Les utilisations possibles de l'Intelligence Artificielle dans la linguistique historique

Benjamin BADOUAILLE, Thomas HORRUT, Eliott CAMOU
Étudiants au Cycle Préparatoire de Bordeaux (CPBx)

Projet tutoré par Rachel BAWDEN
Chercheuse à l'Inria, Paris, dans l'équipe-projet ALMAnaCH

Avril 2023

Résumé

Comprendre la manière dont les hommes s'exprimaient autrefois nécessite de résoudre une grande variété de mystères laissés par le passage du temps.

La linguistique historique étudie les langues anciennes en les comparant à d'autres langues et en considérant que leur évolution suit certaines régularités. Néanmoins, l'accomplissement de certaines tâches est complexe et chronophage.

L'Intelligence Artificielle fait ses preuves depuis quelques décennies dans le Traitement Automatique des Langues, en analysant de manière performante des corpus pour identifier des informations linguistiques. Ses applications ne se sont alors pas arrêtées aux langues contemporaines et ont pu être adaptées aux contraintes de la linguistique historique.

L'intelligence artificielle a notamment pu prouver ses performances dans la restauration de documents écrits, en exploitant l'état de l'art des réseaux de neurones, les Transformeurs, afin de prédire les lettres manquantes de textes dégradés. Une deuxième application consiste à déchiffrer des langues anciennes, en apprenant à associer les mots d'une langue perdue avec ses équivalents dans des langues connues. Enfin, l'intelligence artificielle a aussi permis d'établir des modèles pour la reconstruction d'ancêtres communs de mots apparentés dans différentes langues. Les technologies neuronales ont en effet été mobilisées pour apprendre des règles de changement phonétique à partir de données afin de générer ensuite des solutions hypothétiques. Cependant, la précision des résultats reste discutable.

Mots-clés : Linguistique historique, Intelligence Artificielle (IA), Traitement automatique des Langues (TAL), Restauration de documents anciens, Déchiffrement de langues anciennes, Reconstruction de protoformes.

Abstract

Understanding how people expressed themselves in the past is a challenge that requires solving a wide variety of mysteries left by time and the complexity of the human intelligence.

Historical linguistics studies ancient languages by comparing them with other languages and considering that their evolution follows certain regularities. Unfortunately, the achievement of some tasks in this field is complex and time-consuming. Artificial Intelligence has been proving its worth in Natural Language Processing for several decades, by efficiently analysing corpora to identify linguistic information. Its applications have not been limited to contemporary languages and have been adapted to the constraints of historical linguistics.

In particular, artificial intelligence has been able to prove its performance in the restoration of written documents, by exploiting the state of the art of neural networks, the Transformers, in order to predict the missing letters of degraded texts. A second application consists of deciphering ancient languages, by learning to associate the words of a lost language with their counterparts in known languages. Finally, artificial intelligence has also enabled the reconstruction of common

ancestors of related words in different languages. Neural technologies have enabled the learning of phonetic change rules from data and the generation of hypothetical solutions. However, the accuracy of the results remains questionable.

Keywords : Historical linguistics, Artificial Intelligence (AI), Natural Language Processing (NLP), Restoration of old documents, Ancient languages decipherment, Proto-forms reconstruction.

Remerciements

Nous avons débuté ce projet avec des connaissances infimes en linguistique et en intelligence artificielle. L'objectif était d'appréhender suffisamment de notions dans les deux domaines afin de répondre à notre problématique avec la rigueur attendue et de développer des programmes informatiques. Malgré le peu de connaissances que nous avions nous nous sommes lancés dans un travail assidu jusqu'en avril et tenons à remercier les personnes remarquables qui nous ont soutenus :

— Notre tutrice, Rachel BAWDEN, pour son encadrement, ses précieux conseils, les ressources partagées ainsi que le temps qu'elle nous a accordé. Son expertise en Traitement Automatique des Langues a pu orienter nos apprentissages dans un monde de la recherche dont la découverte a été une expérience inoubliable.

— Notre professeur de Lettres et Communication, Isabelle DORDAN, en charge de superviser les projets. Nous avons eu la chance de croiser nos passions autour du sujet des langues anciennes et avons pu bénéficier de ses connaissances pour enrichir notre culture sur ce thème. Nous retiendrons ses conseils, sa gentillesse et ses encouragements qui ont entretenu notre motivation.

— Notre directeur de Cycle Préparatoire de Bordeaux, Daniel BLAUDEZ, pour nous avoir dédié du temps, au cours de notre cursus, pour la réalisation de notre projet.

— Andre HE, chercheur apprenti à l'Université de Californie (Berkeley) et co-auteur d'un article scientifique sur lequel nous nous sommes grandement appuyés pour rédiger le nôtre. Il nous a transmis les données nécessaires pour l'entraînement neuronal non supervisé que nous souhaitions exécuter et nous a cordialement apporté des explications sur des points de ses recherches sur lesquels nous avions des difficultés.

— Nos professeurs de Mathématiques, Jérôme POIX et Magalie BENEFICE, pour nous avoir aidé dans la compréhension de notions mathématiques encore inconnues.

— Notre professeur d'Informatique, Alexandre Blanché, pour nous avoir facilité l'accès au matériel informatique avancé de l'université.

Table des matières

Table des figures

1 Introduction

À la genèse de ce projet, l'Intelligence Artificielle (IA) était déjà réputée pour ses performances en Traitement Automatique des Langues (TAL), notamment grâce aux logiciels de traduction automatique. Plus récemment, ChatGPT (**chatgpt**) a été rendu public, démontrant au monde entier que la technologie a aujourd'hui la capacité de traiter le sens du contenu textuel. En souhaitant étudier ce domaine, nous nous sommes demandés si ces outils qui vont façonner les sociétés de demain ne pourraient pas être utiles pour comprendre celles d'hier. Ainsi est né ce projet, mêlant les Lettres et l'Histoire avec de l'Informatique, des Mathématiques et de la Cognitique.

À l'occasion du bicentenaire du coup d'« Eurêka ! » de Champollion, qui a débouché sur le déchiffrement des hiéroglyphes égyptiens, nous nous questionnions sur la capacité d'une machine à accomplir la même prouesse pour décrypter d'autres langues perdues encore incomprises. Mais en nous intéressant plus précisément aux préceptes de la linguistique historique, nous avons pris conscience que les problématiques liées à la compréhension de ces langues ne peuvent pas uniquement se résumer à celles d'un code à déchiffrer. Plusieurs d'entre elles disposent néanmoins d'une bonne configuration pour automatiser la production de réponses hypothétiques, étudiées par la suite par les linguistes. Ceci est permis grâce à la mise en place de modèles informatiques, dans lesquels les réseaux de neurones, technologies caractéristiques de l'IA, ont de l'utilité. La question suivante à donc été amenée à être posée : **quel est le potentiel et les applications de l'IA dans la linguistique historique ?**

Dans un premier temps, cette question sera traitée par l'introduction des concepts fondamentaux de la linguistique historique et de quelques-uns de ses enjeux, et nous montrerons en quoi les linguistes ont parfois intérêt à utiliser des modèles informatiques pour appuyer leurs travaux. La réflexion sera suivie de la présentation des bases de l'IA, afin de comprendre dans quelles mesures les réseaux de neurones sont de bons outils pour construire des modèles. Dans une seconde partie, deux tâches de la linguistique historique où l'IA a su faire ses preuves seront synthétisées, ce qui sera l'occasion de découvrir différentes technologies. Dans une dernière partie, une expérience sur la reconstruction de mots historiques avec de l'IA sera décrite, conduisant à révéler les limites de la technologie dans la linguistique historique.

2 La linguistique historique et l'Intelligence Artificielle

La linguistique historique et l'intelligence artificielle ont leur terminologie et concepts propres qu'il est essentiel d'introduire.

2.1 La linguistique historique

2.1.1 Introduction à la linguistique historique

L'étude de l'évolution des langues permet de comprendre la linguistique en général. Bien que les conditions principales de cette évolution nécessitent une étude détaillée, il est important d'établir dès maintenant quelques principes sur la linguistique historique.

Le langage est la capacité d'exprimer une pensée et de communiquer au moyen d'un système de signes doté d'une sémantique, c'est donc tous les signes et mots qui composent une langue. Fruit d'une acquisition, la langue est une des nombreuses manifestations du langage, c'est un mode d'expression propre à une communauté, c'est une institution sociale.
Chaque communauté possède une histoire qui évolue au fil du temps et de ses interactions avec son environnement, amenant avec elle la langue à évoluer tant dans son système que dans son aspect matériel, représenté par des changements phonétiques, morphologiques, syntaxiques et lexicaux.

La linguistique historique est la discipline qui s'intéresse à l'étude de ces phénomènes et des langues envisagées comme systèmes sous tous leurs aspects, notamment dans les changements de ces aspects. Elle étudie l'histoire et l'évolution des langues, et des familles des langues, permettant de caractériser la nature des évolutions. Elle est utilisée, par exemple, dans le déchiffrement de textes anciens, usant d'hypothèses empiriques décrivant la tendance d'une langue à évoluer suivant certaines *règles* régulières.

2.1.2 Les différents principes

La principale méthode de travail consiste à une comparaison entre les différents états d'une même langue ou entre des langues différentes mais issues d'un même ancêtre. Elle cherche des régularités syntaxiques et/ou sémantiques afin de mettre en évidence la relation entre les différents états d'une langue ou d'un groupe de langues. Par exemple, ce travail de comparaison et de recherche permet de retrouver/reconstruire la langue mère à partir d'une langue fille, comme le latin à partir du français.

L'évolution des langues, et plus précisément l'évolution phonétique, se fait généralement de façon régulière. Ces changements sont appelés correspondances régulières (cherchés par la méthode comparative), ce sont des changements prévisibles. Les transformations phonologiques qui changent sans exception un son (ou phonème qui est un élément sonore du langage articulé considéré d'un point de vue physiologique (disposition des organes vocaux) et d'un point de vue acoustique (perception auditive) [1]) *A* de la langue 1 en un autre son *B* de la langue 2 témoignent d'une relation de parenté entre deux langues (cf tableau).

Les phonèmes sont susceptibles d'évoluer dans le temps suivant des « lois phonétiques ». De plus, les phonèmes, dans une langue, peuvent apparaître ou disparaître, comme par exemple le /h/ aspiré au 17ème siècle, cette disparition s'étalant jusqu'au 19ème siècle. Pour étudier leur évolution, des notions de phonétique, basées principalement sur des caractères physiques qui permettent la prononciation, sont abordées. De nombreux sons peuvent être produits et sont répertoriés dans des tableaux phonétiques (**saussure**) (voir annexe B).
L'évolution des phonèmes est due principalement à la loi appelée « *paresse articulatoire* » : ce qui est trop difficile à articuler est automatiquement simplifié. C'est ainsi que les mots ont été raccourcis, par disparition des syllabes les plus faibles. Des consonnes se sont affaiblies : placées entre 2 voyelles (intervocaliques), elles ont été influencées, se sont sonorisées, et ont pu disparaître. Ce phénomène aboutit à la longue à de profondes transformations de la morphologie, comme la chute des déclinaisons, et la modification des conjugaisons. Par exemple, "pater" en latin est devenu "père" en français, et le « s » final de "plus" était autrefois prononcé, mais est devenu muet au fil du temps.

Cette loi de simplification est compensée par la loi d'« *intelligibilité* », la nécessité de clarté dans l'expression : il faut que les mots et les phrases restent compréhensibles ; on a donc conservé certains phonèmes pour éviter que la réduction n'amène des homophones, ou que la phrase devienne obscure. Par exemple, le son [e] a évolué en è ou ê, et le son [o] a évolué en au" ou "eau". De plus, les voyelles suivant une consonne nasale ont tendance à se nasaliser, comme dans le cas de "enfant" où le "a" est nasalisé. [2]

En tenant compte de ces principes, la méthode comparative s'exécute avec des procédures qui permettent d'identifier des mots de plusieurs langues liés entre eux : les cognats. À partir des propriétés phonétiques qu'ils partagent est déduit l'existence hypothétique d'un ancêtre commun : la protoforme, appartenant à une langue ancestrale toute autant hypothétique qualifiée de protolangue.
Le tableau ci-dessous est un exemple de table qui peut être constitué à l'issue de travaux suivant la méthode comparative.

1. Saussure le définit ainsi : « le phonème est la somme des impressions acoustiques et des mouvements articulatoires, de l'unité entendue et de l'unité parlée, l'une conditionnant l'autre : ainsi c'est déjà une unité complexe, qui a un pied dans chaque chaîne » (**saussure**). Ils ont des caractéristiques différentes, se différenciant entre eux de par leur prononciation.
2. Pour plus détails sur les changements phonétiques, nous invitons à regarder l'annexe B.

Sens	Swahili	Dawida
Mâcher	-tafuna	-dafuna
Couper un arbre	-tema	-dema
Arbre	m-ti	m-di
Feu	m-oto	m-odo
Envoyer	-tuma	-duma

Il met en lumière des cognats, en témoignant de correspondances régulières telles qu'entre le /t/ du Swahili et le /d/ du Dawida.

