Advanced Machine Learning & Deep Learning

Compte rendu des TMEs 9-10

Célina Hanouti
Aymen Merrouche

2020-2021

Table des matières

1 TME9 - Attention globale

Dans le TME 6, nous avons vu les modèles Seq2Seq qui consistent à transformer une séquence en entrée en une séquence en sortie, chacune peut avoir une longueur différente. Ces modèles reposent sur une architecture encodeur-decodeur, l'encodeur compresse l'information contenu dans l'input en un vecteur de contexte de *taille fixe* et le décodeur génère la séquence de sortie à partir de ce vecteur de contexte. L'inconvénient principal réside dans le fait que le vecteur de contexte est de taille fixe et donc incapable de mémoriser toute l'information nécessaire, particulièrement lorsque nous avons des séquences de tailles longues. Afin de pallier ce problème, (Bahdanau et al., 2016) ont proposé le mécanisme d'attention que nous allons étudier dans ce TME.

L'attention permet de pondérer les tokens de la séquence en entrée qui ont un intérêt pour la tâche considérée.

1.1 Details d'implémentation

Lors de nos expérimentations, nous avons considéré l'optimiseur SGD avec un learning rate de 0.01, en effet, après quelques expériences, nous avons remarqué qu'en utilisant l'optimiseur Adam, le modèle overfit assez rapidement les données d'entraînement, tandis qu'avec SGD, nous avons une bonne généralisation sur les données de test et validation. On utilise un batch size de 64, on fixe la taille maximum des séquences à 200 et un embedding size de 100. Les modèles sont entrainés sur 50 epochs.

1.2 Modèle de base

Dans ce TME, on considère dans un premier temps une baseline simple qui représente le texte avec la moyenne des embeddings des mots qu'il contient. Pour un texte contenant 4 mots, on aura donc des poids d'attention égales à $\frac{1}{4}$.

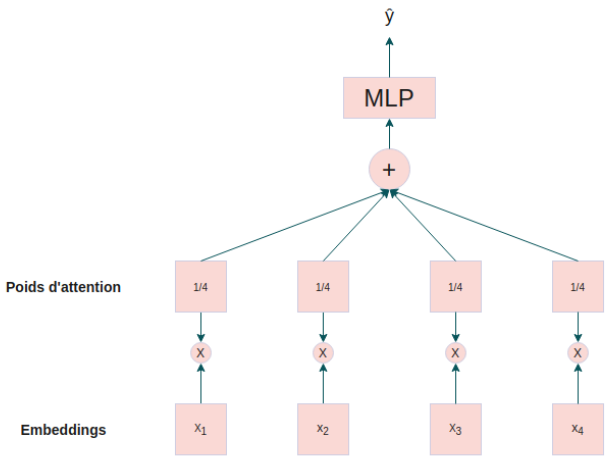
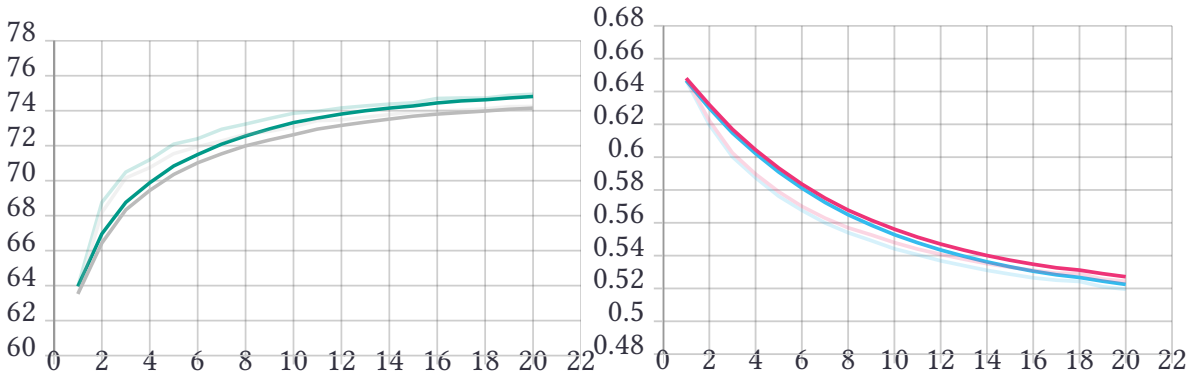


FIGURE 1 – Modèle de base



(a) **train**, **validation** (b) **train**, **validation**

FIGURE 2 – Loss et Accuracy en apprentissage et en validation du modèle Baseline

Avec ce modèle de base, nous obtenons des accuracy de 75.25% et 74.55% en apprentissage et en validation respectivement. Nous avons considéré une pondération uniforme des mots dans un texte, or, un terme peut être plus discriminant qu'un autre par rapport au sentiment du texte. Il serait donc intéressant d'apprendre une distribution sur les mots contenus dans un texte, autrement dit, pondérer fortement les mots ayant un intérêt pour la tâche de classification.

1.3 Attention simple

On considère un modèle d'attention simple où chaque token d'un texte est pondéré par une probabilité qui mesure l'attention portée sur ce token :

$$\hat{\mathbf{t}} = \sum_i p(\mathbf{a_i}|\mathbf{t})\mathbf{x_i}$$

Ces probabilités sont modélisées de la manière suivante :

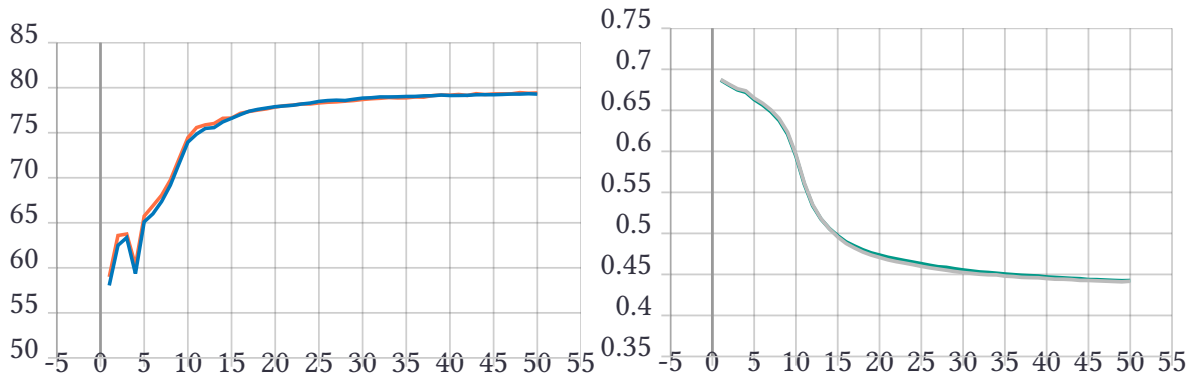
$$\log p\left(\mathbf{a}_i|\mathbf{t}\right) = \text{ constante } + \mathbf{q} \cdot \mathbf{x}_i$$

$$p\left(\mathbf{a}_i|\mathbf{t}\right) = e^{\text{ constante }} e^{\mathbf{q}\cdot\mathbf{x}_i}$$

Afin d'avoir une distribution de probabilité, on pose : $e^{\text{ constante }} = \frac{1}{\sum_j e^{\mathbf{q}\cdot\mathbf{x}_j}}$ On a donc :

$$p\left(\mathbf{a}_i|\mathbf{t}\right) = \frac{e^{\mathbf{q}\cdot\mathbf{x}_i}}{\sum_j e^{\mathbf{q}\cdot\mathbf{x}_j}}$$

On peut donc obtenir $p\left(\mathbf{a}_i|\mathbf{t}\right)$ avec la fonction `Softmax`.



