Rapport Amal 3

Mathieu GROSSO - Thomas Floquet

December 2021

1 TP07 - Régularisation et protocole expérimental

1.1 Préparation du modèle:

On entraine un réseau de neurones simples avec 3 couches linéaires de 100 channels. La sortie est de taille 10 puisqu'on a 10 classe (chiffres de 0 à 9). On utilise une cross entropy pour entrainer le modèle, on optime avec Adam sans lr decay. On choisit un Lr de 0.001, 1000 epochs, et des batchs de 300 données. De même on ne conserve que 0.05 des données de train et de val.
Sans aucune régularisation, on voit que le modèle apprend vite en train et en test mais que rapidement il overfit et que la courbe de train en test augmente après 100 epochs jusqu'à remonter à son niveau de départ. De même l'accuracy diminue. On va visualiser la loss, les poids et le gradient pour essayer de voir l'évolution du modèle au cours de l'entrainement.

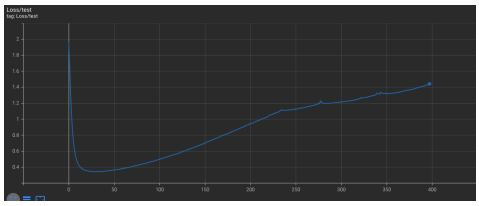

Figure 1: Loss en test sans aucune régularisation, avec un learning rate de 0.0001.

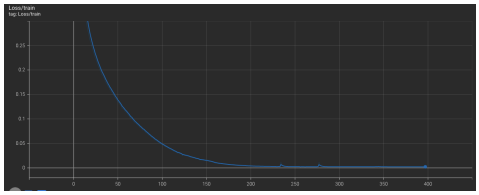

Figure 2: Loss en train sans aucune régularisation, avec un learning rate de 0.0001.

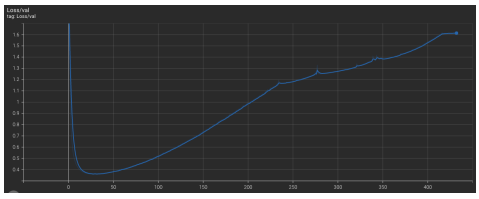

Figure 3: Loss en validation sans aucune régularisation, avec un learning rate de 0.0001.

En train l'apprentissage se passe bien, on apprend bien. Le problème est que sans régularisation on finit par sur apprendre les données de train et que la loss remonte en test et en validation. On se retrouve donc avec de mauvais résultat en test en en validation comme on peut le voir sur le graph.

La loss qui descend jsuqu'à 0.2 après 50 epochs remonte à 2 en validation.

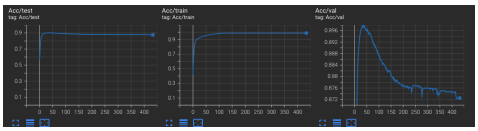

Figure 4: Accuracy en test, val et train sans aucune régularisation.

Le graphe d'accuracy montre la même chose que la courbe de loss, on a surtout une très mauvaise accuracy en validation après 100 epochs. On rajoute également les histogrammes de visualisation des poids et des valeurs du gradient en training, test et validation.

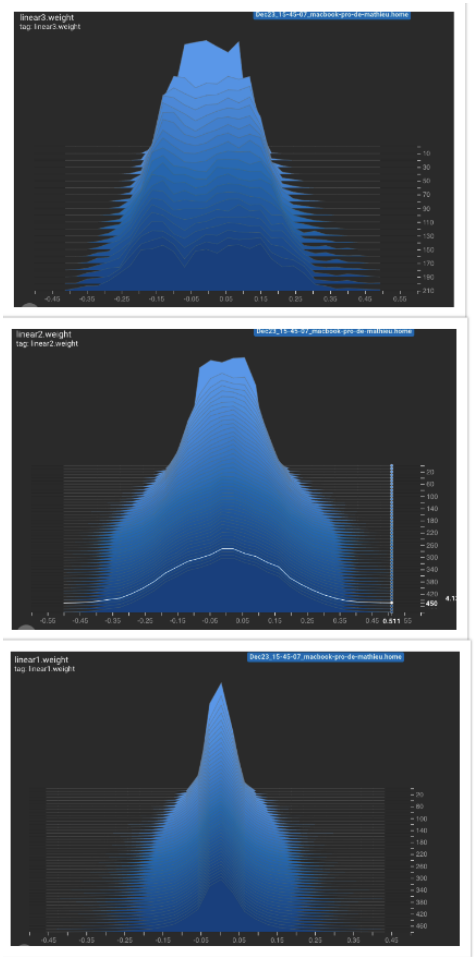

Figure 5: Différents poids de chaque couche au cours de l'apprentissage sans régularisation.

