AMAL

Transformers, Attention et Self-Attention

[redacted]
[redacted]

Master 2 DAC
2020-2021

Table des matières

Le travail dépend de la base IMDB qui contient des critiques de films. Le but est d'associer une classe (positif, négatif) à ces critiques. Pour cela, le mieux est de prendre en compte les sentiments exprimés dans les phrases isolées ; or le sens est quelque chose de diffus, comment peut-on savoir où regarder, à quels mots les associer ?

Ce problème est résolu grâce aux mécanismes d'attention.

Ils permettent à un modèle d'apprendre à diriger son intérêt vers des points pertinents en même temps qu'il s'assure de la qualité des classifications.

1 TME 9 - Attention globale

1.1 Modèle baseline

La baseline pour ce TME est illustrée dans la Figure 1.

Elle n'inclut aucune notion de séquentialité comme on aurait pu l'attendre en traitement de texte. Comme entrées, le modèle considère en fait les critiques entières, composées de plusieurs vecteurs qui sont les *embeddings* de leurs mots ; et la classification se fera simplement sur la moyenne de ces *embeddings*.

$$\hat{t} = \frac{1}{n} \sum x_i$$

Si attention il y a, elle est ici répartie de façon **uniforme entre les mots**. On considère que tous ont la même importance pour la prédiction associée à une critique.

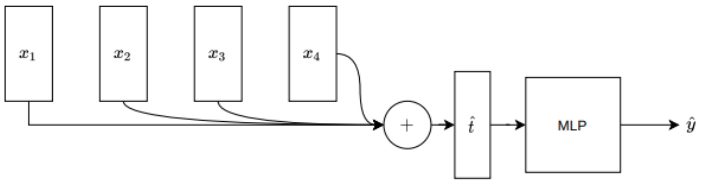


Figure 1 – Modèle naïf

Après 42 epochs, les loss train et test sont confondues, il n'y a pas de surapprentissage ; les scores sont déjà satisfaisants mais perfectibles (75% d'accuracy finale).

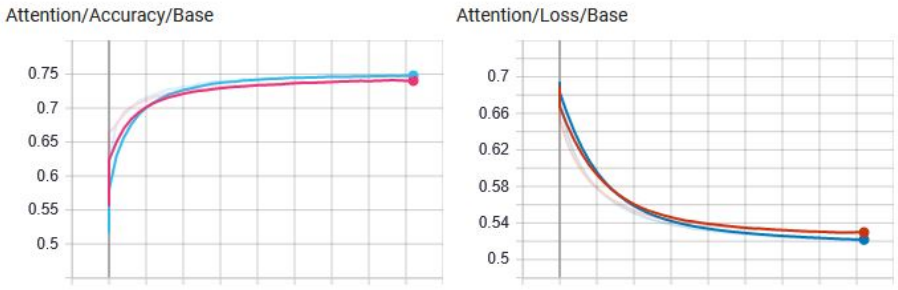


Figure 2 – Résultats du modèle naïf

Les résultats de ce modèle naïf serviront de référence pour comprendre l'intérêt des vrais mécanismes d'attention développés ensuite.

1.2 Modèle d'attention simple

Un mécanisme d'attention se rend utile en quantifiant l'importance des mots d'une critique selon une distribution de probabilité. On souhaite réaliser **une représentation pondérée du texte** en identifiant et en donnant plus de poids aux mots pertinents pour la tâche de classification.

Pour ce faire, on a l'idée d'**apprendre un vecteur *query* noté q, unique pour toutes les critiques** : on peut interpréter ce vecteur comme une question globale sur la base, qui nous aiderait à contextualiser chaque critique, et dont la réponse apporterait une information additionnelle pour aider à la classification.

Formellement, la représentation pondérée d'une critique s'exprimera par

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

où $p(a_i|t)$ est la probabilité de porter l'attention sur le i^{eme} mot :

$$\log p(a_i|t) = C + q \cdot x_i$$

Et comme prévu, elle dépend de la query q. L'expression des $p(a_i|t)$ se simplifie en passant à l'exponentielle (ce qui conserve l'ordre). On constate que

$$p(a_i|t) = e^{C+q\cdot x_i} = e^C \times e^{q\cdot x_i}$$

et en posant $C = -\sum_i e^{q\cdot x_i}$ on a :

$$p(a_i|t) = softmax(q \cdot x_i)$$

ce qui permet de simplifier l'implémentation. On réalise cette implémentation avec l'optimiseur Adam, learning rate = 0.001 :