(a) **train**, **validation** (b) **train**, **validation**

FIGURE 3 – Loss et Accuracy en apprentissage et en validation du modèle avec le mecanisme d'attention simple

En considérant cette pondération sur les termes du texte, nous obtenons une accuracy de **80.11**% et **79**% en apprentissage et en validation. Afin de savoir si la distribution apprise pondère fortement des termes du texte et que celle-ci n'est pas uniforme (ce qui est, dans ce cas, equivalent à la baseline), on affiche sur la figure 4 l'histogramme de l'entropie des attentions calculé sur chaque batch lors de l'apprentissage. On peut observer qu'au début de l'apprentissage, l'entropie des distributions d'attention est élevée pour la majorité des exemples. Au bout de 10 epochs, l'histogramme devient plus "plat", certaines distributions d'attention ont une entropie basse ce qui est équivalent à avoir une distribution "piquée" avec des probabilités élevées pour les termes discriminants. Certaines distributions restent uniformes, celles-ci peuvent correspondre à des séquences où on ne retrouve pas de mots liés au sentiment ou à la négation de celui-ci.

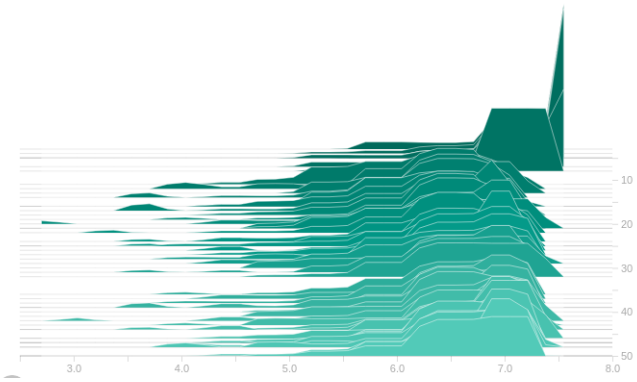


FIGURE 4 – Histogramme de l'entropie des attentions durant l'apprentissage

Afin d'observer sur des exemples précis l'attention portée sur les mots, nous avons implémenté un outil de visualisation d'attention qui permet de visualiser le texte sous forme de heatmap en prenant en compte les poids d'attention. Cette implémention est inspiré du code disponible sur ce répertoire GitHub attention-heatmap.

__OOV__ not into reality shows that __OOV__ but this one is exceptional because at the __OOV__ the viewer gets something useful out of it besides __OOV__ i __OOV__ have children but __OOV__ lessons will be a great help when __OOV__ __OOV__ i ever __OOV__ my only complaint is that the show has been watered down since it has been in the __OOV__ i prefer the british __OOV__ with the sterner nanny __OOV__ is it just me or does anyone else find the us version to be a bit __OOV__ now __OOV__ the __OOV__ __OOV__ __OOV__ i guess we yanks need

(a) Exemple positif

how much do i love this __OOV__ now __OOV__ not a fan of bad __OOV__ but i do love a film that is so bad __OOV__ __OOV__ this is one of __OOV__ juan pablo di pace has a great __OOV__ looks fab on __OOV__ and definitely __OOV__ make a bad turn at his acting debut __OOV__ __OOV__ billy zane is suitably mean and __OOV__ though i still constantly feel that there is something more in __OOV__ i felt it in __OOV__ the look on his face when la winslet spat on him for __OOV__ totally __OOV__ __OOV__ and __OOV__ ...

(b) Exemple positif

this is a poorly written and badly directed short __OOV__ pure and __OOV__ what is interesting and keep me __OOV__ to some __OOV__ was the production __OOV__ shot on video it __OOV__ with a bad script and bad __OOV__ one would think it would also have horrible production __OOV__ that is what the viewer expects when they watch a film that is terrible and shot on __OOV__ but not in this __OOV__ they spent some money and it __OOV__ it keep me very mildly interested to see what was coming __OOV__ just to __OOV__ __OOV__ __OOV__ the worst short film

(c) Exemple négatif

__OOV__ read the positive comments on this __OOV__ i assume people who were in this movie __OOV__ come to this site to give it some good press because this was one of the worst movies i have ever __OOV__ i always watch the whole film despite the quality or lack there of which explains why i watched this whole __OOV__ but i __OOV__ think i laughed even once during the duration of this __OOV__ the jokes were mostly very __OOV__ but when the jokes had some __OOV__ the delivery was __OOV__ if you liked __OOV__ maybe you should lay off

(d) Exemple négatif

Sur les exemples ci-dessus, l'attention se porte globalement sur les mots pertinents pour la prédiction du sentiment : "*good*", "*great*" et "*love*" par exemple, pour un sentiment positif.

1.4 Question et valeur

Dans le modèle précédent, nous avons considéré une seule direction dans l'espace (c'est à dire un seul $\mathbf{q}$) pour dire si un mot est important ou non pour la classification. Or, il est compliqué d'en trouver qu'une seule, pour pallier ça, on va faire en sorte que $\mathbf{q}$ dépende du texte considéré. On aura donc :

$$p\left(\mathbf{a}_i|\mathbf{t}\right) = \frac{e^{\mathbf{q}(\mathbf{t})\cdot\mathbf{x}_i}}{\sum_j e^{\mathbf{q}(\mathbf{t})\cdot\mathbf{x}_j}}$$

avec $\mathbf{q}(\mathbf{t})$ une transformation linéaire de la moyenne des embeddings du texte $\mathbf{t}$. On complexifie l'architecture considérée pour la représentation de $\mathbf{q}(\mathbf{t})$ et nous obtenons les performances répertoriées dans le tableau ci-dessous.

