Rapport projet PSTALN :
Pre-train VS Home-made Transformers

Réalisé par Victor Tancrez, Thomas Oxisoglou et Paul Peyssard
Encadré par Carlos Ramisch et Alexis Nasr

15 janvier 2024

Master 2 Intelligence Artificielle & Apprentissage Automatique

Aix-Marseille Université
Centrale Marseille

Contents

1 Introduction

Ce rapport détaille le projet mené par Victor Tancrez, Thomas Oxisoglou et Paul Peyssard, étudiants en Master 2 Intelligence Artificielle et Apprentissage Automatique à Aix-Marseille Université. Ce travail a été réalisé sous la supervision de Carlos Ramisch et Alexis Nasr, dans le cadre de l'unité d'enseignement PSTALN.

L'objectif principal de ce projet est de construire et d'évaluer divers modèles destinés au POS Tagging. Dans notre recherche, nous avons entrepris une étude comparative, opposant les transformers pré-entraînés disponibles sur des plateformes comme Huggingface à des transformers conçu et entraîné par nos soins.

L'intérêt de ce projet est de déterminer si l'utilisation de modèles spécifiques à une tâche, entraînés sur un volume de données moindre, peut être avantageuse, ou si, au contraire, les modèles pré-entraînés surclassent systématiquement les alternatives moins fournies en données.

2 Déroulement du projet

2.1 Organisation du travail en équipe

Au cours de ce projet, nous avons adopté une méthode de travail structurée et collaborative, consacrant environ une journée et demi par semaine, en plus de nos cours, pour avancer sur nos objectifs. Cette approche nous a permis de maintenir une progression constante tout en équilibrant nos autres projets.

La répartition des tâches a été un élément central de notre stratégie de travail. Chaque membre de l'équipe s'est vu attribuer des responsabilités spécifiques, adaptées à ses intérêts. Certains d'entre nous se sont concentrés sur le développement de modèles spécifiques, tandis que d'autres étaient chargés d'affiner les modèles pré-entrainés ou de conduire des recherches pour explorer de nouvelles approches potentielles. Cette distribution des tâches a non seulement optimisé notre efficacité mais a également permis à chaque membre de se spécialiser et de contribuer de manière significative au projet.

Pour faciliter la collaboration à distance, en particulier pendant les périodes de vacances, nous avons utilisé GitHub pour la gestion du code et Discord pour communiquer par message et effectuer des réunions régulières. Ces outils ont été indispensables pour discuter des progrès, résoudre les problèmes techniques et planifier les étapes suivantes.

2.2 Les spécifictés du projet

2.2.1 Tâches de prédiction de structures

À l'origine, notre projet envisageait de couvrir plusieurs tâches de prédiction de structures, notamment le Part-of-Speech (POS) tagging, la lemmatisation et l'identification des traits morphologiques. Cependant, après une discussion approfondie avec notre encadrant et une évaluation réaliste de nos ressources et du temps disponible, nous avons décidé de concentrer nos efforts exclusivement dans un premier temps sur le POS tagging. Nous avons réalisé que cette tâche, en soi, présentait une complexité importante et offrait un terrain nouveau pour explorer en profondeur les capacités et limites de nos modèles. Cette décision nous a permis de nous investir pleinement dans l'optimisation et l'évaluation de nos modèles pour le POS tagging, tout en assurant une gestion efficace du temps et des ressources à notre disposition.

Une fois notre tâche de POS Tagging bien avancée, nous nous sommes brièvement intéressés à une autre tâche, celle de la lemmatisation. Le défi de la lemmatisation consiste à prédire la forme non fléchie du mot dans sa forme la plus réduite possible. Ce principe permet donc de normaliser et réduire la dimension de notre espace tout en améliorant la généralisation. Nous avons dû adapter nos modèles pour répondre aux besoins de cette tâche et avoir un début de multi-tasking.

2.2.2 Langue et données utilisées

Nous avons décidé de nous concentrer sur le français, en utilisant le dataset UD_FRENCH_Sequoia, pour plusieurs raisons stratégiques. Premièrement, le français est une langue riche et complexe en termes de morphologie et de syntaxe, ce qui représente un défi intéressant pour le POS tagging.

Le choix de UD_FRENCH_Sequoia a été motivé par sa qualité et sa diversité. Ce corpus, faisant partie du projet Universal Dependencies, est bien structuré, annoté de manière rigoureuse et représente une variété de genres textuels. Ces caractéristiques garantissent non seulement une base solide pour l'entraînement et le test de nos modèles, mais offrent également un terrain propice à l'analyse comparative de différents approches de modélisation.

Une des autres raisons a également été le fait que c'était le dataset fourni par nos professeurs ; de ce fait, nous pouvons comparer nos modèles avec ceux des différents groupes réalisant des travaux de recherche similaires.

2.2.3 Organisation des données en batch

Nous avons à faire à des phrases en entrées de longueurs variables. Pour traiter efficacement ces phrases dans des lots (batches). Nous avons utilisé le padding, il consiste à ajouter des tokens spéciaux, habituellement désignés par `<PAD>`, aux phrases plus courtes pour les amener à la même longueur que la phrase la plus longue dans le lot. Cela permet d'effectuer des opérations matricielles de manière efficace et est essentiel pour l'utilisation de lots (batch) lors de l'entraînement.