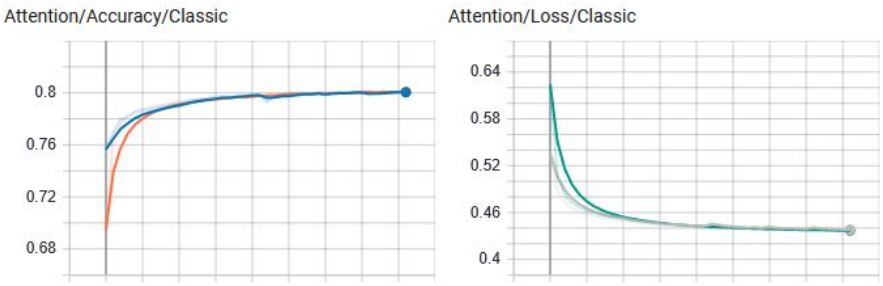


Figure 3 – Résultats du modèle d'attention simple

Les loss avec ce nouveau modèle descendent un peu plus bas qu'avec la baseline. Après le même nombre d'epochs (42), l'accuracy est augmentée de 5%, et comme on n'ajoute qu'un élément global, il n'y a pas là non plus de surapprentissage. C'est donc préférable d'ajouter un mécanisme d'attention, même basique.

Entropies des distributions d'attention

Il est intéressant d'observer l'évolution des entropies sur les attentions au cours de l'apprentissage. On affiche pour cela un histogramme de leurs valeurs pour un batch à chaque epoch en considérant que cette approche stochastique est suffisamment représentative.

Normalement, **la distribution des importances des mots ne doit pas être uniforme** et on s'attend donc à ce que la valeur moyenne des entropies baisse. Elle doit s'écarter de son initialisation chaotique. Cela dit, **il faut qu'elle reste quand même au-dessus d'un certain seuil** : le modèle devrait prendre en compte plusieurs mots dans une phrase pour bien capturer son essence, et non converger vers un Dirac sur un seul mot. Voir le schéma en 1.5 pour une représentation de cet objectif.

Ici, globalement, le mode des entropies s'abaisse par rapport au début de l'apprentissage. Mais sur certains exemples, elles restent élevées (partie droite de l'histogramme). Après vérification, on a remarqué que certaines critiques ne semblaient pas exprimer de sentiment particulier, d'où le fait que le mécanisme d'attention ne soit pas bien ciblé : l'importance des mots a pu légitimement rester à peu près uniforme.

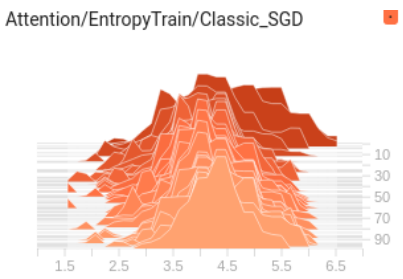


Figure 4 – Entropies sur le modèle d'attention simple

Visualisation des attentions données

Sur les données de test, on visualise l'importance des mots. On constate que **le modèle d'attention n'est pas toujours fiable pour un spectateur humain** : pour décider que cette critique est négative, le seul mot qui parait pertinent dans la liste est "cheap". L'attention est sinon basée sur des mots communs. On note qu'elle aurait pu être focalisée sur des expressions plus intéressantes comme "very tiring" par exemple.

```
The example is  ultra __OOV__ __OOV__ vinnie tries really __OOV__ very tiring script that flashes between past and
__OOV__ cheap __OOV__ saw the boom __OOV__ __OOV__ __OOV__ ultra __OOV__ __OOV__ vinnie tries really __OOV__ very
tiring script that flashes between past and __OOV__ cheap __OOV__ saw the boom __OOV__ __OOV__ __OOV__ ultra __OOV
__ __OOV__ vinnie tries really __OOV__ very tiring script that flashes between past and __OOV__ cheap __OOV__ saw
the boom __OOV__ __OOV__ __OOV__ ultra __OOV__ __OOV__ vinnie tries really __OOV__ very tiring script that flashes
between past and __OOV__ cheap __OOV__ saw the boom __OOV__ __OOV__ __OOV__
********************************
The ten words with the most attention are : ['__OOV__', 'and', 'past', '__OOV__', 'that', 'past', 'between', 'chea
p', '__OOV__', 'that']
```

Figure 5 – Visualisation des mots les plus porteurs d'attention

Cet exemple est l'un des meilleurs sur lesquels on soit tombé. A noter que les visualisations ne deviennent pas beaucoup plus pertinentes avec les améliorations du modèle développées ci-après. On peut potentiellement accuser la taille de la base qui n'a pas l'échelle nécessaire pour entraîner un véritable modèle d'attention (cf. BERT).