Modèle	Architecture de q(t)	Accuracy en train (%)	Accuracy en validation (%)
Baseline	-	75.25	74.55
Attention simple	-	80.11	79.6
Attention avec q dépendant du texte	Simple transformation linéaire	82.1	81.5
Attention avec q dépendant du texte	MLP avec une couche cachée de taille 128	82.50	82.3
Attention avec q dépendant du texte	MLP avec deux couches cachées de taille 256 et 128 chacune	82.45	81.53
Attention avec q dépendant du texte	Simple transformation linéaire + MLP avec une couche cachée pour $v_\theta(x)$	83.51	82.47

TABLEAU 1 – Performances des modèles en apprentissage et en validation

En complexifiant $\mathbf{q}(\mathbf{t})$ et $\mathbf{v}_\theta(\mathbf{x})$, on obtient globalement les mêmes performances. On peut donc conclure qu'une couche est suffisante pour représenter $\mathbf{q}(\mathbf{t})$. On notera également que les modèles généralisent plutôt bien.

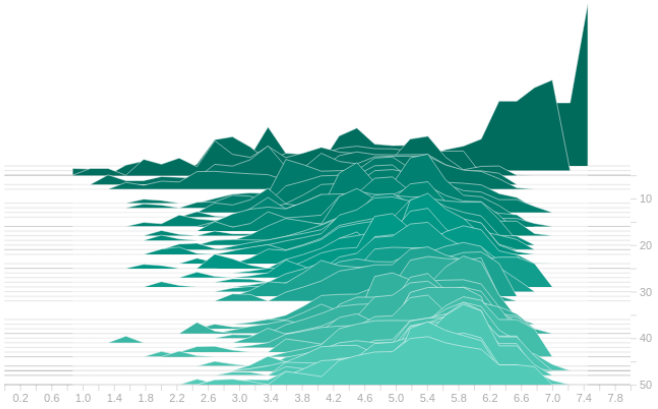


FIGURE 6 – Histogramme de l'entropie des attentions durant l'apprentissage

On considère également quelques exemples afin d'observer sur quels mots se portent l'attention :

__OOV__ get me __OOV__ i really love the __OOV__ arts __OOV__ and i get more and more surprised over how many films like this there are out __OOV__ this one is one of __OOV__ and __OOV__ not even close to be one of the __OOV__ with mathias hues in __OOV__ i thought it would be __OOV__ he __OOV__ save this movie __OOV__ and to be __OOV__ he __OOV__ very good __OOV__ just __OOV__ pay attention to what other people __OOV__ the fighting scenes in this movie are not good at __OOV__ i really know what __OOV__ talking __OOV__ since i

(a) Exemple positif

i usually really enjoy steven seagal __OOV__ they are usually highly entertaining and being somewhat of an adept of __OOV__ i usually like the way steven incorporates these martial art techniques in the fight __OOV__ __OOV__ __OOV__ this film is a really bad movie making effort and it seems obvious to me that the blame lies with the director and the producers who obviously have no idea how to make an action __OOV__ let alone direct someone like steven seagal and to take advantage of his knowledge and __OOV__ __OOV__ __OOV__ never saw the end of this __OOV__ i walked

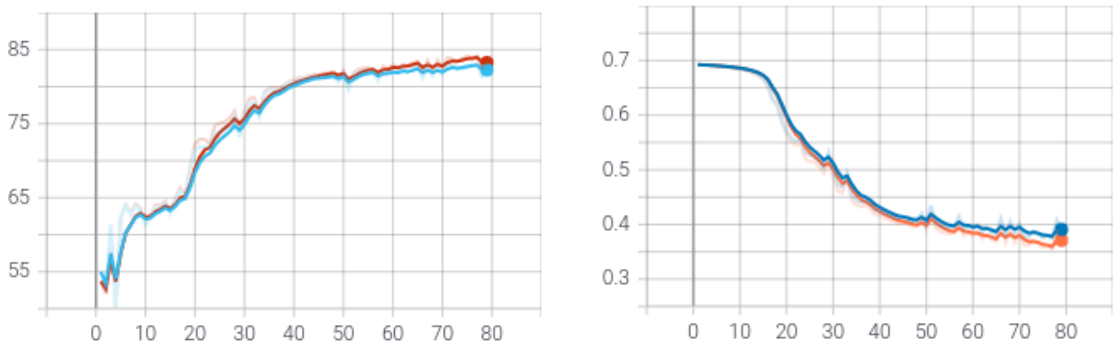
(b) Exemple négatif

1.5 Bonus 1 : Modèles contextuels et attention

On considère à présent un modèle avec des modules LSTM afin d'avoir des plongements contextualisés, nous l'avons notamment comparé à un modèle LSTM sans mécanisme d'attention.

L'apprentissage de ce modèle nécessite plus d'epochs que les modèles précédent, nous avons donc inclus du *Early Stopping* pour l'entraîner pendant suffisamment d'epochs sans pour autant overfitter les données d'apprentissage.

Nous obtenons une accuracy de 84.5% en train et 84.3% en validation, contre 82% en train et 81.3% en test pour le modèle LSTM sans attention.



(a) **train**, **validation** (b) **train**, **validation**

FIGURE 8 – Loss et Accuracy en apprentissage et en validation du modèle avec des modules LSTM

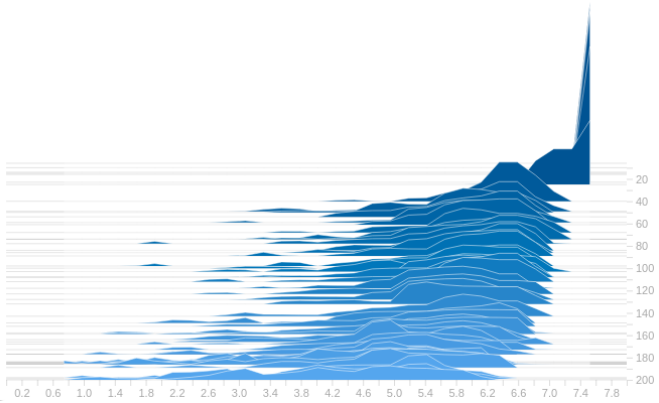


FIGURE 9 – Histogramme de l'entropie des attentions durant l'apprentissage

En combinant un module LSTM avec un mécanisme d'attention, nous obtenons les meilleures performances jusqu'à présent. De plus, étant donné que les LSTMs sont des modèles séquentiels, ils souffrent d'une convergence lente, nous verrons par la suite, dans le TME 10, des architectures plus avancées, à savoir les transformers, où on aura un gain considérable en temps de calcul en introduisant notamment du parallélisme.

