ont vu une énorme percée avec l'utilisation des réseaux de neurones, sous le nom de *Deep Learning*.

D'abord utilisés principalement pour le traitement d'images, les réseaux de neurones modernes, grâce à une puissance de calcul accrue, à des corpus beaucoup plus importants et à des améliorations théoriques, se sont révélés très efficaces pour de nombreuses tâches standard de traitement du langage naturel, telles que la lemmatisation, la reconnaissance d'entités nommées et la traduction automatique.

4.1 Un bref aperçu des réseaux de neurones

Les réseaux de neurones sont des programmes informatiques inspirés du fonctionnement des neurones réels du cerveau. La figure 1, qui montre un système simple et classique, sera utilisée pour expliquer leurs principes de base.

Supposons que nous voulions reconnaître des hiéroglyphes. Nous avons une image d'un signe, et nous voulons savoir à quel code Gardiner il correspond.

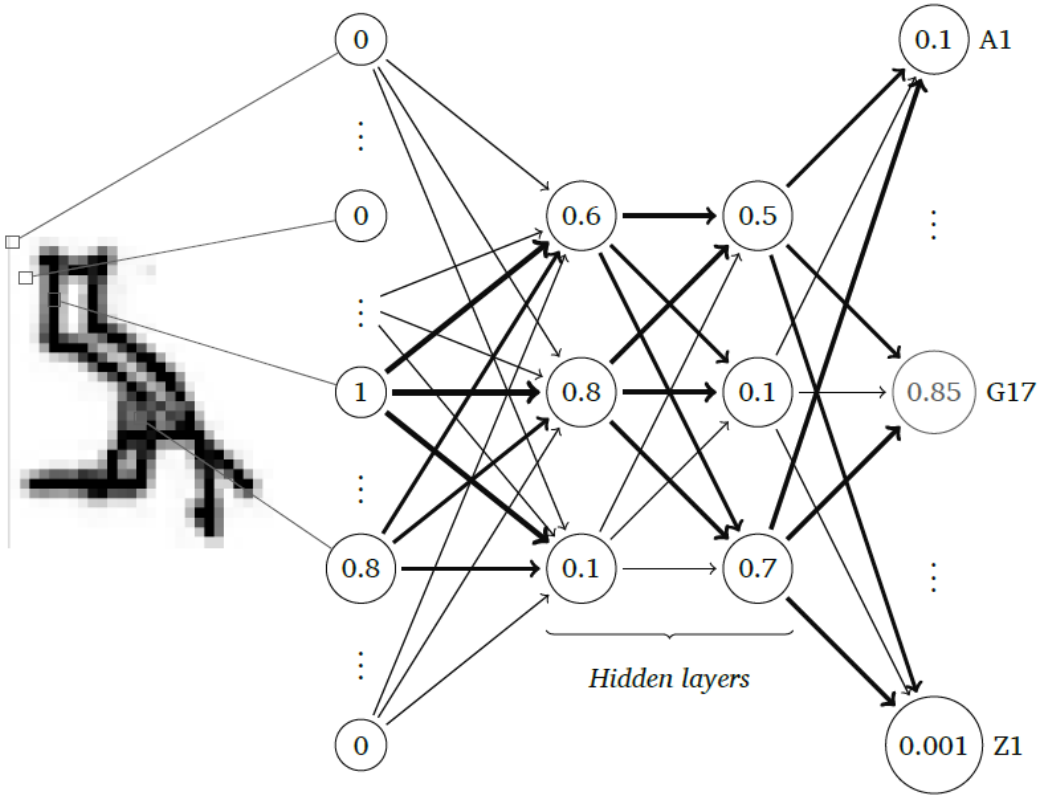


Figure 1 | Un simple réseau de neurones

Le réseau de neurones apprendra comment faire cela à partir d'un large échantillon d'exemples déjà répertoriés. L'entrée est une représentation numérique de l'image du hiéroglyphe et la sortie représentera le code Gardiner du signe. L'entrée serait en fait une longue liste de nombres, qui correspondrait aux pixels de l'image du signe. La sortie dans ce cas serait généralement une autre liste de nombres aussi longue que le nombre de codes Gardiner que nous aimerions discriminer.

Pour chaque code Gardiner, la couche de sortie fournit une valeur numérique comprise entre 0 et 1, qui indique dans quelle mesure l'image originale représente le signe associé au code Gardiner. Par exemple, la valeur de 0,85 dans la figure 1 indique que G17 correspond assez bien à l'image, alors que la valeur de 0,1 du code A1 a tendance à exclure  . Dans le système final, nous pourrions obtenir un résultat intermédiaire, qui indiquera que le système hésite entre un certain nombre de signes similaires.

Entre l'entrée et la sortie, nous avons ce qu'on appelle les « couches cachées » (hidden layers). Elles simulent l'activation neuronale en attribuant un poids numérique à chaque connexion entre deux neurones, qui indique la force de cette connexion et la quantité de signal qu'elle véhicule. Le but du processus de formation est en fait d'apprendre les meilleurs poids possibles pour le système. Le calcul de la sortie de l'entrée obéit à des lois mathématiques relativement simples. La complexité et la capacité d'apprentissage du système proviennent en grande partie de la taille même du réseau.

Nous formerons ensuite le réseau en lui présentant des milliers d'exemples déjà identifiés. Les poids sont à l'origine des valeurs aléatoires, de sorte que les premiers résultats n'ont pas de sens.

Cependant, le système les ajustera pour minimiser la différence entre le résultat calculé

et celui attendu.

Si l'ensemble d'entraînement et le réseau sont suffisamment grands, un entraînement de quelques heures ou de quelques jours donnera un système qui atteindra de bonnes performances (mais généralement pas parfaites). S'il est correctement formé, le système pourra généraliser à partir de ses entrées et classifiera correctement les données qu'il n'a pas encore vues.

4.2 Encodeurs/décodeurs et attention

Les systèmes de réécriture de texte, tels que les traducteurs automatiques ou, dans notre cas, les traducteurs automatiques, sont un peu plus complexes que le système que nous avons présenté ci-dessus. Le principal problème est que leurs entrées et leurs sorties ont une longueur variable.

4.2.1 Encodeur et décodeur

Le système que nous avons utilisé s'appelle un *encodeur/décodeur* (Cho et al., 2014).

L'encodeur construit une représentation numérique pour chaque symbole dans l'entrée. Chaque code Gardiner dans le texte d'entrée sera représenté sous la forme d'une liste de nombres (généralement quelques centaines). Cette représentation est contextuelle : la liste des nombres associés à un symbole sera influencée par les valeurs associées aux symboles voisins (aussi bien avant qu'après le signe).

Le décodeur génère la translittération. Nous commençons par un caractère spécial de « début de translittération » (disons « # »), puis, un caractère à la fois, connaissant à la fois l'entrée hiéroglyphique codée et la translittération déjà générée, il calculera une probabilité pour chaque caractère de translittération existant ; nous choisirons généralement celui avec la probabilité calculée la plus élevée.

Par exemple, si l'entrée est  , et que nous avons déjà généré le texte "*#sw*", le caractère "espace", c'est-à -dire une séparation entre les mots, est le plus susceptible de se produire ensuite.

Une fois que nous avons calculé le caractère suivant, nous le transmettons à son tour au décodeur, et ainsi, caractère par caractère, nous calculons la translittération « la plus probable »[5]. On s'arrête quand on prédit un caractère spécial qu'on a choisi comme "fin de phrase".

L'ensemble du processus de génération du résultat un caractère à la fois peut sembler très local. Pourtant, le système que nous avons construit gère des problèmes qui peuvent nécessiter d'utiliser des informations de la phrase entière pour être résolus. Un certain nombre de caractéristiques architecturales du réseau permettent de le faire. Le codeur étant « bidirectionnel », c'est-à-dire que la représentation construite à une position de signe donnée dépend à la fois du signe précédent et du signe suivant, chaque sortie du codeur dépend de toute la phrase d'entrée. Cette capacité est encore

[5] Ce n'est pas mathématiquement vrai mais fera l'affaire dans cette présentation.

améliorée par le mécanisme d'attention dont nous allons discuter.

4.2.2 Mécanisme d'attention

Lorsque l'encodeur/décodeur calcule la meilleure valeur pour le signe suivant, il doit utiliser une représentation simple pour l'ensemble de l'entrée. La version originale de l'architecture codeur/décodeur utilise la dernière valeur du codeur. Mais il ne fonctionne pas bien sur les longues phrases.

