Projet 7 : Utilisation d'XLNet pour la catégorisation de film par genre

Preuve de concept

Auteur : Antoine Chesnais
Date dernière version : 05/02/2020

Projet réalisé dans le cadre de la formation
« Ingénieur Machine Learning » d'Openclassrooms.

Table des matières

I. Introduction....3

A.Contexte....3

B.La tâche....3

II. Etat de l'art 4

A. Réseaux de neurones séquentiels pour la modélisation du langage....4

B.Les architectures « Transformer » et « Transformer-XL »....8

C.Les derniers nés de la NLP....11

III. Expérimentation....13

A.Les données....13

B.Modélisation baseline :....14

C.XLNet....15

IV. Conclusion....16

V.Sources 17

A.Publications :....17

B.Cours :....17

C.Articles de blog :....17

D.Vidéos :....17

E.Code :....17

I. Introduction

A. Contexte

Le traitement naturel du langage (NLP) a connu de nombreuses avancées fulgurantes ces dernières années, en particulier depuis 2018 avec l'apparition d'algorithmes dits de « modélisation » du langage tel que Elmo, ULMFit ou bien encore plus récemment Bert. L'objet de cette étude sera d'appréhender et de mettre en œuvre le dernier né des recherches en NLP : XLNet, lui aussi un algorithme de modélisation du langage et qui est à ce jour le plus performant sur une majorité des tâches et datasets de benchmarking.

Cette étude est composée de deux parties majeures : l'état de l'art et l'expérimentation. Dans la partie « état de l'art » sera effectuée une retrospective des avancées en NLP de ces dernières années, en détaillant le fonctionnement des modèles fondamentaux. Pour la partie « expérimentation », il s'agira de comparer les performances d'XLNet contre celles d'une approche basique utilisée couramment pour répondre à une même problématique. Cette problématique est détaillée dans la partie suivante.

B. La tâche

Les deux algorithmes seront testés sur une tâche de classification de texte multi-label supervisée. Il s'agit de prédire le genre d'un film à partir de son intrigue parmi 50 catégories, chaque film pouvant avoir plusieurs genres qui lui sont associés.

Le dataset utilisé est le « CMU Movie Summary Corpus », collecté par David Bamman, Brendan O'Connor, et Noah Smith, du « Language Technologies Institute and Machine Learning Department » à l'Université Carnegie Mellon.

Le dataset a été traité au préalable pour ne conserver uniquement que les identifiants des films, leur intrigue et les genres associés.

II. Etat de l'art

Dans cette partie sera exposé l'état de l'art concernant les problématiques de NLP. On présentera dans un premier temps les réseaux de neurones séquentiels, briques essentielles qui ont permis l'établissement d'algorithmes de modélisation du langage et ainsi l'ouverture de nouveaux horizons en la matière. Sera ensuite montré le fonctionnement d'une autre approche de modélisation du langage, les « Transformers », conçu pour palier à certaines lacunes des modèles précédents. Cette partie se finira par la présentation de deux modèles, Bert et XLNet, tous les deux très récents et ayant permis d'établir de nouveaux records en termes de performances.

Cette rétrospective peut paraître conséquente, néanmoins il est important de passer sur chacun de ces points afin de bien comprendre les différentes évolutions qui ont permis à XLNet de voir le jour. A noter que ces évolutions (du moins leur mise en pratique), s'est faite sur un très court laps de temps puisque les publications concernant ces modèles sont sorties courant 2018/2019.

A. Réseaux de neurones séquentiels pour la modélisation du langage

1. Réseaux de neurones récurrents (RNN)

Les réseaux de neurones récurrents ont été mis en place pour la modélisation de données séquentielles, pour lesquelles le partage d'informations entre les différentes features est nécessaire, ce qui n'est pas possible avec des architecture NN classiques. De plus les RNN permettent de travailler avec des séquences de taille variable, ce qui n'est pas le cas des NN classiques.

Par exemple, pour les deux phrases « Henry Roosevelt was an american political man »et « Henry went to the Roosevelt Museum », il est important pour le modèle de savoir que pour la première phrase « Henry » et « Roosevelt » sont à la suite et que ce n'est pas le cas pour le second exemple. Il est donc important de considérer nos données d'entrée comme une séquence de features et non plus un ensemble non ordonné de celles ci.

Le principe des RNNs peut être décrit par le schéma ci dessous (figure 1) :

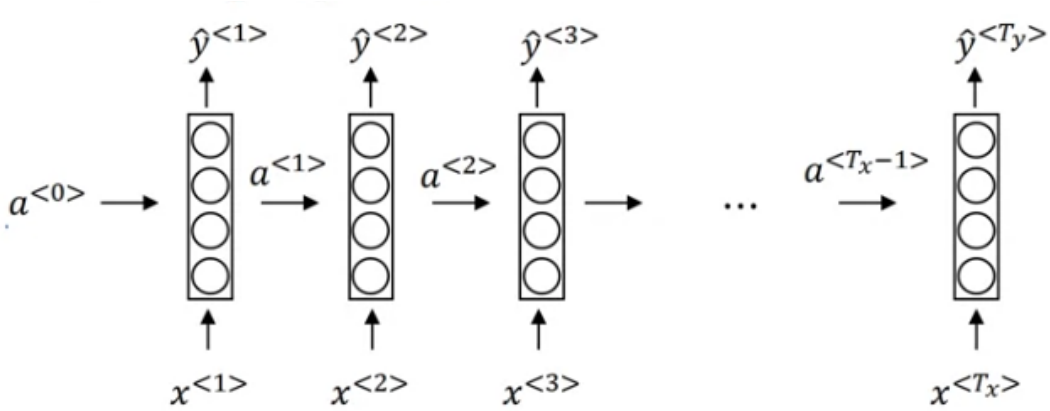

Figure 1: Architecture réseau de neurones récurrents (source [5])

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

Celui est assez simple, il s'agit pour chaque features de notre séquence d'entrée, par exemple les différents mots d'une phrase ($X^{<T>}$...), de calculer la sortie ($\hat{y}^{<T>}$) en fonction d'une valeur ($a^{<T>}$), calculée à partir de l'entrée ($X^{<T>}$) mais également d'une valeur issue du module précédent ($a^{<T-1>}$) . Les sorties sont ainsi déterminées de manière séquentielle et non plus en parallèle comme pour les NN classiques. Pour calculer la sortie issue de ($X^{<2>}$) il est nécessaire d'avoir calculé au préalable ($a^{<1>}$) et donc avoir processé ($X^{<1>}$) .

2. Long Short Term Memory Unit (LSTM)

Le principal défaut des RNN est leur incapacité à détecter des dépendances entre les features sur le long terme, hors il s'agit d'un phénomène que l'on rencontre très souvent dans les données séquentielles, notamment le texte.