1.3 Modèle d'attention complet : questions et valeurs

Plutôt que d'apprendre un vecteur q pour toutes les critiques, il est plus pertinent de le faire dépendre de chacune. En partant de la moyenne des *embeddings* pour chaque critique i, on apprend un $q(t)$ pour elle ; intuitivement, cela revient à se dire que sa classe ne dépend pas uniquement d'un vocabulaire général de commentaires positifs - resp. négatifs, mais surtout d'attributs qui lui sont inhérents ; par exemple, selon son genre, on l'associera à un mode d'expression spécifique. Cette fois,

$$\log p(a_i|t) = C + q(t) \cdot x_i$$

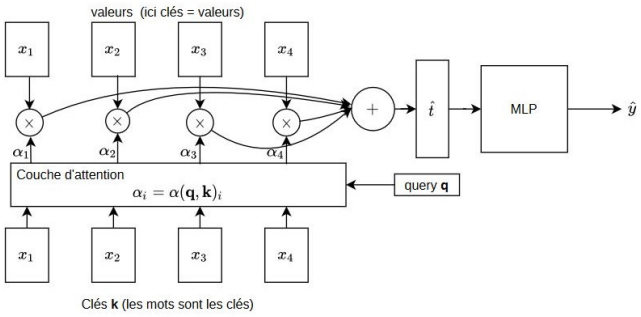


Figure 6 – Schéma du modèle d'attention complet

Le schéma illustratif cite des "valeurs" et des "clefs" qui n'auront pas d'intérêt particulier ici pour améliorer les résultats. Cela dit, l'implémentation va servir à introduire des concepts qui seront précieux pour la self-attention au TME suivant. On commente.

Les k (***keys***, clefs) représentent théoriquement des attributs appris, extraits de chaque entrée, sur lesquels on se fondera pour répondre à la *query* q déjà introduite. L'apprentissage des scores d'attention α par mot dépendra de ces k.

Les v (***values***, valeurs) sont d'abord associées aux entrées indépendamment de q. Une fois reliées aux scores d'attention trouvés, elles aideraient à y répondre : c'est grâce à une somme des v pondérée par les α qu'on aurait une représentation classifiable de toute la phrase.

Clefs et valeurs sont généralement apprises grâce à des transformations linéaires des mots des entrées, mais ici, les deux correspondent tout simplement aux *embeddings* de ces mots. On commence donc par une version simple et on approfondira ces notions plus loin.

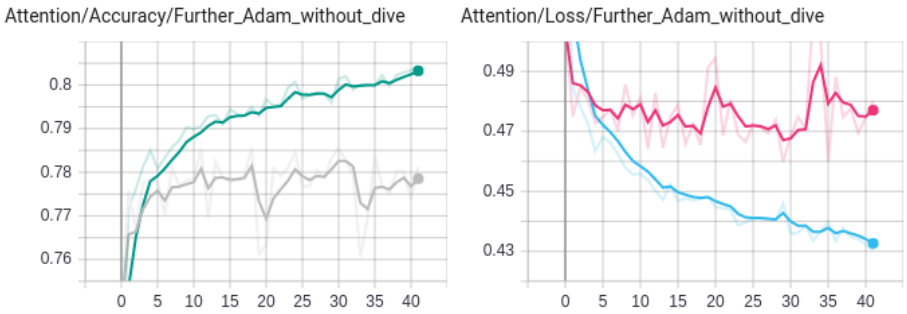


Figure 7 – Résultats du modèle d'attention complet

Sur les performances en train, on remarque qu'il n'y a pas ou peu de changement par rapport à la version précédente - car les clefs et valeurs choisies ne sont pas plus

informatives que le mot seul, la sortie $v * \alpha$ notamment correspond juste à une pondération directe des *embeddings* par les scores d'attention.

Cependant, le surapprentissage est fort (la courbe de la loss en test prend vite du retard sur celle du train). Cela se justifie par la prise en compte détriplée de chaque *embedding* de mot, de façon exacte, sans aucune extraction de *features* qui pourrait mener à une meilleure généralisation.

Amélioration par plongements de valeurs

On tente alors de **changer la valeur d'un mot** (qui jusqu'ici correspondait juste à son *embedding*) **en l'apprenant aussi** avec un réseau de neurones :

$$\hat{t} = \sum p(a_i|t)v(x_i)$$

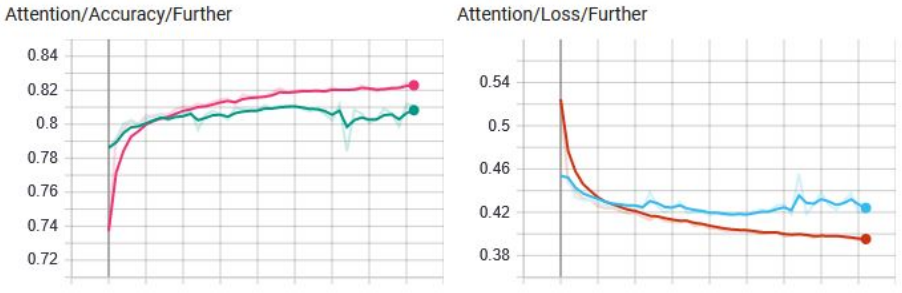


Figure 8 – Résultats du modèle d'attention complet avec plongements de valeurs

On s'en doute, le surapprentissage est désormais moins grave, car on ajoute une extraction d'information qui peut aider à comprendre la dynamique du texte - au lieu de le considérer directement, en l'état, avec toutes ses spécificités. La loss s'affaiblit un peu en train comme en test et mène à de meilleurs résultats en termes d'*accuracy*.