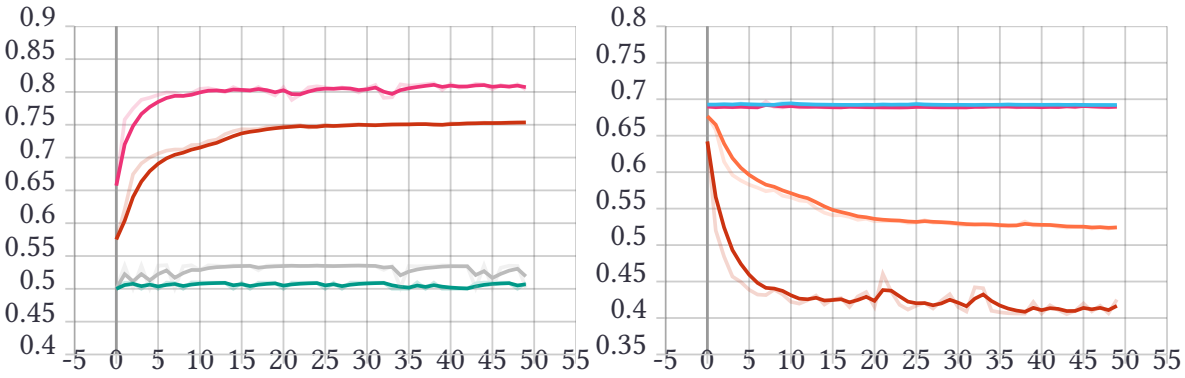
1.6 Bonus 2 : Entropie

Afin d'avoir une attention qui se porte d'avantage sur mots important pour la prédiction, on ajoute à la loss un terme de régularisation qui pénalise les entropies trop élevées.

L = CrossEntropyLoss($\mathbf{y}$, $\hat{\mathbf{y}}$) + $\alpha \times R$

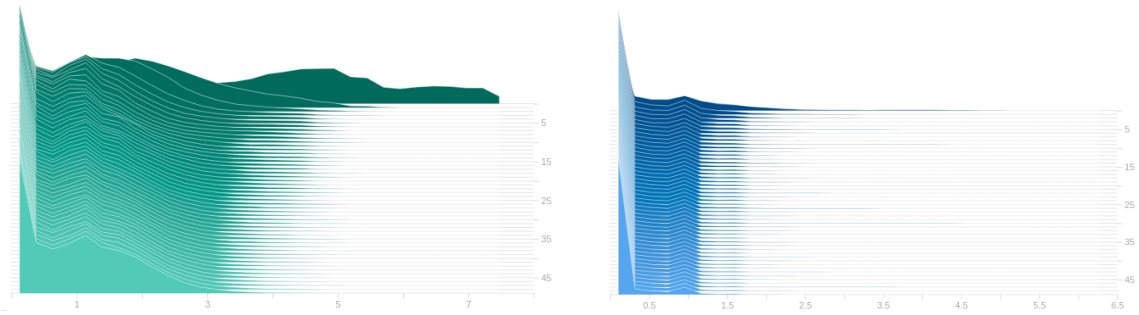
avec R la moyenne des entropies sur un batch.

Nous testons plusieurs valeurs de α : 0.01, 0.05, 0.1 et 1, cela dans le but d'observer l'influence du terme de régularisation sur l'apprentissage.



(a) Accuracy. α=0.01, α=0.05, α=0.1, α=1 (b) Loss. α=0.01, α=0.05, α=0.1, α=1

FIGURE 10 – Loss et Accuracy en validation en faisant varier le α



(a) $\alpha = 0.05$ (b) $\alpha = 1$

FIGURE 11 – Histogramme de l'entropie des attentions durant l'apprentissage

On peut observer que lorsqu'on pénalise d'avantage les entropies, les performances du modèle diminuent considérablement, en effet, après avoir parcouru quelques exemples du dataset, nous avons remarquons qu'il peut y avoir des phrases où il peut exister plusieurs mots pertinents pour la prédiction, surtout que nous avons considéré des séquences assez longues (200 mots) et par conséquent, cette pénalité n'a pas l'effet que l'on souhaite sur ce type d'exemples. En particulier, pour $\alpha > 0.05$, le modèle n'apprend rien et l'accuracy stagne à 50%.

2 TME 10 - Self-attention & Transformers

2.1 Introduction

Dans la partie précédente, nous avons exploré les mécanismes d'attention permettant d'attribuer un poids d'importance, par rapport à une tâche considérée, aux éléments d'une séquence. Le modèle le plus performant conjuguait ces mécanismes avec une architecture récurrente utilisant des modules LSTM. Pour palier au problème de parallélisation hérité des architectures récurrentes, nous allons nous intéresser dans cette partie à une nouvelle architecture complètement parallélisable, qui repose entièrement sur les

mécanismes d'attention. Cette architecture appelée **Transformer**, permet une réduction considérable du temps d'entraînement tout en améliorant les performances.

2.2 Modèle de base :

2.2.1 Module d'attention propre :

L'opération d'attention propre (self-attention ou intra-attention) est une opération séquence à séquence. Elle prend en entrée une séquence de taille n $\{x_1, .., x_n\}$ et de dimension k (la dimension de l'espace de représentation "embedding"), et renvoie une séquence contextualisée de même taille $\{y_1, ..., y_n\}$ où y_i est la iémé entrée contextualisée de même dimension k. y_i est simplement obtenue à partir d'une somme pondérée des entrée $\{x_1, .., x_n\}$:

$$y_i = \sum_{j=1}^{n} p(a_{ij}) \times x_j$$

avec :

$$\sum_{j=1}^{n} p(a_{ij}) = 1$$

$p(a_{ij})$ est obtenu grâce à une fonction sur x_i et x_j (pour calculer y_i on mesure la similarité de x_i avec toutes les autres entrées x_j, permettant ainsi de *contextualiser* la représentation de x_i). On note $p(a_{ij}) = W_{ij}$. W sera donc la matrice de poids avec W_i les poids sur les entrées $\{x_1, .., x_n\}$ pour calculer la iéme entrée contextualisée y_i. On remarque que dans ce mécanisme d'attention, chaque x_i joue trois rôles que l'on va *decorréler* et *apprendre* :

- Pour calculer la iéme entrée contextualisée y_i, on multiplie chaque x_j avec son poids respectif W_{ij}. C'est le rôle "value", qui va être appris grâce à la fonction value $v_\theta(x_j)$ renvoyant un résultat de même dimension que x_j.
- Afin de calculer les poids W_i de la sortie y_i, on mesure la compatibilité de x_i avec toutes les autres entrées x_j. C'est le rôle "query", qui va être appris grâce à la fonction query $q_\theta(x_i)$ renvoyant un résultat de même dimension que x_i.
- Afin de calculer les poids W_j de la sortie y_j, on mesure la compatibilité de x_j avec toutes les autres entrées x_i. C'est le rôle "key", qui va être appris grâce à la fonction key $k_\theta(x_i)$ renvoyant un résultat de même dimension que x_i.