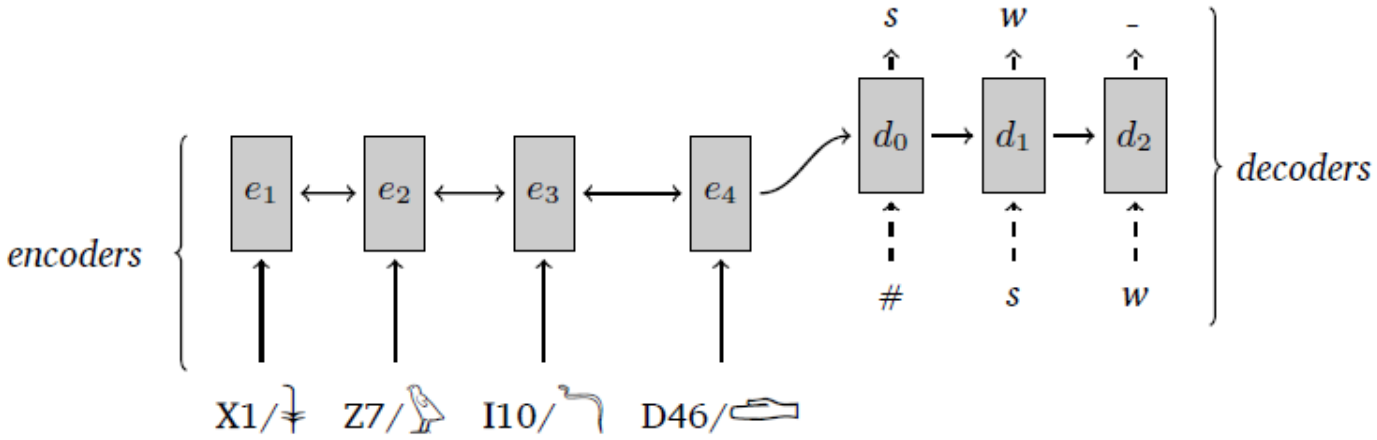


Figure 2 | Encodeur/Décodeur

En conséquence, un nouveau mécanisme, appelé *mécanisme d'attention*, a été introduit (Bahdanau et al. 2015). Lors de la prédiction du signe suivant dans la sortie, il donne un poids, de 0 à 1, à chaque signe dans l'entrée. Ce poids rend compte de la pertinence du signe hiéroglyphique par rapport à la lettre de translittération sur le point d'être générée. Cette attention s'apprend aussi ; ce n'est pas quelque chose de défini à l'avance.

Dans la figure 2, le texte source est  . Si nous avons déjà généré le début de la sortie, « s,w », et essayons de générer le caractère suivant, notre système actuel donne une attention de 0,85 sur  , 0,09 sur  et 0,06 sur le code « début de phrase » (« # » sur la figure). Les autres signes, ( et ), ont des valeurs d'attention négligeables. En quelque sorte, le système, qui "sait" qu'il a déjà généré « sw », concentre son attention sur les glyphes de début de phrase (dont le "#" qui indique précisément leur position de début de phrase). Il décide alors que « sw » est probablement un mot en soi et propose un espace comme caractère suivant.

Si nous allons un peu plus loin, le système donne un poids très lourd (0,991) pour  et propose en fait un "(" comme caractère de sortie le plus probable après l'espace. Au final, nous générons correctement la translittération « *sw* (*ḥr*) *ḏd* ».

Les mécanismes d'attention ont deux caractéristiques intéressantes : ils améliorent la précision du système de translittération et ils donnent un aperçu du fonctionnement interne du système, atténuant l'aspect "boîte noire" des réseaux de neurones en général.

5 Résultats

Comme nous l'avons expliqué plus haut, le corpus est divisé en trois parties :

- un corpus d'apprentissage, sur lequel s'effectue l'apprentissage automatisé proprement dit ;
- un corpus de validation, qui sert à évaluer si l'apprentissage s'est réellement amélioré ou si le système fait un apprentissage par coeur sur le corpus de formation ;
- un corpus de test qui permet de comparer les performances respectives de différents systèmes d'apprentissage.

Les résultats quantitatifs ci-dessous ont été calculés sur le corpus de test, et nous discuterons des résultats qualitatifs du corpus de validation. Nous avons fait un certain nombre d'expériences, en utilisant divers systèmes et architectures, dont *OpenNMT* et *Tensor2Tensor*. Le meilleur résultat correspondait à notre propre implémentation d'encodeur/décodeur avec mécanisme d'attention, à la suite de Luong et al. (2015). Nous avons utilisé *Python 3* et le framework *Keras*. Les couches cachées ont une largeur de 500 neurones ; le réseau a un poids de 8 693 256 et utilise 101 mégaoctets de mémoire. Une description détaillée de notre système sera donnée dans un article technique.

5.1 Résultats quantitatifs

Le corpus de test est composé de 2 728 phrases. Les résultats de notre meilleure sortie sur le corpus de test nous donnent une distance d'édition moyenne de 0,094. Cela signifie que si nous faisons une translittération automatique d'un texte néo-égyptien, nous devrions nous attendre à éditer environ une lettre sur dix.

Le système a trouvé la translittération « or » (0 erreur) pour 1 246 phrases – un peu moins de la moitié de toutes les phrases. Une étude plus approfondie de la distribution des erreurs montre que le système est en moyenne plus performant pour les phrases dont la translittération est comprise entre 25 et 60 caractères et se dégrade légèrement avec des phrases plus courtes ou plus longues.

Si l'on tient compte des différents systèmes d'écriture, la distance de Levenshtein moyenne est de 0,092 sur les textes hiératiques et de 0,111 sur les textes hiéroglyphiques. Cela était attendu, car le hiératique a plus de redondances que le système peut utiliser, et le corpus actuel est plus riche en termes de textes hiératiques.

5.2 Résultats qualitatifs

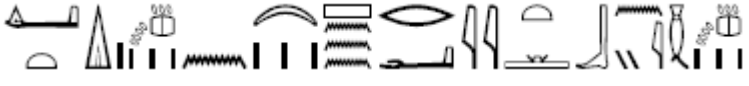
Pour rester en ligne avec nos principes, les résultats qualitatifs sont extraits de la validation du corpus.

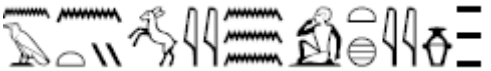
La phrase la plus longue qui a été translittérée avec une précision de 100% (c'est-à-dire en parfait accord avec la translittération construite à partir de l'analyse du corpus) est la suivante (P. BM 10685, v° 2,1) :

Avec la translittération :

ḥry sȝw.w sš.w jmn-ḥtp n šnw.wt pr-ᶜȝ ᶜws n sš pȝ-n-tȝ-wr.tn tȝ ḥw.t nswt bjty wsr mȝᶜ.t rᶜ-stp.n-rᶜ ᶜws m pr jmn m ᶜws m ḥs.t jmn-rᶜ nswt nṯr.w.

De nombreuses petites différences entre le texte généré et le corpus « or » sont liées à des conventions différentes dans les terminaisons de mots.

Par exemple,  (O. DeM. 852, ro, 1,1) a la translittération « or » *rdj.t dj.wn ȝbd 3 šmw rdy.t bnr.w*, alors que le système produit : *rdj.t djw n ȝbd 3 šmw rdy. t bnr*. Un coup d'oeil au lexique de la base de données Ramsès montre que les orthographes singulières et plurielles de *djw* et *bnr.w* ont été enregistrées avec une terminaison Z2 - ce qui n'est pas surprenant en néo-égyptien.

Il y a quelques erreurs flagrantes, telles que  (O. Turin N 57001, r° 2), « Or » : *nȝ nty jb tḫ* ; Calculé : *nȝ nty jw.t* (sic).

Nous avons expliqué plus haut qu'une grande partie du corpus utilisé pour cette expérience utilisait des textes de la base de données Ramsès qui n'ont pas encore été entièrement relus. Dans de nombreux cas, les différences entre les textes « or » et « calculé » indiquent une erreur dans l'analyse initiale ou un problème dans le texte lui-même.

Dans le cas de 

, (O. DeM 571, r° 5) « or » est *jw=w ḥr ᶜš pȝ ḥm-nṯr tpy LACUNA*, alors que la version calculée restitue un n manquant : *jw=w ḥr ᶜš (n) pȝ ḥm-nṯr tpy LACUNA*.

Dans la translittération de P. BM 10335, le système corrige l'analyse actuelle et rend  comme *twtw jw r=j mtr.t* au lieu de l'erreur « or » *twtw.(ḥr) jy.tr=j mtr.t*[6]

Dans la plupart des cas, les phrases pour lesquelles les phrases calculées diffèrent beaucoup de la phrase originale contiennent beaucoup de lacunes. Leur contenu est souvent restauré à partir des textes environnants dans la translittération « or », mais ce contexte n'est pas disponible pour le système actuel, qui fonctionne par phrase. Un exemple simple est 

(P. BM. 10190, 4), « Or » : *jmy* [*snb=ṯ*], Calculé : *jmy* [*ᶜnḫ=k*]. Ici le procédé est simple : la séquence de souhaits, usuelle dans les *Late Ramesside Letters,* implique que la lacune doit être comblée par *snb=ṯ*, alors que le

[6] Bien sûr, dans les deux versions y compris la version « or », il manque un « m » devant mtr.t.

système, qui ne connaît pas du tout le contexte, propose une suite pour *jmy*, qui accidentellement tombe pile sur la bonne solution.