Par exemple, entre deux phrases très similaires :

« The cat, which already ate a bunch of delicious food composed of fresh meat and fish, was full »

« The cats, which already ate a bunch of delicious food composed of fresh meat and fish, were full »

Il est important pour un modèle de se « souvenir » que le mot « cat » est soit au singulier, soit au pluriel, pour pouvoir déterminer le verbe « be » qui se retrouve bien plus tard dans la phrase Les RNN rencontrent des difficultés face à ce genre de problématique. Pour palier à celle ci les unités LSTM ont été crées, ci dessous en figure 2 un exemple de plusieurs de celles ci à la suite, formant un réseau LSTM :

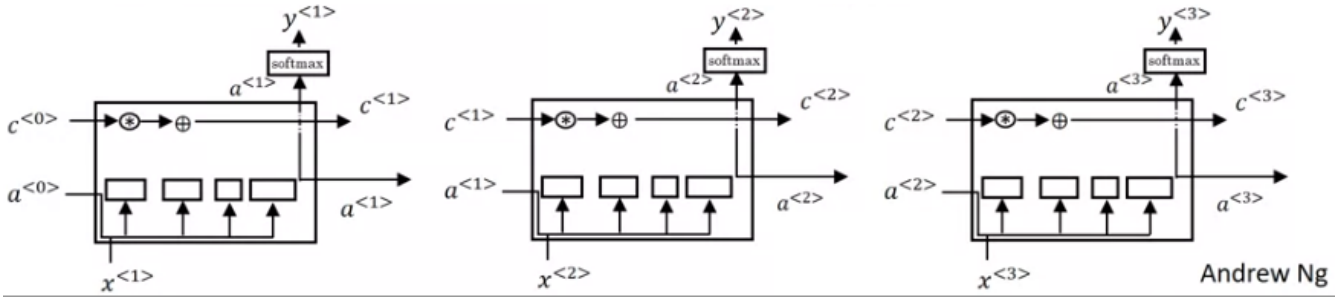

Figure 2: Architecture réseau LSTM (source [5])

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

De la même manière que pour les RNN la sortie est calculée à partir d'une valeur ($a^{<T>}$). Néanmoins cette fois ci le calcul de ($a^{<T>}$) est beaucoup plus complexe. Il ne se base plus uniquement sur ($X^{<T>}$) et ($a^{<T-1>}$) mais également sur une valeur de « mémoire » ($C^{<T>}$), qui permet de conserver des informations issues de features utilisées préalablement. Cette valeur ($C^{<T>}$) peut être mise à jour à chaque étape en fonction d'une nouvelle valeur de mémoire potentielle ($\mathcal{C}^{<T>}$) et de la mémoire précédente ($C^{<T-1>}$) . Cette mise à jour optionnelle permet, au long de l'avancement du processing des features successives, de pousser des informations utiles plus profondément dans le réseau.

3. Modélisation du langage

La notion de modélisation du langage est apparue pour palier aux limitations des « words embeddings », c.a.d la description d'un mot par un vecteur exprimant sons sens. Cette représentation, bien que plus intéressante que de représenter un mot par un simple vecteur « one hot » issu d'un vocabulaire, ne prenait pas en compte les multiples sens qu'un même mot peut avoir selon son contexte.

Par exemple si l'on prend les deux phrases suivantes, le sens du mot « bank » diffère totalement :

- « Henry was playing along the river bank »
- « I've been to the bank to get some cash »

Modéliser un langage consiste à entraîner un modèle à prédire le mot suivant à partir des mots précédents, et ce de manière successive, prenant donc en compte le contexte dans lequel chaque mot est employé. On entraine ainsi le modèle sur une grande collection de texte et l'on peut ensuite le réutiliser en l'adaptant pour différentes tâches.

Les aboutissements de cette démarche les plus marquants sont « ELMo » et « ULMFit », qui ont marqué le début du transfer learning en NLP. Par exemple « ULMFit » est un ensemble d'unités LSTM, comme on peut le voir sur l'exemple ci dessous (figure 3). Lors de son entrainement, il est conditionné sur un grand corpus de texte à prédire le mot suivant à partir de ceux qui le précédent. On peut ensuite supprimer les couches hautes (dernière couche LSTM, decoder) pour les adapter à une tâche spécifique (classification , génération de texte …) et entraîner à nouveau notre modèle, c'est le « fine tuning ». Le processus est comparable à celui effectué pour le traitement d'images avec des CNN.

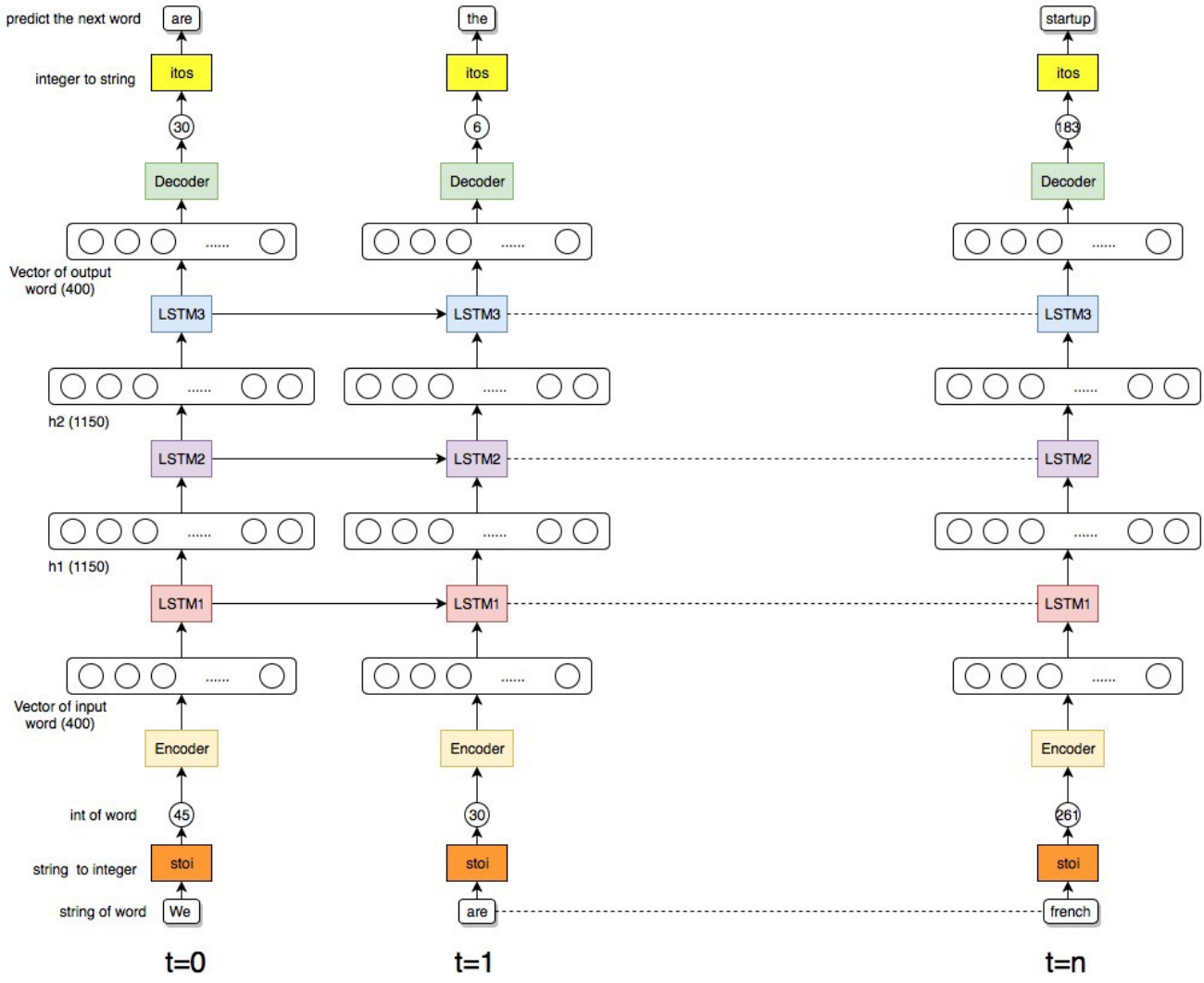

Figure 3: Exemple de réseau ULMFit (source [8])