Cependant, il faut aussi identifier les mots provenant de d'autres langues, afin d'éviter de biaiser les mesures et de conduire à une sous-estimation de la profondeur d'un ancêtre commun à plusieurs langues. Pour y remédier, les linguistes peuvent se baser sur des mots simples existant dans chaque langue, avec l'aide par exemple de listes comme celles de Swadesh. (`https://en.wiktionary.org/wiki/Appendix:Latin_Swadesh_list`).

2.1.3 Le besoin de modèles informatiques

L'évocation des listes de Swadesh est suffisante pour comprendre que la discipline de la linguistique historique dispose de ressources composées de tables et de bases de données dont la taille dépasse certainement les capacités de la mémorisation humaine pour pouvoir les exploiter efficacement durant des travaux. L'outil informatique est donc nécessaire pour pouvoir produire des résultats à partir d'un parcours rapide de ces données. Mais, pour l'ordinateur, la difficulté de l'abstraction de la langue est rencontrée. Des outils de modélisation sont donc à mettre en place afin d'obtenir des hypothèses qui vont permettre aux linguistes, grâce à leur niveau de connaissances, d'être plus efficace dans leurs recherches.

Concevoir ce type d'outil fait parti des enjeux du TAL. Dans la partie suivante, les technologies de l'IA qui y sont utiles sont introduites.

2.2 L'intelligence artificielle dans le Traitement Automatique des Langues

2.2.1 Introduction à l'apprentissage automatique

Un nombre important de problèmes informatiques peut être résolu à travers la détermination d'une fonction mathématique f d'un espace vectoriel $\mathbb{K}^n$ vers un espace vectoriel $\mathbb{K}^{n'}$ (avec $\mathbb{K}$ correspondant à $\mathbb{R}$ ou $\mathbb{C}$).

De nombreux cas demeurent où il est difficile – voire impossible – de poser une expression mathématique ou un algorithme pour répondre à certains problèmes. On considère alors f comme hypothétique et on cherche à l'approcher à partir d'un **modèle**, qu'on construit à partir des informations qu'on a à , comme un ensemble de ses points $\{(x_k, y_k = f(x_k)), k \in S\}$, à travers une tâche

dite de **régression**.

Les **réseaux de neurones** sont des outils performants pour établir des modèles. Mathématiquement, ce sont des compositions d'applications non-linéaires et linéaires recevant un vecteur d'entrée représentant une donnée et sortant un vecteur de sortie représentant un résultat dans un format cohérent avec le problème.

Définitions, du neurone au réseau

Le neurone artificiel le plus élémentaire effectue la **somme pondérée** des coefficients du vecteur d'entrée, à laquelle il ajoute une valeur de **biais** pour enfin calculer l'image de la somme à travers une fonction non-linéaire dite **d'activation**. La sortie du neurone est donc un réel ou un complexe. Si on la note y_i, qu'on note x le vecteur d'entrée dans $\mathbb{K}^n$, w_i le vecteur de **poids** associé au neurone, b_i son biais et σ sa fonction d'activation, on obtient :

$$y_i = \sigma(b_i + \sum_{j=0}^{n} w_{ij}x_j) = \sigma(b_i + < w_i, x >) \tag{2.1}$$

(**jurafsky**)

Une **couche de neurones** est la mise en commun d'un nombre arbitraire N de neurones devant prédire des sorties y_i différentes. Leurs vecteurs de poids w_i diffèreront donc. En revanche, leur fonction d'activation est identique. La sortie d'une couche est donc un vecteur y pouvant s'écrire comme dans l'équation 2.2 [3].

$$\begin{aligned} y &= \begin{pmatrix} \sigma(b_1 + < w_1, x >) \\ \sigma(b_2 + < w_2, x >) \\ \vdots \\ \sigma(b_i + < w_i, x >) \\ \vdots \\ \sigma(b_N + < w_N, x >) \end{pmatrix} = \sigma \begin{pmatrix} b_1 + \sum_{j=0}^{n} w_{1j}x_j \\ b_2 + \sum_{j=0}^{n} w_{2j}x_j \\ \vdots \\ b_i + \sum_{j=0}^{n} w_{ij}x_j \\ \vdots \\ b_N + \sum_{j=0}^{n} w_{Nj}x_j \end{pmatrix} \\ &= \sigma \left(\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_N \end{pmatrix} + \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1j} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2j} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \dots & w_{Nj} & \dots & w_{Nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{pmatrix} \right) = \sigma(b + Wx) \end{aligned} \tag{2.2}$$

Un réseau neuronal est ainsi formé à partir de la mobilisation d'une ou plusieurs couches. L'"utilisation de couches intermédiaires, qu'on nomme des **couches cachées**, induit que la machine

3. On y confond la fonction d'activation avec la fonction vectorielle σ s'appliquant indépendamment à chaque coefficient.

puisse être capable d'apprendre à construire des **représentations adéquates** des données pour effectuer la prédiction finalement voulue avec pertinence. On parle alors d'**apprentissage profond** et cette technique offre des réponses face aux difficultés d'abstraction soulevées par les problèmes de TAL.

L'agencement des couches et la manière de calculer la sortie finale à partir de chacune d'elles, qu'on peut rassembler sous le terme de « mode de **propagation** », est un **paramètre architectural** à part entière qu'il faut judicieusement définir en fonction de la tâche à réaliser. Pour traiter de l'utilisation de l'IA dans le TAL, au moins deux principaux types de réseaux de neurones seront introduits au cours de ce chapitre, différant par la nature récurrente ou non de l'enchaînement de leurs couches internes (**jurafsky**).

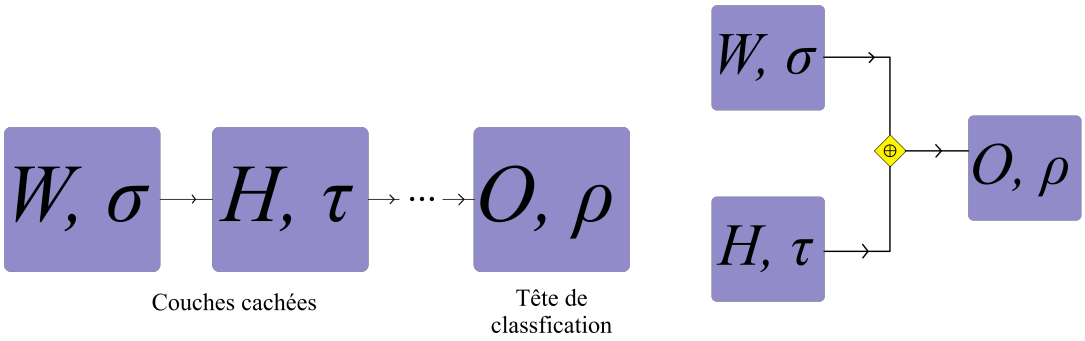


FIGURE 2.1 – Schéma de deux architectures possibles de réseaux de neurones. À gauche est représenté l'empilement le plus simple de couches neuronales, qu'on désigne par le terme de « réseau de neurones à propagation avant ». À droite, une manière de combiner les informations de couches séparées – ici par concaténation vectorielle – a été schématisée.

Processus d'apprentissage

L'entraînement d'un réseau de neurones s'effectue à travers des **ajustements des poids et des biais dans chaque couche**, dans le cadre d'une **minimisation d'une fonction de perte** [4]. Cette fonction doit être choisie judicieusement selon le problème puisqu'elle exprime l'écart entre les sorties prédites et les sorties ciblées au passage de données d'entrée d'entraînement. Sa minimisation est réalisée par un algorithme de **descente du gradient**, avec le gradient de la fonction de perte selon tous les poids du réseau qui est calculé grâce à un procédé de **rétropropagation de l'erreur**, une adaptation pour les réseaux de neurones de celui de la discrimination rétropropagative pour des graphes d'exécution quelconques. Plusieurs implémentations de cet algorithme, qu'on nomme des **optimiseurs** [5], existent et la nature de l'optimiseur dans un modèle fait partie de ses paramètres d'entraînement.

4. aussi appelée fonction de coût, ou d'erreur
5. Adam (**kingma2017adam**) en est un assez connu et est celui qui était prévu d'être utilisé pour le codage de l'expérience.

Des hyperparamètres pour l'entraînement sont également amenés à être définis en pratique, tels que le **taux d'apprentissage**, fixant la valeur maximale de variation qu'un poids ou biais peut subir au cours d'une rétropropagation, le nombre d'**époques** (*epochs*), i.e. de balayages de l'ensemble des données du set d'entraînement pour l'exécution des descentes, et enfin la taille des **lots (*batchs*)**, i.e. des groupes de données d'entraînement à envoyer au modèle en une fois [6] (**jurafsky**)(**fourrier**).

Concevoir correctement

L'évaluation d'un modèle neuronal nécessite un jeu de données dont la nature dépend du problème à résoudre. En TAL, du fait de la diversité des structures rencontrées dans une langue, les données doivent être capables de passer en revue un large éventail de cas pour vérifier que de bonnes généralisations aient été faites par la machine. Pour la même raison, quantifier la qualité des résultats prédits pour comparer plusieurs modèles nécessite d'établir une métrique avec parcimonie. (**fourrier**) Un exemple d'évaluation est synthétisé dans la sous-section 4.1.2.
Le choix des paramètres d'un réseau de neurones pour améliorer sa capacité à pertinemment réaliser une certaine tâche ne se justifie *a priori* que par l'expérience. (**fourrier**) Il est néanmoins bon de noter que des recherches ont été menées pour automatiser le réglage de certaines configurations, comme par exemple celles de **auto_sizing_layer_dim**, qui ont étudié des algorithmes pour trouver les dimensions minimales de couches cachées d'un modèle de langue neuronal aux performances fixées.

2.2.2 Traitement des données

Avant de donner à un réseau de neurones des données au format quelconque, que ce soit pour son entraînement ou pour réaliser des inférences dessus, il est nécessaire de les prétraiter. Le but est de les rendre correctes et compréhensibles pour un traitement numérique. Leur transformation se fait selon le type de tâche souhaitée. Néanmoins, dans sa généralité, les étapes de préparation des données en TAL restent les mêmes.

Normalisation

Tout d'abord, il faut normaliser (nettoyer) la base de données qui peut être, par exemple, un corpus de textes, une liste de mots, provenant de la toile, ou d'un système de transcription audio-visuelle. Dans ces données se trouvent des chaînes de caractères sous un certain format, où de la ponctuation peut apparaître, de même que des lettres en capitale, des chiffres, des caractères spéciaux (comme le dièse que l'on retrouve souvent dans les *tweets*)... Mais une partie de ces éléments peut ne pas avoir d'intérêt à être traitée par les algorithmes ou peut même apporter du bruit aux données d'entraînement pour l'intelligence artificielle. Une première étape consiste à les

6. Cette quantité est pensée pour prévenir les problèmes de saturation de la mémoire des ordinateurs, qui sont favorisés par les ordres de grandeur souvent importante des tailles des jeux de données d'entraînement.

supprimer ou bien à apporter des modifications à l'aide d'outils de traitement de texte [7] (**jurafsky**).

Tokenisation

Après avoir nettoyé la base de données, il faut définir une segmentation des textes en éléments d'entrée pertinents pour faciliter le traitement par l'IA : les *token*s. La tokenisation correspond à la segmentation de chaînes de caractères (comme notre texte) en tokens (mots, sous-mots, caractères, ponctuations). L'ensemble des tokens uniques d'un texte se nomme : vocabulaire.

Prenons un exemple « J'aime les bananes », l'approche traditionnelle est de tokeniser notre texte suivant les blancs et les signes de ponctuation [8] ce qui nous donne les tokens [« J », « ' », « aime », « les », « bananes », « . »] avec un vocabulaire de taille 6 (dans cette phrase tous les tokens sont uniques).

Néanmoins, dans certains cas, cette approche peut être insuffisante, par exemple, lorsque les mots ne sont pas séparés ou lorsque la quantité de mots différents est importante. Ainsi, une technique récente vient compléter l'approche traditionnelle en segmentant un texte en sous-unités linguistiques, appelées sous-mots. Par exemple, le mot « bananes » sera segmenté en deux tokens [« banane », « s »]. Cette méthode peut s'approcher de deux façons, soit par l'application de *règles* (adaptées à la langue) pour découper notre texte en sous mots (comme avec le mot « bananes » dont on sait qu'en français le « s » indique, pour un nom, le pluriel), soit par un algorithme purement statique segmentant de plus en plus un mot en sous-mots au fil que fréquence diminue dans le texte, dont l'un des plus connues pour ses performances et sa simplicité de mise en place est le *Byte-Pair Encoding* (BPE) (**bpe**) [9].

Encodage

Enfin, après avoir tokenisé un texte, il faut convertir ses tokens en vecteurs, car une machine, un réseau de neurones, ne comprend que des nombres et ne sait procéder qu'à des calculs. Il existe différentes façons de convertir un *token* en un vecteur numérique, mais on retiendra deux méthodes : l'encodage *one-hot* (ou 1 parmi *n*) et le plongement lexical (*word embedding*).