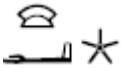
Pour voir quel genre d'erreurs le système peut faire, et ce qu'il fait bien, nous pouvons aller regarder l'une des "pires" lignes en nombre absolu de modifications (KRI 4, 434,3) :

La translittération « or » est ici :

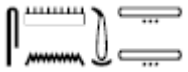
s3-w3ḏyt nḫt-sw ḥy-nfr jmn-m-jn.t **bw-ḳn.tw.f w3w-rn-f** *ḫ ʿ-m-dw3w nb-nfr ṯqut ḳḥ ḫ ʿ-m nwn jmn-m-ḥb* **jpy** *ḫ3m smn -t3.wy* **gs wr** *60 LACUNA*

alors que le résultat calculé est :

s3-w3ḏyt nḫt-sw ḥy-nfr jmn-m-jn.t **ḳn.tw.f w3 rn=f** *ḫ ʿ-sb3-nb-nfr ṯqut ḳḥ ḫ ʿ-m-nwn jmnm ḥb* **jpwy** *ḫ3m smn-t3.wy* **-m-w3st** *60 LACUNA*

Ce cas est complexe car le document est une liste, écrite avec des formes très abrégées. Dans certains cas, notre rendu est meilleur que la translittération « or », par exemple pour *jpwy*. Dans le cas du nom *ḳn.tw.f* vs *bw-ḳn.tw.f*, le système reste plus proche de l'orthographe réelle. Le nom  est aussi probablement plus *ḫ ʿ-(m)-sb3* que *ḫ ʿ-m-dw3* : la base de données de *Deir-el-Medine* préfère le premier, et une recherche rapide dans tout le Corpus Ramsès (dans lequel les deux translittérations ont été utilisées) révèle que le nom ne prend jamais un déterminatif comme  , ce qu'on attendrait du mot *dw3w*, *matin*. Le système est évidemment faux dans les autres cas : il lie *ḫ ʿ-sb3* à *nb-nfr* en un seul nom, et fait deux mots de *w3 rn=f*.

Les signes  à la fin du texte font allusion à un certain nombre de problèmes dans les données actuelles. Premièrement, ces signes ne sont en fait pas analysés dans le corpus de Ramsès. Aucun lemme ne leur est attaché : la translittération *gs wr* est en fait un artefact du système que nous avons utilisé pour produire le corpus de cette étude, et non la responsabilité de l'encodeur (KRITA suggère *m j3w* ici).

Pour ce texte, la façon dont nous avons préparé le corpus écarte certaines informations intéressantes, en particulier sur la disposition du texte et les fins de ligne sur le manuscrit. Dans le cas particulier des listes, qui sont souvent tabulées, cette information est tout à fait pertinente. Le saut de ligne avant  implique que ces signes n'appartiennent pas au nom propre  de la ligne précédente.

On peut aussi remarquer que le système peut raisonnablement bien traiter les soi-disant « écritures de groupe » (voir l'orthographe du nom propre *ḳḥ*, par exemple).

Chaque groupe se voit attribuer une valeur consonantique et, dans les cas où les déterminatifs apparaissent à la fin du groupe, ils ne sont pas confondus avec les terminaisons de mots. Comme les résultats calculés imitent son corpus d'entraînement, il ne tente pas de restituer une voyelle.

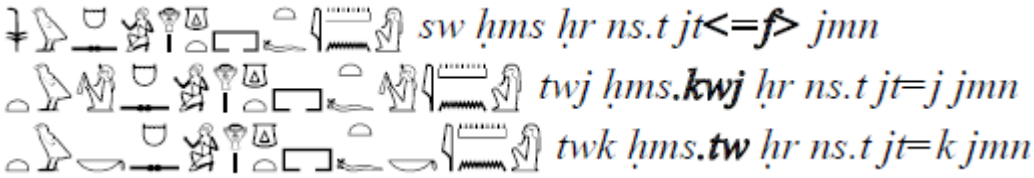
5.3 Accord sujet-verbe

Le système « apprend » au niveau général. Il prend une phrase en hiéroglyphes et produit une translittération. Dans ce processus, les informations sont réparties sur les valeurs numériques calculées par le réseau. Il englobe à la fois des informations au niveau des signes et des connaissances plus globales au niveau de la phrase.

Les outils utilisés par les linguistes, en remplacement paradigmatique, permettent d'explorer ce que système sait et ce qu'il a pu généraliser.

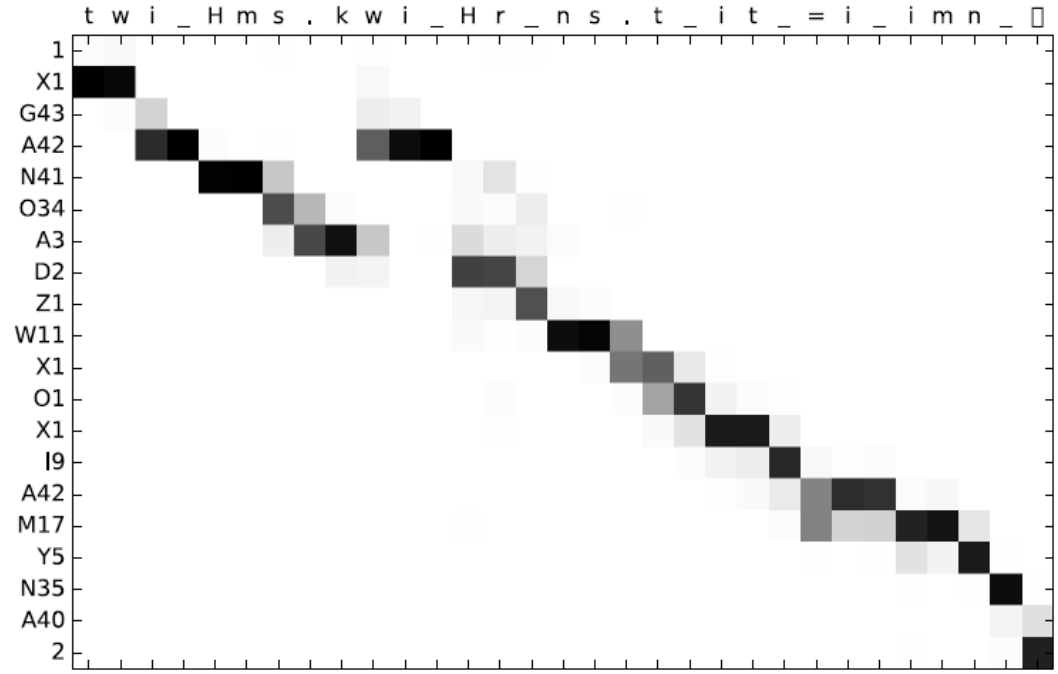
Un point très intéressant est que le réseau est capable « d'apprendre » certaines caractéristiques grammaticales ou, pour être plus précis, que d'une manière ou d'une autre sa représentation interne capture un certain nombre de caractéristiques que nous interprétons comme grammaticales. Pour le démontrer, nous avons testé le système avec des phrases où des parties des morphèmes grammaticaux sont absentes de l'entrée hiéroglyphique.

Nous obtenons les résultats suivants :



Le système a correctement fourni l'ancienne flexion perfective pour le verbe *ḥmsj*, ce qui montre qu'il a une certaine représentation des formes verbales attendues (un premier présent aurait également été possible), et de l'accord sujet - verbe. Les flexions manquantes ne sont pas écrites entre parenthèses comme on pourrait s'y attendre, comme « *twj ḥms(.kwj)* », par exemple, car dans le corpus d'entraînement actuel, les parenthèses ne sont utilisées que pour englober des mots entiers. Les parties manquantes du mot sont silencieusement restaurées.

Les valeurs d'attention pour la phrase à la première personne (*twj ḥms.kwj*) sont affichées dans la figure 3 ci-dessous. Le point intéressant ici est que, lors de l'émission de l'ancienne inflexion perfective "*.kwj*", l'attention du système est focalisée sur le groupe X1-G43-A42, c'est-à-dire  . L'accent est même particulièrement mis sur le signe A42.