Afin que ces tokens de padding n'influencent pas l'apprentissage du modèle, lors du calcul de la fonction de perte, telle que la *Cross Entropy Loss*, les tokens de padding sont ignorés. Dans PyTorch, cela est réalisé en utilisant l'argument `ignore_index` de la fonction de perte. Les indices correspondant aux tokens de padding sont passés à cet argument, ce qui a pour effet de ne pas prendre en compte ces positions dans le calcul de la perte.

3 Les différents modèles

3.1 Une première baseline LSTM

Pour établir une première baseline, nous avons opté pour un modèle LSTM, qui diffère du GRU abordé en travaux pratiques. Ce choix a été motivé par le désir d'explorer une architecture différente et de comprendre ses performances dans notre contexte spécifique.

Notre modèle LSTM est basique, il utilise une couche d'embedding pour transformer les identifiants de mots en vecteurs denses, suivie d'une couche LSTM pour capturer les dépendances séquentielles dans les phrases. La sortie de la couche LSTM est ensuite passée à une couche linéaire pour prédire les étiquettes de POS pour chaque mot de la phrase.

Durant l'entraînement, nous avons utilisé une fonction de perte CrossEntropy, adaptée pour gérer les séquences de longueurs variables. Le modèle a été entraîné sur 50 époques, avec un batch size de 16, et nous avons utilisé l'algorithme d'optimisation SGD pour ajuster les poids du modèle.

Cette première baseline a fourni une fondation solide pour notre projet, en nous permettant de comparer les performances de ce modèle simple avec des architectures plus complexes et des modèles pré-entraînés par la suite.

Model	Learning Rate	Batch Size	Epochs	Embedding_dim	Hidden_dim	Optimizer
baseline_LSTM	0.01	16	50	64	128	SGD

Table 1: Paramètres de la baseline

3.2 Les modèles pré-entrainés

3.2.1 Architecture

Nous avons décidé d'utiliser 2 modèles pré-entrainés afin d'aller plus loins dans nos comparaisons et voir si un modèle entrainé sur plusieurs langue sera plus performant qu'un modèle seulement entraîné sur le français.

1) **Camembert-base** :

- *Architecture* : Basé sur le model RoBERTa, il est composé de 12 couches de transformers, `Camembert-base` possède 111 millions de paramètres. Chaque couche transformer intègre une couche de self-attention multi-têtes et une couche feed-forward, conformément à l'architecture originale de BERT.
- *Entraînement* : Ce modèle a été spécifiquement entraîné pour comprendre le français. Il a été formé sur un vaste corpus de textes français, le OSCAR dataset, couvrant une grande variété de genres et de styles, ce qui lui permet de capter finement les nuances linguistiques et les spécificités syntaxiques du français.

2) **Bert-base-multilingual-cased** :

- *Architecture* : `Bert-base-multilingual-cased` partage la même structure de base que `Camembert-base`, avec 12 couches de transformers et environ 179 millions de paramètres. Chaque couche comprend une self-attention multi-têtes et des réseaux de neurones feed-forward pour traiter l'information contextuelle.
- *Entraînement* : Conçu pour traiter 104 langues différentes, ce modèle a été entraîné sur un large corpus multilingue. Il est capable de comprendre le contexte et les subtilités de plusieurs langues, y compris le français, ce qui le rend utile pour des applications nécessitant une compatibilité linguistique étendue.

- *cased* : La variante "cased" signifie que le modèle prend en compte la casse des lettres (majuscules/minuscules), ce qui lui permet de mieux comprendre la structure des phrases et les noms propres, important pour de nombreuses langues, notamment le français.

3) **Paramètres Fine-tunés** :
Pour améliorer les modèles et trouver les paramètres optimaux, nous avons utilisé du Grid-Search avec les paramètres suivant :

- *Taux d'apprentissage* : Un taux d'apprentissage bien ajusté est crucial pour l'équilibre entre apprentissage efficace et évitement du surajustement. Un taux trop élevé peut conduire à un apprentissage instable, tandis qu'un taux trop bas peut entraîner une convergence lente et potentiellement un sous-apprentissage.

- *Tailles de batch* : La taille du batch influence directement la stabilité et la vitesse de l'apprentissage. Des batchs plus grands fournissent des estimations de gradient plus stables, mais peuvent être moins efficaces en termes de mémoire et de calcul. Des batchs plus petits, en revanche, peuvent conduire à une convergence plus rapide mais plus instable.

- *Nombre d'époques* : Le nombre d'époques détermine combien de fois le modèle verra l'ensemble du jeu de données. Un nombre insuffisant d'époques peut empêcher le modèle de converger, tandis qu'un nombre trop élevé peut conduire à un surajustement. Un ajustement adéquat est nécessaire pour assurer un apprentissage optimal. De plus, pour prévenir le surajustement, nous avons implémenté une technique d'early stopping, qui arrête l'entraînement si le modèle ne montre plus d'amélioration importante sur l'ensemble de validation sur un certain nombre d'époques consécutives. Pour effectuer cette early stopping, nous avons utilisé un dataset de validation. Nos différents dataset étaient découpés en 70% train, 15% validation et 15% test.