En détails, l'entrainement d'un tel model s'effectue de la manière suivante :

- On fournit au premier bloc LSTM le premier mot d'une phrase ($X^{<1>}$: « We »)
- Le modèle effectue une prédiction du second mot ($\hat{y}^{<1>}$: « are »)
- On fournit au second bloc LSTM le **réel** (et non pas la prédiction) second mot de notre phrase ($X^{<2>}$ « are »)
- Le modèle effectue la prédiction pour le troisième mot ($\hat{y}^{<2>}$: « the»)
- et ainsi de suite jusqu'à prédire $\hat{y}^{<n>}$.
- On calcule pour chaque étape (1, 2, …, n) la fonction de coût :

$$L\left(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}\right)=-\sum y_i^{\langle t\rangle}\log\left(\hat{y}_i^{\langle t\rangle}\right)$$

$$\hat{y}_i^{\langle t\rangle}=P\left(\hat{y}^{\langle 1\rangle}\right)*P\left(\hat{y}^{\langle 2\rangle}|\hat{y}^{\langle 1\rangle}\right)*P\left(\hat{y}^{\langle t\rangle}|\hat{y}^{\langle 1\rangle}, \hat{y}^{\langle 2\rangle}, \ldots, \hat{y}^{\langle t-1\rangle}, \hat{y}^{\langle t\rangle}\right)$$

- On calcule la fonction de perte que l'on cherche à optimiser :

$$L=\sum L^{\langle t\rangle}\left(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}\right)$$

- A partir de l'erreur obtenue obtenue à l'étape précédente on effectue une descente du gradient pour optimiser les paramètres

B. Les architectures « Transformer » et « Transformer-XL »

La principale faiblesse des modèles basés sur des RNN ou bien LSTM est que de part leur architecture séquentielle, la parallélisation des calculs est impossible, chacun devant être effectué après le précédent. Cela peut générer des temps de calcul très importants, limitant la mise en œuvre de réseaux de grande taille. Afin de palier à cela, une nouvelle architecture a vu le jour, le « Transformer », plus proche d'un réseau de neurones classique et introduisant une mécanique « d'attention » pour palier à l'incapacité de ceux ci à partager de l'information entre les différentes features. Cette mécanique « d'attention » remplace celle de récurrence des RNN ou LSTM pour la modélisation de dépendances entre les features. Néanmoins, à l'instar de ses prédécesseurs, ce modèle est entrainé sur des tâches de prédictions des mots suivants en prenant en compte uniquement les mots précédents ou suivants, dans un seul sens à chaque fois.

Cette architecture telle que présentée pour la première fois est décrite ci dessous en figure 4 :

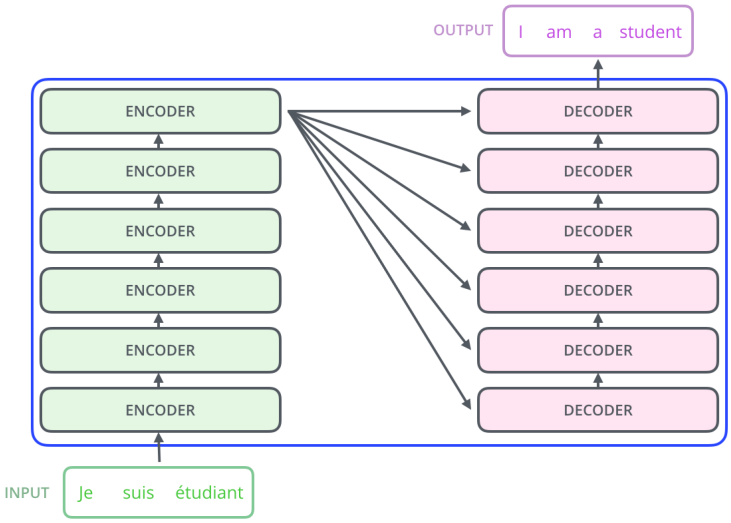

Figure 4: Architecture globale du réseau "Transformer" (source [6])

Il s'agit d'un ensemble de plusieurs blocs d'encodage, suivi par des blocs de décodage. A noter que les couches de décodages sont ici présentes car cette architecture a été utilisée au départ pour de la traduction de texte. Pour ce qui nous intéresse, la modélisation du langage, seules les couches d'encodage sont utilisées, on se concentrera sur ces couches en particulier par la suite. Le détails de ces couches sont donnés ci dessous en figure 5 :

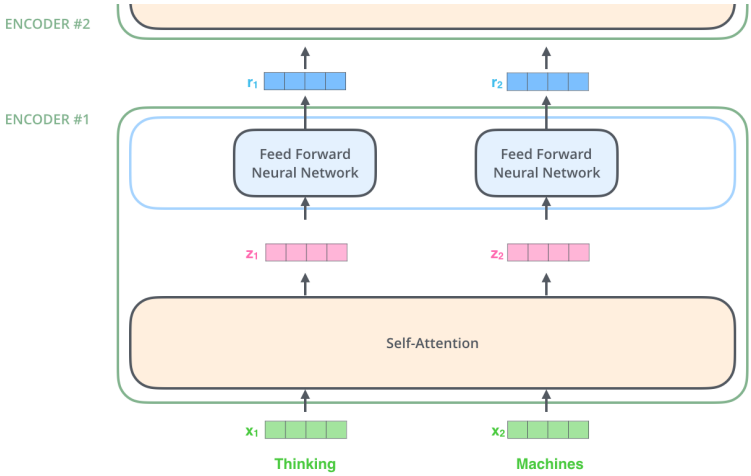

Figure 5: Détails d'un bloc "encoder" (source [6])

Un bloc d'encodage est composé de deux couches, une couche dite « d'attention » suivie d'une couche de neurones classique. C'est cette couche « d'attention » qui constitue la principale innovation de ce modèle et qui permettra au modèle de se focaliser sur certaines parties de la séquence pour la prédiction d'une feature / mot en particulier. Elle prend en entrée soit le « word embedding » (X, pour la première couche), soit la sortie du bloc d'encodage précédent (R) et en sort une nouvelle représentation (Z) qui sera transmise à la couche de neurones classique.

Dans le détails, la couche « d'attention » fonctionne de la manière suivante (figure 6) :

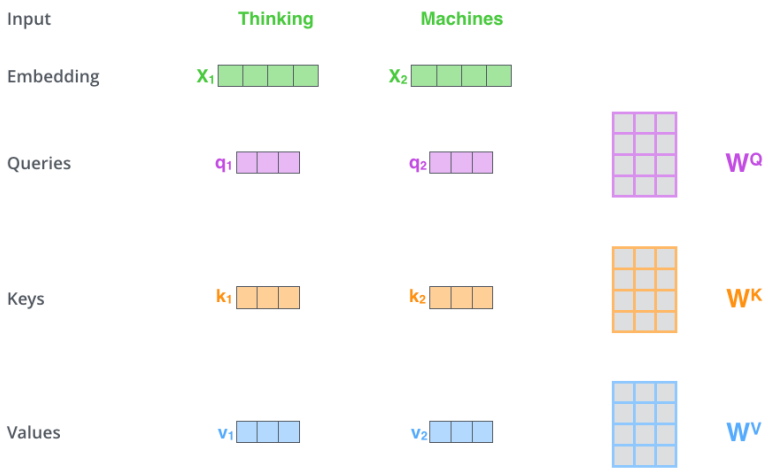

Figure 6: Calcul des vecteurs "Queries", "Keys" et "values" de la couche d'attention (source [6])