En notant $y_i = f_\theta(x_i; x_1, ..., x_n)$, on obtient donc que :

$$f_\theta(x_i; x_1, ..., x_n) = \sum_{j=1}^{n} W_{ij} \times v_\theta(x_j)$$

On normalise les poids avec un softmax :

$$W_{ij} = p(a_{ij}) = \frac{\frac{1}{\sqrt{k}} \times q_\theta(x_i)^T k_\theta(x_j)}{\sum_{j=1}^{n} \frac{1}{\sqrt{k}} \times q_\theta(x_i)^T k_\theta(x_j)}$$

Nous mettons à l'échelle le produit scalaire $q_\theta(x_i)^T k_\theta(x_j)$ par $\frac{1}{\sqrt{k}}$. Car la magnitude de ce dernier augmente avec la dimension des embeddings k. Pour de grandes valeurs de k, le gradient de la fonction softmax sera très petit ce qui peut entraver l'apprentissage. Cette opération permet donc d'obtenir des gradients de même magnitude quelque soit la dimension k.

Dans notre implémentions, les fonction v_θ, v_θ et q_θ sont des transformations linéaires des entrées. L'architecture du module d'attention propre est illustrée sur le schéma suivant :

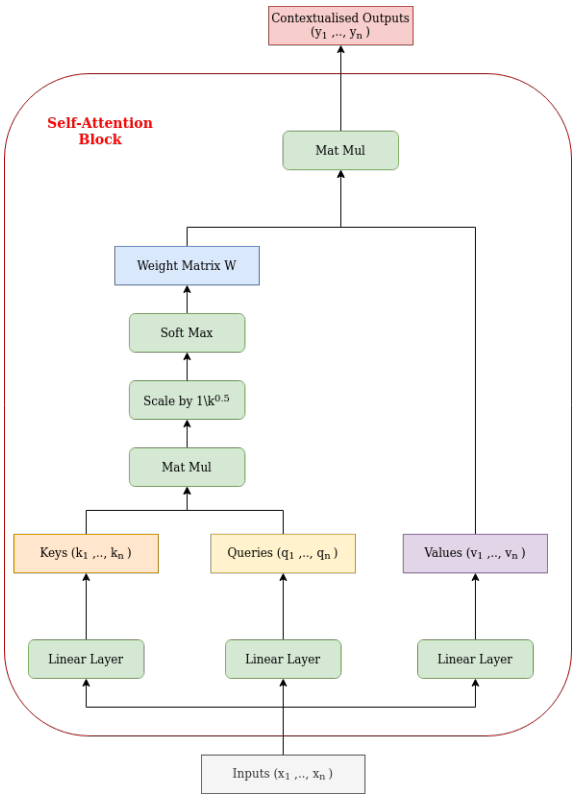


FIGURE 12 – Module d'attention propre f_θ - Le module prend en entrée la séquence d'embeddings $\{x_1, ..., x_n\}$. Les keys $\{k_1, ..., k_n\}$, les queries $\{q_1, ..., q_n\}$, et les values $\{v_1, ..., v_n\}$ sont tous des transformations linéaires de $\{x_1, ..., x_n\}$. La matrice de poids $(W_{ij})_{1\leq i,j\leq n}$ est obtenue à partir d'une multiplication matricielle des keys et des values, suivie par une mise à l'échelle par $\frac{1}{\sqrt{k}}$. Le résultat de ces opérations est passé à une fonction softmax qui permet la normalisation des poids. On pondère les values avec cette matrice de poids pour obtenir les sorties contextualisées $\{y_1, ..., y_n\}$.

2.2.2 Module Transformer :

Le module transformer est composé de deux couches consécutives. On emploi des connections résiduelles autour de chaque couche, suivi d'une normalisation (Layer Norm) :

- Une couche d'attention propre f_θ, qui calcule les embeddings de séquences contextualisées. Cette couche est suivie par une connexion résiduelle et une Layer norm.
- Un perceptron multicouches suivie par une connexion résiduelle et d'une Layer norm : g_θ qui transforme le résultat de l'attention propre.

Nous employons aussi du droupout pour éviter le sur-apprentissage après chaque couche. L'architecture du module transformer que nous considérons est illustrée sur le schéma suivant :

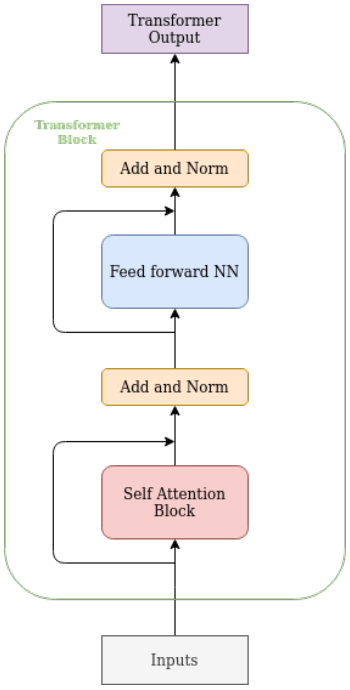


FIGURE 13 – Module Transformer

2.2.3 Module de classification de base :

L'architecture de base consiste en plusieurs block transformer comme defini dans la section 2.2.2 empilés, le nombre de bloques est donnée par L. Pour effectuer la classification des séquences, on utilise du "global average pooling" où on récupère, pour chaque séquence, la moyenne de ses embeddings que l'on projette linéairement vers le nombre de classes du problème (2 dans notre cas). Cette opération est suivie par un softmax pour récupérer une distribution de probabilité sur les différentes classes possibles. L'architecture de ce module est illustrées sur le schéma suivant :

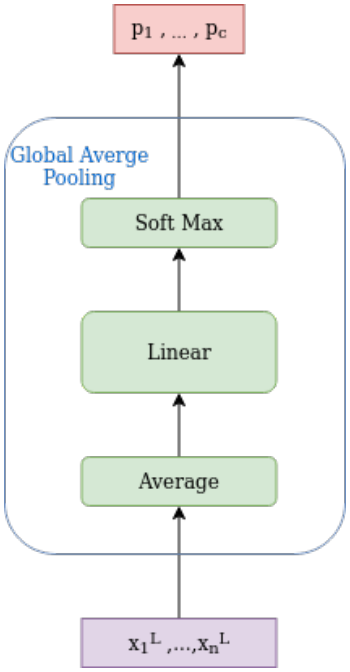


FIGURE 14 – Module "Global Average Pooling"

Le modèle de classification de base complet est donc :