L'encodage *one-hot* consiste à créer un vecteur binaire de la taille du vocabulaire $|V|$, c'est-à-dire, que chaque dimension correspond à un mot (ou sous-mots) [10] dans le vocabulaire. Dans ce vecteur binaire, la valeur est 1 pour la dimension correspondant au mot (ou sous-mot) dans le vocabulaire, et 0 pour toutes les autres dimensions. Ainsi, en supposant que la dimension du mot

7. Par exemple, les Expressions Régulières (*REGEX* pour *Regular Expression*) est un outil puissant pour manipuler le contenu textuel. Un autre exemple est la librairie Python `https://www.nltk.org/` fournissant une variété de méthodes et d'outils pour traiter du texte.

8. Vous remarquerez déjà que ce processus ne s'applique pas aux langues, comme le Japonais ou le Chinois, manquant d'espace entre les caractères.

9. Il est bon de citer trois autres algorithmes qui sont tout aussi connues : *unigram language modeling* (**unigramkudo2018**), *WordPiece* (**wordpiece**), *SentencePiece* (**sentencepiece**) qui est une implémentation du BPE et de l'*unigram language modeling*.

10. Remarque : la dimension du vecteur dépend du degré de segmentation en sous-mots.

« bananes » se trouve à la troisième dimension (c'est le troisième mot de notre vocabulaire V), sa représentation correspondra à un 1 à la troisième dimension du vecteur et à des 0 sur les autres dimensions, soit le vecteur :

$$\begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \\ 1 & 2 & 3 & 4 & \dots & |V| \end{bmatrix}$$

(**jurafsky**)

Comme vous pouvez le voir l'avantage de cet encodage est sa simplicité de mise en œuvre. Mais le problème survient quand le vocabulaire devient très grand, les vecteurs générés, par définition, deviennent à leurs tours très grands (le nombre de dimension pour décrire les données augmente) et très dispersés (beaucoup de 0 et peu de 1), ce qui entraîne une augmentation de la complexité et des temps de calcul. De plus, cet encodage ne prend pas en compte les relations sémantiques entre les mots, la similarité entre deux mots ne peut être mesuré, car ils sont encodés de façon indépendante les uns des autres, ce qui peut être particulièrement problématique pour les nombreuses tâches de TAL tel que la compréhension ou traduction d'une langue.

Le plongement lexical, en revanche, permet de représenter les mots en encapsulant leur *sens sémantique* par des vecteurs denses de plus faibles dimensions, c'est à dire des vecteurs de nombres réels de petites tailles. Pour obtenir le plongement des tokens, c'est-à-dire, transformer des tokens en vecteurs, une matrice de plongement est utilisée [11], obtenus par l'entraînement d'un modèle [12]. Cette matrice contient, à chaque ligne, le plongement d'un *token* du vocabulaire, et à chaque colonne correspond une dimension du vecteur de plongement. Par ailleurs, les dimensions de ces vecteurs n'ont pas une représentation claire (**jurafsky**). Comme cette matrice représente un espace vectoriel, ainsi, les propriétés vectorielles peuvent s'appliquer pour nos tokens vectorisés. Par exemple, étudier la similarité entre deux mots revient à calculer la distance entre les deux vecteurs correspondants, ou encore comme la figure 2.2 le montre, les vecteurs peuvent s'additionner/se soustraire permettant d'obtenir un nouveau vecteur qui garde une cohérence sémantique face à ces opérations. (**rumel**) (**lund**)

11. Techniquement, la conversion s'effectue d'abord par un plongement en *one-hot* qui est ensuite converti en plongement lexical via la multiplication matricielle en théorie, mais via un tableau de correspondance (entre les mots et leur plongement lexical) en pratique

12. Généralement, l'entraînement s'effectue de façon non supervisée à partir d'un grand corpus de texte, pour permettre au modèle de se représenter les relations sémantiques et syntaxiques qui peuvent exister entre les mots. Nous pouvons citer, comme exemples de modèles, Word2Vec et GloVe qui sont souvent utilisés pour créer des matrices de plongement lexical. Une méthode plus moderne consiste à apprendre les plongements directement à l'intérieur du modèle, comme le modèle Bert. Pour approfondir le sujet, nous vous invitons à regarder (**jurafsky**).

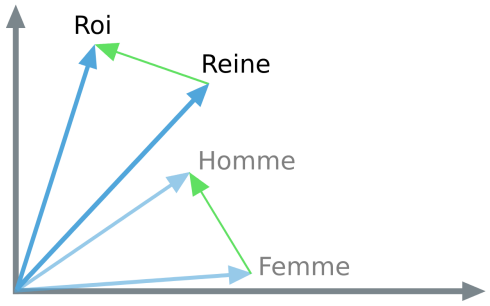


FIGURE 2.2 – Analogie connu montrant les relations sémantiques et algébriques entre les mots dans un espace de plongement lexical. La soustraction du vecteur $\overrightarrow{Femme}$ avec l'addition du vecteur $\overrightarrow{Homme}$ permet d'obtenir le vecteur vert entre ces deux vecteurs. Et en appliquant ce nouveau vecteur au vecteur $\overrightarrow{Reine}$, on obtient le vecteur $\overrightarrow{Roi}$.

2.2.3 Architectures neuronales utiles au TAL

En TAL, le besoin d'identifier des informations **contextuelles** s'est fait ressentir. Pour par exemple relever des diphtongues, dans une tâche de modélisation du changement phonétique, un réseau de neurones devra traiter des **séquences** de *token*s représentant des décompositions de mots en chacun de leurs sons, et ensuite apprendre à établir des liens entre les *token*s.
Une bonne intuition est que des représentations calculées en profondeur dans un réseau de neurones pour certains éléments de la séquence seront utiles pour le calcul de prédictions sur d'autres éléments de cette même séquence.

Dans cette section, des types d'architectures vont être présentés pour leur capacité à construire de manière puissante des contextes dans des séquences d'information.

Réseaux de neurones récurrents

Les Réseaux de Neurones Récurrents (*Recurrent Neural Networks* (RNN)) permettent de prendre en compte un contexte. Étant donné une séquence de n entrées $(x_i)_{1\leq i\leq n}$, le RNN le plus élémentaire est un enchaînement d'une couche cachée avec une autre couche.(**origine_srnn**) La sortie est alors inférée comme dans l'équation 2.3 Les vecteurs des biais seront dorénavant gardés implicites dans les expressions afin de les simplifier.

$$y_i = \sigma(Vh_i) \tag{2.3}$$

La valeur ajoutée du RNN réside dans le calcul de cette couche cachée, qui en plus d'être une fonction de l'entrée x_i de la séquence est également une fonction de la couche cachée qui a été calculée en ayant passé en revue les éléments précédents de la séquence. Une somme pondérée des deux vecteurs est alors effectuée dans la définition de h_i, ce pourquoi deux matrices de poids

interviennent :

$$h_i = \sigma'(Hh_{i-1} + Wx_i) \tag{2.4}$$

La propagation du flux d'informations amenant à la sortie d'un modèle RNN pour un certain élément d'une séquence d'entrée peut être illustrée avec le figuré 2.3. Avec la simplification faite dans le schéma, on remarque aisément qu'on peut construire des empilements d'unités RNN, de la même manière qu'on approfondit l'apprentissage en empilant des couches plus simples.(**jurafsky**)

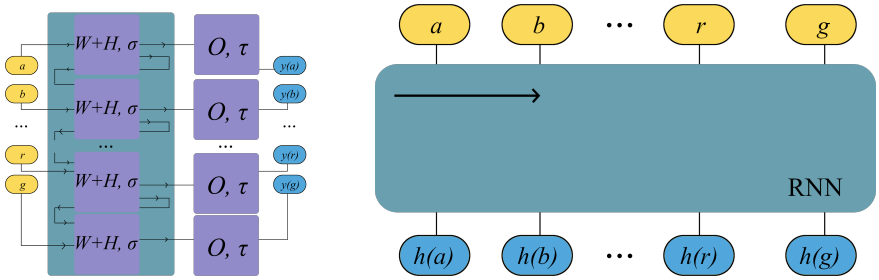


FIGURE 2.3 – Schématisation d'un réseau de neurones récurrent. À droite, il est montré qu'il est possible de directement exploiter les vecteurs cachés, sortis par la couche RNN. Ces derniers sont d'une dimension arbitraire, à définir avec l'unité de RNN.

Dans ces schémas, la séquence d'entrées n'a été parcourue que de gauche à droite, mais il existe des problèmes, comme celui mentionné en introduction de cette section, où le contexte à gauche comme à droite d'un élément de la séquence doit être pris en compte. Une configuration architecturale des RNNs peut justement permettre de construire des contextes de manière *bidirectionnelle* (**bi_rnn**). Dans celle-ci, la couche cachée pour l'entrée x_i est produite à partir de la concaténation des couches cachées ayant balayée respectivement les entrées $(x_1, x_2, ..., x_{i-1}, x_i)$ et dans l'autre sens les entrées $(x_n, x_{n-1}, .., x_{i+1}, x_i)$. (**jurafsky**)

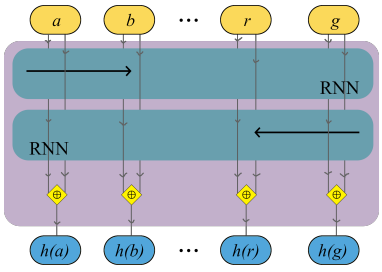


FIGURE 2.4 – Schéma de la structure interne d'un RNN bidirectionnel.

Avec des séquences de tokens dans les données, les RNNs sont alors entraînables. En revanche, il a été observé que la rétropropagation du gradient ne se déroule pas correctement pour des séquences de taille trop importante, ce qui empêche ce type de réseau d'exploiter efficacement une information contextuelle se situant au-delà d'un certain voisinage autour du *token* à traiter (**fourrier**).

Architecture *Long Short-Term Memory*

Les *Long Short-Term Memories*(LSTMs) sont des modèles neuronaux dont l'architecture est une variante à celle des RNNs conçue expressément pour repousser les limitations de ces derniers. Le réseau de neurones est alors entraîné à focaliser son **attention** sur les bons éléments de contexte.

Une unité LSTM est de ce fait plus complexe qu'une unité RNN, mais elle bénéficie de la même modularité. Ainsi, on peut empiler des LSTMs et construire des LSTMs bidirectionnels de la même manière que décrite plus tôt avec les RNNs.

La complexification architecturale peut dans un premier temps se remarquer d'un point de vue externe en notant que, là où une unité RNN n'est composée que d'une couche cachée qui est une fonction de l'entrée et d'une ou plusieurs couches cachées périphériques, l'unité LSTM est composée en plus d'une **couche de contexte** qu'elle renvoie et qui est utilisée pour le calcul des couches cachées et de contexte des autres entrées dans la séquence.

D'un point de vue interne, ces deux couches interagissent à travers des **portes** qui sont entraînées pour rejeter de l'information tout en promouvant d'autre. (**jurafsky**)

Transformeurs

Un type de réseau de neurones a été introduit il y a moins d'une décennie en ayant surpassé les difficultés de parallélisation causées par les connexions récurrentes des modèles introduits précédemment : le *Transformer* (Transformeur en français) (**transformer**). Ce modèle de type encodeur-décodeur se base sur un mécanisme d'**attention multi-têtes** [13].

L'attention multi-têtes permet d'étudier tous les vecteurs d'entrées, comme des mots, en même temps (de façon parallèle). Chaque tête qui la compose se focalise sur un aspect des *interactions* entre les différents éléments de la séquence. Par exemple, dans la phrase « Toto aime les bananes vertes », un tête fera attention à *qui* aime (« Toto »), une autre à l'objet aimé « les bananes », et encore une à l'adjectif utilisé pour l'objet (« vertes »).

Une tête contient un module d'auto-attention (*self-attention*) qui est utilisé pour identifier les interactions de chaque *token* x_i avec tous les autres de la séquence $X = (x_1, x_2, \ldots, x_n)$. Ceci est permis grâce au calcul d'un trio de vecteur pour chaque x_i : un vecteur de requête (*query*) q_i, un de clé (*key*) k_i et un de valeur (*value*) v_i. Le vecteur requête i correspond à celui sur lequel l'attention est portée et sera comparé à tous les vecteurs clés de la séquence ($j \neq i$). Le vecteur valeur sera ensuite multiplié par le poids d'attention calculé avec les autres vecteurs (requêtes et clés). Pour créer ces vecteurs, les vecteurs x_i sont multipliés avec les matrices de poids (W^Q, W^K, W^V) qui sont obtenus à l'entraînement du modèle :

$$q_i = W^Q x_i \quad ; \quad k_i = W^K x_i \quad ; \quad v_i = W^V x_i \tag{2.5}$$

13. Par simplification, nos efforts se concentrerons sur l'attention multi-têtes, en ignorant la décomposition en blocs de *transformer*, contenant chacun une unité d'attention multi-têtes (masqué ou non) ainsi qu'une tête de classification, accompagné de connexions résiduelles et de couches de normalisation **transformer**.