On calcule dans un premier temps 3 nouveaux vecteurs pour chaque mot en entrée : Les « Queries », « keys » et « Values ». Il s'agit du produit de l'entrée X avec 3 matrices différentes (Wq, Wk et Wv), qui sont des paramètres du modèle qui seront à apprendre. Ces 3 nouveaux vecteurs Q, K et V sont ensuite utilisés pour calculer la sortie Z selon le principe ci dessous (figure 7) :

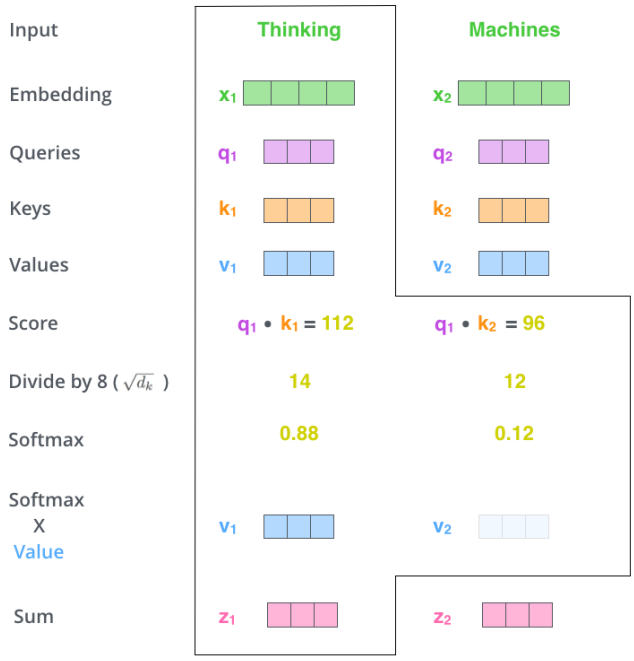

Figure 7: Calcul du vecteur Z de sortie de la couche d'attention (source [6])

Pour chaque entrée X sont calculés de multiples scores, un pour chaque mot composant la séquence. Chaque score est calculé en effectuant le produit du vecteur Q du mot d'entrée X concerné et le vecteur K pour lequel on cherche à calculer le score.

Ainsi pour pour calculer les scores obtenus par la première entrée dans l'exemple ci dessus, on calcule dans un premier temps le score de X1 par rapport à lui même, avec le produit Q1*K1, puis celui de X1 par rapport à X2 en calculant Q1*K2. Cela est fait pour chaque mot de la séquence. Les scores sont ensuite divisé par un nombre (dépendant du modèle, ici 8) puis passés à une fonction softmax pour obtenir un score entre 0 et 1.

De nouveaux vecteurs V sont ensuite obtenus en multipliant les vecteurs valeurs et les scores obtenus, l'idée étant de conserver les valeurs initiales et de se focaliser sur les parties pertinentes (en les multipliant par des valeurs importantes) tout en diminuant l'impact de celles qui ne le sont pas (avec une multiplication par un nombre « petit »). La sortie de la couche « d'attention » est ensuite obtenue en effectuant la somme de ces vecteurs.

Cette opération « d'attention » est répétée plusieurs fois au sein d'une même couche, processus connu sous le nom de « multiple attention head » (voir figure 8). Cela permet d'obtenir de multiples représentations de notre mot d'entrée, chaque « head » pouvant se focaliser sur un aspect particulier de la séquence (Genre, nombre, lien entre nom et pronom …). On obtient ainsi en sortie de la couche « d'attention » autant de représentations (Z) de notre entrée (X) que de « attention head ». Ces matrices sont ensuite concaténées et multipliées par une autre matrice (W), qui est un paramètre que le modèle doit apprendre et permet d'exprimer ces représentations aux bonnes dimensions pour qu'elle soit transmise à la couche suivante.

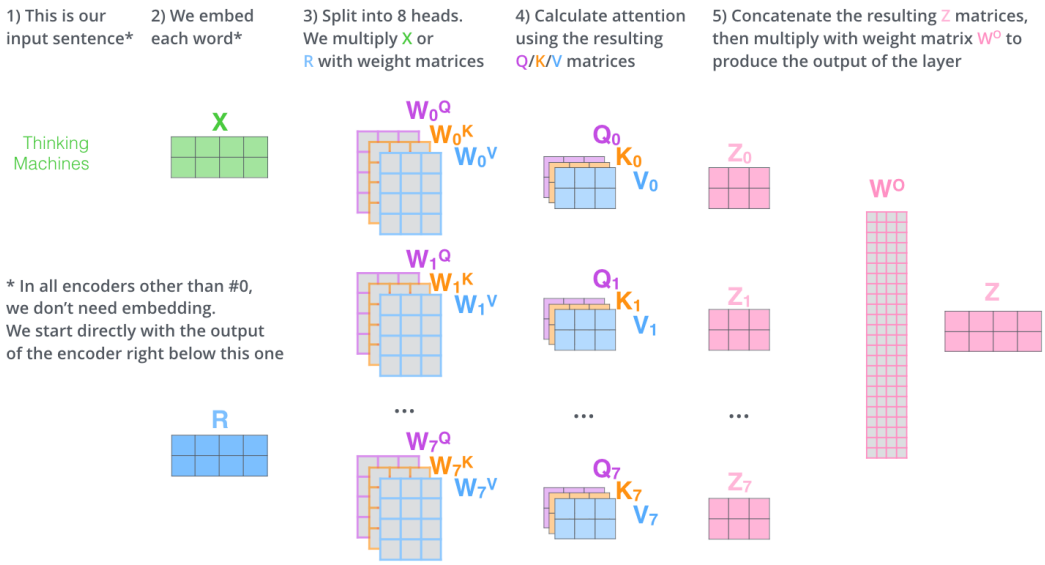

Figure 8: Détails du processus avec plusieurs "attention head" (source [6])

L'opération est ainsi répétée au sein des différents blocs « d'attention » , avec quelques autres particularités comme des connexions résiduelles et normalisation mais qui ne seront pas détaillées. Une autre mécanique qui n'est pas détaillée dans cette présentation du modèle « Transformer » mais qu'il convient de mentionner est le fait que le modèle est conscient de la position relative du mot étudié par rapport aux autres features. En effet, en entrée du modèle, en plus de convertir notre mot en « word embedding », on ajoute à cette représentation un autre vecteur encodant la position du mot au sein de la séquence.

Ce modèle « Transformer » a permis des gains conséquents en termes de temps de calcul, puisque bon nombre d'entre eux peuvent être effectués en parallèle. Il serait également plus efficace pour modéliser des dépendances entre des features éloignées au sein d'une même séquence. Néanmoins il présente également une limitation majeure : la taille des séquences qu'il peut traiter est limitée et fixe. Afin de palier à cette limitation une extension de cette architecture a vu le jour : Transformer-XL.

Transformer-XL est basé sur une idée simple pour surpasser le problème de limite de taille des séquences : faire revenir la notion de récurrence. Pour ne pas retomber sur un modèle séquentiel classique et profiter des avantages de l'architecture « Transformer » néanmoins la récurrence ne peut se faire au niveau du mot, mais de la séquence.

Ainsi il est possible de traiter des séquences de longueur infinie en théorie, en la fragmentant en sous segments qui seront traités à la suite. Le modèle stocke d'une étape à la suivante du traitement d'une même séquence les valeurs de sortie de chaque couche, qu'il peut fournir ensuite en tant qu'informations complémentaires à l'étape suivante, établissant un lien entre les séquences.