- *Optimiseurs* : Le choix de l'optimiseur affecte la manière dont le modèle ajuste ses poids en réponse à l'erreur observée. Différents optimiseurs, tels qu'Adam, SGD ou AdamW que nous avons testés, ont des caractéristiques propres qui influencent la vitesse et la qualité de la convergence, affectant ainsi les performances globales du modèle.

4) **Paramètres Finaux obtenus** :
Les paramètres finaux sélectionnés pour les modèles `camembert-base` et `bert-base-multilingual-cased` après le processus de fine-tuning sont les suivants :

Model	**Learning Rate**	**Batch Size**	**Epochs**	**Optimizer**
Camembert-base	0.0001	32	6	Adam
Bert-base-multilingual-cased	0.0001	64	7	Adam

Table 2: Paramètres finaux des modèles pré-entrainés

3.2.2 Encodage des entrées

L'encodage des entrées pour nos modèles pré-entraînés, `camembert-base` et `bert-base-multilingual-cased`, s'appuie sur les tokenizers de BERT.

Pour `camembert-base`, spécifiquement conçu pour le français, l'encodage utilise une tokenisation SentencePiece, qui décompose les mots en sous-unités plus petites. Cette méthode est particulièrement efficace pour gérer les mots complexes ou inconnus, les segmentant en fragments familiers au modèle. Les phrases sont d'abord converties en minuscules, puis le tokenizer ajoute des tokens spéciaux au début ('[CLS]') et à la fin ('[SEP]') de chaque séquence.

"Le `bert-base-multilingual-cased` suit un processus similaire, mais avec une portée multilingue. Ce modèle est entraîné pour reconnaître et traiter 104 langues, y compris le français. Ce modèle utilise la tokenisation WordPiece qui traite chaque mot séparément en prenant en compte les espaces. La capacité de ce modèle à gérer plusieurs langues signifie qu'il peut ne pas être aussi spécialisé dans une langue spécifique comme le français comparé à `camembert-base`. Cependant, il est entraîné sur plus de données et avec plus de paramètres, donc ils ne sont pas forcément comparables (Le but n'étant pas de comparer les transformers pré-entraînés entre eux).

Dans les deux cas, les mots ou tokens sont convertis en identifiants uniques (input IDs), avec un traitement spécial des masques d'attention et du padding pour normaliser la longueur des séquences. Les identifiants sont ensuite transformés en vecteurs d'entrée comprenant les tokens, les marqueurs de segmentation et les embeddings de position. BERT traite ces vecteurs pour comprendre le contexte bidirectionnel des mots dans une phrase.

L'efficacité de ces méthodes réside dans leur capacité à gérer les mots hors vocabulaire (OOV) grâce à la tokenisation basée sur des sous-mots. Même les mots non rencontrés dans le corpus d'entraînement peuvent

être représentés par une combinaison de ces sous-unités, permettant aux modèles pré-entraînés de traiter un large éventail de mots avec précision.

3.3 Le model fait maison

Dans le cadre de nos expériences, nous avons entraîné ici également deux modèles distincts basés sur l'architecture Transformer, chacun avec une approche différente pour l'encodage des entrées.

3.3.1 Encodage des entrées

1) **Méthode d'Apprentissage des Embeddings orienté sur la tâche :**
Dans ce modèle, chaque mot du corpus d'entraînement est associé à un indice unique.
Le mot inconnu (UNK) est associé à la valeur 1. Ceci est important pour gérer les mots non rencontrés pendant l'entraînement (OOV).
Fonctionnement :

- Initialisation du Vocabulaire :
  - Les mots sont comptés et un indice unique est attribué à chaque mot.
  - Le token "UNK" est explicitement associé à l'indice 1.
- Préparation des Données :
  - Lors de l'entrainement un certain nombre de mots sont remplacés par le token "UNK".
  - Ceci nous permet d'apprendre son embedding.
  - Lors de l'inférence, si un mot n'est pas dans le vocabulaire de test alors, il est remplacé par "UNK".
  - Les phrases sont converties en séquences d'indices.
- Embedding des Mots :
  - Une couche d'embedding transforme les indices en vecteurs denses.
  - L'embedding pour "UNK" est appris pendant l'entraînement.
  - Cet embedding est utilisé pour tous les mots inconnus pendant la prédiction.