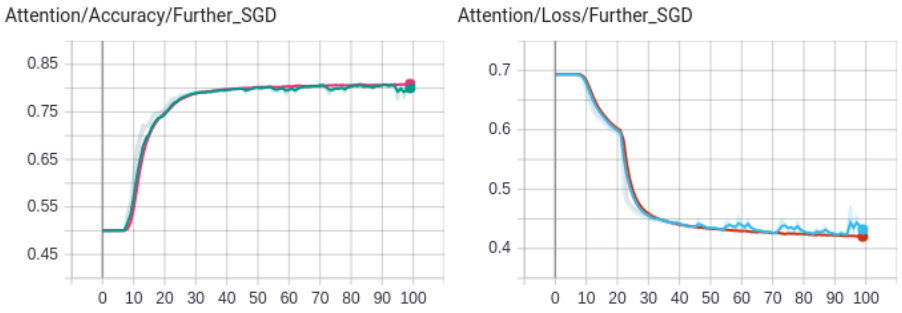


Figure 9 – Résultats du modèle d'attention complet avec plongements, SGD

Toujours en conservant ces plongements de valeurs, notons que l'optimiseur SGD permet d'annihiler totalement le surapprentissage. Cela dit, son usage pénalise légèrement les scores finaux qui ne montent plus aussi haut qu'avant malgré un nombre d'epochs accru. La convergence n'est cependant pas terminée après 100 epochs, et il se peut aussi que raffiner le learning rate aide à aller plus loin.

Entropies des distributions d'attention

En comparant avec la page 4, on remarque là encore une baisse des entropies moyennes au cours de l'apprentissage. Selon l'interprétation déjà donnée, cela veut dire que **l'attention perd sa pseudo-uniformité initiale et se met bien en place avec des focalisations sur des éléments** particuliers. D'ailleurs ici, le dernier histogramme est plus plat que pour la version précédente, et il est plus massif sur sa gauche : il y a donc plus d'exemples à entropie faible qu'avant.

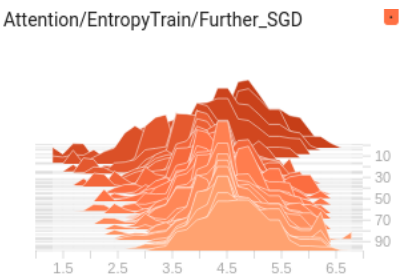


Figure 10 – Entropies obtenues avec questions et valeurs

Complémenter les questions par des clefs et valeurs a donc l'avantage de favoriser l'effet de focalisation. Toute cette structure sera étudiée plus avant dans le TME 10.

1.4 Ajout d'une LSTM

Depuis les TME précédents, il nous semble logique de traiter des séquences (comme ici) avec un principe de récurrence. Intégrer **une représentation de la suite temporelle des mots** permettrait de comprendre lesquels appellent un intérêt particulier.

En augmentant le système précédent d'une LSTM, on arrive de fait à de meilleurs résultats en entraînement. On constate par contre un grave surapprentissage, car la LSTM intègre des rapports entre les mots qui peuvent dépendre des exemples précis avec lesquels elle a été entraînée. Nous n'avons pas réussi à nous en débarrasser avec l'optimiseur Adam, quels que soient les hyperparamètres ; seul un passage à SGD a permis de le réduire, mais en revenant alors aux mêmes scores que précédemment.