(plus sombre signifie une plus grande attention)

Figure 3 | Attention pour les inflexions omises

Maintenant, toutes les anciennes formes perfectives ci-dessus sont documentées dans le corpus du verbe *ḥmsj*. Mais nous voulions savoir si cela fonctionnait pour certaines formes inconnues dans le corpus d'apprentissage, ce qui signifierait que le système est capable d'une sorte de généralisation. Par exemple, nous avons essayé  . La translittération résultante est "*twj nfr.kwj m p3 j.jr n=j nb*", et la structure d'attention, de même, se concentre sur le premier préformant présent.

Une mise en garde cependant. Les réseaux de neurones n'apprennent pas de « règles ». Si l'on considère l'analogie entre eux et le cortex biologique, l'analogie la plus proche pourrait être faite avec les premières couches de neurones dédiées à la vision. En réalité, ils calculent simplement un tas de nombres et optimisent leur poids pour minimiser une perte calculée, mais nous pourrions métaphoriquement les considérer comme modélisant une sorte de traitement de langage « instinctif ». Aucun raisonnement formel n'est utilisé pour choisir entre les différentes possibilités.

Cela signifie que parfois, des exemples très proches peuvent donner des résultats très différents. Par exemple, la première version du réseau que nous avions construit pour cet article traitait correctement les exemples construits sur le verbe *ḫʿj*, « apparaître ». Avec notre dernier réseau, dont les résultats sont globalement meilleurs, l'ancienne flexion perfective n'a pas été restaurée pour le *ḫʿj* à la deuxième personne, et, pour la première personne, elle n'a été restaurée que lorsque le premier préformant présent *twj* a été orthographié comme  et  , et non pas  .

5.4 Décider entre l'ancien perfectif et l'infinitif

Le système capture également des informations sur des verbes spécifiques et leurs environnements possibles. Par exemple,  donnera *twk* (*ḥr*) *wnm db.w*, fournissant la préposition *ḥr*. Cette fois, le système, au lieu de choisir un ancien perfectif comme dans les exemples précédents, fournit correctement un explétif (*ḥr*). entre parenthèses. Dans ce cas, l'information d'attention est également intéressante. L'attention, lors de la production du (*ḥr*), est portée sur l'ensemble du segment  et en particulier sur le verbe *wnm*, lui-même. Cela pourrait suggérer que le système prend en compte à la fois le verbe, son sujet et le début de l'objet.

5.5 Complétion du texte et phraséologie

Comme dans le corpus d'entrée une partie du texte est endommagée, le système dispose de quelques fonctionnalités limitées pour compléter les mots ou les signes manquants.

L'un des exemples les plus intéressants que nous ayons trouvés dans le corpus est le suivant : 

(KRI IV, 266, 5). Les codes « MISSING » sont utilisés lorsqu'un mot du corpus n'est pas associé à une orthographe[7]. La translittération générée est « *jw=tn ḥr ḏd n pr ꜥ3 ꜥws p3y=tn šrj nfr m mn.t* ». Le résultat n'est pas correct mais il est relativement raisonnable, linguistiquement et philologiquement parlant. Notre système a fourni un mot pour chaque élément MISSING et fourni une phrase grammaticale cohérente. On devine qu'il a extrapolé *ḥr ḏd n* à cause de la formule épistolaire trouvée par exemple dans O. DeM 327, 3 : *r-ntt twj ḥr ḏd n jmn m-mn.t* « *Je dis à Amon quotidiennement* ».

5.6 Noms propres

Dans le corpus, les noms propres longs sont coupés. Cela permet au lecteur de les distinguer des phrases normales. Le système apprend à repérer les phrases qui ressemblent à un nom propre et les traite généralement correctement. Nous nous sommes demandé si les noms étaient en quelque sorte mémorisés ou si le système avait développé une représentation qui lui permettrait de traiter les nouveaux noms.

Pour tester cela, nous l'avons essayé avec un nom imaginaire, mais plausible, que nous avons inséré dans une phrase :

[7] Ce décret de Sethy II n'est pas vraiment abîmé ; l'orthographe manquante est une erreur d'encodage. La translittération correcte est *jw=tn (r) ḏd n=w ssnb pr-ꜥ3 ꜥws p3y=tn šrj nfr*.

Il est « correctement » translittéré comme *ḏd=jn sš ḳdw.t* ***ḥr-m-pr-ȝs.t***. Le nom propre est complètement inventé de sorte qu'il ne pouvait pas être appris à travers le corpus.

La valeur d'attention lors de la production de la première lettre du nom propre est portée à la fois sur le signe G5 Horus, et sur le déterminatif final du titre précédent, *sš ḳdw.t*. Cela nous a incités à vérifier ce qui se passerait lors de la suppression du titre. Dans ce cas, le système n'analyse pas *ḥm-m-pr-ȝs.t* comme un nom propre, et opte pour des mots indépendants :

*  *ḏd=jn ḥr m pr ȝs.t*.

Ainsi, le choix de rendre une séquence de signes en nom propre est déclenché à la fois par son organisation interne, mais aussi par le contexte. L'un des différents réseaux que nous avons construits avait un parti pris très intéressant (mais on le surinterprète peut-être) : face à un nom inconnu mais évidemment royal, il générait « *wsr-mȝʿ.t-rʿ* ». Cela montre que le système pondère à la fois les informations contextuelles, y compris ce qu'il a déjà généré, et des informations locales. Dans ce cas, si la chaîne de signes n'avait pas de sens, elle prendrait les informations de ce qui était le plus attendu après un groupe *nsw.t-bjty*.

5.7 Grammaire, phraséologie, contexte et réseaux de neurones

En guise de conclusion partielle, les capacités de notre système suggèrent parfois qu'il a appris quelque chose sur la grammaire du texte. Cela conduit à un certain nombre de questions. La première consiste à décider quelle est la limite entre la grammaire et la phraséologie. Une analyse beaucoup plus approfondie serait nécessaire, par exemple, pour déterminer si notre système choisit de fournir un "*ḥr*" avant un verbe Y parce que le contexte indique que Y doit être une forme active (par exemple, parce qu'il est suivi d'un objet direct), ou simplement parce que le corpus contient plus d'occurrences de "(*ḥr*) Y" que de l'ancien perfectif Y.

Cela étant dit, les études sur les capacités des réseaux de neurones indiquent qu'ils peuvent au moins s'approcher de façon approximmative de certaines caractéristiques grammaticales. En particulier, Linzen (2016) a montré qu'un réseau pouvait apprendre à calculer l'accord sujet-objet en anglais, même lorsque divers substantifs se produisaient entre le sujet et l'objet, ce qui signifie que le système ne prenait pas simplement le nom le plus proche du verbe comme l'objet.

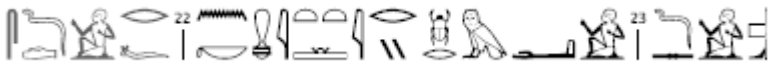
6 Utilisations sur le moyen égyptien

Nous avons testé notre système sur le texte du *Conte du Naufragé*[8], avec une erreur moyenne de 0,175. Il est bien plus grand que celui que nous avons obtenu pour le néo-égyptien, mais reste raisonnable. Comme il est possible de recycler un réseau déjà

[8] The Tale of the Shipwrecked Sailor.

formé sur différentes données en utilisant une technique appelée *apprentissage par transfert*, utiliser notre système actuel comme point de départ pour la formation sur des corpus moyen égyptiens pourrait être une approche valable.

Si nous creusons plus profondément dans nos résultats, le mauvais score est causé par quelques phrases avec un taux d'erreur très élevé, alors que beaucoup d'autres phrases sont très bien rendues. Par exemple, L. 21–24 comporte 16 erreurs pour une longueur de translittération « or » de 62 (c'est-à-dire une erreur moyenne de 0,25) :



translittération « or » : *sḏd=j rf n=k mjtt jry ḫpr.w mᶜ=j ḏs=j šm.kwj r bjꜣ n jty*
translittération calculée : *sḏd=jr=fn=k mjt.t jry ḫpr m-dj=j* (*ḏs=j* est manquant) *šm.kwj r bḥm n sbk*

Parmi ces différences, *m-dj* au lieu de *m-ᶜ* est attendu, car à la fois le *m-dj* néo-égyptien et le *m-ᶜ* moyen égyptien ont la même orthographe hiéroglyphique. D'autres différences sont causées par des habitudes d'encodage (*mjtt* vs. *mjt.t*), par des lacunes dans le corpus d'entraînement (*sbk* vs. *jty*) ou sont de purs échecs (le rendu manquant de *ḏs=j*).