2) **Méthode d'apprentissage d'embeddings à base de convolution 1D :**

- La convolution à partir d'une séquence d'encodage de caractère nous permet d'éviter les mots inconnus.
- Il est extrêmement improbable d'avoir des lettres inconnues dans de grands corpus de texte. Et si nous observons des lettres inconnus alors elles sont très certainement extrêmement peu utilisées.
- Traitement des Mots :
  - Chaque mot dans une phrase est converti en une séquence d'indices de caractères, en utilisant un mapping pré-établi.
  - Lors de l'inférence si la lettre n'est pas connu alors on utilise l'indice de "UNK" à la place. Nous n'avons pas jugé nécessaire de faire passer des lettres "UNK" au modèle durant l'entraînement car il y a un nombre extremement restreint de lettre inconnu.
  - Pour standardiser la taille des mots, ceux-ci sont soit complétés avec un caractère spécial de padding ("PAD"), soit tronqués, pour atteindre une longueur fixe (max_word_len) de 26 caractères. Cette standardisation est cruciale pour l'application uniforme de la convolution sur les mots.
  - Les POS tags de chaque mot sont également converties en indices.
- Encodage des Caractères des Mots :
  - Un embedding est utilisé pour transformer chaque indice de caractère en un vecteur dense, capturant des informations sémantiques et syntaxiques subtiles au niveau des caractères. (Char Embedding)
  - Ces vecteurs de caractères sont ensuite traités par une couche de convolution 1D, qui analyse les motifs locaux au sein des séquences de caractères. (Convolution)

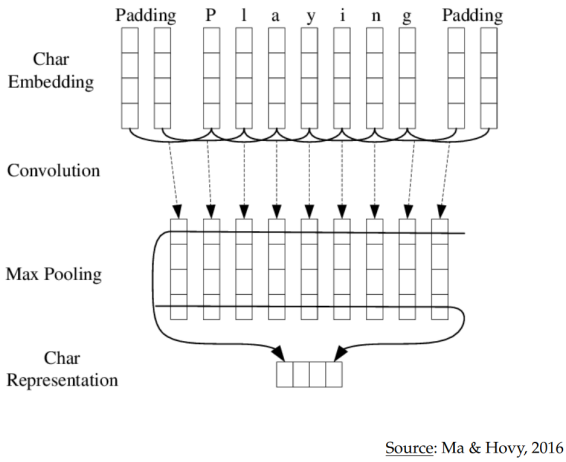

Figure 1: Architecture de l'embedding fait à partir des caractères.

  - La couche de convolution aide à extraire des caractéristiques locales pertinentes des séquences de caractères, telles que les préfixes, les suffixes et les racines des mots, qui sont importants pour la compréhension du langage.
  - Le max-pooling est utilisé pour réduire la dimensionnalité des sorties de la convolution. (Max pooling)
  - On obtient après ces étapes notre représentation du mot basé sur l'embedding des caractères

- Formation de la Représentation de la phrase :
  - Après le traitement par convolution et pooling, la sortie est restructurée en une matrice où chaque ligne correspond à la représentation condensée d'un mot dans la phrase, obtenue grâce au pooling qui sélectionne les caractéristiques les plus importante de chaque fenêtre de convolution.
  - Les dimensions de cette matrice sont définies par la longueur de la phrase et le nombre de filtres utilisés dans la couche de convolution.
  - Cette matrice, représentant une version enrichie et condensée des mots de la phrase, sert d'entrée pour les couches subséquentes du Transformer.

3) **Positional Encoding :**

Le Positional Encoding ajoute des informations de position aux embeddings des mots. Cela permet au modèle de prendre en compte l'ordre des mots dans une phrase, ainsi, il permet de conserver une notion de séquence dans le traitement du langage.

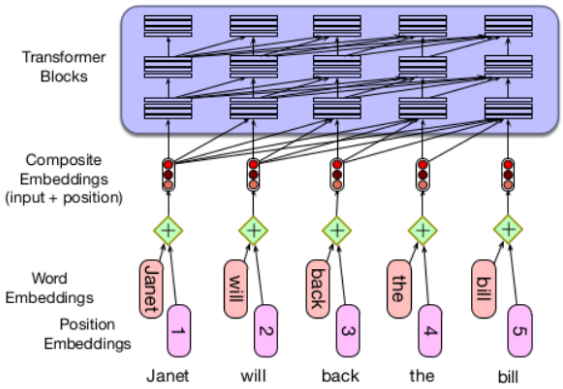

Figure 2: Principe du positional encoding. (Cours PSTALN, M2 IAAA, Carlos Ramisch & Alexis Nasr)

- Initialisation :
  - d_model : La dimension des embeddings de mots.

- dropout : Le taux de dropout pour la régularisation.
- max_len : La longueur maximale des phrases prises en charge.

- Calcul des positions :
  - Les positions des mots sont représentées par des indices allant de 0 à max_len - 1.
  - Un terme de division (div_term) est calculé pour alterner entre le sinus et le cosinus.
- Création de la matrice d'encodage positionnel (pe) :
  - Une matrice de zéros de dimension [max_len, 1, d_model] est initialisée.
  - Les valeurs sinus et cosinus sont appliquées alternativement sur les positions.
- Fonction forward :
  - L'embedding d'entrée (x) est additionné avec la matrice d'encodage positionnel.
  - Le dropout est appliqué après l'addition pour la régularisation.

Cette approche permet au modèle de distinguer les mots en fonction de leur position relative dans la phrase, améliorant ainsi la capacité du modèle à comprendre le contexte et la structure séquentielle du texte.