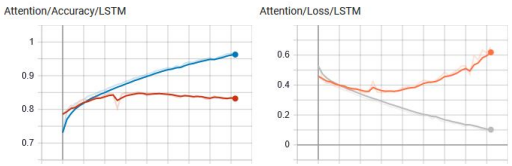


Figure 11 – Résultats du modèle LSTM Adam

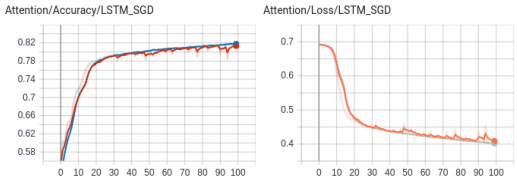


Figure 12 – Résultats du modèle LSTM SGD

Entropies des distributions d'attention

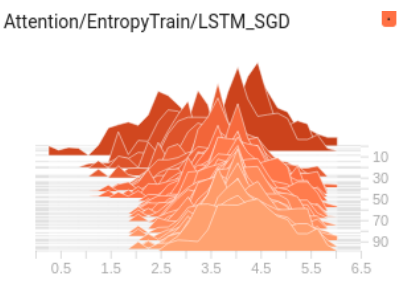


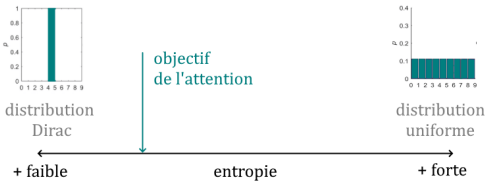
Figure 13 – Entropies avec LSTM

En observant les entropies des attentions, on se rend compte que la LSTM permet d'obtenir des résultats plus proches de ceux qu'on espérait, avec une baisse sensible des entropies moyennes en train comme en test d'epoch en epoch (surtout sur les 30 premières). Le mode plus bas qu'avant nous permet de déduire que la LSTM apprend à se focaliser sur un ensemble beaucoup plus restreint de mots (par exemple parce qu'il en suffit parfois d'un seul pour résumer ceux qui précèdent).

Une piste d'amélioration (mais qui ne peut rien contre le surapprentissage) serait d'utiliser une LSTM bidirectionnelle pour aussi prendre en compte les successeurs d'un mot dans le calcul de son score. On réduirait probablement encore plus l'entropie en restreignant encore la liste des mots d'intérêt : si un mot sert de résumé à des éléments qui le précèdent et le suivent, on peut s'attarder sur un petit nombre seulement.

1.5 Pénalisation entropique

L'illustration ci-dessous explicite exactement ce qu'on recherche au niveau des entropies et justifie notre interprétation des graphiques précédents.



Plus haut, les entropies n'ont jamais été aussi basses que théoriquement prévu. On propose donc de pénaliser par leurs valeurs pour les affaiblir, et faire en sorte que l'attention ne reste pas uniforme et se porte bien sur un minimum de mots.

La loss du modèle devient alors $L_{reg} = L + \lambda E$, à minimiser.

La pénalisation devait rester suffisamment légère pour respecter les ordres de grandeurs des loss et des entropies ; la seule constante de régularisation trouvée qui ne fasse pas exploser les loss (entraînant une divergence du modèle) était λ=0.001, et plus basse, elle n'avait plus d'effet.

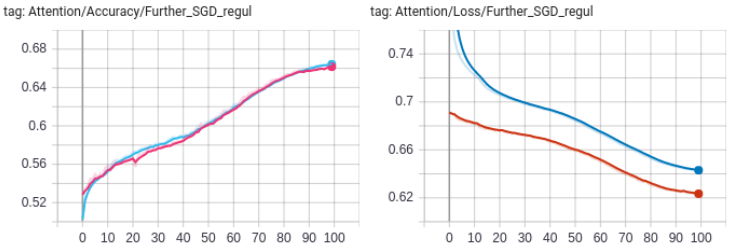


Figure 14 – Résultats en pénalisant les entropies des attentions

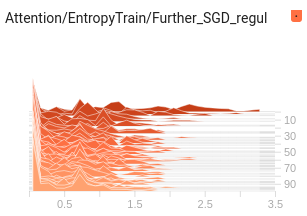


Figure 15 – Effet sur les valeurs des entropies

On se rend néanmoins compte que cette régularisation ne permet pas d'améliorer les performances. De fait, il est souvent bon de s'intéresser quand même à plus d'un mot à la fois, idée que cette forme de loss pénalise. Les entropies ont certes une forme plus proche de celle qu'on désire : elles tendent toutes vers zéro... mais si l'accuracy baisse, cela n'en vaut pas la peine.

Remarque sur l'optimiseur choisi

Notons que depuis le début de ce rapport, tous les tests liés à l'entropie ont été obtenus avec l'optimiseur SGD, qui se caractérise par sa stabilité à long terme et sa sensibilité réduite au learning rate. L'optimiseur Adam qu'on avait utilisé par défaut (et dont relèvent les graphiques des *loss* et *accuracies*) converge plus vite à learning rate optimal, mais les entropies affichées n'avaient quasiment jamais le comportement typique - elles avaient plutôt tendance à augmenter avec l'apprentissage.

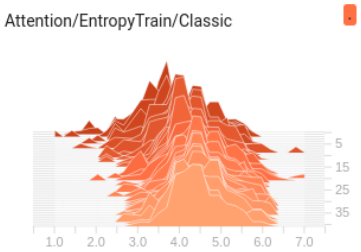


Figure 16 – Exemple d'entropies obtenues avec Adam

1.6 Conclusions

Modèle	Optimiseur	Loss		Accuracy	
		Train	Test	Train	Test
Baseline (uniforme)	Adam	0.522	0.530	74.78	74.03
	SGD	0.555	0.560	72.96	72.11
Attention Simple	Adam	0.436	0.437	80.07	80.04
	SGD	0.446	0.446	79.55	79.54
Attention sans plongemt val.	Adam	0.433	0.477	80.44	78.12
Attention avec plongemt val.	Adam	0.419	0.395	82.29	80.83
	SGD	0.420	0.432	80.99	80.63
Attention avec LSTM	**Adam**	**0.102**	**0.619**	**96.49**	**82.99**
	SGD	0.402	0.409	81.75	81.56
Pénalisation entropique	SGD	0.643	0.623	66.50	66.24

Tableau 1 – Ensemble des résultats sur le TME9

Ces résultats ont tous été obtenus sur 42 epochs pour Adam, et sur 100 pour SGD, ce qui prend de 30 minutes à une heure. Les courbes de loss montraient une convergence apparente qui ne nous encourageait pas à continuer l'entraînement.