Pour réussir cette modification du modèle, les auteurs ont dû retravailler le mécanisme d'encodage de la position du mot au sein de la séquence pour que celui ci prenne en compte la position du mot au sein de la sous séquence et de la sous séquence au sein de la séquence complète.

Voici les principales innovations apportées par Transformer-XL, qui ne seront pas plus détaillées ici, mais qu'il est important de mentionner puisque c'est cette architecture qui a servi de base à l'algorithme XLNet.

C. Les derniers nés de la NLP

Dans cette parties seront abordés les derniers nés en termes de NLP : BERT(Octobre 2018) et XLNet (Juin 2019). Il s'agit de deux algorithmes de modélisation issus de l'architecture « Transformer » (Transformer-XL pour XLNet) et qui ont chacun à leur tour établis de nouveaux records de performances sur de nombreuses tâches de NLP.

L'architecture « Transformer » ayant été détaillée à la partie précédente, on se contentera dans cette partie de rester sur une vue d'ensemble de ces deux modèles en se focalisant plus sur la spécificité de leur approche.

1. BERT

BERT part du constat que l'ensemble des algorithmes de modélisation du langage possèdent un défaut majeur. En effet, ils sont entrainés à prédire la suite d'une séquence en se basant soit sur les features précédant le terme à prédire, soit sur celles qui le suivent. Le modèle ne prend donc jamais en compte le contexte dans son intégralité (de chaque côté du mot) pour évaluer sa pertinence.

Afin de palier à cette limite, BERT n'est plus entrainé à prédire le mot suivant mais à prédire des mots manquants (voir figure 9). Il masque de manière aléatoire 15% de la séquence en entrée du modèle, et cherche à prédire les termes masqués. Les termes présents de chaque côté de ceux à prédire sont ainsi pris en compte simultanément.

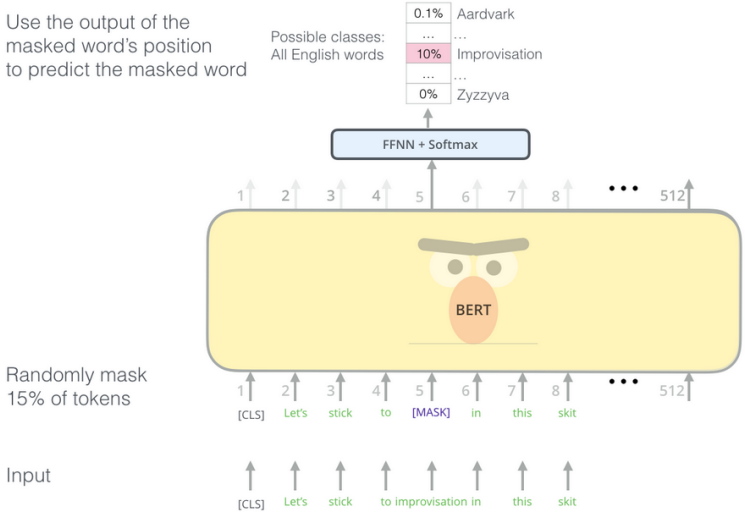

Figure 9: Fonctionnement de BERT en mode apprentissage

(source [7])

Après cette phase d'entrainement sur un large corpus de texte, il est très facile de l'adapter pour de nouvelles tâches en remplaçant la partie classification en sortie du modèle et en l'adaptant pour la tâche souhaitée.

2. XLNet

Sur la lancée de BERT, XLNet propose une nouvelle approche pour la prise en compte d'un contexte bidirectionnel, en relevant les faiblesses suivantes de son prédécesseur :

- BERT « corrompt » l'entrée en introduisant un token [Mask] qui ne sera jamais présent dans des données réelles, pouvant introduire un biais.
- BERT néglige de potentielles dépendances entre les termes masqués, par exemple :
  - When she goes to the [Mask], she buy [Mask].
  - When she goes to the mall, she buy clothes.
  - When she goes to the cinema, she buy candies.
  - When she goes to the cinema, she buy clothes.

  Le modèle n'a aucun moyen de savoir quelle combinaison est la plus pertinente.

Afin d'éviter de corrompre l'entrée, mais de tout de même prendre en compte le contexte de manière bidirectionnel, XLNet se base sur une architecture Transformer-XL, tout en effectuant des permutations lors de la prédiction d'un terme en fonction des mots qui le précède. Pour illustrer cela il est intéressant de se reporter à la figure ci dessous qui illustre pour une séquence de 4 termes X1 à X4 les différentes manières de prédire X3 (figure 10):

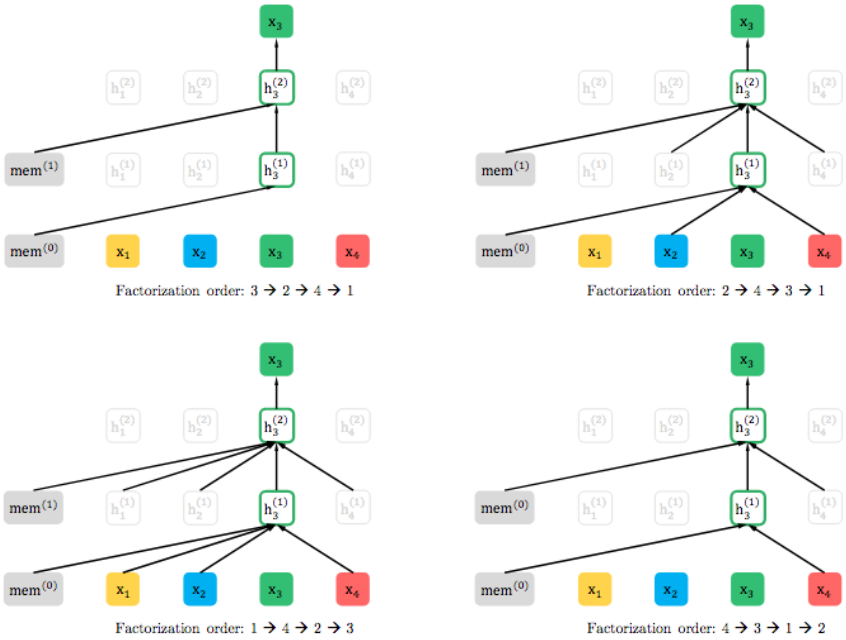

Figure 10: Exemple de calculs de X3 pour différents ordres de factorisation

(source [4])

Par exemple, pour la séquence ci dessus, il existe plusieurs manière de calculer la probabilité de X3 selon l'ordre de factorisation de celle ci. Si l'ordre de factorisation de la séquence est 4, 3, 1 puis 2 on se basera sur X4 et potentiellement l'unité de mémoire, pour évaluer les probabilités que chaque terme du vocabulaire soit X3.

En évaluant ainsi de multiples permutations, le modèles peut apprendre à la fois des termes précédents le mot à évaluer, mais aussi ceux qui le suivent. On a donc bien une prise en compte bidirectionnelle du contexte.