3.3.2 Architecture

1) **Architecture utilisé :**

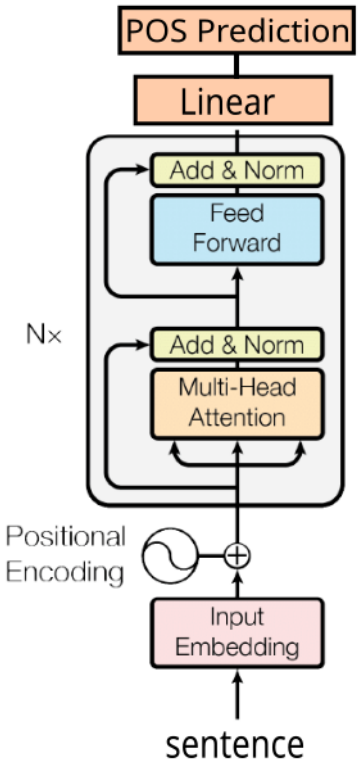

Figure 3: Principe général de notre architecture (Source : Documentation PyTorch, modifié pour notre implémentation).

- **Modèle avec Embedding de mots :** Le premier modèle utilise un embedding classique des mots. Ce modèle est conçu pour convertir chaque mot du corpus en un vecteur dense via une couche d'embedding. L'architecture est détaillée comme suit :
  - Une couche d'embedding transforme les indices de mots en vecteurs.

- Le Positional Encoding est appliqué après l'embedding pour ajouter des informations sur la position relative des mots dans une phrase.
- Un encodeur Transformer traite les embeddings positionnels.
- Une couche linéaire finale convertit les sorties de l'encodeur en scores pour les étiquettes POS.

- **Paramètres du modèle :**
  - Taille du vocabulaire (`vocab_size`) : Défini par la taille du corpus.
  - Dimension d'embedding (`embedding_dim`) : Exemple, 512.
  - Nombre de têtes d'attention (`nhead`) : Exemple, 4.
  - Dimension de la couche interne du Transformer (`nhid`) : Exemple, 1024.
  - Nombre de couches de l'encodeur Transformer (`nlayers`) : Exemple, 3.
  - Taille de l'ensemble des étiquettes (`tagset_size`) : Dépendant du nombre d'étiquettes POS.

- **Modèle avec Convolution sur les Caractères :** Le second modèle utilise une approche basée sur la convolution des caractères. Cette méthode se concentre sur l'encodage au niveau des caractères pour chaque mot, offrant une robustesse accrue face aux mots inconnus. Les éléments clés de ce modèle sont :
  - Une couche d'embedding de caractères suivie d'une convolution 1D pour encoder chaque mot.
  - Le Positional Encoding est également appliqué ici pour maintenir la connaissance de la position des mots.
  - L'encodeur Transformer traite ces embeddings enrichis par la convolution.
  - Une couche linéaire transforme la sortie de l'encodeur en scores des étiquettes POS.

- **Paramètres du modèle :**
  - Nombre de caractères (`num_chars`) : Défini par le corpus.
  - Dimension d'embedding de caractères (`char_embedding_dim`) : Exemple, 256.
  - Nombre de filtres dans la couche de convolution (`num_filters`) : Exemple, 100.
  - Taille du kernel dans la couche de convolution (`kernel_size`) : Exemple, 3.
  - Nombre de têtes d'attention (`nhead`) : Exemple, 4.
  - Dimension de la couche interne du Transformer (`nhid`) : Exemple, 1024.
  - Nombre de couches de l'encodeur Transformer (`nlayers`) : Exemple, 3.
  - Taille de l'ensemble des étiquettes (`tagset_size`) : Dépendant du nombre d'étiquettes POS.

Chaque modèle vise à exploiter les avantages de l'architecture Transformer tout en adaptant l'approche d'encodage (mots vs caractères) pour améliorer la performance sur la tâche de Part-of-Speech tagging.

3.3.3 Etude de l'architecture :

Dans le cadre de notre étude, nous avons examiné l'impact de divers hyperparamètres sur la précision du modèle sur un ensemble de données de test. Afin d'optimiser l'apprentissage à chaque époque, nous avons évalué les performances du modèle sur un ensemble de données de validation.

Nous avons adopté une stratégie d'early stopping : si la meilleure perte de validation n'est pas surpassée pendant 5 époques consécutives, l'apprentissage est interrompu. Cette méthode vise à prévenir le surapprentissage et à améliorer l'efficacité de l'entraînement.

Compte tenu de la nature combinatoire du problème, notre approche a consisté à étudier initialement différentes architectures une par une pour identifier les hyperparamètres les plus influents. Après avoir fixé ces hyperparamètres, nous avons varié chacun d'entre eux individuellement, en ajustant les hyperparamètres fixés au besoin.

- char_embedding_dim = 512
- num_filters = 128
- nhead = 1
- kernel_size = 1

- nhid = 1024
- nlayers = 1
- patience = 5

Avant cette étude nous avons constaté que les hyperparamètres ayant le plus grand impact sur les performances sont la taille du noyau de convolution, le nombre de têtes d'attention, le nombre de filtres.