On pose l'ensemble des vecteurs q_i, k_i, v_i sous forme de matrice, respectivement Q, K, V.

Ensuite, le calcul d'auto-attention pour une tête s'effectue par la multiplication du score d'attention (QK^T), normalisé par la racine carré de la dimension d_k de la matrice Q et K, et par la fonction *softmax*, avec la matrice des valeurs V :

$$Attention(Q,K,V) \quad = \quad softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.6}$$

Enfin, l'ensemble des matrices d'auto-attention correspondant à chaque tête est concaténé en une unique matrice qui est multipliée par une matrice de poids W^O (également obtenus à l'entraînement) pour former la matrice résultante de l'attention multi-têtes :

$$MultiHeadAttention(X) \quad = \quad Concat(head_1, head_2, ...)W^O \tag{2.7}$$

où $head_i = Attention(Q_i, K_i, V_i)$
avec $Q_i = XW_i^Q$; $K_i = XW_i^K$; $V_i = XW_i^V$
et i allant de 1 au nombre de têtes définis.

Cependant, au tout début, il a été évoqué que le modèle étudiait l'ensemble des tokens x_i en même simultanément. Ainsi, le modèle ne tient pas compte de l'ordre des tokens, une information pourtant capitale. Par conséquent, pour injecter l'information de la position de nos tokens dans les vecteurs d'entrées, le papier **transformer** propose une solution en additionnant, à nos tokens plongés (vectorisés), des vecteurs positions créés à base de fonction sinus et cosinus [14]. Cette méthode permet, ainsi, à notre modèle de travailler avec tous les tokens x_i, de notre séquence X, en même temps, tout en ayant l'information de leur position (dans la séquence) dans leur vecteur.

Dans certaines situations, notamment dans la tâche de modélisation du langage en TAL, où le but est de prédire le prochain mot d'une phrase, l'entraînement avec ce type de Transformeur est assez inapproprié, car le mot suivant est déjà connu. Le modèle apprendra juste, à son entraînement, de renvoyer en sortie les tokens d'entrées. Pour résoudre ce problème, les chercheurs [15] proposent d'appliquer un masque qui remplacent par des $-\infty$ (qui seront transformés en 0 par la fonction *softmax*) les valeurs de la partie triangulaire supérieure de la matrice QK^T (le score d'attention). Cette variante se nomme l'auto-attention masquée. De ce fait, le champ de visibilité de notre modèle sera réduit au mot qu'il est en train de voir et à ceux qu'il a déjà vus, et pourra prédire de façon plus pertinente (sans triche) les prochains mots de la phrase.

14. Nous invitons le lecteur à regarder le papier de (**transformer**) pour se convaincre de la pertinence de ce choix d'encodage de position. Simplement, retenez que cela permet au modèle de s'entraîner avec la position relative des tokens, plutôt que la position absolue.

15. Ils ont proposé cette variante car leur modèle de *base* répondait aux problématiques des réseaux récurrents pour la traduction automatique.

3 Les contributions de l'IA dans la linguistique historique

Après avoir énoncé quelques principes de la linguistique historique et de l'intelligence artificielle dans le traitement automatique des langues, nous allons aborder des applications de l'intelligence artificielle à deux problématiques de la linguistique historique, la restauration de documents anciens et le déchiffrement de langues anciennes.

3.1 Restauration de documents anciens

La restauration d'inscriptions endommagées nécessite que les épigraphistes s'appuient sur de vastes bases de données pour trouver des parallèles textuels et contextuels. Ces référentiels sont principalement constitués du répertoire mnémotechnique des parallèles d'un chercheur et, plus récemment, de corpus numériques permettant d'effectuer des recherches par « correspondance de chaînes de caractères ». Cependant, des différences dans la requête de recherche peuvent exclure ou obscurcir des résultats pertinents, et il est difficile – voire impossible – d'estimer la véritable distribution de probabilité des restaurations possibles. L'attribution d'une inscription pose aussi problème : si elle a été déplacée ou si des éléments de datation internes utiles manquent, les historiens doivent trouver d'autres critères pour attribuer le lieu et la date de l'écriture (formes de lettres, dialectes, etc). Et cela implique inévitablement, un niveau élevé de généralisation, les intervalles d'attribution chronologique pouvant être très longs.

Ainsi, l'utilisation d'un modèle de langue nommé Ithaca (**deepmind2022**) permet de surmonter les méthodes épigraphiques actuelles en utilisant l'apprentissage automatique sur de grandes quantités de données (quand celles-ci sont disponibles). L'augmentation récente de la puissance de calcul a permis à ces modèles de relever des défis de plus en plus sophistiqués dans de nombreux domaines, y compris l'étude des langues anciennes. Ithaca est un réseau de neurones de type Transformeur, entraîné à effectuer simultanément les tâches de restauration textuelle, d'attribution géographique et d'attribution chronologique. Il a été formé sur des inscriptions écrites en grec ancien et dans le monde méditerranéen entre le VIIe et le XXe siècle. Ce choix s'explique par deux raisons principales. La variabilité du contenu et le contexte des documents épigraphiques grecs en fait un excellent défi pour le traitement des langues, de plus, la disponibilité de corpus numérisés pour le grec ancien est une ressource essentielle pour l'entraînement des modèles d'apprentissage automatique.

Pour commencer, le texte d'entrée est représenté sous forme conjointement de caractères et de mots, en remplaçant les mots endommagés, manquants ou inconnus (*unknown*) par un symbole

spécial « [unk] ».
Ithaca utilise le mécanisme d'attention pour évaluer l'influence des différentes parties de l'entrée **deepmind2022** (telles que les caractères, les mots) sur le processus de prise de décision du modèle. Le modèle est constitué de blocs de transformeur empilés : chaque bloc produit une séquence de représentations traitées dont la longueur est égale au nombre de caractères d'entrée, et la sortie de chaque bloc devient l'entrée du suivant. La sortie finale de l'ensemble des blocs est transmise à trois têtes [1] de tâches différentes qui traitent respectivement la restauration, l'attribution géographique et l'attribution chronologique. Chaque tête est constituée d'un réseau neuronal *feedforward*, spécifiquement entraîné pour chaque tâche.

Ithaca est conçu pour assister et étendre le travail de l'historien. Seul, il a atteint une précision de 62% lors de la restauration de textes endommagés, mais l'utilisation d'Ithaca par des historiens a amélioré leur précision de 25% à 72%, confirmant l'effet synergique de cet outil de recherche **deepmind2022**. Ithaca peut de plus attribuer des inscriptions à leur emplacement d'origine avec une précision de 71% et peut les dater à moins de 30 ans de leur période de vérité, redonnant ainsi vie à des textes clés de l'Athènes classique et contribuant à des débats d'actualité sur l'histoire ancienne. Cette recherche montre comment des modèles tels qu'Ithaca peuvent libérer le potentiel de coopération entre l'intelligence artificielle et les historiens, en transformant la façon dont nous étudions et écrivons sur l'une des périodes les plus importantes de l'histoire de l'humanité.

Ithaca est ainsi un exemple type de l'utilisation des intelligences artificielles dans le domaine de la linguistique historique. Cependant, l'un des enjeux des chercheurs est d'étendre l'action des intelligences artificielles jusqu'au déchiffrement de langues anciennes, indéchiffrables à ce jour. La plupart de ces langues sont mortes, n'ayant plus de locuteurs ni suffisamment de supports physiques permettant son étude approfondie. Le linear B, une écriture mycénienne du deuxième millénaire av. J.-C., a été découvert en Crète en 1900 et compris seulement en 1952, tandis que le linear A, découvert au même moment, n'a toujours pas été déchiffré à l'heure actuelle. On estime qu'à l'heure actuelle, une douzaine de langues sont toujours indéchiffrées. Plusieurs projets en cours de développement, exploitant les données récoltées depuis le début de l'utilisation des intelligences artificielles.

3.2 Déchiffrement de langues anciennes

Le prochain défi des intelligences artificielles est le déchiffrement de langues anciennes. Pour déchiffrer une langue ancienne, l'objectif des linguistes et archéologues est de trouver (s'ils existent) des textes bilingues ou parallèles (un texte et sa traduction), qui permettent à la fois de comparer ces deux langues, et de traduire la langue inconnue, de la même manière que Champollion en 1821. Plus de 200 ans après la découverte de la pierre de Rosette, on peut maintenant affirmer que Champollion a pu déchiffrer l'égyptien autrement que grâce à cette dernière. Elle a en effet servi à amorcer la compréhension du système alphabétique de l'égyptien, mais Champollion a surtout

1. Il faut tenir compte du fait que ces têtes sont différentes de celles contenus dans les blocs de transformeur, les multi-têtes d'attention.

déchiffré l'égyptien grâce au copte, la langue liturgique des chrétiens d'Egypte. L'apport du copte a permis 95 % du déchiffrement, la pierre de Rosette - ou ses équivalents - à peine 5%. Cependant, ces bilingues sont très rares et parfois trompeurs, car rédigés différemment ou erronés dans leur traduction, et les linguistes doivent s'appuyer sur d'autres textes et méthodes.

D'abord, les linguistes essaient de déterminer le type de système d'écriture utilisé : une écriture alphabétique ou alpha-syllabique, une écriture qui note chaque syllabe indépendamment, ou alors un système qui comporte des logogrammes c'est-à-dire des signes utilisés pour noter des mots entiers. Il est important de connaître le système d'écriture pour la déchiffrer, car la méthode et l'approche diffèrent. Il y a parfois des écritures sans espaces (**luo-et-al-2020**), et il faut donc identifier les mots dans les phrases.

Ils travaillent également avec les noms propres contenus dans les textes anciens, permettant de fournir des pistes pour le déchiffrement de l'écriture, comme a pu le faire récemment François Desset pour déchiffrer l'élamite linéaire. C'est grâce à huit vases en argent que l'archéologue français est parvenu à *craquer* le code. Ces huit vases, nommés *vases Gunagis*, vieux de 4 000 ans, présentent des séquences de signes identiques d'une pièce à l'autre. L'archéologue a ainsi pu repérer les noms de deux souverains, Shilhaha et Ebarti II, ainsi que de la divinité Napirisha, qui l'ont petit à petit amené à déchiffrer le reste de l'écriture en élamite linéaire (**linear-elamite-writing**).

Plusieurs travaux sur le déchiffrement de langues anciennes avec de l'intelligence artificielle ont été menés, notamment par le MIT en 2020. (**luo-et-al-2020**) ont réalisé une avancée majeure dans ce domaine : ils ont mis au point un nouveau système capable de déchiffrer automatiquement une langue perdue, sans avoir besoin d'une connaissance *approfondie* de ses relations avec d'autres langues.

L'article propose un modèle de déchiffrement pour extraire les cognats des textes non-segmentés (ou sous-segmentés), sans supposer une proximité entre les langues perdues et les langues connues, ce qui n'était pas le cas dans leur papier précédent(**ugaritic-and-linear-B**).

Les propriétés linguistiques de chaque langue ont été incorporées dans la conception du modèle, telles que la plausibilité phonétique du changement de son, la préservation des sons, et la monotonicité d'alignement [2].

D'abord, l'algorithme apprend à associer les caractères de la langue perdue, par combinaison linéaire, avec ceux de l'API dans un espace de plongement phonétique particulier au déchiffrement (afin d'utiliser les propriétés phonétiques). Ensuite, il essaie de segmenter la séquence en morceaux (censé être un mot dans la langue ancienne), pour les mettre en correspondance avec leurs équivalents dans une langue connue (apparentée).

Les résultats de l'étude sur le gothique, l'ougaritique et l'ibérique montrent que leur modèle peut traiter efficacement les textes sous-segmentés, même lorsque les langues source et cible ne sont pas apparentées. En outre, avec leur modèle, ils introduisent une méthode d'identification des langues proches, dont il trouve correctement les langues apparentées pour le gothique et l'ougaritique. Pour l'ibérique, la méthode n'apporte pas de preuves solides en faveur du basque

2. Cette propriété permet de rendre alignement entre les caractères monotones, l'ordre des éléments à l'entrée du modèle et à la sortie du modèle est préservé.

comme langue apparentée, ce qui correspond à la position privilégiée par les chercheurs actuels (**deepmind2022**).

Finalement, les valeurs phonétiques des caractères perdus mis en correspondance avec les caractères apparentés connus, peuvent servir de point de départ de recherche à la reconstruction des sons perdus qui établiront leur pertinence. Au delà, de l'intégration des caractéristiques phonologiques, ce papier ouvre la voie à de futures améliorations pour la détection des cognats dans la linguistique historique computationnelle. Actuellement, la méthode fonctionne sur une paire de langues. Pour traiter simultanément plusieurs langues, comme c'est souvent le cas dans la tâche de détection de cognats, des travaux supplémentaires sont nécessaires pour modifier ce modèle actuel et sa procédure d'inférence.