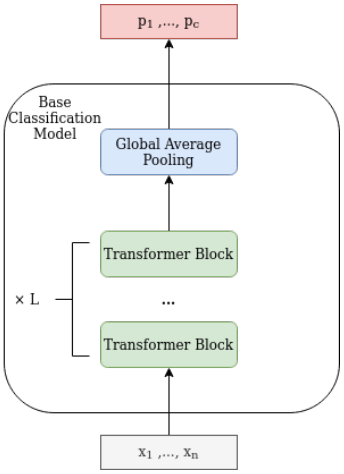


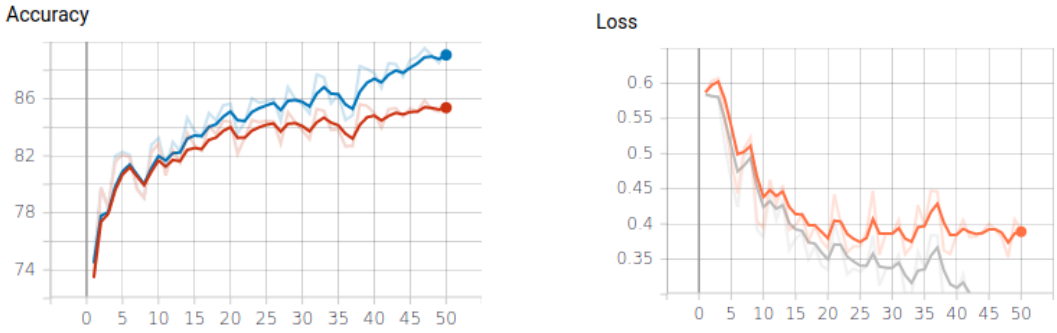
FIGURE 15 – Modèle de classification de base

2.2.4 Expérimentations :

On teste notre modèle sur la tâche de classification de sentiment proposée. On utilise des embeddings glove et des séquences de taille fixée. Les hyper-paramètres sont les suivants :

- Optimiseur : SGD.
- lr : 0.01.
- batch size : 16.
- dimension de l'espace des embeddings : 100.
- taille de séquences max : 500.

— nombre de couches transformer : $L = 3$.
— couche latente du MLP (activation relu) : 5×100.
— droupout : 0.2.



(a) **train**, **validation** (b) train, validation

FIGURE 16 – -Modèle de base avec- Loss et Accuracy en apprentissage et en validation. Meilleure accuracy atteinte sur le validation set : 85.26.

Le modèle atteint une accuracy maximale de 84.92 sur le validation set. On observe une forte variance, ceci est notamment du à l'optimiseur SGD qui a un learning rate 0.01 relativement élevé par rapport à la taille des batchs 16. Ceci est aussi du au droupout utilisé qui est fixé à 0.2, qui augmente la variance sur certaines couches. On a fixé le dropout à 0.2 car on a observé un sur-apprentissage avec un dropout à 0.1 et pour plus d'époques d'apprentissage. Le modèle a atteint une accuracy de 85.82 comme le montre la figure suivante :

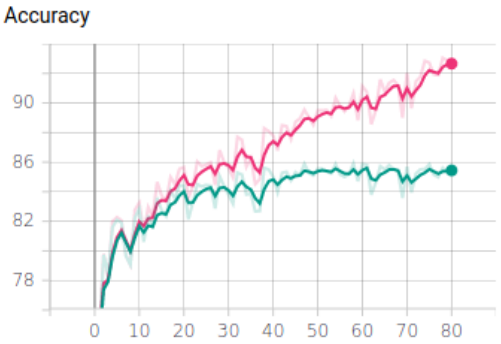


FIGURE 17 – -Modèle de base avec- Accuracy en apprentissage et en validation. Meilleure accuracy atteinte sur le validation set : 85.82.

Comparons les résultats avec plusieurs valeurs de L :

On note une légére amélioration pour des valeurs de L plus grandes. Cependant les modéles restent assez comparables. On peut dire que pour le dataset considéré (relativement petit). Un transformer stack de $L = 3$ est suffisant, augmenter la complexité du modèle au delà ne permet pas une amélioration notable des performances.

TABLEAU 2 – Comparaison du modèle de base avec différentes valeurs de L

L	PosEnc	$\text{CLS}_{\text{Embedding}}$	Sequence Length	Embedding size	$\text{Accuracy}_{\text{validation}}$
$L = 3$	No	No	500	100	85.82
$L = 4$	No	No	500	100	86.22
$L = 6$	No	No	500	100	86.30

2.3 Multi-head attention :

Comme dans le papier original (Vaswani et al., 2017), on se propose d'explorer les résultats qu'on peut obtenir avec plusieurs têtes d'attention (*wide attention*). Chaque tête d'attention h, apprend indépendamment des autres têtes, une fonction value v_θ^h, une fonction key k_θ^h et une fonction query q_θ^h. Chaque tête h calcule ensuite une représentation contextualisée des entrées. Ces représentations sont enfin concaténées et projetées linéairement vers la dimension de l'espace de représentation ($n\_heads * k \times k$). Le résultat de cette projection est le résultat de l'attention multi-têtes.

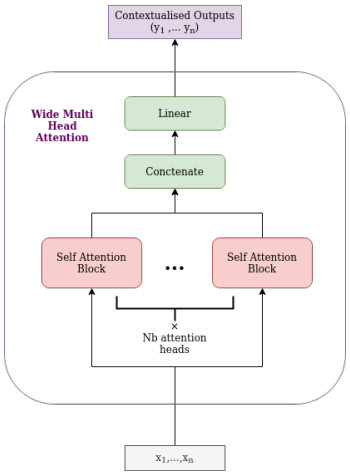


FIGURE 18 – Module Multi-head Attention

2.3.1 Expérimentions :

On teste les performances de l'attention multi-têtes en fonction de plusieurs valeurs de nb_{heads}. On garde les mêmes hyper-paramètres que précédemment :

TABLEAU 3 – Comparaison du modèle de base avec différentes valeurs du nombre de têtes d'attention

L	nb_attention_heads	PosEnc	$\text{CLS}_{\text{Embedding}}$	Sequence Length	Embedding size	$\text{Accuracy}_{\text{validation}}$
$L = 3$	4	No	No	500	100	83.45
$L = 3$	8	No	No	500	100	84.02

On ne note pas d'amélioration par rapport au modèle de base. Le modèle avec plusieurs têtes d'attention est plus complexe et n'est pas forcement adapté à la tâche que nous considérons vu la taille relativement réduite des données en apprentissage dont nous disposons.