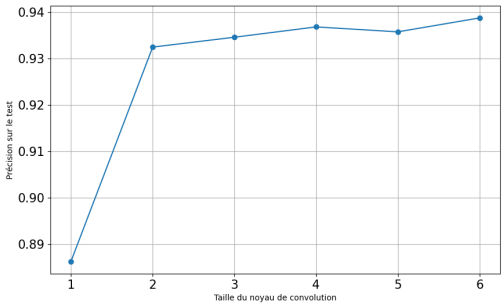

Figure 4: Précision sur le test en fonction de la taille du noyau de convolution

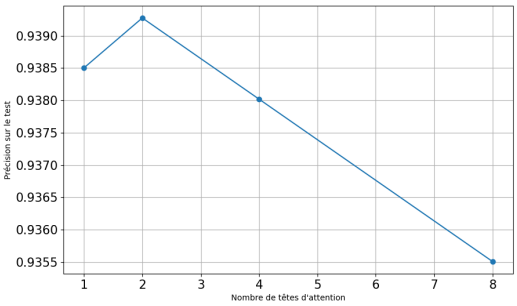

Figure 5: Précision sur le test en fonction du nombre de têtes d'attention avec une taille de noyau de convolution fixé à 6

Basés sur les résultats de ces tests, nous avons décidé de fixer la taille du noyau de convolution à 6 et le nombre de têtes d'attention à 2.

L'amélioration des performances observée lors de l'augmentation de la taille du noyau de convolution peut être attribuée à une meilleure capture du contexte. En effet, un noyau de taille 1 ne peut traiter qu'un seul caractère à la fois, se limitant ainsi à l'analyse de caractéristiques extrêmement locales. À l'opposé, un noyau de taille 6 est capable de traiter simultanément six caractères. Cette capacité accrue permet au modèle de détecter des motifs et des dépendances plus étendus au sein des mots, tels que les syllabes, qui jouent un rôle essentiel dans la compréhension du sens et de la structure des mots.

Nous avons également testé, en partant de cette taille de noyau de convolution, si la taille de la couche cachée (nhid) et la taille de l'embedding des caractères étaient optimales. Il s'avère que la taille de l'embedding des caractères est adéquate, mais pour la taille de la couche cachée, une valeur inférieure à 1024 pourrait être suffisante. Cependant, les résultats étant très proches, nous avons décidé de conserver cette valeur de 1024 pour nos tests ultérieurs.

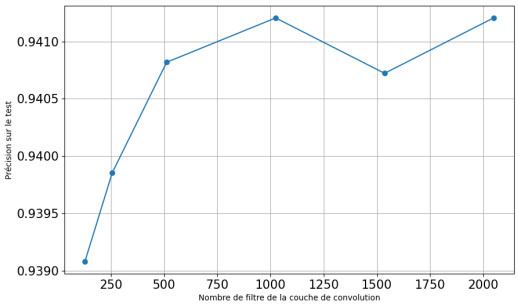

Figure 6: Précision sur le test en fonction du nombre de filtre de convolution

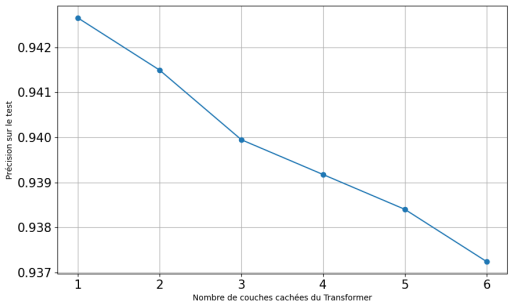

Figure 7: Précision sur le test en fonction du nombre de couche de Transformer avec un nombre de filtre à 1024

Nos graphiques indiquent que l'utilisation de 1024 filtres dans la couche de convolution offre d'excellentes performances. Un nombre aussi élevé de filtres enrichit considérablement le modèle, lui permettant de reconnaître un large éventail de caractéristiques linguistiques, ce qui se traduit par une analyse textuelle plus profonde et précise. Bien qu'il ait été observé que l'utilisation de 2048 filtres pourrait également atteindre des performances optimales, nous avons choisi de maintenir le nombre de filtres à 1024. Cette décision a été motivée par la volonté de réduire le temps d'apprentissage.

Cependant, il est à noter que l'accroissement du nombre de couches dans l'encodeur Transformer a un effet légèrement négatif sur les performances. Cette baisse peut s'expliquer par l'insuffisance des données disponibles

pour entraîner de manière efficace un modèle de cette complexité. Il devient alors essentiel de disposer d'un volume de données suffisant pour tirer pleinement parti des capacités d'un modèle ainsi élaboré.

En conclusion, il apparaît que les facteurs déterminants pour l'efficacité de l'apprentissage dans cette configuration sont la taille du noyau de convolution et le nombre de filtres. Ces deux éléments ont une influence notable sur la capacité du modèle à interpréter et à comprendre les données, soulignant l'importance d'un réglage approprié de ces hyperparamètres pour optimiser les performances.