Le système fonctionne mieux sur la phrase suivante qui est presque aussi longue :

translittération « or » : *hꜣ.kwj r wꜣḏ-wr m dp.t nt mḥ 120 m ꜣw=s mḥ 40 m wsḫ=s*
translittération calculée : *hꜣb.kwj r wꜣḏ-wr m dp.t nt mḥ 100 20 m ꜣw=s mḥ 40 m sḫ.wt=s*

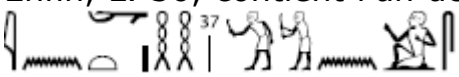
Ici, le principal problème est le faux « b » dans le verbe initial, qui n'est pas dû à un manque d'exemples similaires dans le corpus, car la chanson d'amour de O. DeM 1266 et O CG 25218, l 12 contient la protase « *hꜣ. kwj r mw* ».

Les erreurs systématiques, basées sur les différences grammaticales entre l'égyptien moyen et le néo-égyptien, se produisent comme prévu : le système comprend souvent *sḏm n=f* au lieu de *sḏm.n=f*, par exemple. Il a aussi des difficultés avec l'ancien perfectif après *ᶜḥᶜ.n* comme dans l. 109. Elle tend à les interpréter comme un perfectif *sḏm=f*.

Translittération « or » : *ᶜḥᶜ.n jn.kwj r jw pn jn wꜣw n wꜣḏ-wr*
Translittération calculée : *ᶜḥᶜ.n jnj=k wj r jw pn jn wꜣw n wꜣḏ-wr*
Ou l. 129 :

Translittération « or » : *ᶜḥᶜ.n sbꜣ hꜣ.w*
Translittération calculée : *ᶜḥᶜ.n sbꜣ (ḥr) hꜣj.t*

Dans ce dernier cas, le système a privilégié le modèle narratif de l'Égypte tardive par rapport à celui de l'Égypte moyenne.

Enfin, L. 36, contient l'un des exemples les plus alléchants que le système ait fourni :  est rendu par *m ḫt ḥw.tn=j sy*. Cela donne l'impression que le système "traduit" d'une manière ou d'une autre l'égyptien moyen en néo-égyptien. Cependant ce n'est qu'un artefact de la façon dont notre corpus est construit. Les marqueurs de phrase clivées *jn* et *m* sont enregistrés sous le même lemme dans Ramsès. Par conséquent, lorsque nous avons construit le corpus d'entraînement, tous les *jn* initiaux de la phrase clivées ont été translittérés en *m*. Il reste que le système a correctement identifié le *jn* initial comme un marqueur de phrase clivée. Pour vérifier cela, nous avons essayé de voir s'il différencierait ce *jn* du marqueur de question homonyme en testant la phrase  , qui est correctement rendue par *jn jnk sš*. Il démontre que le système est capable de différencier deux types de contexte de début de phrase  en fonction du contexte.

7 Apprendre les valeurs des signes ?

Les valeurs d'attention ouvrent une fenêtre intéressante sur les parties internes du système. Il était tentant d'essayer de les utiliser systématiquement. Ainsi, nous avons décidé d'essayer d'extraire une sorte de « valeur de signe » du système d'attention. Nous avons exécuté le système sur le corpus d'entraînement (en raison de sa taille) et analysé chaque matrice d'attention. Pour chaque phrase, nous avons extrait les informations suivantes :

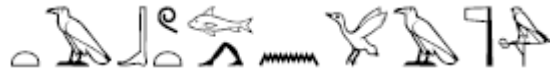
A. pour chaque lettre de translittération, quel hiéroglyphe retiendra le plus l'attention ?
B. pour chaque lettre de translittération, quels hiéroglyphes, pris ensemble, expliquent jusqu'à 99 % de l'attention ?

Pour être plus précis, pour chaque lettre de translittération, chaque hiéroglyphe dans l'entrée a une valeur d'attention, et la somme des attentions est 1. Nous choisissons le signe hiéroglyphique avec la plus grande valeur d'attention. Cela sera utilisé dans la colonne A. Ensuite, nous choisissons la deuxième plus grande valeur d'attention, et nous continuons jusqu'à ce que nous ayons atteint une attention totale de 0,99. C'est la base de la colonne B.

Notez que dans le cas A, nous attribuerons chaque lettre de translittération à un seul hiéroglyphe. Notez également qu'un hiéroglyphe peut ne pas être répertorié du tout, si aucune attention n'est portée dessus.

Le résultat final va des hiéroglyphes à la translittération. Pour chaque signe hiéroglyphique en entrée, nous collectons toutes les lettres de translittération pour lesquelles ce signe est dans le groupe A, et chaque lettre de translittération pour laquelle le signe est dans le groupe B.

Ce que nous calculons réellement n'est pas strictement la valeur du signe telle qu'elle est apprise par un lecteur humain. Un signe hiéroglyphique sera associé à une lettre de translittération lorsqu'il est déterminant dans la décision de choisir cette lettre de translittération spécifique. Pour être plus clair, dans certains cas, la simple apparition d'un  dans un texte hiéroglyphique suffit à décider qu'il faut translittérer *ḏd*, sans même regarder le *d* suivant ; dans ce cas, toute l'attention pour les deux lettres de *ḏd* pourrait être concentrée sur le seul glyphe. Par conséquent, la notion de « valeur » utilisée ici est un peu tirée par les cheveux et différente de l'idée intuitive.

Considérons le petit fragment de texte  "*t3 bw.tn p3 nṯr*". Le tableau 1 décrit les valeurs que nous avons extraites pour chaque occurrence de signe.

Sign												
A	*t3*	SEP	*bw*	.	*t*		SEP	*n* SEP	*p3*	SEP	*nṯr*	SEP
B				*w*	.; SEP	*t* SEP	*t; n*	SEP				

SEP indique les séparations entre mots

Tableau 1 | Valeurs des signe de l'attention

Dans le cas des idéogrammes, cette approche donne souvent leur valeur phonétique réelle. Par exemple,  est correctement identifié comme *nṯr*. En général, cet algorithme tend à segmenter le texte en petits groupes dont les signes sont interprétés ensemble, et à attacher une translittération au premier hiéroglyphe du groupe. Cela fonctionne bien pour  et *p3*, mais a tendance à échouer pour les signes unilitéraux. Ainsi, *bw* dans *bw.t* est principalement lié au signe *w* n'ayant qu'une importance secondaire dans le processus. ,

Le déterminatif et un certain nombre de signes unilitéraux sont également correctement identifiés comme terminaisons de mots ( ,  et ).

L'annexe A énumère les valeurs les plus courantes pour un certain nombre de signes. Pour un grand nombre de signes, ils peuvent apparaître en début ou en fin de mots, d'où le grand nombre d'occurrences de SEP dans les valeurs de signe.

Pour les signes unilitères, qui apparaissent souvent comme des « compléments phonétiques », leur valeur « réelle » se trouve généralement dans la liste B. Cette liste est construite sur la base des signes nécessaires pour expliquer jusqu'à 99 % de l'attention, et non sur les signes qui reçoivent le maximum d'attention. En tant que tel, il est plus approprié pour les compléments phonétiques. Pour des signes comme (M17), comme l'attention semble se concentrer sur les débuts de groupe, il capture dans la liste A un certain nombre de groupes qui commencent par « *j* » : *jw*, *jm*. Il en est de

même pour  ou  .

Pour quelques signes, ce que nous considérons comme leur valeur « standard » ne figure tout simplement pas dans nos cinq principales valeurs. Par exemple, est correctement répertorié comme « *=f* » lorsqu'il apparaît comme un pronom, mais pas comme « *f* » lorsqu'il apparaît au milieu des mots.

La compréhension du système des signes bilitères est plus fiable. Pour la plupart d'entre eux, leur valeur habituelle apparaît au début de la liste A : par exemple, les premières valeurs de  et  sont respectivement *w3* et *šw*. Pour un signe tel que  les premières valeurs sont *p3* et *wsf*.

D'autres types de signes sont généralement relativement bien reconnus.  et des signes similaires sont compris soit comme un suffixe à la première personne, soit comme des limites de mots (SEP), qui est la valeur attendue pour les déterminatifs. De même,  attire l'essentiel de son attention sur les terminaisons de mots SEP.

Le signe  s'entend comme un mot se terminant en SEP ou comme une écriture de consonnes *w, t* ou *s*.

La dichotomie entre  et  est assez intéressante. Dans la base de données, il semble que le premier soit principalement une terminaison de mot (donc un déterminatif), et que le second, beaucoup moins fréquemment encodé, s'encode le plus souvent en *jw* ou *nm.t*.

8 Limites du système

Le système présente un certain nombre de lacunes. Certains proviennent de l'organisation du corpus, et d'autres du système d'apprentissage en profondeur.