Les meilleurs sont ceux qui impliquent également la LSTM pour une prise en compte de la séquentialité dans la critique, avec un score en test d'un peu plus de 82%. Il se trouve qu'il a été établi qu'il n'était pas possible de dépasser 85% d'accuracy avec ce genre d'approche sur un dataset comme le nôtre. La marge n'est pas très grande.

Dans tous les cas, les mots qui attirent le plus l'attention du modèle ne sont pas ceux qu'un humain aurait remarqués : il y voit apparemment des indices qui ne sont pas évidents pour nous. Cela dit, tant qu'ils améliorent la classification, on lui fait confiance.

2 TME 10 - Self-attention

La *self-attention* est le principe de base des *Transformers*. La différence avec l'attention globale est qu'en plus de considérer les attentions sur les mots pris séparement, on inclut à présent dans le score d'attention d'un mot son rapport à tous les autres. Le concept n'est pas lié à la séquentialité de ces mots, qu'on a pu mettre à profit avec la LSTM; il s'agit vraiment - si l'entraînement fonctionne - d'un rapport de sens. Cela passera notamment par un apprentissage un peu plus complexe des q, des v et des k déjà décrits dans le TME précédent.

Les modèles de self-attention sont très gourmands en espace mémoire du fait du très grand nombre de paramètres - comme il est nécessaire de considérer des relations entre tous les mots. La taille des batchs s'en est donc vue réduite (à 16) pour pouvoir exploiter correctement les GPU à notre disposition.

2.1 Principe de l'attention propre

La *self-attention* dépend de l'empilement de plusieurs structures dites "couches d'attention". La première couche d'attention va donc pouvoir proposer un vecteur (les valeurs v des mots multipliées par les scores d'attention α) qui représente chaque critique, et les suivantes pourront reprendre cette représentation pour l'améliorer.

La représentation du i^{eme} élément à la couche l (la couche 0 étant l'entrée) est donnée par la fonction résiduelle :

$$x_i^{(l)} = g(x_i^{(l-1)} + f(\tilde{x}_i^{(l-1)}; \tilde{x}_1^{(l-1)}, ..., \tilde{x}_n^{(l-1)}))$$

où $\tilde{x}^{(l-1)}$ est la version normalisée (BatchNorm) de la sortie de la couche précédente
g est une non-linéarité qui transforme sa représentation pour la suivante
f est une fonction d'attention qui permet d'apprendre les k, q et v pour chaque mot d'entrée x_i : le paramétrage de cet apprentissage est partagé pour toute la couche l.

$$f(x_i^{(l-1)}; x_1^{(l-1)}, ..., x_n^{(l-1)}) = \sum_{j=1}^{n} p(a_{ij}^l) v^{(l)}(\tilde{x}_j^{(l-1)})$$

Chaque couche sera suivie d'un Dropout. On utilise une moyenne sur la sortie de la dernière couche pour pouvoir effectuer une classification.

Application du modèle de base

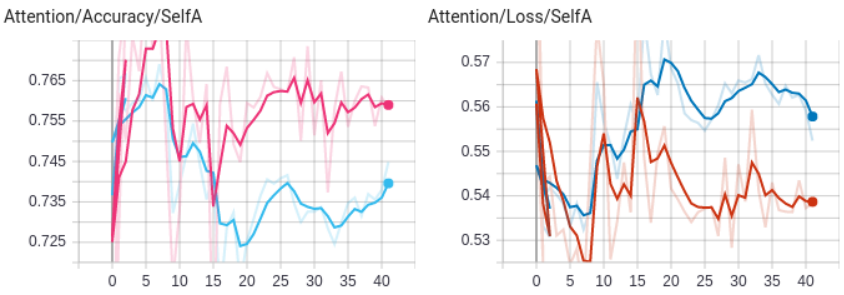


Figure 17 – Performances de base en self-attention

Ici, on retient L = 3 comme demandé pour un empilement de trois couches.