4 Étude du cas de l'application de l'IA pour la reconstruction des protoformes

La méthode comparative décrite en section 2.1.2 permet d'émettre des hypothèses dans certains cas quant au sens et à la prononciation de mots dans une langue perdue à partir de comparaisons avec d'autres langues apparentées dont on dispose de plus de connaissances. Ce problème a de l'importance pour la construction de lexiques et pour la traduction de documents dans des langues déjà déchiffrées. Le cas de l'Étrusque l'illustre bien puisque, bien que déchiffré (des informations sur la phonétique et la syntaxe ont été trouvées), les linguistes ignorent encore à quelles langues le comparer pour comprendre le sens d'un grand nombre de ses mots (**bnf_etrusque**).

Des solutions informatiques appliquant cette méthode-là dans des directions différentes ont été développées. Par exemple, l'identification des cognats dans des langues aux origines communes est une tâche pour laquelle des travaux de recherche ont été entrepris et à l'issue desquels de performantes solutions sont sorties (**fourrier**)(**meloni-etal-2021-ab**). Cependant, notre projet s'est focalisé sur celle de la reconstruction des protoformes uniquement, dont de récentes solutions neuronales sont apparues. Puisqu'une d'entre-elles a fait l'objet de la rédaction de notre article scientifique et de l'élaboration d'une expérience, cet exemple d'application de l'IA va être étudié en détail dans ce chapitre. Il a été choisi d'aborder une approche supervisée et une non supervisée afin de pouvoir comparer brièvement les techniques.

4.1 Démarches neuronales récentes

4.1.1 Conceptualisation du problème

Le problème que cette tâche doit résoudre est de prédire la protoforme la plus probable d'être l'ancêtre d'un ensemble donné de cognats, afin de fournir des ressources hypothétiques pouvant orienter le travail des linguistes. Les indices à suivre pour effectuer la reconstruction sont ici les règles de changement phonétique de la linguistique diachronique.

Puisque ces changements sont supposés être réguliers d'une langue à une de ses descendantes, il est possible de déployer des réseaux de neurones pour essayer de les apprendre à partir d'un grand nombre de données, qui sont des paires composées d'un groupe de cognats dans des langues filles et de la protoforme supposée peut être correctement associée [1]. Toutes les chaînes de caractères sont converties au format phonétique afin que les tokens à manipuler ne soient que des caractères de l'API, représentant des sons.

1. En précisant que, dans beaucoup de cas, il y a des ambiguïté. Par exemple si $/b/ > /p/$ dans certains contextes, un $/p/$ dans la langue fille peut venir de plusieurs contextes - un ancien $/b/$ ou un $/p/$ qui n'a pas changé

4.1.2 Description de solutions

Approche supervisée

En disposant des données décrites plus tôt, une des démarches supervisées conçues ces dernières années s'inspire directement de celles mobilisées usuellement dans les problèmes de **traduction automatique**. Le modèle neuronal, conçu ici par **meloni-etal-2021-ab**, est alors doté de l'architecture **encodeur-décodeur**.

L'encodeur reçoit les séquences de plongements associés aux phonèmes (sons) chaque cognat et encode les informations qui vont être passées dans le décodeur pour générer une protoforme plausible. Les deux parties sont chacun des réseaux de neurones à part (**jurafsky**).
Ici, les architectures choisies sont des *Gated Recurrent Units* (**gru**), des variantes aux unités LSTM. Pour de la traduction entre des langues contemporaines, les Transformeurs sont privilégiés, mais il a été révélé que ce dernier type de modèle est plutôt efficace pour des quantités de données d'un ordre de grandeur qui n'est pas satisfait dans le contexte de la tâche ci-présente (**fourrier** ; **araabi-monz-2020-optimizing** ; **bawden-etal-2022-automatic**). Nous avons pu l'expérimenter en tentant d'entraîner des modèles avec ce type d'architecture, en appliquant un procédé dit de *fine-tuning*. Des détails y sont apportés dans l'annexe D.

Les tests menées lors des travaux de **meloni-etal-2021-ab** illustrent bien dans quelles mesures la méthode d'évaluation d'un modèle en TAL est propre à la tâche linguistique à accomplir. Ici, les règles de changement phonétique apprises ont pu être analysées en exécutant des inférences sur des données artificielles, contenant uniquement des syllabes d'intérêt dans les langues romanes et dont on a des connaissances sur leur ancêtre latin. Les erreurs identifiées ont témoigné des limitations du dispositif pour modéliser un phénomène de changement phonétique qui est naturellement complexe. Il aurait en effet fallu que le réseau de neurones soit capable de traiter des informations supplémentaires au format différent, pour être capable par exemple de reconstruire correctement des sons qui ont évolué dans chaque langue fille à travers un processus de neutralisation – et dont il est donc difficile de retrouver le son disparu à partir seulement des cognats.

Approche non supervisée

L'ultime modèle étudié dans ce mémoire s'écarte du précédent en proposant de s'affranchir des bonnes reconstructions associées aux cognats pour s'entraîner. Au contraire, il détermine les protoformes en même temps que d'entraîner les réseaux servant à le faire (**he2022neural**).

Pour cela, le système général est probabiliste. Celui-ci admet que la transformation d'une protoforme vers son cognat dans une certaine langue fille suit un processus dans lequel chaque caractère phonétique de la protoforme peut subir une suppression ou une substitution suivie d'éventuelles insertions de caractères supplémentaires. Les modèles neuronaux interviennent pour apprendre à estimer les probabilités que ces opérations aient lieu sur le i-ème caractère d'une protoforme x, en sachant que la forme dans une certaine langue fille actuellement construite avec les précédentes opérations appliquées sur x soit y' (la nouvelle chaîne de caractères). Les réseaux sont alors composés d'une couche avec une architecture LSTM encodant un contexte à partir de x et de y', suivie

d'une couche FFNN de classification renvoyant une distribution de probabilité sur le vocabulaire [2] (**he2022neural**).

L'heuristique principale de l'approche est, qu'étant donné une reconstruction temporaire x_0 pour un ensemble de cognats $\{y_l, l \in L\}$ donné (avec L désignant l'ensemble des langues filles concernées pour la reconstruction), la bonne protoforme se situe sur un chemin d'édition de taille minimale entre cette reconstruction temporaire et chacun des cognats. Il est donc possible d'établir une liste de candidats $(x_i)_{0\leq i\leq \Gamma}$ dans laquelle elle se trouve. Le critère de sélection du candidat devrait se faire en calculant la probabilité suivante, qu'on développe grâce aux règles de Bayes [3] :

$$\begin{aligned} p(x|\{y_l, l \in L\}) &= \frac{p(x, \{y_l, l \in L\})}{p(\{y_l, l \in L\})} \propto p(x, \{y_l, l \in L\}) \\ &\propto p(x)p(\{y_l, l \in L\}|x) = p(x)\prod_{l\in L} p(y_l|x) \end{aligned} \tag{4.1}$$

Grâce à un programme dynamique utilisant des inférences dans les modèles neuronaux, le calcul des $p(y_l|x)$ est possible. La probabilité $p(x)$ est calculée à l'aide d'un modèle de langue qui est censé déterminer si une chaîne de caractères phonétiques forme un mot ayant des chances d'exister dans la protolangue (**he2022neural**).

Tant que les réseaux de neurones ne sont pas bien entraînés, les échantillons x tirables ne sont pas fiables. Pourtant, c'est bien avec de tels échantillons qu'il va falloir travailler pour créer des données d'entraînement, si l'on souhaite que la non supervision soit conservée.
Les modèles neuronaux gagnent dans ce cas en pertinence à mesure qu'ils reçoivent des entraînements, qui sont alors multiples, et où l'ensemble de couples entrée-sortie pour ajuster les poids neuronaux évolue à chaque fois. Il s'agit d'un algorithme d'espérance-maximisation(**EM**), où un entraînement maximise la vraisemblance d'évènements attendus à un certain moment. Ici, lors d'une itération, pour chaque paire de cognats, on tire aléatoirement une forme ancestrale selon une distribution de probabilités sur les candidats définie à partir des probabilités $p(x_i|\{y_l, l \in L\})$. À partir de cet échantillon x, les évènements attendus sont que certaines modifications aient eu lieu dessus pour obtenir le cognat y_l. La probabilité de ces modifications est calculable à l'aide à nouveau d'un programme dynamique faisant appel aux modèles neuronaux. On construit de cette façon les données d'entraînement de manière non supervisée.(**he2022neural**)

Le modèle a pu être testé pour reconstruire le latin à partir des mêmes données que la démarche supervisée, hormis que les formes latines ont intégralement été utilisées pour son évaluation. Il a pu surpasser en performances un ancien modèle entraîné avec une méthode non supervisée probabiliste. En revanche, aucune étude approfondie des règles de changement phonétique apprises n'a encore été entamée (**he2022neural**).

2. L'ensemble des caractères phonétiques auquel un caractère spécial est ajouter pour symboliser la suppression ou la fin d'une insertion. Le coefficient j du vecteur de sortie représente alors la probabilité que le j-ième caractère du vocabulaire soit inséré sur y'.

3. La dernière égalité est justifiée par la supposition de l'indépendance des apparitions conditionnées des cognats dans les langues filles.

4.1.3 Limites d'applicabilité

Étant donné que le recueil d'informations phonétiques sur une langue ancienne est facilité par l'identification de cognats phonétisés dans des langues descendantes, on en déduit qu'il existe des cas pratiques de reconstruction où la quantité de ressources sur la protolangue peut devenir significativement plus mince que celle des ensembles de cognats procurables. Mettre en place un modèle le moins gourmand possible en données sur la protolangue est donc un objectif présentant de l'intérêt.

Pour aller dans ce sens, la démarche non supervisée dispose d'un net avantage par rapport à l'approche supervisée. Néanmoins, il a été mentionné que la présence d'un modèle de langue de la protolangue était nécessaire, ne la dispensant pas totalement du besoin de mots phonétisés dans la protolangue à reconstruire.

De ce problème a émergé la question de notre article scientifique : celle des propriétés que le modèle de langue doit respecter pour que le modèle non supervisé puisse prédire des résultats de confiance.

4.2 Expérience sur une approche non supervisée

Pour tenter de répondre à la question posée à la fin de la section précédente, une expérience a été imaginée afin d'avoir un premier aperçu de l'impact que la qualité du modèle de langue peut avoir sur la pertinence des résultats prédits par le modèle de reconstruction proposé par (**he2022neural**). Une description synthétique en sera effectuée, mais certains détails seront esquivés, car consultables dans un article rédigé antérieurement (**notreArticle**).

Énormément de points sur cette problématique assez complexe ayant été négligés, une réflexion sur les limites de la méthodologie sera aussi abordée. Enfin, une précision sur le déroulé du développement de l'expérience et de son état de conception sera ajoutée dans ce chapitre.

4.2.1 Méthode

Dans un premier temps, les réseaux de neurones d'édition sont initialisés et tous les algorithmes décrits par le papier sont implémentés afin de pouvoir reproduire la reconstruction du latin dans un maximum de conditions expérimentales en commun avec celles des chercheurs qui leur ont fourni les meilleurs résultats.

Dans un second temps, une multitude de modèles de langue aux architectures parmi celles à base de bi-grammes, de tri-grammes (**huang** ; **simons**) ainsi que la neuronale de type RNN (**jurafsky**) sont entraînés sur des jeux de données de taille différente. Les données proviennent d'un autre jeu de données [4] que celle décrite dans le papier des chercheurs.

Enfin, les itérations des espérance-maximisations sont exécutées pour chaque configuration du modèle de langue et les niveaux de pertinence de chaque modèle sont comparés. Comme pour les expériences décrites dans les papiers associés aux approches non supervisée et supervisée,

4. `https://data.statmt.org/cc-100/` (**conneau** ; **wenzek**)

la métrique utilisée est la distance d'édition moyenne (non normalisée et normalisée) entre les reconstructions latines prédites et les vraies reconstructions.

4.2.2 Critiques

Une première limitation présentée par l'expérience est qu'à vouloir uniquement observer un lien de causalité entre la quantité de données fournies à un modèle pour l'entraîner et sa pertinence, l'influence bien plus déterminante de leur hétérogénéité est négligée. Cette dernière propriété est définie ici par la diversité des structures que l'ensemble de mots phonétisés du latin expose. L'expérience a en effet déjà montrée que cette qualité avait bien plus d'importance que la quantité de données pour l'entraînement de modèles de langue (**camembert**).

Ensuite, rien ne garantit que les résultats pour la reconstruction du latin soient comparables avec ceux pour des reconstructions d'autres langues. Ce fait est d'autant plus vrai que l'approche non supervisée de reconstruction en elle-même n'admet pas de performances identiques selon la configuration généalogique des langues concernées par la reconstruction (**he2022neural**). De plus, il a été mis en évidence que la modélisation du langage semble admettre des limites de performances intrinsèques à la langue en question (**cotterell-etal-2018-languages**).