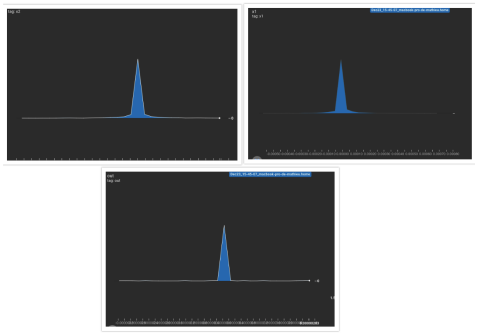

Figure 6: Gradient au cours de l'apprentissage sans régularisation.

1.2 Régularisation:

On va ensuite s'interesser à la régularisation du modèle; On va tester différentes approches, le dropout, la regularisation L1 et L2, la batchnorm et la layer norm. Au final, on retiendra le dropout, la régularisation L1 et un decay du learning rate de 0.009. En rajoutant cette régularisation, on a des résultats qui convergent pus vite et surtout le modèle ne sur-apprend pas en train puisque même après 400 epochs la loss en val et en test en remonte pas.

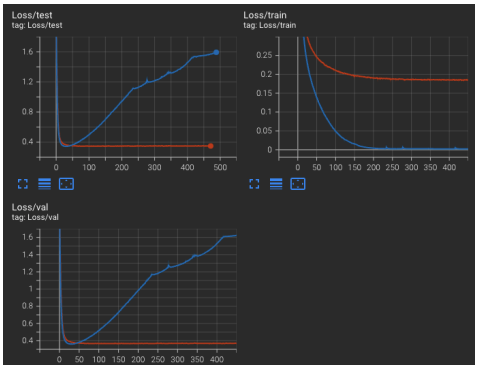

Figure 7: Loss en train,test et validation avec régularisation (dropout, L1,lr decay).

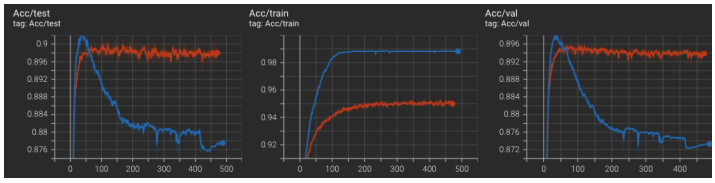

Figure 8: Accuracy en train,test et validation avec régularisation (dropout, L1,lr decay).

En rouge la courbe avec régularisation *(dropout= 0.47,lambda = 0.001 (pour la régularisation L1), et decay = 0.009)*. Sur la courbe de la loss, on observe clairement une amélioration en val et en test. Contrairement à avant ou le modèle ne convergeait pas, maintenant il converge et plus haut que sans la régularisation. C'st particulièrement vrai en validation où la différence est importante. De même sur les courbes d'accuracy on a une bien meilleure accuracy en val et en test. A l'inverse en train, les scores sont légerement moins bon mais c'est normal puisqu'on sur apprend moins sur les données de train.
Ensuite on va comparer les poids et gradients avec régularisation et sans régularisation.

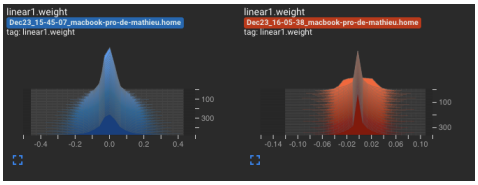

Figure 9: Différents poids de chaque couche au cours de l'apprentissage avec régularisation.

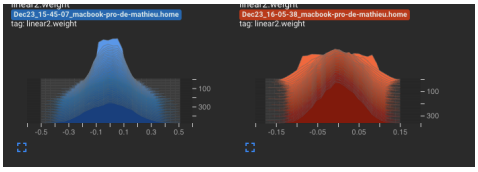

Figure 10: Différents poids de chaque couche au cours de l'apprentissage avec régularisation.

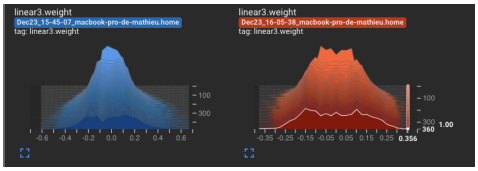

Figure 11: Différents poids de chaque couche au cours de l'apprentissage avec régularisation.

En rouge les poids pour l'apprentissage avec régularisation. On voit que le modèle en rouge semble converger mieux.

1.3 Augmentation de données:

l'augmentation des données réduit grandement l'accuracy. Les opérations effectuées sont: $data_transforms2 = transforms.Compose([transforms.ToPILImage(), transforms.RandomRotation(degrees = 45), transforms.Random$
$(18, 18)), transforms.ToTensor()])$.