2.4 Ajout des encodages de positions :

L'architecture précédente souffre d'un problème important : il n'y a pas de notion de position dans la séquence. Les séquences d'embeddings sont données au modèle sans ordre contrairement aux architectures récurrentes comme le modèle étudié dans la section 1.5, où les mots sont passées l'un après l'autre. Pour régler ce problème on rajoute un signal qui aide le modèle à incorporer la notion de position de la manière suivante (**sinusoidal positional encoding**) :
Soit le mot à la position i dans la séquence et k la dimension de l'espace des embeddings. On calcule pour chaque position i, des prolongement de position pe_{ih} :

$$pe_{ih} = \begin{cases} \sin \frac{i}{1000^{\frac{h}{k}}} & \text{si } h \text{ est impair} \\ \sin \frac{i}{1000^{\frac{h-1}{k}}} & \text{sinon} \end{cases}$$

Affichons les encodages de positions correspondants à une séquence de taille 10 avec un espace d'embeddings de dimension 20 :

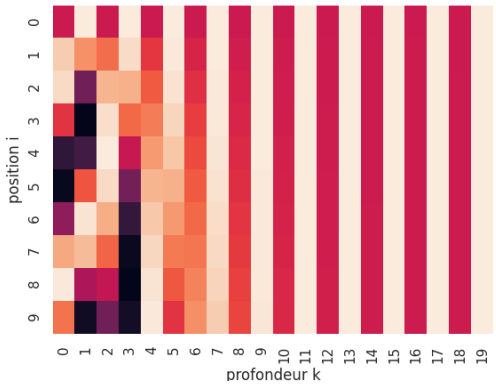
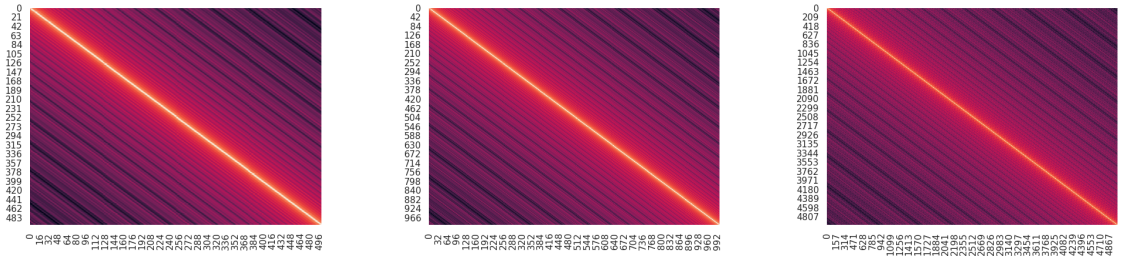


FIGURE 19 – -sinusoidal positional encoding pour une séquence de taille 10 avec un espace d'embeddings de dimension 20- Chaque ligne correspond à un pe_i.

On remarque que l'encodage de position est attribué de façon unique à chaque mot dans la séquence.

Pour comprendre mieux comment fonctionne cet encodage de positions, on affiche la heatmap de produits scalaire entre p_i et p_j pour $i, j \in \{1..n\}$ et ce pour différentes tailles de séquences n :



(a) $n = 500$ (b) $n = 1000$ (c) $n = 5000$

FIGURE 20 – Produit scalaire entre chaque pe_i et pe_j pour des séquences de tailles variables.

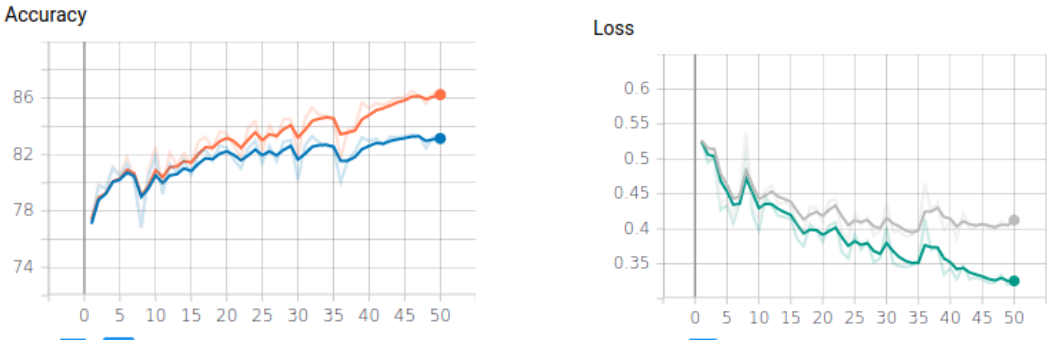
On remarque que la distance (similarité) entre les mots voisins est symétrique (x_i est à distance égale de x_{i+1} et de x_{i-1}. On remarque aussi que cette distance (resp. similarité) croit (resp. décroît) plus on s'éloigne du mot x_i dans la séquence. Il est a noter que plus la séquence est grande mois bons sont les encodages de positions, puisque plus on s'éloigne d'un mot plus la similarité décroît. Au delà d'un certain seuil, cette quantité devient négligeable et ne peut pas correctement représenter les positions des mots.

Incorporation dans le modèle : Les encodages de positions ne font pas partie du modèle, on équipe chaque mot d'informations sur sa position dans la séquence en passant au modèle la séquence des embeddings qu'on somme à ses encodages de positions $\{x_i + pe_i\}_{1 \leq i \leq n}$. De plus, les connections résiduelles dans l'architecture transformer, permettent aux informations sur la position de se propager vers les couches plus profondes.

2.4.1 Expérimentations :

On utilise les mêmes hyper-paramètres que précédemment :

- Optimiseur : SGD.
- lr : 0.01.
- batch size : 16.
- dimension de l'espace des embeddings : 100.
- taille de séquences max : 500.
- nombre de couches transformer : $L = 3$.
- couche latente du MLP (activation relu) : 5×100.
- droupout : 0.2.



(a) train, validation (b) train, validation

FIGURE 21 – - Positional Encodings avec Global Average Pooling- Loss et Accuracy en apprentissage et en validation. Meilleure accuracy atteinte sur le validation set : 83.42.

On ne note pas d'amélioration par rapport au modèle de base. Cependant, il est difficile de juger de l'efficacité des encodages de positions avec un dataset relativement petit comme le notre. Pour des taches réelles avec des datasets conséquents, l'amélioration est certaine, comme il est documenté dans différentes expériences.

2.5 Ajout d'un embedding CLS :

Jusqu'à présent, la classification était faite à l'aide du global average pooling. Dans cette partie, on va utiliser un pseudo token CLS à cet effet. On rajoute au début de chaque séquence un mot vide "CLS", ce mot est le même pour toutes les séquences. L'embedding de ce mot est appris, et on utilise sa représentation contextualisée de la dernière couche pour classifier la séquence. La représentation contextualisée de la dernière couche pour chaque séquence capture toute l'information nécessaire pour la classification.