Ci dessous une illustration du processus complet d'évaluation de la probabilité de la séquence, pour la même séquence de X1 à X4 que précédemment, avec un ordre de factorisation 3, 2, 1 puis 4 (figure 11) :

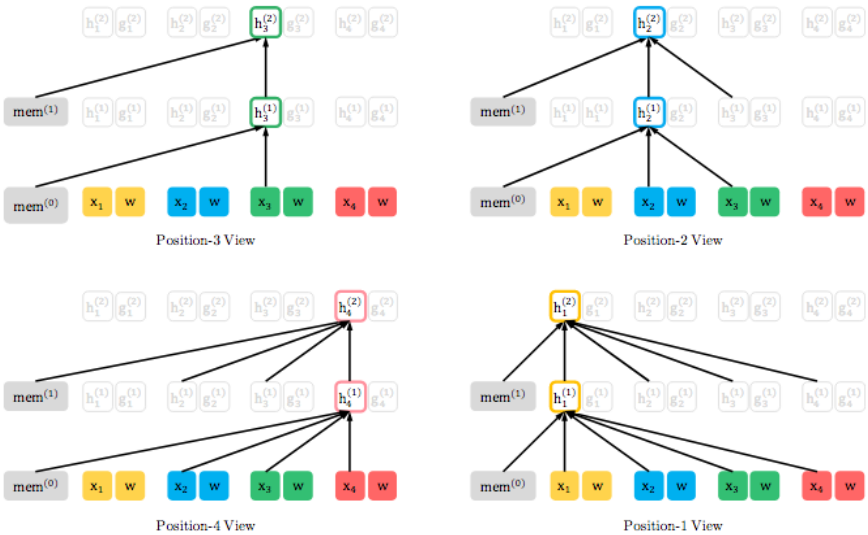

Figure 11: Exemple calcul complet (ordre factorisation 3, 2, 1 et 4) (source [4])

Les auteurs ont également dû adapter l'architecture du Transformer-XL pour la faire fonctionner avec leur stratégie de permutations multiples, mais les détails ne seront pas abordés ici.

Avec cette nouvelle approche XLNet a pu surpasser BERT dans de nombreuses tâches de benchmark en NLP, faisant de ce modèle la nouvelle référence en la matière, que l'on testera sur un cas d'étude dans la partie suivante.

III. Expérimentation

Après avoir vu les aspects théoriques d'XLNet, il est intéressant de le voir à l'oeuvre.

Pour cela on le confrontera à un modèle naif et une approche classique sur une tâche de classification multi-label de texte. Comme annoncé en introduction, il s'agit de prédire le ou les genres d'un film à partir de son intrigue.

A noter que l'utilisation d'XLNet est gourmande en ressources, le notebook a donc été exécuté dans Google Colab, afin de profiter d'une GPU avec 12Gb de Ram.

A. Les données

Comme mentionné en introduction les donnés sont issues de la base de données « CMU Movie Summary Corpus », pour lesquelles ont été conservés uniquement l'identifiant du film, son intrigue et les genres associés. A noter que le texte de l'intrigue a été légèrement nettoyé, avec le retrait de la ponctuation, des caractères spéciaux et des chiffres. Le texte a également été passé en minuscule et les stopwords anglais ont été retirés.

Sous sa forme initiale, le dataset contient 363 genres de films différents, certains étant très peu représentés. Il a été choisi de conserver uniquement les 50 plus courants, cela n'empêchant pas un important déséquilibre de la distribution des tags. Enfin, les données ont été séparées entre set d'entrainement et set de test. Le set d'entrainement contient 29 806 échantillons et celui de test 3 302. Les distributions des cibles sont similaires entre les deux sets (figures 12 + 13).

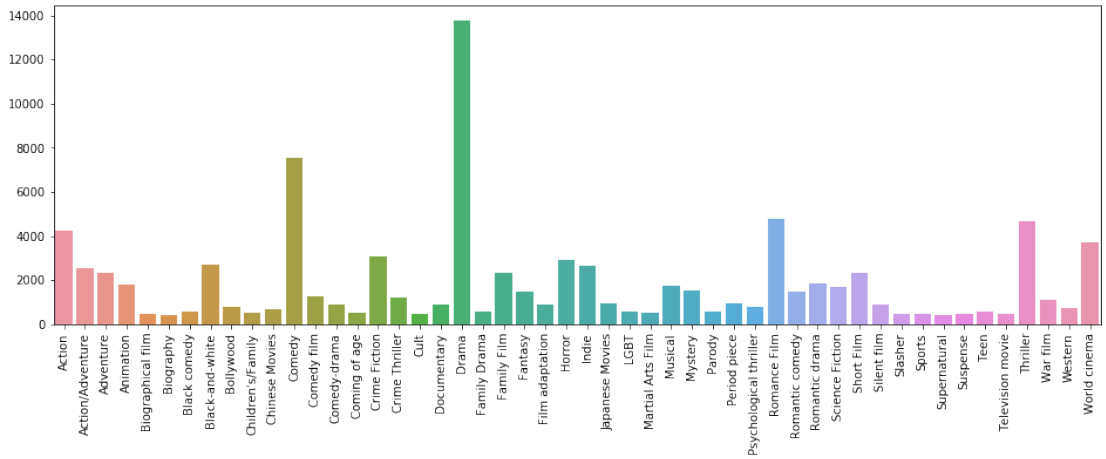

Figure 12: Distribution des genres de films sur le set d'entrainement

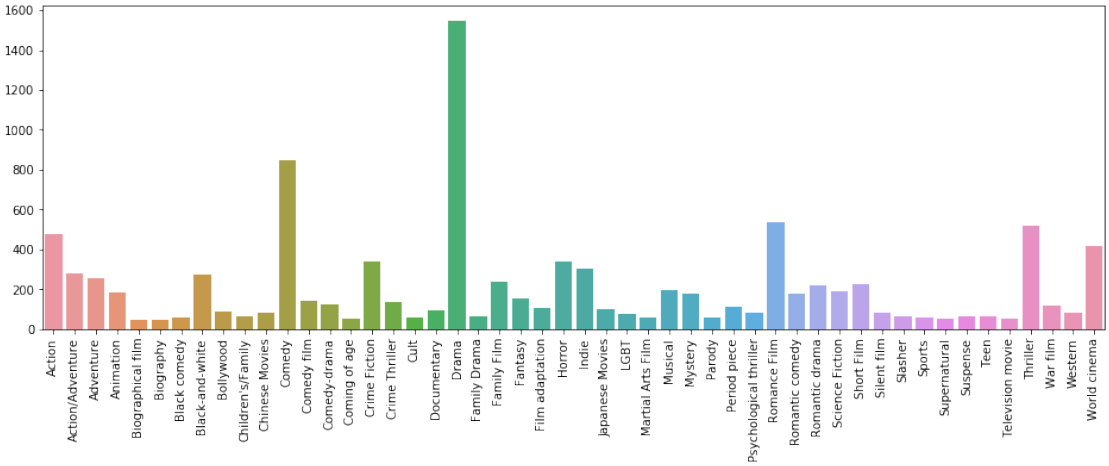

Figure 13: Distribution des genres de films sur le set de test

B. Modélisation baseline :

Afin d'avoir un point de comparaison des performances d'XLNet, une baseline a été implémentée, basée sur un **classifieur SVM**.

Pour pouvoir effectuer cette modélisation de référence, il a fallu dans un premier temps extraire les features du texte. Cela a été fait via une représentation **TF-IDF** de notre corpus de textes, en excluant les termes apparaissant dans moins de 10 textes et dans plus de 20% de notre corpus. On obtient ainsi une matrice de données composées de 29 806 échantillons et 23 638 features. Ce dataset a ensuite été séparé en train / validation sets avec un ratio 80/20.