Bien qu'on arrive immédiatement à un résultat prometteur (au-dessus de 75% d'*accuracy*), même un lissage conséquent ne permet pas d'avoir des courbes typiques. L'apprentissage n'a pas l'air de converger vers quoi que ce soit et le résultat n'est pas beaucoup mieux que celui du modèle baseline au TME9 - sans attention - à nombre d'epochs égal. On constate même un décrochage important de l'*accuracy* en test (bleu clair), liée à une évolution chaotique de la loss qui fait quelques embardées et remonte brusquement après 10 epochs. Le même genre de chiffres a été obtenu en remplaçant les BatchNorms par des LayerNorms.

Ayant écarté la possibilité d'une erreur de code (revérifications constantes depuis le jour du TP), il faut noter que nos batchs sont très petits (16 éléments par contrainte mémoire) et que cela rapproche la situation d'un apprentissage stochastique online. Peut-être faut-il alors allonger l'apprentissage pour faire mieux, mais il est trop chronophage et contraignant pour considérer sérieusement des tests supplémentaires.

2.2 Ajout des plongements de position

Les *Positional Encodings* (PE) permettent de prendre en compte l'ordre des mots. Il ne s'agit pas de considérer simplement le contexte immédiat d'un mot (ou ses prédécesseurs pour une LSTM monodirectionnelle) ; on va plutôt agir directement sur son *embedding* et y intégrer une information intelligemment construite concernant sa position dans la phrase. Cette information va se répercuter sur sa valeur apprise ainsi que sur celles des autres mots qui sont en relation avec lui.

Pour le mot de position i, sachant que la taille de l'embedding visé est d, la h^{eme} composante de cet embedding s'écrit

$$pe_{ih} = \begin{cases} sin(i/10000^{h/d}) & \text{si h est pair} \\ cos(i/10000^{(h-1)/d}) & \text{sinon} \end{cases}$$

La matrice ci-contre représente les produits scalaires entre les *embeddings* PE des paires de mots dans une critique. L'ajout des PE permet de faire alterner ces produits scalaires entre des valeurs fortes et faibles - de sorte que lorsqu'on calcule ces produits scalaires pour un mot, on gagne une idée de sa distance à celui auquel il est en train de se comparer. Il pourra ainsi prendre en considération différemment des mots proches ou plus éloignés.

Note : cette critique est courte, donc la partie intéressante de la matrice (la "grille" alternée) est concentrée en haut à gauche. On voit l'intérêt de bien adapter les paramètres des PE à la longueur des critiques.

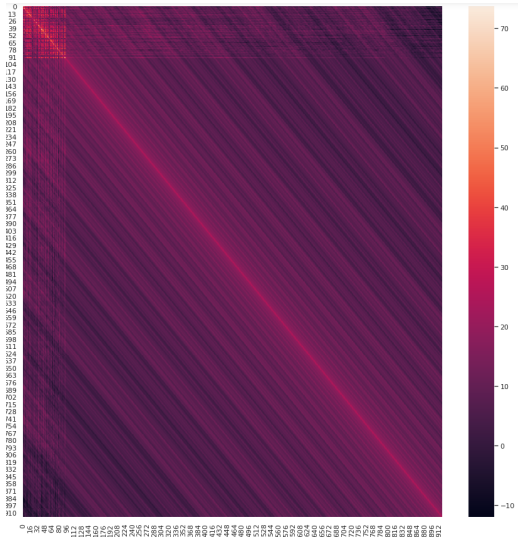


Figure 18 – Visualisation de l'effet des plongements sur les encodings

Visualisation de l'effet

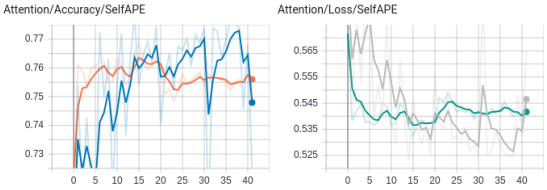


Figure 19 – Positional Encodings Adam

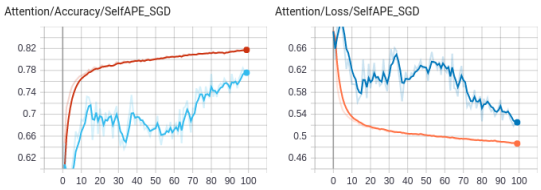


Figure 20 – Positional Encodings SGD

On constate pour les deux optimiseurs que l'apprentissage est plus stable que sans les PE - même si les courbes des loss présentent toujours des pics, les deux continuent globalement à baisser tout du long. Il y a donc une information vitale qui est contenue dans les plongements de position et qui permet de mieux *comprendre* les critiques. Mais - surtout avec SGD - l'usage des PE engendre un surapprentissage évident en favorisant la mémorisation exacte de la suite des changements dans une phrase. Les performances en train sont donc bien plus satisfaisantes qu'avant, mais on regrette que celles en test ne s'améliorent pas autant.