Nous utiliserons les meilleurs hyperparamètres obtenus avec ces résultats pour le modèle transformer avec embedding sur les mots. Ainsi voici les hyperparamètres que nous allons utiliser pour les résultats finaux :

- nhead = 2
- nhid = 1024
- nlayers = 1
- batch_size = 16
- embedding_dim = 512 (spécifique embedding sur les mots)
- char_embedding_dim = 512 (spécifique à la convolution)
- num_filters = 1024 (spécifique à la convolution)
- kernel_size = 6 (spécifique à la convolution)

4 Résultats

Pour l'entrainement du home-made transformer avec encodage convolutif et avec embedding sur les mots, le modèle a été entrainé avec une patience de 20 ce qui signifie qu'on arrête l'entrainement après qu'il n'y ait plus eu d'amélioration du modèle pendant 20 époques. Cet hyper paramètre est plus élevé comparé aux expériences sur les hyperparamètres et cela explique les meilleurs performances.

Modèle	**Train Accuracy**	**Test Accuracy**	**Test f1**
base_line_LSTM	0.8869%	0.8664%	0.7892%
bert-base-multilingual-cased	0.9806%	0.9507%	0.8628%
camembert-base	0.9758%	0.9324%	0.8518%
home-made transformer encodage convolutif	0.9708%	0.9434%	0.8531%
home-made transformer embedding sur les mots	0.9678%	0.9110%	0.8217%

Table 3: Comparaison des performances des différents modèles

La comparaison des performances des différents modèles présentée dans la Table 3 révèle des insights importants sur l'efficacité de différentes architectures dans la tâche de Part-of-Speech (POS) tagging. Le modèle LSTM de base, avec une précision de test de 86.64%, sert de point de référence initial. Bien que sa performance soit la plus basse parmi les modèles évalués, elle reste respectable, soulignant l'efficacité des architectures LSTM dans les tâches de traitement du langage naturel.

On note qu'une différence notable est observée entre `bert-base-multilingual-cased` et `camembert-base`. Le modèle `bert-base-multilingual-cased` présente une légère supériorité avec 95.07% de précision, comparativement à 93.24% pour `camembert-base`. Cette différence peut s'expliquer par le nombre de paramètres et les corpus de pré-entraînement différent des deux modèles.

`Bert-base-multilingual-cased` bénéficie d'un entraînement sur un corpus diversifié multilingue, lui permettant une compréhension linguistique plus généraliste et potentiellement une meilleure capacité de généralisation, y compris pour le français. À l'inverse, `camembert-base`, conçu spécifiquement pour le français, se focalise sur les particularités de cette langue grâce à son entraînement sur un corpus unilingue. Bien que cela lui confère une spécialisation avancée, la diversité limitée de son corpus peut restreindre sa capacité à généraliser, en comparaison avec `bert-base-multilingual-cased`. Ces observations suggèrent l'importance de la diversité et de la quantité des données d'entraînement dans la performance globale des modèles de langage naturel.

Ces résultats soulignent également l'avantage des modèles pré-entraînés grace à leur vaste corpus d'entraînement qui les rend aptes à mieux comprendre les subtilités du langage naturel que les modèles entrainés par nos soins.

Parmi les modèles conçus spécifiquement pour ce projet, le transformer avec encodage convolutif se distingue avec une précision de test de 94.34%, rivalisant étroitement avec les modèles pré-entraînés et dépassant même camembert-base pour ce dataset. Cette performance remarquable illustre l'efficacité de l'approche personnalisée, en particulier l'utilisation de l'encodage convolutif pour capturer des caractéristiques linguistiques fines.

En revanche, le transformer avec embedding sur les mots affiche une précision de test de 90.71%. Bien que cette performance soit inférieure à celle des autres modèles, elle démontre néanmoins l'efficacité d'une approche d'embedding spécifique à la tâche. Ces résultats suggèrent qu'une exploration plus approfondie des méthodes d'encodage et de paramétrage pourrait être une voie d'améliorations.

Comme les classes étaient légèrement déséquilibrées, nous avons également analyser les f1_scores des différents modèles et pour pouvoir garder un oeil sur différentes métriques afin d'avoir un "avis" en plus.

Parmi les modèles pré-entraînés, `bert-base-multilingual-cased` et `camembert-base` enregistrent des f1_scores de 86.28% et 85.18%, respectivement, reflétant leur efficacité dans la gestion des classes moins représentées. La baseline LSTM, malgré une précision de test adéquate, affiche un f1_score de 78.92%, indiquant des difficultés à équilibrer précision et rappel.

Les modèles maison, notamment le transformer à encodage convolutif, atteignent un f1_score de 85.31%, surpassant légèrement `camembert-base` et démontrant une bonne gestion des déséquilibres de classe. Le transformer avec embedding sur les mots obtient un f1_score de 80.17%, inférieur aux modèles pré-entraînés mais remarquable.

Ces résultats mettent notemment en lumière l'importance d'adopter diverses métriques d'évaluation, en particulier dans des contextes avec des déséquilibres de classe, où le f1_score se révèle être un indicateur essentiel pour juger de la qualité globale d'un modèle de POS tagging.

En somme, ces différents résultats mettent en évidence la robustesse des modèles pré-entraînés dans les tâches de POS tagging, tout en ouvrant la voie à l'exploration de modèles spécifiques plus ciblés, capables de performances comparables, voire supérieures, dans certaines configurations.