Les modèles réalisés sont évalués sur différents critères : présence d'overfitting, niveau de **précision**, **rappel** et **F1 score** (agrégé de façon 'macro' sur les différentes classes). Afin d'optimiser la baseline, une recherche du meilleur paramètre de régularisation a été effectuée (résultats en figure 14) :

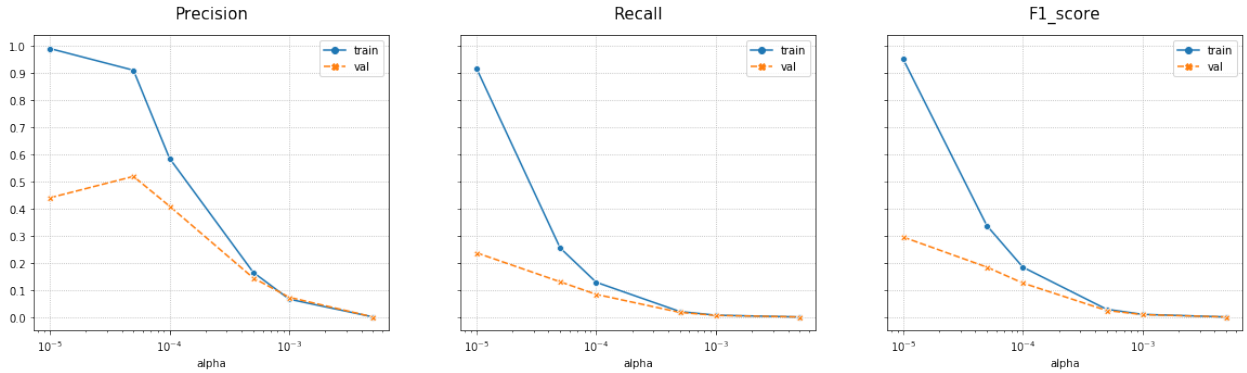

Figure 14: Score obtenus par le classifieur SVM avec différentes valeurs de alpha

Au final le meilleur compromis entre overfitting et performances semble être le modèle avec alpha=0.0001, c'est la valeur de régularisation qui a été retenue. Ce modèle a ensuite été évalué sur le set de test :

- Précision : 0.41
- Rappel : 0.085
- F1 score : 0.13

Il est intéressant de regarder les résultats par genre, (voir figure 15) que l'on comparera plus tard avec XLNet :

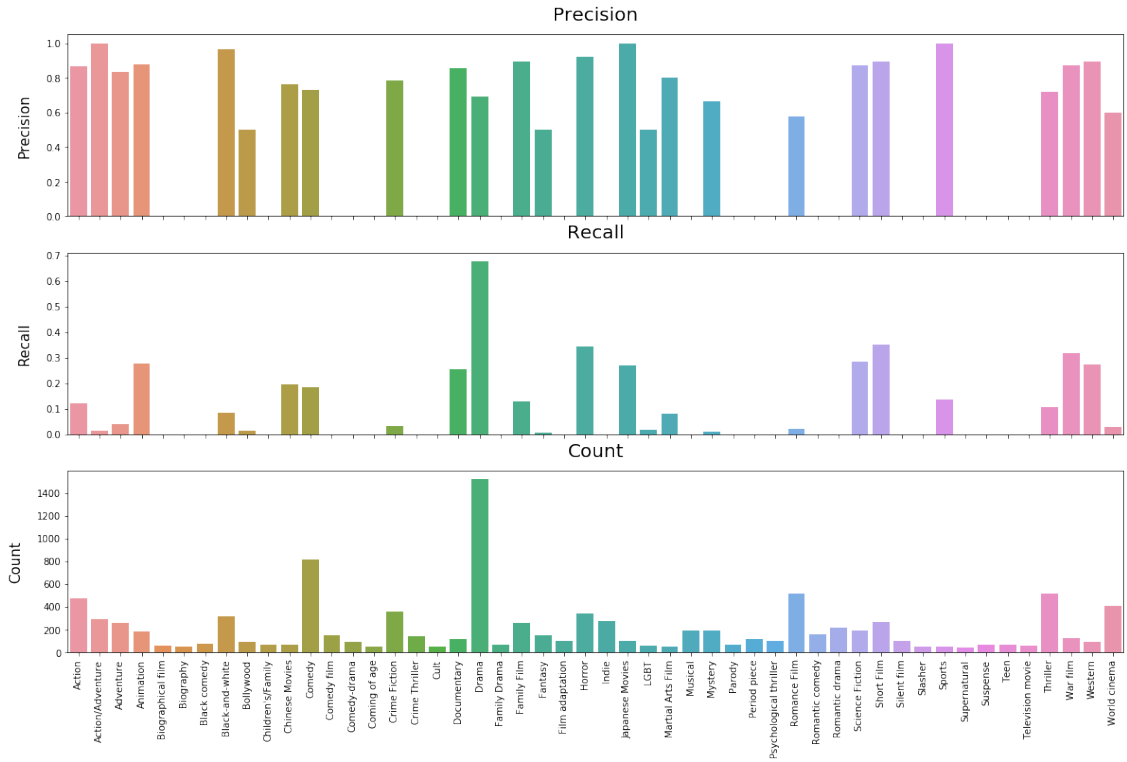

Figure 15: Détails de résultats SVM par classe sur le set de test

C. XLNet

Pour l'utilisation d'XLNet sur notre dataset, le code joint à l'article de blog [11] a été adapté. Il a fallu modifier la dernier couche du modèle pour qu'elle effectue des prédictions sur les 50 classes que contient le dataset. Le modèle génère des probabilités pour chaque genre, et différents seuils de décision ont été testés pour choisir le plus adéquat.

Cette fois ci les données fournies à algorithme sont une chaine de caractère complète représentant l'intrigue du film pour les features et matrice one hot encoded des genres de film en cible. Deux longueurs de séquence ont été testées, réduisant les intrigues à 250 tokens dans un cas et 500 tokens dans l'autre. Ci dessous les résultats de l'utilisation d'XLNet sur le set de test (figures 16 et 17):

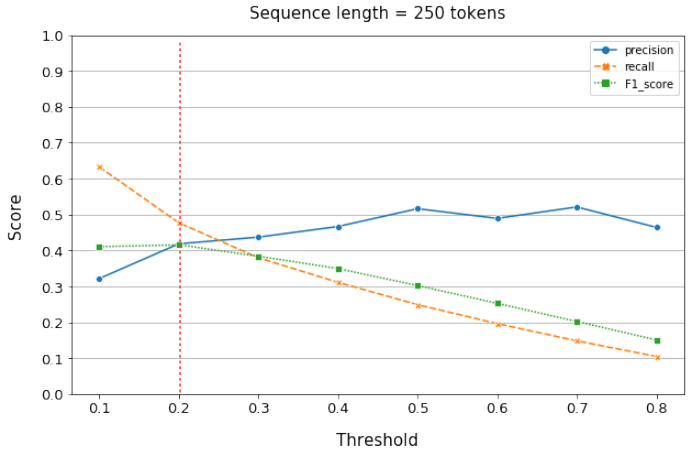

Figure 16: Résultats XLNet avec 250 tokens max

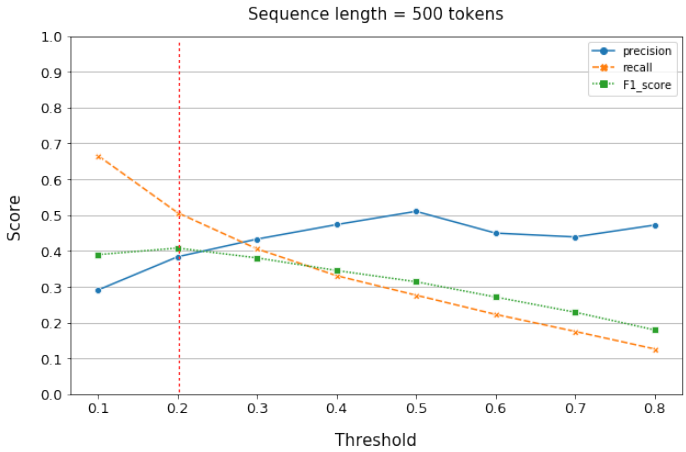

Figure 17: Résultats XLNet avec 500 tokens max

Les comportements sont relativement similaires quelle que soit la taille de séquence choisie, un seuil de 0.2 donne les meilleures performances en terme de F1 score. Le modèle avec une séquence de 250 mots donne une précision et un rappel légèrement plus proches, on retiendra donc celui ci. Le niveau de précision atteint est similaire à celui de la baseline (environ 0.4) mais les performances sont bien meilleures en termes de rappel (0.48 vs 0.085). Ce résultat flagrant lorsque l'on regarde le détail des scores par classe (figure 18):