Par conséquent, la question de l'applicabilité des dernières démarches neuronales de reconstruction nécessiterait d'obtenir des réponses qui font encore l'objet de débats au sein de la communauté scientifique en TAL. En s'abstenant de ces éléments de complexité, une mise en place de l'expérience pourrait toutefois permettre d'observer si les itérations d'espérances-maximisation sont sensibles aux variations de propriétés des modèles de langue et si leur architecture y joue un rôle. La section suivante décrit les avancées de nos travaux de développement allant dans ce sens.

4.2.3 Compte-rendu de l'élaboration

L'exécution de l'expérience nécessitait d'implémenter la quasi-entièreté des algorithmes posés dans **he2022neural**. Nous avons alors eu l'ambition de le faire avec Python, en utilisant la librairie PyTorch, qui permet de manipuler les réseaux de neurones avec modularité et performance. Du fait de la nouveauté qu'a représenté l'entraînement en pratique de tels réseaux, le défi était déjà de taille pour nous, mais des difficultés encore plus importantes sont apparues lorsqu'il a fallu programmer d'autres fonctionnalités périphériques, imposées par la non supervision de l'apprentissage.

Par manque de temps, nous n'avons pas pu surmonter tous les obstacles qui se sont dressés face à nous pour terminer de coder l'ensemble des étapes voulues. Néanmoins, une description détaillée de nos avancées va être faite afin d'apporter des réflexions supplémentaires sur l'approche posée par les chercheurs de Berkeley [5].

Tout d'abord, des questionnements se sont posés à propos des données d'entraînement du modèle de reconstruction que nous avions à disposition. Comme annoncé dans les remerciements, nous avons eu la chance de bénéficier du soutien d'un des auteurs de l'article étudié. Nous l'avions

5. Nos implémentations en Python sont consultables dans ce répertoire

sollicité au départ sur conseil de notre tutrice afin d'obtenir les échantillons de protoformes des trois premières itérations d'espérance-maximisations de son expérience, qui était issus d'un ancien modèle probabiliste de reconstruction conçu par **bouchard**. Avec ces échantillons, il nous a alors joint sous formes phonétique et orthographique les cognats français, roumains, portugais, espagnols et italiens ainsi que les bonnes reconstructions latines qui leur étaient associées. Comme précisé dans son article, leur format différait de celui de **meloni-etal-2021-ab** avec des informations sonores qui ont été retirées. Ce jeu de données a servi de base à notre encodage des tokens sous forme de vecteur *one-hot*. Cependant, des anomalies y ont été repérées au niveau de certains cognats français que la librairie espeak [6] a phonétisé par erreur comme des mots anglais.
Plusieurs causes sont derrière ces coquilles. D'abord, la librairie ne sait pas phonétiser tous les mots du dictionnaire français. Ensuite, certains cognats, même sous forme orthographique, n'étaient pas des mots français ! Nous pensons que ces erreurs n'ont toutefois pas impacté grandement l'entraînement de leurs modèles neuronaux d'édition.
En effectuant des comparaisons avec les reconstructions latines et les cognats dans les autres langues, il a été possible de corriger ces défauts. Nos corrections sont affichées dans le tableau ci-dessous [7]. Il est bon de noter qu'il y a une certaine vigilance à avoir pour déterminer des cognats descendants du latin. En effet, le français par exemple dispose de nombreux mots inspirés de cette langue-là, mais dont la création provient de choix savants et non d'un processus de changement phonétique. Ces mots-ci ne sont alors pas des cognats.

Français	IPA incorrect	Latin	Espagnol	Italien	Français corrigé	IPA corrigé
absinthe	(en)ˈabsɪnθ(fr)	absinthium	ajenjo	assenzio	absinthe	apsɛ̃t
arithmetique	(en)ˌaɹɪθmɪtˈiːk(fr)	arithmetica	aritmética	aritmetica	arithmétique	aʁitmetik
arthrite	(en)ˈɑːθɹaɪt(fr)	arthritis	artritis	artrite	arthrite	aʁtʁit
arthritique	(en)ˌɑːθɹɪtˈiːk(fr)	arthriticus	artrítico	artritico	arthritique	aʁtʁitik
bathe	(en)bˈeɪð(fr)	batt	batir	battere	battre	batʁ
it	(en)ˈɪt(fr)	asi	casi	quasi	quasi	kazi
dos	(en)dˈɒs(fr)	dorsum	dorso	dosso	dos	do
dual	(en)djˈuːəl(fr)	dualis	dual	duale	dual	dɥal
eurythmie	(en)jˈuːɹɪθmi(fr)	eurhythmia	euritmia	euritmia	eurythmie	ø.ʁit.mi
express	(en)ɛkspɹˈɛs(fr)	expressus	expreso	espresso	express	ɛkspʁɛs
hyacinthe	(en)hˈaɪəsˌɪnð(fr)	hyacinthus	jacinto	giacinto	hyacinthe	jasɛ̃t
in	(en)ˈɪn(fr)	in	lino	lino	lin	lɛ̃
indult	(en)ɪndˈʌlt(fr)	indultum	indulto	indulto	indult	ɛ̃.dylt
insipide	(en)ɪnsˈɪpaɪd(fr)	insipidus	insípido	insipido	insipide	ɛ̃sipid
labyrinthe	(en)lˈabɪɹˌɪnð(fr)	labyrinthus	laberinto	labirinto	labyrinthe	labiʁɛ̃t
léthargique	(en)lˌɛɪθɑːdʒˈiːk(fr)	lethargicus	letárgico	letargico	léthargique	letaʁʒik
léthargie	(en)lˈɛɪθɑːdʒi(fr)	lethargia	letargo	letargia	léthargie	letaʁʒi
menthe	(en)mˈɛnð(fr)	mentha	mienta	menta	menthe	mɑ̃t
menthe	(en)mˈɛnð(fr)	mens	mienta	menge	mens	mɑ̃
recreation	(en)ɹˌɛkɹiːˈeɪʃən(fr)	recreatio	recreación	ricreazione	récréation	ʁekʁeasjɔ̃
escheer	(en)ɛʃˈiə(fr)	excadere	escaecer	scadere	échoir	eʃwaʁ
spiritual	(en)spˈɪɹɪtʃˌuːəl(fr)	spiritualis	espiritual	spirituale	spirituel	spiʁituɥɛl
térébinthe	(en)tˈɛɹɪɛɪbˌɪnð(fr)	terebinthus	terebintos	terebinto	térébinthe	te.ʁe.bɛ̃t

FIGURE 4.1 – Tableau détaillant les corrections (en vert) apportées aux données du français. Les informations erronées sont en rouge.

6. `https://github.com/espeak-ng/espeak-ng`
7. Des erreurs dans les cognats espagnols et italiens ont également été identifiées.

Ensuite, un algorithme de détermination des candidats étant donné, un échantillon actuellement sélectionné et ses cognats associés a pu être établi. Une description est proposée dans l'annexe E [8]. Ce qu'il faut retenir est que le nombre de candidats à une certaine étape d'échantillonnage est dominé par un terme en $(2\sqrt{2})^d$, avec d la plus grande distance d'édition entre la reconstruction actuelle et les cognats. Comme chaque candidat aura droit à sa probabilité calculée avec le programme dynamique fourni dans (**he2022neural**), le temps d'exécution risque d'être coûteux. Le modèle a donc tout intérêt à ne pas débuter ses premiers échantillonnages à partir d'échantillons éloignés trop excessivement des cognats, d'où l'utilité des premières itérations d'un autre modèle de reconstruction pour l'initialisation.

Enfin, il a été possible de directement exploiter le code des chercheurs pour les programmes dynamiques, sur lesquels nous avions eu des difficultés de compréhension. Nous ne l'affichons donc pas dans le répertoire de Github, mais ce script a eu une importance centrale et a servi de base pour notre implémentation des inférences dans les modèles neuronaux d'édition.

Dans les fonctions qui restent à développer, une première doit effectuer le tirage aléatoire d'échantillons, qui sera simulé à l'aide d'une chaîne de Markov construite à partir de l'algorithme de Metropolis-Hastings (**metro_has**), et dont la loi stationnaire finale sera donnée par l'équation ci-dessous, étant donné une variable aléatoire X à laquelle on attribue le numéro de l'échantillon tiré dans la liste des N candidats :

$$p(X = i) = \frac{p(x_i|\{y_l, l \in L\})}{\sum_{j=1}^{N} p(x_j|\{y_l, l \in L\})} \tag{4.2}$$

L'entraînement des modèles d'éditions à partir des données établies lors d'une espérance-maximisation est aussi encore à programmer. Pour terminer, l'entraînement des modèles de langue du latin n'a toujours pas été géré, de même que la construction des différents jeux de données pour le réaliser.

8. Elle aura eu l'intérêt d'apporter des détails qui ont été esquivés par **he2022neural**.

5 Conclusion

L'intelligence artificielle a déjà commencé sa révolution dans la linguistique historique. Les différents travaux étudiés ont montré que ces technologies ont su s'adapter aux contraintes du domaine en produisant des modélisations convenables, dans la limite où des données sur une langue sont disponibles.

Bien que performants de par leur rapidité, il a été cependant démontré que les modèles neuronaux font des prédictions dont la fiabilité doit souvent être remise en question, du fait de la variété des cas possibles rencontrés dans une ou plusieurs langues, qu'ils savent difficilement gérer dans son ensemble. Néanmoins, des pistes de réflexion sont fournies aux linguistes grâce à ces outils, permettant de prétraiter leur travail.

L'amélioration du niveau de confiance qu'il sera possible d'accorder aux hypothèses sorties par les machines à base d'intelligence artificielle progressera certainement avec les avancées de la recherche sur la modélisation des langues en général. Les progrès dans le domaine des langues contemporaines étant tout aussi important à considérer que ceux dans le domaine des langues anciennes

A Histoire de la linguistique historique

La linguistique historique aurait été introduite par Sir William Jones (1746-1794) lorsqu'il émet l'hypothèse que le grec, le latin et le sanskrit ont des origines communes, menant à une langue proto-indo-européenne. Une partie de ses hypothèses se seront révélées incorrectes plus tard, mais la langue proto-indo-européenne est toujours étudiée aujourd'hui, se voulant être la protolangue du latin, du gothique, du celte, du grec et du perse.

Si toutes les langues indo-européennes descendent d'une langue primitive commune, quel était donc le peuple qui parlait cette langue, où se situait-il et à quelle époque ? Pour essayer de répondre à cette question, on se base généralement sur des éléments de linguistique et d'archéologie. Plusieurs hypothèses existent, mais c'est actuellement l'hypothèse admise est que le proto-indo-européen viendrait d'un peuple semi-nomade ayant vécu il y a environ 6000 à 7000 ans dans la steppe située au nord de la Mer Noire, aux environs de l'actuelle frontière entre la Russie et l'Ukraine. Ce peuple de guerriers et de cavaliers conquérants aurait entrepris de nombreuses migrations, permettant ainsi la diffusion de leur langue en Europe et en Asie.

B Phonologie et Phonétique

Pour la description de l'appareil, nous nous référerons une figure schématique, où *A* désigne la cavité nasale, *B* la cavité buccale, *C* le larynx, contenant la glotte ε entre les deux cordes vocales.

Dans la bouche on distingue les lèvres α et *a*, la langue β - γ (β désignant la pointe et γ tout le reste), les dents supérieures *d*, le palais, comprenant une partie antérieure, osseuse et inerte *f* et *h*, et une partie postérieure, molle et mobile ou voile du palais *i*, enfin la luette δ.

Les lettres grecques désignent les organes actifs dans l'articulation, les lettres latines les parties passives. La glotte ε, formée de deux muscles parallèles ou cordes vocales, s'ouvre par leur écartement ou se ferme par leur resserrement. La fermeture complète n'entre pour ainsi dire pas en ligne de compte ; quant à l'ouverture, elle est tantôt large, tantôt étroite. Dans le premier cas, l'air passant librement, les cordes vocales ne vibrent pas ; dans le second, le passage de l'air détermine des vibrations sonores. Il n'y a pas d'autre alternative dans l'émission normale des sons. La cavité nasale est un organe tout à fait immobile ; le passage de l'air peut être arrêté par le relèvement de la luette δ, rien de plus ; c'est une porte ouverte ou fermée.

Le système phonologique français est composé de 35 phonèmes : 17 sont dits consonantiques ; ils mettent en jeu les 20 consonnes de l'alphabet, 15 sont dits vocaliques ; ils mettent en jeu les 6 voyelles de l'alphabet et 3 semi-consonantiques ou semi-vocaliques.

Phonème consonantique : [p] de paon, [f] de faon, [b] de banc, [v] de vent, [t] de temps [d] de dent, [s] de sans, [z] de zan, [ʃ] de chant [ʒ] de Jean, [g] de gant, [k] de quand, [l] de lent, [ʀ] de rend, [m] de ment, [n] de non, [ɲ] de pagne

Phonèmes vocaliques : [a] de patte, [œ] de œuf, [ø] de feu, [ã] de pente [o] de côte, [ə] de petit, [o] de cotte, [e] de pré, [ɔ̃] de conte, [ε] de prêt [i] de nid, [ε] de brin, [y] de nu, [œ̃] de brun, [u] de nous.