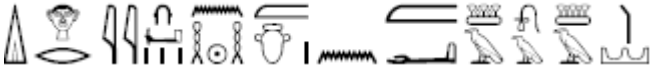
La première limite est l'organisation du corpus en phrases. L'ensemble sera formé phrases, le système essaiera de faire une phrase avec n'importe quel texte. Par exemple,  sera translittéré comme *j.3b*, considérant la phrase d'un mot comme un impératif. Or, ce choix précis est tout à fait raisonnable, mais compliquera certains usages possibles, comme les recherches dans les dictionnaires. Surtout, le système aura des difficultés lorsque le texte n'a pas été segmenté en phrases. Heureusement pour nous, les encodeurs humains ne sont pas toujours cohérents dans la segmentation de leurs textes, ce qui signifie que le corpus contient des phrases qui sont en fait des séquences de propositions relativement indépendantes. Le problème est aussi légèrement atténué par la présence dans le corpus de textes endommagés et de listes diverses.

Plus fondamentalement, la technologie d'encodeur/décodeur construit toujours une

sorte de représentation de longueur fixe de l'ensemble de son texte d'entrée : les données que l'encodeur envoie dans le décodeur. De ce fait, la mémoire système peut devenir relativement déficiente sur des phrases longues, surtout si ladite phrase a une structure relativement uniforme. Dans l'extrait suivant du *Poème de Qadesh*, le décodeur se trouve incapable de garder une trace de la position exacte dans le texte d'entrée et saute un passage entier.

La translittération résultante est :

(§ 1) *ḥ3t-ʿ m p3 nḫtw n nswt bjty wsr-m3ʿ.t-rʿ stp.n-rʿ s3 rʿ rʿ-ms-sw-mry-jmn dd ʿnḫ ḏt*
(§ 2) *jr.n=fm p3 t3 n ḫt3 nhrn p3 t3 n jrṯ*
(§ 3) *m pds* (le système saute *m p3 drdny*)
(§ 4) *m p3 t3 n ms*) *m p3 t3 n ḳrḳš ḥnʿ rk*
(§ 5) *m krkmš ḳdy t3 n ḳdš*
(§ 6) *m p3 t3 n jkrt mšnt*

Dans d'autres cas, une segmentation assez simple peut être manquée (KRI IV, 14,3–4) :  est translittéré *j.dj ḥry.tn nḥḥ m* ***jbwš***, alors que l'ajout explicite d'un génitif « n » impose un meilleur résultat :  *j.dj ḥry.tn nḥḥ m jb n mšwš*. Dans le texte original, non seulement le système a proposé une segmentation erronée, mais il a également complètement négligé les groupes  qu'un système basé sur des règles ou un lecteur humain est peu susceptible de manquer.

Dans ce cas précis, le système fonctionne mieux si on lui fournit un génitif indirect au lieu d'un direct. Cependant, même si les génitifs indirects sont plus usuels en néo-égyptien, la construction directe est largement prédominante dans le cas de *jb*. L'erreur commise par le système ne peut être imputée au contenu du corpus.

Le problème le plus flagrant est que le système peut échouer là où un simple logiciel basé sur des règles fonctionnerait bien. Sa formation lui permet d'utiliser des informations de haut niveau, comme la morphologie, la syntaxe et, en quelque sorte, la phraséologie, pour translittérer une phrase. Mais d'un autre côté, sa compréhension de la valeur du signe individuel est limitée. Ce comportement est d'une certaine utilité

pour le néo-égyptien, car il permet au système d'ignorer les aleph superflus, les consonnes faibles ou les remplissages d'espace. D'un autre côté, il a la fâcheuse habitude de négliger ou de mal interpréter certains signes unilitéraux. Par exemple, si le texte à analyser est le groupe unique  , il le translittère comme "LACUNA" (sic). D'autre part,  est correctement rendu comme *j.gr* : en interprétant le mot comme un impératif, le système en fait une phrase complète et donne une translittération raisonnable.

Un autre type d'erreurs qu'un système basé sur des règles ne commettrait pas est la génération de parties de phrases apparemment déplacées. En translittérant  le système produit ***mry-rˁ*** *ḥˁ-ḫpr-rˁ-snb*, l'initiale étant traitée deux fois.

9 Sensibilité à la taille du corpus

La majeure partie du temps passé en humanités numériques, notamment lors de l'application d'algorithmes d'apprentissage automatique, est consacrée à la préparation du corpus. Trouver sa taille minimale est donc une préoccupation raisonnable.

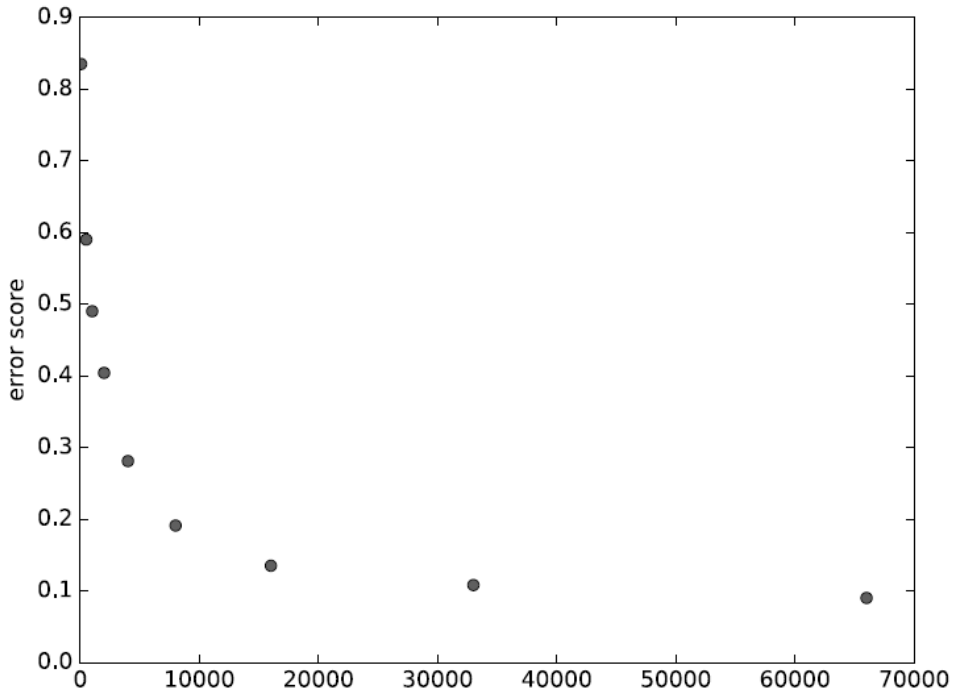


Figure 4 | Score d'erreur par rapport à la taille du corpus

Nous avons entraîné le système sur des corpus réduits pour évaluer l'impact de la taille du corpus, de 62 phrases (environ 1 300 signes) à 66 000 (environ 1 400 000 signes). La figure 4 montre les scores moyens de distance *Levenshtein* correspondants. Il semble que l'on puisse s'attendre à des résultats raisonnablement bons pour un corpus de 16 000 à 20 000 phrases, ou de 340 000 à 400 000 signes, de préférence issus de divers types de textes. Pour donner une idée de la taille minimale du corpus, il est 25 fois plus grand qu'Horus et Seth, notre texte littéraire le plus long, et 17 fois plus grand

que P. BM. 10052, le plus grand texte de Ramsès.

10 extensions possibles

10.1 Améliorations du système actuel

Les extensions et les améliorations de ce système proviendront de deux sources. Premièrement, de meilleurs algorithmes : chaque année depuis 2014 a vu une petite révolution dans le *Deep Learning*. Notre système actuel est basé sur Luong (2015), qui a depuis été surpassé par le convertisseur (Vaswani et al. 2017) et le BERT (Devlin et al. 2019).

L'autre amélioration viendra des données elles-mêmes. Le corpus original contient beaucoup d'informations que nous n'avons pas utilisées. Tout d'abord, pour surmonter le problème des « limites de phrases », nous avons l'intention de produire un nouveau corpus qui sera constitué d'échantillons arbitraires de textes, certains couvrant plus d'une phrase, certains commençant au milieu d'une phrase, etc. L'idée serait de permettre au système de commencer à n'importe quel point arbitraire dans un texte, et même de segmenter les textes en phrases si nécessaire.

Une deuxième amélioration consisterait à introduire intentionnellement des lacunes dans les textes. Il s'agit d'une technique bien connue en apprentissage automatique pour améliorer les capacités de généralisation d'un système et pour le préparer à une entrée « bruyante ». Si nous alimentons le système avec quelques phrases supplémentaires qui contiennent des entrées lacunaires mais une translittération complète, nous pourrions nous attendre à ce qu'il améliore ses capacités à restituer une partie d'un texte manquant. La quantité exacte d'apport lacunaire qui donnera des résultats raisonnables est une question ouverte.