2.3 Ajout du token CLS

Généralement, les Transformers de l'état-de-l'art - comme BERT - ajoutent un token en entrée de chaque séquence qui ne correspond à aucun mot déjà existant. C'est le token [CLS] pour "classification". Au lieu de classifier chaque critique sur une moyenne pondérée des $v * \alpha$, mot par mot, il est possible d'utiliser seulement la sortie des couches d'attention sur un token dédié - celui-ci. Voici le raisonnement :

— Ayant passé à travers les couches d'attention, le token [CLS] dans une phrase aura subi les mêmes transformations que les autres éléments contenus dedans.

— Comme l'attention propre rapporte les mots les uns aux autres, il aura été confronté à toutes les autres parties de la critique, et la sortie qui lui est associée finit par porter en elle toute l'information pertinente pour la classification.

— Comme il est initialement neutre du point de vue du sens, l'ajouter ne va pas biaiser les scores d'attention des autres mots.

— Comme c'est initialement le même pour absolument toutes les critiques, il est juste un reflet de ce que le mécanisme d'attention propre apporte spécifiquement à telle ou telle autre.

C'est donc la sortie sur le [CLS] qui est seule fournie au classifieur, avec le résultat fourni en page suivante. Les loss deviennent particulièrement stables, même avec Adam, et le surapprentissage disparaît. On atteint 80% d'accuracy en train comme en test en considérant seulement le token CLS ; il permet donc de s'affranchir un peu de la stochasticité des entrées (avec tout ce que la prise en compte de la moyenne impliquait d'approximations) en profitant de tous les avantages décrits ci-dessus.

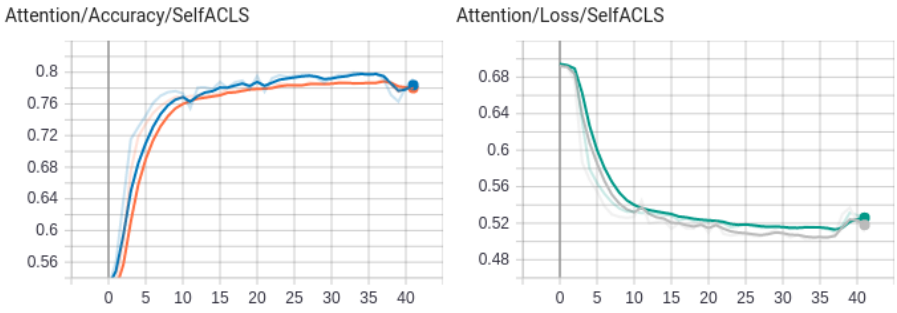


Figure 21 – Classification sur CLS avec Adam

Amélioration : ajout des plongements de position

Pour pousser un peu plus loin, on a pensé à compléter CLS avec des plongements de position. Mauvaise idée : on récupère l'aspect surapprentissage et pas du tout l'amélioration des résultats. Les performances en test s'effondrent et celles en train ne sont plus aussi bonnes. Apparemment, le modèle s'y perd.

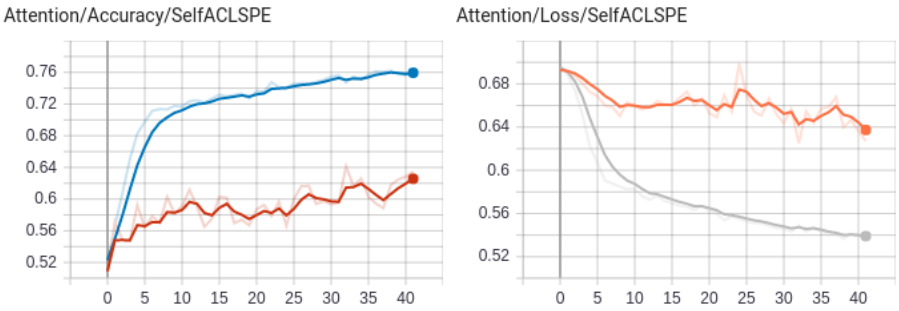


Figure 22 – Résultats du modèle augmenté SGD

2.4 Conclusions

Nos résultats sur ce TME ne sont pas très satisfaisants malgré de très nombreux essais. Les améliorer requerrait un paramétrage bien choisi. Il aurait fallu tout de même que les résultats avec l'attention propre soient un peu meilleurs que ceux avec l'attention globale ; seulement, étant donnée la difficulté de l'entraînement et de la convergence, il est difficile de bien obtenir cette conclusion sur de petits datasets comme le nôtre.

On retient néanmoins que la classification sur le token [CLS] est particulièrement efficace pour les raisons explicitées dans la section dédiée.

En ouverture, on rappellera que le système implémenté ici est souvent lié à une unité de décodage qui permet de fonder des générations sur les attentions calculées. Ensemble, les deux forment un Transformer, très utilisé par exemple pour les chatbots - et comme le montrent les soutenances d'AMAL, où les Transformers tenaient une place prépondérante, pour beaucoup d'autres applications.