Phonème semi-consonantique ou semi-vocalique : [w] de oui, [j] maille, [ɥ] de pluie.

C Quelques principes de changements de phonèmes

La palatalisation : La palatalisation est un processus phonétique qui consiste à prononcer une consonne avec un son de palais mou. Par exemple, en ancien français, le son "ch" était prononcé "k" dans certains mots, mais a évolué vers un son de palais mou en français moderne, comme dans le mot "chien".

La diphtongaison : La diphtongaison est un processus phonétique qui consiste à ajouter un deuxième son vocalique à la fin d'un son vocalique existant. Par exemple, en ancien français, le son "i" dans le mot "pierre" était prononcé comme une voyelle simple, mais a évolué vers un son diphtongué "ie" en français moderne.

La disparition de voyelles : La disparition de voyelles est un processus phonétique qui consiste à éliminer une voyelle en position finale ou médiane d'un mot. Par exemple, le mot "femme" était autrefois prononcé avec une voyelle en position médiane, mais cette voyelle est tombée au fil du temps, laissant le mot avec deux consonnes.

La liaison : La liaison est un processus phonétique qui consiste à lier la dernière consonne d'un mot avec la première voyelle du mot suivant. Par exemple, dans la phrase "les amis", la liaison est faite entre le "s" de "les" et le "a" de "amis".

La nasalisation : La nasalisation est un processus phonétique qui consiste à produire un son nasal en passant de l'air par les fosses nasales. Par exemple, en français, les voyelles suivant une consonne nasale, comme le "n" ou le "m", sont nasalisées, comme dans le mot "enfant".

D Fine-tuning

Actuellement, en traduction automatique, le modèle neuronal de type encodeur-décodeur le plus utilisé sont les Transformeurs 2.2.3 qui ont su montré leur performance dans la tâche de traduction automatique. L'une des raisons de ces résultats est qu'ils peuvent être entraînés sur de grande quantité de données, et se crée une « bonne » représentation de celles-ci.

Mais, dans le cadre de la reconstruction d'une langue, il faut s'attendre à avoir *peu* (ou pas) de données sur la langue à reconstituer. Dans notre cas, nous souhaitons reconstruire le latin à partir de langue romanes (français, espagnol, italien, portugais), dont nous possédons une base de données contenant environ 5 000 mots phonétisés par langue (latin inclus). Cette base de données est très petite face aux quantités gigantesques (environ des millions de mots) que peuvent traiter les grands modèles de langue.

Ainsi, nous avons décidé d'utiliser des modèles multilingues pré-entraînés dans la tâche de traduction. En effet, pour pallier le « manque » de données d'entraînement, et espérer utiliser les représentations que ce sont construits ces modèles, c'est-à-dire, les relations sémantiques et syntaxiques qu'ils ont pu apprendre, il a été choisi d'utiliser des modèles qui ont déjà été entraînés sur de grandes quantités de données dans les langues romanes. Ici, nous avons choisi les modèles *mBart* (**mbart**) et *mT5* (**mt5**) de la librairie *transformers* de *HuggingFace* (**hf_lib_transformer**), qui respectent les critères précédents [1].

Maintenant, nous devons affiner *fine-tuning* les modèles choisis pour notre tâche de reconstruction, c'est-à-dire, que nous allons continuer l'entraînement de ces modèles pré-entraînés avec notre base de données pour qu'ils effectuent la tâche de reconstruction. L'affinement d'un modèle neuronal est un procédé courant quand on souhaite utilisé l'apprentissage (les connaissances) d'un modèle pour l'adapter à la tâche souhaitée (similaire à la tâche initial), surtout quand on ne possède pas assez de données pour entraîner notre modèle en partant de zéro. À préciser que ces modèles ont été pré-entraînés sur des données orthographiques, et non phonétiques, ainsi, pour espérer utiliser les représentations de ces modèles, nous avons fournis notre base de données orthographiques.

La mise en place de l'approche s'est effectué par le téléchargement des modèles, puis la configuration de leur entraînement supervisé et de leur évaluation (par distance d'édition (**levenshtein**)).

Après plusieurs tests d'affinement et d'optimisation d'hyperparamètres de nos modèles, nous n'avons obtenus aucun résultat concluant [2]. Plusieurs causes sont possibles, d'abord le manque de données d'entraînement empêchant le modèle d'apprendre la tâche demandée [3], on sait d'ailleurs

1. Les modèles choisis ne sont pas directement ces modèles, mais des modèles reprenant la même architecture affinés pour la tâche de traduction.

2. Une des causes supposées était que les modèles étaient trop grands pour apprendre avec notre base de données, ainsi, des versions plus petites ont été utilisé *tiny-mbart* (`https://huggingface.co/sshleifer/tiny-mbart`) et *mT5-small* (`https://huggingface.co/google/mt5-small`), sans pour autant obtenir de résultats significatifs.

3. Possible que les modèles apprennent avec de plus grandes quantités de données, mais nous sortons du cadre de notre étude consistant à travailler avec des langues où l'on possède peu de données.

que les Transformeurs (et cela est amplifié quand on utilise des grands modèles de langue) ne sont pas très bon dans des situations à basses ressources, mais aussi la supposition que la tâche de traduction automatique est proche de la tâche de reconstruction, enfin on ne repose plus sur le principe de régularité de changement phonétique en travaillant qu'avec des données orthographiques (dont les changements sont plus irréguliers) ainsi il est plus difficile pour nos modèles d'apprendre la tâche. Par conséquent, nous en concluons que l'utilisation de Transformeurs n'est pas approprié pour notre tâche de reconstruction, nous devons nous tourné vers d'autres type de modèle.

E Détermination des nœuds dans des chemins d'édition de taille minimale.

Soient x et y deux chaînes de caractères quelconques. L'objectif est de trouver toutes les chaînes intermédiaires qui ont possiblement été créées pour transformer x en y avec le nombre minimal de modifications appliquées à x. Dans l'illustration de **he2022neural** ci-dessous, les éléments à trouver sont ceux entre « absɛns » et « assɛnte » [1].

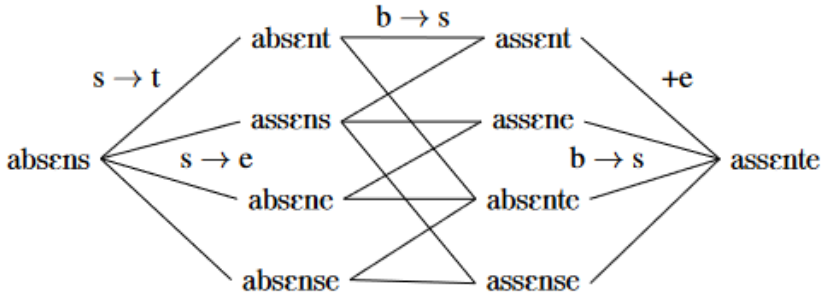


FIGURE E.1 – Représentation des intermédiaires possibles (figuré de **he2022neural**). Ils sont obtenus par application de modifications faisant partie d'un chemin d'édition de taille minimale (ici de 3)

La taille minimale des chemins d'édition est donnée par la distance d'édition entre les deux chaînes, en choisissant d'attribuer à la substitution un coût d'opération de 1, égal à celui de l'insertion et de la suppression. Le calcul de cette valeur d s'effectue avec un programme dynamique proposé par **levenshtein**. On pose M sa matrice de calcul associée, de dimension $(|x|+1, |y|+1)$ [2] et dont le coefficient $M[i, j]$ à la ligne i et à la colonne j contient la distance d'édition entre les i premiers caractères de x ($x[:i]$) et les j premiers caractères de y ($y[:j]$) (pour $i \in [\![0, |x|]\!]$ et $j \in [\![0, |y|]\!]$).

On parcourt récursivement M à partir de la position $(|x|, |y|)$ de sorte à établir toutes les combinaisons possibles de d modifications à appliquer à x pour obtenir y. Il sera alors possible de modéliser chaque modification distincte afin de pouvoir appliquer chacune d'elle indépendamment sur x. À chaque combinaison de k modifications appliquées à x, avec k entre 1 et $d-1$ compris, un nouvel intermédiaire est calculé. Toutes les combinaisons seront donc déterminées.

1. Les mots « absɛnts » et « assɛnts » devraient y être ajoutés pour obtenir l'ensemble des solutions.
2. avec $|a|$ qui désigne ici le nombre de caractères de la chaîne a

E.1 Parcours matriciel

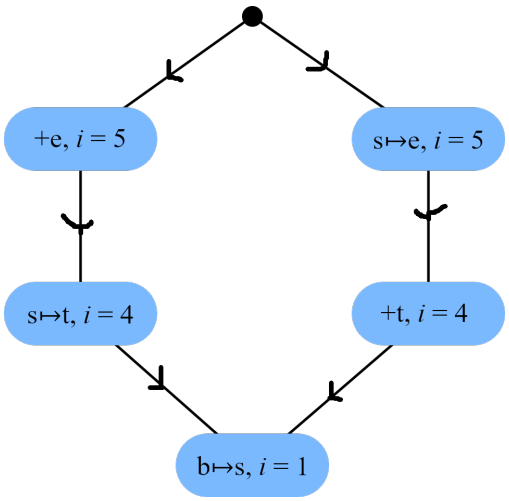
Les insertions, substitutions et suppressions de caractères à effectuer pour modifier le moins possible x s'identifient en remontant la matrice M. Si on note n le coefficient de M qui est rencontré à un certain moment dans le parcours, il est possible de s'assurer que le chemin emprunté balaiera toujours une séquence de d éditions ayant transformé x vers y dès lors qu'on s'oriente toujours vers « le haut et la gauche » et que n diminue en permanence. (**jurafsky**) On créé un graphe orienté G ne contenant d'abord qu'un nœud vide O et dans lequel on stocke les modifications trouvées. Pour $n = M[i, j]$, l'algorithme de parcours est alors le suivant :

- si $n = 0$, l'algorithme est terminé.
- sinon,
  - si $M[i-1, j-1] = n-1$, alors le caractère $x[i]$ a été substitué par le caractère $y[j]$. On ajoute cette modification en tant que nœud du graphe, en le connectant au noeud actuel et on poursuit récursivement le parcours à la position $(i-1, j-1)$.
  - si $M[i-1, j] = n-1$, alors l'insertion au moins du caractère $y[j]$ a eu lieu après le caractère $x[i]$. Les mêmes actions que précédemment sont effectuées, sauf qu'on poursuit le parcours à la position $(i-1, j)$
  - si $M[i, j-1] = n-1$, alors le caractère x[i] a été supprimé. On ajoute cette modification au graphe de la même façon et on poursuit le parcours à la position $(i, j-1)$

Pour les deux chaînes du figuré E.1, le parcours effectué dans la matrice et le graphe d'édition construit sont illustrés dans le figuré ci-dessous.

		a	s	s	ε	n	t	e
	0	1	2	3	4	5	6	7
a	1	0	1	2	3	4	5	6
b	2	1	1	2	3	4	5	6
s	3	2	1	1	2	3	4	5
ε	4	3	2	2	1	2	3	4
n	5	4	3	3	2	1	2	3
s	6	5	4	3	3	2	2	3

(a) Illustration du parcours de la matrice de calcul de la distance d'édition pour les mots « absents » et « assents »



(b) Représentation du graphe d'édition calculé pour les mots « absents » et « assents »

E.2 Modélisation d'une modification

Avec la représentation par un graphe des modifications comme illustrée ci-dessus, presque toutes les informations sont correctement définies pour construire les intermédiaires. Cependant, dans le cas où plusieurs caractères ont été insérés à la même position i de x, il faut veiller à ce qu'ils restent ordonnés correctement dans l'intermédiaire produit, indépendamment de l'ordre avec lequel les insertions seront appliquées à x — puisqu'il est intéressant ici de travailler avec des combinaisons de modifications. Il est donc nécessaire d'ajouter une propriété supplémentaire lors de l'étiquettage des nœuds. La position j dans y du caractère inséré répond à cette demande, d'autant plus que disposer de la coordonnée du coefficient de la matrice M à partir duquel une modification a été identifiée permet de distinguer des modifications impliquant des caractères identiques dans des chemins d'édition différents, évitant alors à ces chemins d'être croisés à tort (ce cas a déjà été rencontré lors du développement de l'algorithme).

E.3 Combinaisons d'éditions

À partir du graphe orienté, il est possible grâce à un parcours en largeur depuis le nœud vide O d'établir toutes les combinaisons de k modifications distinctes parmi les ensembles à d éléments qui correspondent avec les chemins d'édition, après quoi toutes les chaînes intermédiaires sont déterminables.
L'algorithme permet d'établir toutes les combinaisons sans doublon. Pour chaque nœud N, d'une profondeur k [3] et dont on note u la modification contenue dans l'étiquette, les seules combinaisons construites sont en effet les unions du singleton $\{u\}$ avec les combinaisons construites auparavant par l'ensemble des sommets ancestraux à N (i.e. ceux d'une profondeur entre 0 et $k-1$ inclus, qui auront été obligatoirement déjà traités puisque le parcours du graphe s'effectue en largeur). Puisque u n'a pas été vu par ces sommets-ci, les nouvelles combinaisons sont donc bien distinctes de celles construites précédemment, de mêmes que celles construites chez les autres sommets avec la même profondeur k.
Néanmoins, plusieurs combinaisons distinctes de modifications peuvent déboucher sur le même intermédiaire généré. Des améliorations dans l'étape de l'établissement des combinaisons est donc à faire afin de s'assurer de ne sortir que des intermédiaires uniques.