On implémente la classification avec l'embedding CLS de la manière suivante :

- On genre un tensor aléatoire de même dimension que l'espace des embeddings k (généré une fois seulement). Il représente l'embedding du mot vide CLS.
- Cet embedding est appris en le passant par une couche linéaire ($k \times k$). On rajoute la sortie de la couche linéaire au début de chaque séquence en entrée.
- Les séquences sont ensuite passées dans l'architecture (la pile de transformers) en rajoutant les encodages de position.
- On récupère la représentation contextualisée de la dernière couche de ce token. On projette cette dernière linéairement vers le nombre de classes. On utilise ensuite un softmax pour générer une distribution de probabilité sur les classes possibles.

L'architecture du modèle est illustrée sur le schéma suivant :

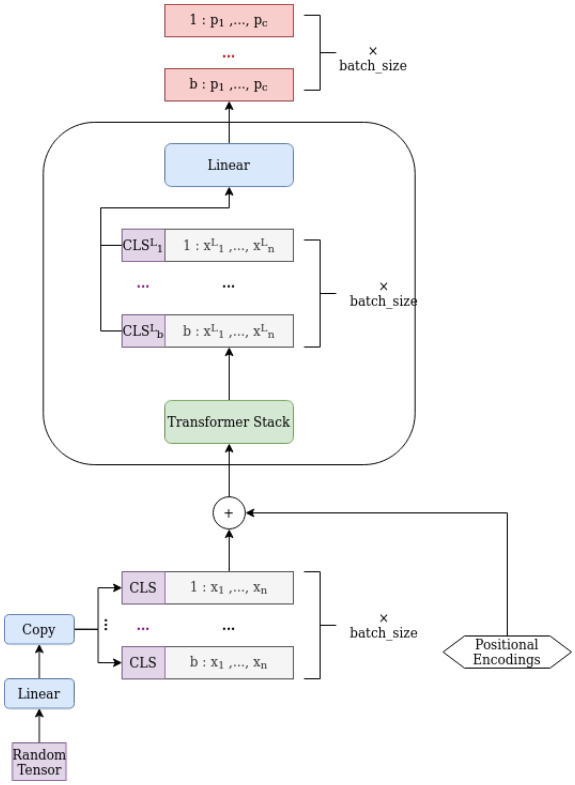


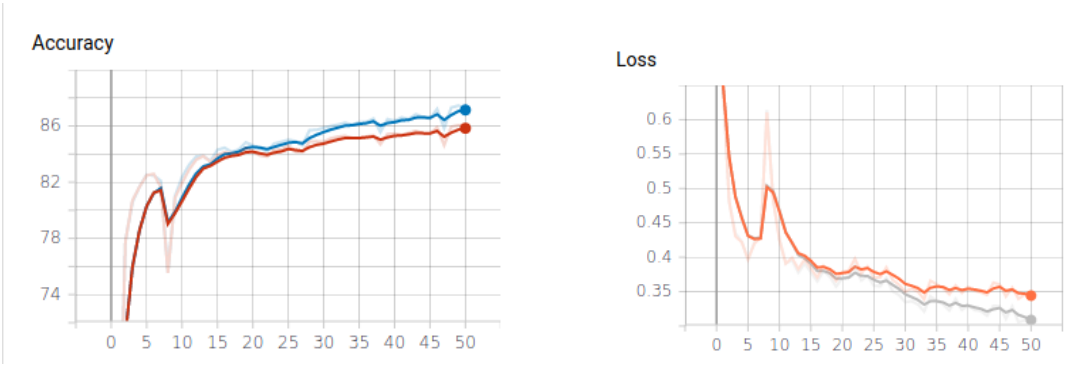
FIGURE 22 – Classification avec un embedding CLS

2.5.1 Expérimentations :

On utilise les mêmes hyper-paramètres que précédemment :

- Optimiseur : SGD.
- lr : 0.01.
- batch size : 16.
- dimension de l'espace des embeddings : 100.
- taille de séquences max : 500.
- nombre de couches transformer : $L = 3$.
- couche latente du MLP (activation relu) : 5×100.
- droupout : 0.2.

Dans un premier temps on va expérimenter cette méthode sans encodages de positions, pour observer sa performance face au modèle de base :

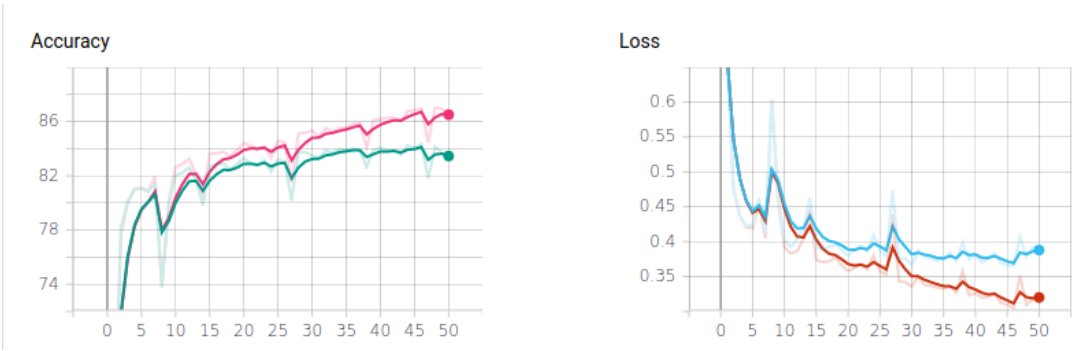


(a) **train**, **validation** (b) **train**, **validation**

FIGURE 23 – -Embeddings cls sans positional encoding- Loss et Accuracy en apprentissage et en validation du modèle avec $L = 3$. Meilleure Accuracy atteinte : 86.06.

On note une amélioration par rapport au modèle de base. On obtient une accuracy de 86.06 sur le set de validation contre 85.26. On a testé ce modèle sur plus d'époques et pour un droupout=0.1, on obtient une accuracy maximale de 86.98.

Maintenant observons ce que donnes la combinaison postional encoding + pseudo token CLS pour la classification :



(a) **train**, **validation** (b) **train**, **validation**

FIGURE 24 – -Embeddings cls avec positional encoding- Loss et Accuracy en apprentissage et en validation du modèle avec $L = 3$. Meilleure Accuracy atteinte : 84.32.

Là encore une fois on ne note pas d'amélioration par rapport au modèle de base. Cependant on observe une amélioration quand à l'utilisation du pseudo token CLS pour la classification au lieu du global average pooling 84.32 vs 83.42.

Références

Bahdanau, D., Cho, K. & Bengio, Y. (2016). *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv : 1409.0473 [cs.CL].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention Is All You Need. In : *CoRR* abs/1706.03762. arXiv : 1706.03762.