5 Bonus : Une dimension multi-tâches

Modèle	**Train Accuracy**	**Test Accuracy**
base_line_LSTM	0.8379%	0.8580%
home-made transformer encodage convolutif	0.9915%	0.9722%

Table 4: Comparaison des performances des différents modèles pour prédiction de lemme

Comme l'un de nos objectifs initiaux, bien qu'ambitieux, était de réaliser une prédiction multi-tâches en plus de la comparaison des différents modèles, nous avons seulement eu le temps de réaliser certaines expériences sur la baseline et le transformer convolutif, mais nous avons préféré parfaire la dimension de comparaison au niveau du POS plutôt que de nous concentrer sur un objectif optionnel.

Dans la Table 4 ci-dessus, on peut observer les résultats de nos modèles sur une tâche de lemmatisation. Nous sommes donc partis des modèles utilisés pour la tâche de POS-Tagging et avons ajusté l'entrée des modèles pour cette nouvelle tâche qu'est la lemmatisation. En ce qui concerne les hyperparamètres utilisés, ce sont les mêmes que pour la tâche de POS-Tagging.

En ce qui concerne les résultats, on peut observer que le modèle LSTM, qui nous sert de base pour pouvoir effectuer toute autre comparaison, a une précision en test de 85.80%. Cette précision modérée peut s'expliquer par le fait que notre modèle LSTM a des difficultés à garder des dépendances sur de longs termes et donc du mal à apprendre une association entre les mots et les lemmes. Pour ce qui est du résultat de notre transformer avec encodage convolutif, on atteint une précision en test de 97.22%, on dépasse donc largement le résultat du modèle LSTM. Cette amélioration importante se justifie par le fait qu'un modèle comme le nôtre capte mieux les dépendances sur le long terme et apprend une meilleure association entre les mots et les lemmes.

Un axe d'amélioration serait de tester si la prédictions multi-tâches peut avoir un impact positif pour chaque prédictions par rapport à un model uni-tâche. Egalement, ce serait intéressant de comparer, ici aussi, avec les transformers pre-train.

6 Conclusion

Les défi principaux lors de ce projet ont été le processus de conception, d'entraînement et de fine-tuning des différents modèles. Grâce à ça, nous avons développé des compétences en matière de gestion du temps et d'organisation, tout en renforçant notre persévérance face aux défis. Ce projet a non seulement enrichi notre compréhension du deep learning, du NLP et des transformers, mais a aussi renforcé notre capacité à gérer efficacement des situations complexes et exigeantes. De plus, lors de l'entraînement de nos modèles, nous les avons exécutés sur des GPU. Ce fut un défi, car plusieurs membres de l'équipe ne disposaient pas forcément d'ordinateurs suffisamment puissants. En conséquence, nous avons également utilisé les GPU de Google Colab pour faire tourner nos modèles. Cette expérience nous a donc permis de maîtriser l'utilisation de PyTorch sur GPU et d'approfondir nos connaissances sur Google Colab.

En termes de perspectives futures, plusieurs voies pourraient être explorées. Premièrement, allouer plus de temps au fine-tuning de nos modèles, pourrait permettre d'améliorer davantage leurs performances. De même, l'acquisition et l'utilisation de jeux de données plus volumineux, surtout pour les modèles faits maison, seraient bénéfiques pour tester la robustesse et l'efficacité de ces modèles dans des contextes plus variés et leur permettre d'obtenir de meilleures performances.

L'exploration d'autres architectures, méthodes d'encodage et de manières de traiter les OOV pourrait également ouvrir la porte à des améliorations importantes. Par exemple, tester nos modèles dans un cadre multilingue ou sur des tâches multiples pourrait révéler des aspects intéressants de leur adaptabilité et de leur efficacité dans des scénarios plus complexes.

En conclusion, bien que nous soyons satisfaits des performances atteintes, notamment avec des modèles faits maison surpassant les modèles pré-entraînés dans certains cas, il reste encore un large éventail d'opportunités d'amélioration et d'exploration. Ce projet a posé les bases pour de futures recherches dans le domaine du NLP et du deep learning, et nous sommes impatients de voir comment ces technologies continueront à évoluer et à influencer le monde de l'intelligence artificielle.

De plus, nous sommes donc maintenant confiant sur le fait qu'utiliser des modèles de plus petites tailles mais entrainés sur des tâches spécifique peut s'avérer bénéfique.

7 Bibliographie

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*. Disponible sur : `https://arxiv.org/abs/1810.04805`

2. Martin, L., Muller, B., Ortiz Suárez, P. J., Dupont, Y., Villemonte de la Clergerie, É., Romary, L., Seddah, D., & Sagot, B. (2020). CamemBERT: a Tasty French Language Model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7203-7219. Disponible sur : `https://www.aclweb.org/anthology/2020.acl-main.645`

3. Hugging Face's Transformers: State-of-the-art Natural Language Processing. Disponible sur : `https://huggingface.co/models`

4. Universal Dependencies - French Sequoia. Disponible sur : `https://universaldependencies.org/treebanks/fr_sequoia/index.html`

5. Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, 173-180. Association for Computational Linguistics.

6. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780. MIT Press.

7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 5998-6008. Disponible sur : `https://arxiv.org/abs/1706.03762`