Dans le meilleure des cas, le graphe d'édition forme un simple chemin. Le nombre de combinaisons construites sera alors de

$$\sum_{k=0}^{d} \binom{d}{k} = 2^d \tag{E.1}$$

En revanche, en pratique, il est fréquent que des chemins se séparent en deux, ce qui implique que le nombre de candidats peut être plus important. Cette séparation est cependant nécessairement suivie d'un renouement plus en profondeur dans le graphe, puisque les chemins d'édition mènent tous à l'obtention de y (voir le figuré E.3 pour un exemple). De plus, l'observation de graphes d'édition

3. i.e. qu'il est possible de le rejoindre depuis O en traversant k arêtes orientées

pour d'autres mots semble confirmer qu'un nœud ne puisse pas réunir plus de deux chemins.
Étant donné ces conditions, le graphe imaginable avec un nombre maximal de sommets pour une distance d'édition d fixée est celui dont le nombre de sommets est multiplié par deux à chaque descente en profondeur sur la première moitié du graphe pour ensuite être divisé par deux sur l'autre moitié, où les chemins fractionnés se réunissent à nouveau. Dans cet exemple de graphe, il va être montré que le nombre de combinaisons calculées est dominé par une valeur allant au-delà de $O(2^d)$. Déterminer si cette valeur constitue une majoration de la quantité d'intermédiaires générés est un problème mathématique dont le traitement est ici laissé pour des recherches ultérieures.

Soit $(c_k)_{0\leq k\leq d}$ la suite des nombres de combinaisons générées par l'algorithme lorsqu'il travaille sur un nœud de profondeur k et soit $(n_k)_{0\leq k\leq d}$ celle des nombres de sommets que le graphe possède à cette même profondeur. On a donc $c_0 = n_0 = 1$. À une profondeur $k \in [\![1, \lceil \frac{d}{2} \rceil]\!]$, un sommet est situé sur un chemin simple et le nombre de combinaisons qui y sont générées est alors de $c_k = 2^{k-1}$. Puisque le nombre de sommets est multiplié par deux d'une profondeur k à $k+1$, pour $k \leq \lceil \frac{d}{2} \rceil - 1$, on a donc $n_k = 2^k, \forall k \leq \lceil \frac{d}{2} \rceil$.
Pour $k > \lceil \frac{d}{2} \rceil$, si on note $m := k - \lceil \frac{d}{2} \rceil$, on démontre sans mal que $n_{(\lceil \frac{d}{2} \rceil + m)} = 2^{\lceil \frac{d}{2} \rceil - m}$. Chaque nœud renoue deux sommets de sorte à avoir le plus de sommets ancestraux possibles, dont le nombre à la profondeur j est alors de $\min\{2^m, n_j = 2^j\}$, dans le cas où $j \leq \lceil \frac{d}{2} \rceil$ sinon de 2^{k-j}, pour $\lceil \frac{d}{2} \rceil < j < k$.
Le nombre de combinaisons dans un nœud à la profondeur k vaut donc, $\forall 1 \leq m \leq \lceil \frac{d}{2} \rceil$:

$$\begin{aligned} c_k = c_{(\lceil \frac{d}{2} \rceil + m)} &= \sum_{j=0}^{m-1} 2^j c_j + \sum_{j=m}^{\lceil \frac{d}{2} \rceil} 2^m c_j + \sum_{j=\lceil \frac{d}{2} \rceil + 1}^{k-1} 2^{k-j} c_j \\ &= 1 + \sum_{j=1}^{m-1} 2^j 2^{j-1} + 2^m \sum_{j=m}^{\lceil \frac{d}{2} \rceil} 2^{j-1} + \sum_{p=1}^{m-1} 2^{m-p} c_{(\lceil \frac{d}{2} \rceil + p)} \\ &= 1 + \frac{1}{2} \sum_{j=1}^{m-1} 4^j + 2^m \left(2^{\lceil \frac{d}{2} \rceil} - 2^{m-1} + \sum_{p=1}^{m-1} 2^{-p} c_{(\lceil \frac{d}{2} \rceil + p)} \right) \end{aligned} \tag{E.2}$$

Pour $m = 1$, on a alors $c_{(\lceil \frac{d}{2} \rceil + 1)} = 1 + 2\left(2^{\lceil \frac{d}{2} \rceil} - 1\right) = 2^{\lceil \frac{d}{2} \rceil + 1} - 1$ et on a donc bien $c_{(\lceil \frac{d}{2} \rceil + 1)} = 2(2c_{(\lceil \frac{d}{2} \rceil)}) - 1$, ce qui confirme que les combinaisons calculées dans les premiers sommets de renouement sont bien générés à partir de l'union de deux ensembles de combinaisons calculées sur des chemins d'édition simples de longueur allant jusqu'à $\lceil \frac{d}{2} \rceil$, dont le cardinal de chacun vaut $2c_{(\lceil \frac{d}{2} \rceil)} = 2^{\lceil \frac{d}{2} \rceil}$. Ces deux ensembles contiennent seulement la combinaison vide en commun. Le cardinal de l'union vaut donc le double des cardinaux auquel on ôte 1.
Pour $m > 1$, la réécriture suivante est valide :

$$\begin{aligned} c_k = c_{(\lceil \frac{d}{2} \rceil + m)} &= 1 + \frac{1}{2}\left(\frac{4^m - 1}{3} - 1\right) - \frac{1}{2}4^m + 2^m \left(2^{\lceil \frac{d}{2} \rceil} + \sum_{p=1}^{m-1} 2^{-p} c_{(\lceil \frac{d}{2} \rceil + p)} \right) \\ &= \frac{1}{3}(1 - 4^m) + 2^m \left(2^{\lceil \frac{d}{2} \rceil} + \sum_{p=1}^{m-1} 2^{-p} c_{(\lceil \frac{d}{2} \rceil + p)} \right) \end{aligned} \tag{E.3}$$

En notant $(v_m)_{1\leq m\leq\lceil\frac{d}{2}\rceil}$ la suite $(c_{(\lceil\frac{d}{2}\rceil+m)})_{1\leq m\leq\lceil\frac{d}{2}\rceil}$, il est ainsi possible de déterminer une relation de récurrence entre les termes, pour $m \geq 2$:

$$\begin{aligned} v_{m+1} =& \frac{1}{3}(1-4^{m+1}) + 2^{m+1}\left(2^{\lceil\frac{d}{2}\rceil} + \sum_{p=1}^{m} 2^{-p}v_p\right) = \frac{1-4\cdot 4^m}{3} + 2\cdot 2^m\left(2^{\lceil\frac{d}{2}\rceil} + \sum_{p=1}^{m-1} 2^{-p}v_p + 2^{-m}v_m\right) \\ =& \frac{1-4\cdot 4^m}{3} + 2\cdot 2^m\left(2^{\lceil\frac{d}{2}\rceil} + \sum_{p=1}^{m-1} 2^{-p}v_p\right) + 2v_m = \frac{1-4\cdot 4^m}{3} + 2\left(v_m - \frac{1-4^m}{3}\right) + 2v_m \\ =& 4v_m - \frac{2}{3}4^m - \frac{1}{3} \end{aligned} \tag{E.4}$$

On déduit donc une expression explicite pour $(v_m)_{2\leq m\leq\lceil\frac{d}{2}\rceil}$ [4] :

$$v_m = 4^{m-2}v_2 + \sum_{j=2}^{m-1}\left(\frac{-2}{3}4^j - \frac{1}{3}\right)4^{m-1-j} = 4^{m-2}v_2 - \frac{2}{3}(m-2)4^{m-1} - \frac{1}{3}4^{m-3}\sum_{j=0}^{m-3}4^{-j} \tag{E.5}$$

D'après l'équation E.3 et l'expression de $c_{(\lceil\frac{d}{2}\rceil+1)}$, on a $v_2 = 2^{\left(\lceil\frac{d}{2}\rceil+3\right)} - 7$. Pour $m \geq 3$, on obtient donc :

$$\begin{aligned} v_m =& 4^{m-2}\left(2^{\left(\lceil\frac{d}{2}\rceil+3\right)} - 7\right) - \frac{2}{3}(m-2)4^{m-1} - \frac{1}{3}4^{m-3}\frac{4}{3}(1-4^{2-m}) \\ \text{D'où, } v_m = c_{\left(\lceil\frac{d}{2}\rceil+m\right)} =& 2^{\left(2m+\lceil\frac{d}{2}\rceil-1\right)} - \frac{2}{3}m4^{m-1} - \frac{4^m-1}{9} \end{aligned} \tag{E.6}$$

La quantité totale de combinaisons construites avec le graphe est donnée par la somme $N = \sum_{k=0}^{d} n_k c_k$. Si d est paire, on remarque alors à partir de la première expression dd l'équation E.2 que $N = 2 * c_d$. Si d est impaire, on remarque de la même façon que $N = c_{d+1}$ Dans tous les cas, on a donc

$$N = O(c_{2\lceil\frac{d}{2}\rceil}) = O(v_{\lceil\frac{d}{2}\rceil}), \text{avec } v_{\lceil\frac{d}{2}\rceil} = \frac{1}{2}2^{3\lceil\frac{d}{2}\rceil} - \frac{1}{12}2\lceil\frac{d}{2}\rceil 2^{2\lceil\frac{d}{2}\rceil} - \frac{1}{9}2^{2\lceil\frac{d}{2}\rceil} + \frac{1}{9} = O(2^{\frac{3}{2}d}) \tag{E.7}$$

4. On s'intéresse évidemment à étudier des valeurs infinies de d, ce qui ne pose donc pas de problème pour définir $(v_m)_{2\leq m\leq\lceil\frac{d}{2}\rceil}$

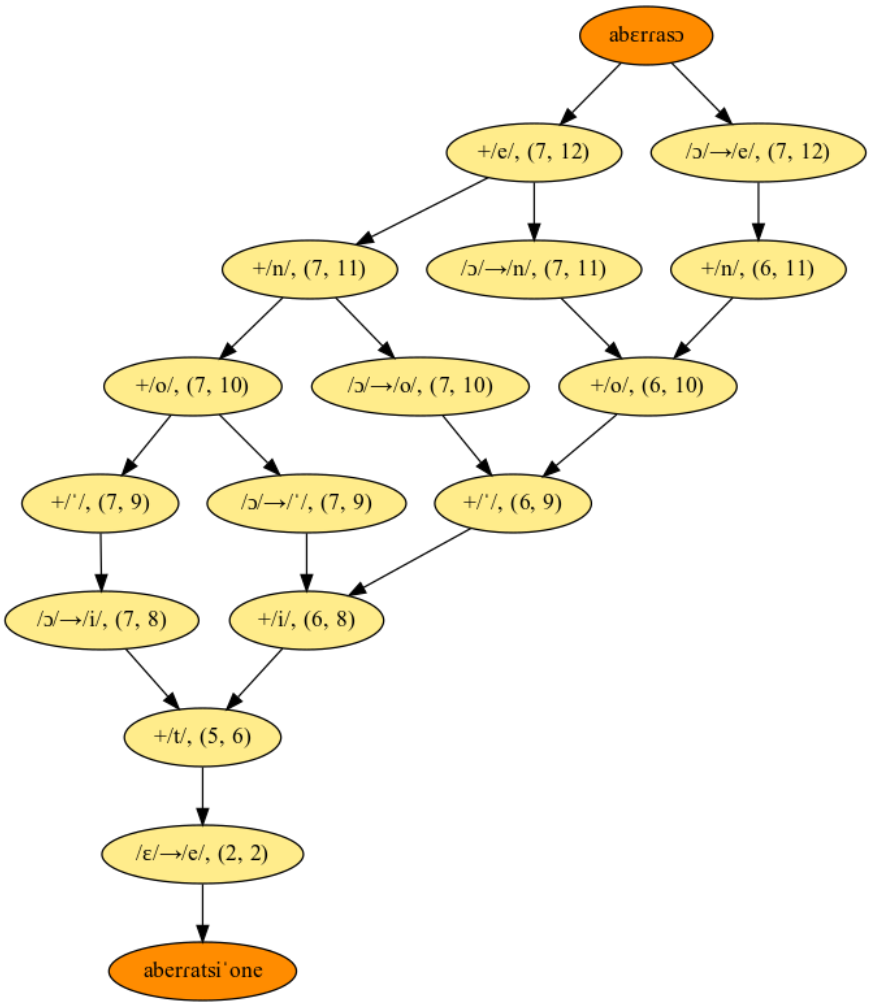


FIGURE E.3 – Représentation d'un graphe d'édition plus complexe.