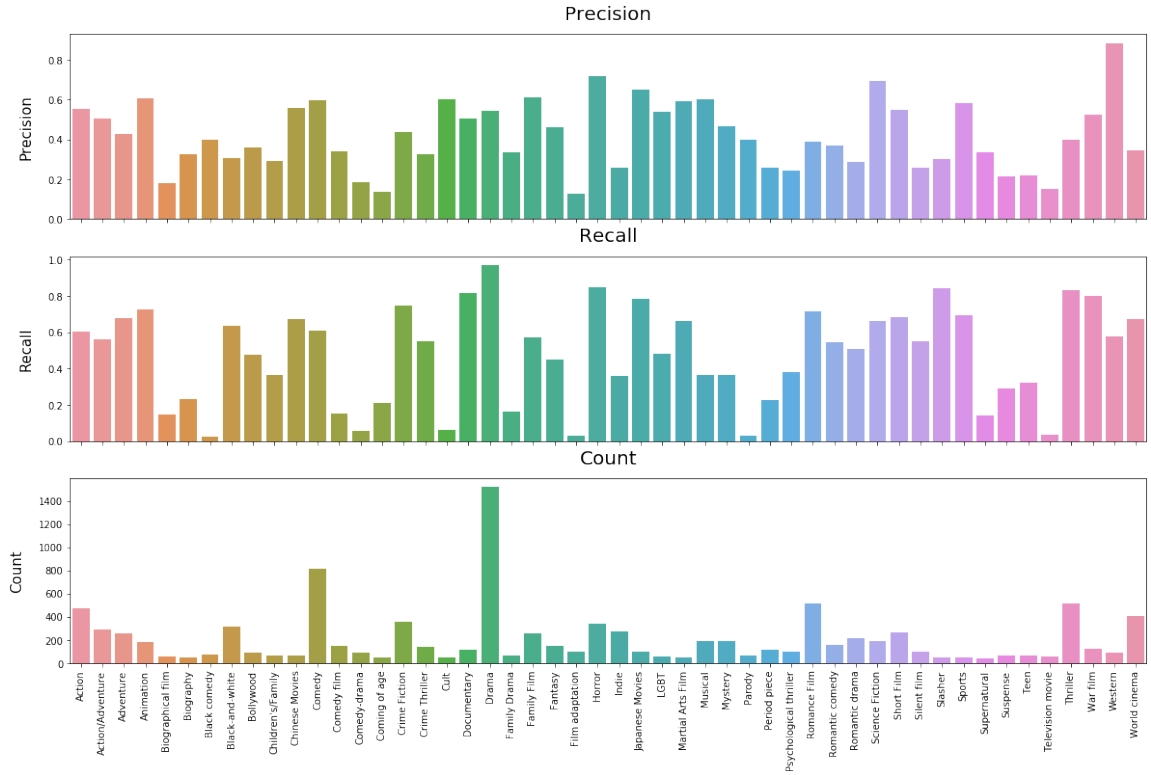

Figure 18: Résultats XLNet par classe sur le set de test

Les classes les moins représentées sont cette fois beaucoup mieux prédites qu'avec la baseline. Les scores les plus faibles sont souvent obtenus pour des genres redondants, dérivés d'un autre ou peu explicites, et donc difficile à discerner comme par exemple « Télévision movie », « parody » ou bien encore « black comedy ».

IV. Conclusion

Le domaine de la NLP ayant connu de nombreuses et importantes évolutions ces dernières années, il était intéressant dans cette étude de se pencher à la fois sur les aspects théoriques et pratiques de celui ci.

Nous avons ainsi pu aborder dans ce document les aspects théoriques de la modélisation du langage via les réseaux de neurones et leurs évolutions à travers le temps, chaque nouveau modèle capitalisant sur le précédent et cherchant à combler ses lacunes.

Concernant ces évolutions, l'apparition en 2017 de la publication « Attention is all you need », présentant une nouvelle architecture de réseau de neurones « Transformer », a marqué un tournant dans le milieu de la NLP. Deux nouveaux algorithmes basés sur cette architecture ont ensuite vu le jour et ont battus chacun à leur tour des records sur de nombreuses tâches de NLP. Il s'agit des modèles BERT (2018) et XLNet (2019), tous les deux provenant de Google. XLNet étant à l'heure actuelle la référence, surpassant BERT sur de multiples tâches de Benchmark.

Pour ce qui est des aspects de mise en pratique de l'algorithme XLNet, nous avons pu le tester sur une tâche de catégorisation multi-label de texte, la prédiction du genre ou des genres d'un film à partir de son intrigue sous forme de texte. Afin d'avoir une performance de référence, une baseline a été établie sur base d'une représentation TF-IDF du texte et d'un classifieur SVM , permettant de quantifier les apports de XLNet.

Sur notre dataset, la baseline a donné les performances suivantes :

- Précision : 0.41
- Rappel : 0.085
- F1 score : 0.13

et XLNet :

- Précision : 0.41
- Rappel : 0.48
- F1 score : 0.41

XLNet est donc plus efficace que notre baseline, avec notamment un rappel beaucoup plus élevé. Cela se traduit par une capacité du modèle à prédire également les classes les moins représentées, là où la baseline se contentait de ne jamais les faire apparaître dans les prédictions. A noter que les scores les plus faibles sont souvent obtenus pour des genres redondants, dérivés d'un autre ou peu explicites, et donc difficile à discerner comme par exemple « Télévision movie », « parody » ou bien encore « black comedy ».

Après cette expérimentation, il est possible de conclure qu'XLNet est donc capable de détecter plus de nuances au sein des textes et ainsi déceler les genres de films plus finement que des méthodes classiques.

V. Sources

A. Publications :

[1] : « Attention Is All You Need » , de Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

[2] : « Character-Level Language Modeling with Deeper Self-Attention » , de Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, Llion Jones

[3] : « Transformer-XL: Attentive Language ModelsBeyond a Fixed-Length Context » , de Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov

[4] : « XLNet: Generalized Autoregressive Pretrainingfor Language Understanding » , de Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le

B. Cours :

[5] : Cours « Sequence models » de Andrew Ng sur Coursera

C. Articles de blog :

[6] : Article « The illustrated Transformer » de Jay Alammar

[7] : Article « The Illustrated BERT, ELMo, and co. » de Jay Alammar

[8] : Article « A NLP Use Case: An application of Universal Language Model Fine-Tuning to the classification of multiclass company descriptions » de Guanguan ZHANG

D. Vidéos :

[9] : « [Transformer] Attention Is All You Need » de la chaîne AISC

[10] : « XLNet: Generalized Autoregressive Pretraining for Language Understanding » de la chaîne AISC

E. Code :

[11] : Issu de l'article « Multi-Label Text Classification with XLNet » par Josh Xin Jie Lee