Enfin, il serait intéressant d'essayer de mélanger l'approche actuelle de la boîte noire avec un système basé sur des règles ; dans l'apprentissage automatique, combiner différents systèmes, qui font des erreurs différentes, est une approche plutôt habituelle ; cependant, dans le cas présent (et dans le traitement du langage naturel en général), il n'y a pas de moyen évident de le faire.

10.2 Extension à d'autres corpus

Le système actuel est également une preuve de concept. Cela prouve que la translittération automatisée est possible et donne une estimation de la taille du corpus dont on a besoin pour obtenir des résultats raisonnables. Les limitations liées à l'utilisation du corpus de Ramsès concernant les limites de mots ou les transpositions honorifiques pourraient être levées avec un corpus translittéré clair.

Pour étendre le présent travail au moyen égyptien, une technique dite d'*apprentissage par transfert* (Goodfellow et al 2016) pourrait éventuellement être utilisée pour limiter la taille du corpus nécessaire en réutilisant une partie du réseau actuel. Pour les textes ptolémaïques, plus de travail serait nécessaire, mais les principes généraux utilisés ici s'appliqueraient. Nous serions heureux de travailler avec des collègues qui ont ces

corpus sous forme électronique.

10.3 Extension à d'autres tâches

Le système actuel est bien une boîte noire, même si le système d'attention permet de comprendre partiellement comment la translittération finale est produite. Un certain nombre de lecteurs de cet article ont exprimé leur souhait d'obtenir de meilleures informations sur chaque valeur de glyphe. Étant donné un corpus annoté suffisamment grand, ce serait une tâche relativement facile - l'annotation de symboles dans une séquence est une application bien connue des réseaux de neurones. L'apprendre à partir de notre corpus actuel est beaucoup plus complexe et laissé à de plus amples explorations.

Comme le système actuel « apprend » une certaine grammaire dans une certaine mesure, il est également tentant d'essayer de l'entraîner à étiqueter des mots individuels avec leur partie du discours et leurs inflexions. Une partie du balisage vocal est une application classique du traitement du langage naturel, mais elle est généralement effectuée avec des mots entiers comme entrées. Ici, cependant, l'entrée serait une liste de signes. De tels systèmes traitent automatiquement la variation orthographique, car ils ne reposent pas sur un lexique. Ils ne sont pas très répandus sur les langues modernes, qui ont une orthographe relativement rigide, car leurs performances ont tendance à être plus faibles lorsqu'il s'agit de textes normalisés.

11 Conclusion

Nous avons fourni ici un premier corpus utilisable pour la translittération automatique. Les résultats sont très alléchants. La translittération en tant que telle n'est peut-être pas le principal problème pour les chercheurs, mais c'est une preuve de concept pour d'autres applications : études lexicales, aides pédagogiques, détection d'erreurs dans le corpus, pour n'en nommer que quelques-unes. Le principal inconvénient de la méthode actuelle est l'énorme quantité de données nécessaires. D'autres améliorations pourraient réduire cette exigence.

12 Bibliographie

Bahdanau, Dzmitry, Kyunghyun Cho et Yoshua Bengio. 2015. Traduction automatique neuronale par apprentissage conjoint pour aligner et traduire, dans : 3e Conférence internationale sur les représentations d'apprentissage, ICLR. arXiv preprint arXiv:1409.0473v7.

Barthélemy, François & Serge Rosmorduc. 2011. Intersection of Multitape Transducers vs. Cascade of Binary Transducers: The Example of Egyptian Hieroglyphs Transliteration, dans: Actes du 9e atelier international sur les méthodes d'état fini et le langage naturel, Blois, France, 74–82.

Billet, Sophie 1995. Apports à l'acquisition interactive de connaissances contextuelles. Thèse de doctorat, Université Montpellier II.

Billet-Coat, Sophie & Danièle Hérin-Aime. 1994. Une architecture multi-agents pour un module de système expert en évolution, dans: Dimitris Karagiannis (éd.), Database and Expert Systems Applications, Lecture Notes in Computer Science, Berlin & Heidelberg, 581–590.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk et Yoshua Bengio. 2014. Apprentissage des représentations de phrases à l'aide de l'encodeur-décodeur RNN pour la traduction automatique statistique, dans : Actes de la conférence 2014 sur les méthodes empiriques de traitement du langage naturel (EMNLP), Doha, 1724-1734.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee et Kristina Toutanova. 2019. BERT : Pré-formation des transformateurs bidirectionnels profonds pour la compréhension du langage, ArXiv : 1810.04805 [Cs].

Goodfellow, Ian, Yoshua Bengio et Aaron Courville. 2016. Apprentissage en profondeur, Cambridge, Massachusetts.

Linzen, Tal, Emmanuel Dupoux et Yoav Goldberg. 2016. Évaluation de la capacité des LSTM à apprendre les dépendances sensibles à la syntaxe, dans : Transactions de l'Association for Computational Linguistics 4, 521–535.

Luong, Thang, Hieu Pham et Christopher D. Manning. 2015. Approches efficaces de la traduction automatique neurale basée sur l'attention, dans : Actes de la conférence 2015 sur les méthodes empiriques dans le traitement du langage naturel, Lisbonne, 1412-1421.

Nallapati, Ramesh, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre et Bing Xiang. 2016. Résumé de texte abstrait à l'aide de RNN de séquence à séquence et au-delà, dans : Actes de la 20e conférence SIGNLL sur l'apprentissage informatique du langage naturel, Berlin, 280-290.

Nederhof, Mark-Jan & Fahrurrozi Rahman. 2017. Un modèle probabiliste de l'écriture égyptienne antique, dans : Journal of Language Modeling 5 (1), 131–163.

Neveu, François. 1996. La Langue Des Ramsès – Grammaire Du Néo-Égyptien, Paris.

Rosmorduc, Serge. 2008. Automated Transliteration of Egyptian Hieroglyphs, in: Nigel Strudwick (ed.), Information Technology and Egyptology in 2008: Proceedings of the Meeting of the Computer Working Group of the International Association of Egyptologists (Informatique et Egyptologie), Vienne, 8–11 Juillet 2008, Piscataway, 167–83.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser et Illia Polosukhin. 2017. L'attention est tout ce dont vous avez besoin, dans : Advances in Neural Information Processing Systems 30, 5998–6008.

Vinyals, Oriol, ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever et Geoffrey Hinton. 2015. Grammaire comme langue étrangère, dans : Advances in Neural Information Processing Systems 28, 2773– 2781.

Winand, Jean, Stéphane Polis & Serge Rosmorduc (2015). Ramsès. An annotated corpus of late Egyp tian, in: P. Kousoulis (ed.), Actes du 10e Congrès international des égyptologues, Université de la mer Égée, Rhodes 22– 29 mai 2008, Orientalia Lovaniensia Analecta, Louvain, 1513–1521.

Annexe A. Sélection de « valeurs de signe » calculées

La section suivante contient un certain nombre de valeurs pour les signes sélectionnés, en utilisant les listes de valeurs A et B (section 7) Le nombre après chaque valeur est le nombre d'occurrences de cette valeur trouvée lors de la translittération du corpus d'apprentissage. Pour chaque signe, nous n'avons répertorié en A et B que les cinq valeurs les plus fréquentes. Pour éviter de perdre de l'espace imprimé avec une liste trop longue, nous avons sélectionné pour chaque type de signes quelques cas représentatifs. La liste complète est cependant disponible dans l'archive git du projet (https://gitlab.cnam.fr/gitlab/rosmorse/ramses-trl).

Unilitère

 (G1) **A.** SEP (12737), *y* (5451), - (2650), *w* (1196), . (790)
B. *ȝ* (5216), **SEP** (3483), . (2599), *t* (533), *n* (481)

 (M17) **A.** *iw* (6599), . (6038), **SEP** (5149), *im* (4510), *i* (3840)
B. **SEP** (10399), . (4717), *n* (3757), *w* (2800), *t* (1913)

 (D36) **A.** **SEP** (4219), . (1536), ꜥ **SEP** (989), - (599), *w* (478)
B. **SEP** (2702), ꜥ (1546), . (651), *t* (493), *n* (351)

 (W11) **A.** *gr* (371), *ns.* (216), *gȝ* (159), *ȝ* (110), **SEP** (109)
B. **SEP** (409), . (127), *g* (86), *w* (85), *t* (70)

 (D58) **A.** **SEP** (1513), *bn* (995), *bw* (963), . (631), *bi* (425)
B. *b* (1439), **SEP** (1115), . (303), *w* (255), *t* (164)

 (Q3) **A.** **SEP** (1959), *pt* (1066), . (965), *pn* (743), *pȝ* (622)
B. **SEP** (1384), *p* (1314), - (596), *t* (483), *y* (277)

 (I9) **A.** =*f* **SEP** (12364), *f* **SEP** (2451), **SEP** (1296), *r* (713), **SEP** =*f* **SEP** (215)
B. **SEP** (3274), . (1304), = (1013), *r* (511), *r* **SEP** (471)

 (G17) **A.** *m* **SEP** (9227), *m-* (4618), **SEP** (2764), *y* (1061), *mt* (1009)
B. **SEP** (3684), *m* (2030), . (693), *t* (578),) (494)

 (F32) **A.** *ḥ.* (196), *ḥr* (101), *ḥd* (95), *ḥs* (82), *ḥ*ꜥ (22)
B. **SEP** (57), *t* (53), *ḥ* (27), . (13), *w* (13)

 (N37) **A.** *šm* (870), **SEP** (837), *šr* (335), *š*ꜥ (278), *šf* (130)
B. *w* (628), **SEP** (422), *š* (415), . (116), *t* (104)

 (X1) **A.** *t* (8740), *t* **SEP** (7090), .*t* (6741), **SEP** (5208), *tȝ* (3935)
B. **SEP** (22346), . (8120), *t* (6441), *t* **SEP** (3777), *w* (1733)

 (D46) **A.** **SEP** (5029), *ḏr* (1489), . (1160), *dm* (313), ((206)
B. . (1692), **SEP** (1480), *d* (762), *t* (443), *ḏ* (301)

 (I10) **A.** *ḏd* (3215), *ḏ.* (568), *d* (356), **SEP** (236), (*ḥr*) (206)
B. **SEP** (705), *ḏ* (260),) (131), *t* (81), - (78)

bilitère

 (O29) **A.** ꜣ (1148), ꜣ **SEP** (1076), *pr* (395), ꜣ. (250), *pr*- (114)
B. **SEP** (1255), - (716), . (328), ꜥ (153), *t* **SEP** (86)

 (O29v) **A.** *ȝ*- (2), ꜣ **SEP** (2), **SEP** ꜣ. (1), ꜣ- (1), ꜣ*y* (1)
B. - (61), *pr*- (32), *b*- (23), *b* (21), *n*- (13)

 (V4) **A.** *wȝ* (164), *ȝ* (137), *š* (56), **SEP** (29), . (24)
B. *w* (142), **SEP** (120), . (51), *r* (20), *t* (20)

 (G29) **A.** *bȝ* (1068), *bȝk* (88), *bḏ* (41), (*ḥr*) (37), *bn* (29)
B. *k* (891), **SEP** (149), *b* (37), - (29), *r* (25)

 (G40) **A.** *pȝ* **SEP** (1177), *pȝy* (278), *pȝ* (243), *pȝ*- (55), **SEP** *pȝ* **SEP** (20)
B. **SEP** (305), *y* (85), - (37), *n* (29), . (27)

 (G41) **A.** *pȝ* (9588), *wsf* (444), <*n*> (115), *ȝ* (104), **SEP** (64)
B. **SEP** (4193), *y* (1002), - (981), > (214), > **SEP** (185)

 (M16) **A.** *ḥȝ* (520), **SEP** (65), *ȝ* (58), *ḥȝ* **SEP** (40), *ḥl* (27)
B. **SEP** (188), *w* (80), *y* (78), *ḳ* (65), *p* (43)

 (M12) **A.** *ḫȝ* (647), *ȝ* (356), *ḫb* (160), *ḫr* (123), *1000* **SEP** (122)
B. **SEP** (597), *0* (219), *00* **SEP** (195), *0* **SEP** (175), *000* **SEP** (77)

 (U30) **A.** **SEP** (252), *š* (89), *ȝ* (22), . (20), *y* (5)
B. *ȝ* (236), *ȝ* **SEP** (129), *ȝ.* (53), **SEP** (34), *tȝ* (14)

 (H6) **A.** *mȝꜥ.t* (125), *šw* (119), *mȝꜥ* (76), *mȝꜥ.* (41), **SEP** (27)
B. - (173), **SEP** (153), . (69), *.t* **SEP** (67), *t* (46)

 (H6A) **A.** *šw* (101), *šwy* (42), *ȝꜥ* (7), **SEP** (4), *mȝꜥ* (2)
B. **SEP** (113), ꜥ (51), - (45), *y* (36), *pḥ* (12)

 (U23) **A.** *mḫr* (241), *mḫr* **SEP** (165), *ȝb* (128), *mḥ* (82), *r* (54)
B. **SEP** (300), *r* (67), *mḫr* **SEP** (62), *mḫr* (48), *r* **SEP** (41)

 (K1) **A.** *n* (42), *rmw* (7), *in* (3), *w* (2), *wn* (2)
B. *n* (95), **SEP** (26), *w* (23), . (16), *n.* (14)

 (D4) **A.** *ir.* (1655), *iri* (1224), *ir* (938), *iry* (511), *ir* **SEP** (391)
B. **SEP** (1450), . (1053), *i* (281), *i.* (197), *w* (135)

Sélection d'autres signes

 (Ff1) **A.** **SEP** (3900), *w* (735), *t* (533), . (528), *s* (348)
B. **SEP** (7423), . (1253), *w* (1136), *t* (1040), *s* (717)

 (Z1) **A.** **SEP** (17074), *1* **SEP** (3271), - (1513), **SEP** *1* **SEP** (1040), *7* **SEP** (754)
B. **SEP** (13073), *t* (4431), *n* (2003), = (1750), *w* (1126)

 (Z2) **A.** **SEP** (4293), *w* **SEP** (1316), *5* **SEP** (1077), *3* **SEP** (955), **SEP** *3* **SEP** (794)
B. **SEP** (7963), *w* (1485), *n* (1436), *w* **SEP** (1251), . (1174)

 (V20) **A.** *10* **SEP** (2828), **SEP** (1853), *0* **SEP** (1376), *2* (1338), *20* (570)
B. **SEP** (11652), *0* **SEP** (4925), *0* (1947), *s* (317), **SEP** *60* (230)

 (A1) **A.** =*i* **SEP** (4037), **SEP** (2259), *i* **SEP** (1189), *w* **SEP** (443), . (239)
B. **SEP** (6178), . (771), *n* (667), *w* **SEP** (489), *w* (485)

 (A2) **A.** **SEP** (1418), . (565), *t* **SEP** (291), *n* (206), *w* **SEP** (120)
B. **SEP** (1400), *i* (1223), *r* (608), *t* (604), . (526)

 (A4) **A.** **SEP** (31), . (3)
B. **SEP** (9), *w* (3), .*w* **SEP** (3), *w* **SEP** (3), **SEP** = (2)

 (A5) **A.** **SEP** (1), *imn* (1), *imn* **SEP** (1)
B. **SEP** (4), *t* (2), = (1), - (1), *13* (1)

 (A24) **A.** **SEP** (721), *nḫt* (512), *t* **SEP** (172), *w* **SEP** (108), - (104)
B. **SEP** (2235), - (456), . (362), = (245), **SEP** *m* (220)

 (A42) **A.** =*i* **SEP** (554), **SEP** (269), *i* **SEP** (74), *t* **SEP** (23), **SEP** =*i* **SEP** (13)
B. = (284), **SEP** (169), **SEP** = (53), *t* **SEP** (39), *i* (37)

 (B1) **A.** **SEP** (724), =*t* **SEP** (285), *t* **SEP** (205), *w* (108), *t* **SEP** (95)
B. **SEP** (872), *t* (585), . (317), *t* **SEP** (180), *n* (152)

 (D6) **A.** **SEP** (398), *ptr* (59), *ptr* **SEP** (51), . (21), *t* **SEP** (21)
B. **SEP** (510), **SEP** = (94), = (86), *n* (53), . (45)

 (D40) **A.** **SEP** (1017), . (175), .*t* **SEP** (132), *nḫt* **SEP** (110), *t* **SEP** (110)
B. **SEP** (1342), *w* (295), = (275), . (261), **SEP** = (216)

 (D54) **A.** **SEP** (982), *t* **SEP** (744), .*t* **SEP** (531), *iw* **SEP** (320), . (168)
B. **SEP** (1891), . (866), = (491), *r* (484), **SEP** = (351)

 (D54A) **A.** *iw* **SEP** (25), *nmt* (13), **SEP** (2), *iw* (2), *iw.* (2)
B. **SEP** (11), .*t* (7), - (2), *10* (2), .*t* **SEP** (2)

 (P5) **A.** *ṯ3w* (191), *nfw* (70), *ṯ3* (45), *ṯ3w* **SEP** (38), **SEP** (24)
B. **SEP** (94), *w* (51), - (28), *ṯ* (24), *t* (15)

 (P1) **A.** **SEP** (181), *t* **SEP** (69), .*t* **SEP** (63), *wi3* **SEP** (37), *imw* (34)
B. **SEP** (248), *i* (72), . (70), *n* (35), *t* (35)