On a donc décidé de ne pas garder cela dans la suite.

1.4 Optuna, Lightning, et optimisation des hyperparamètres:

On a effectué une grid search de 100 itérations sur le coefficient de régularisation L1 et sur le coefficient de dropout. Pour le dropout on a placé le coef entre 0.1 et 0.5 et pour la régularisation L1 on a choisi le coefficient entre 0.0001 et 0.1. Puis on a optimisé grace à optuna en créant un objectif et une étude sur le modèle précédemment utilisé.
Les résultats sont les suivants:

2 TP08 - Détection de sentiments :

Nous avons étudié 3 architectures différentes :

1. 1 couche de convolution avec 16 out_channels
2. 2 couches de convolution allant de 16 à 32 out_channels

3. 4 couches de convolution allant de 16 à 128 out_channels

Tous suivis d'un MLP à deux couches.

Nous analysons la performance relative de notre modèle en comparant l'accuracy à un modèle renvoyant toujours la classe majoritaire 0.

Question 3 :

$$S_(i+1) = s_(i+1) * s_(i) \tag{1}$$

$$W_(i+1) = w_(i+1) + k_(i-1) \tag{2}$$

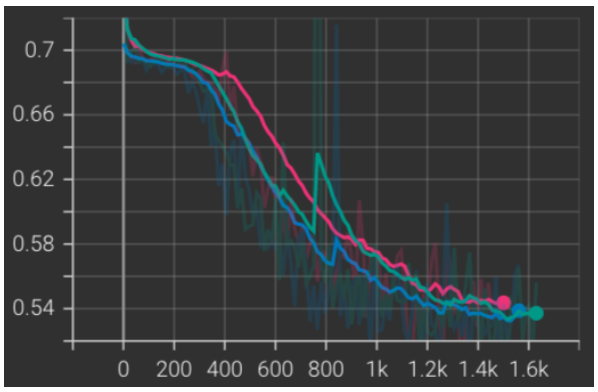

Figure 12: Loss of the 3 models

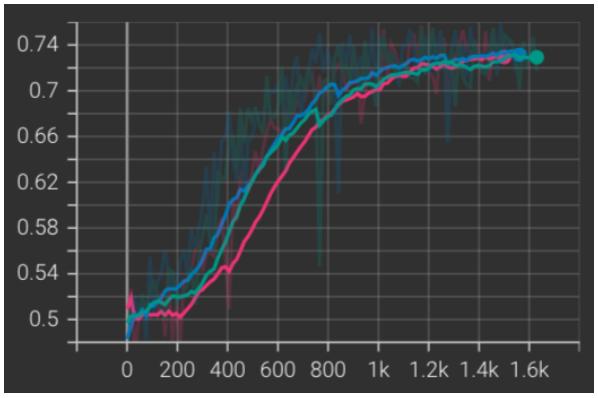

Figure 13: Accuracy of the 3 models

Légende : En vert le modèle 1, en bleu le modèle 2 et en rose le modèle 3.

On remarque que les 3 modèles ont des performances assez proches, avec un léger avantage pour le modèle 2 qui semble converger un peu plus vite et avoir une très légère meilleure performance. L'accuracy relative montre bien que nos modèles sont plus performants qu'un modèle renvoyant toujours la classe majoritaire (ici ayant une accuracy d'environ 0.5).

3 TP09 - CNN et classification de sentiments

BasicNet : modèle simple de la question 1 où chaque mot est utilisé avec la même pondération.
SimpleAttention : modèle d'attention de la question 2 qui pondère les mots qui ont un intérêt pour la classification. Une fonction simple qui permet d'obtenir $p(a_i|t)$ est la fonction softmax
ComplexAttention : modèle de la question 3 où la query dépend du texte lui-même.

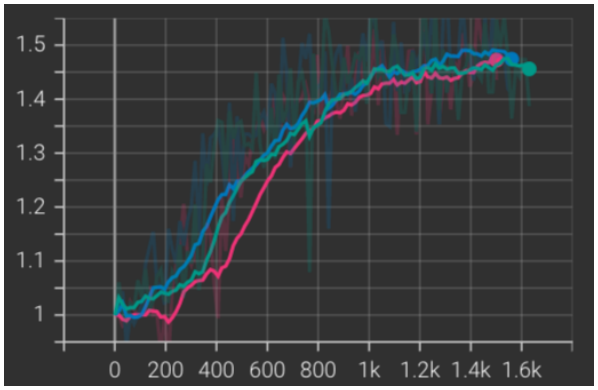

Figure 14: Relative accuracy of the 3 models of TP8

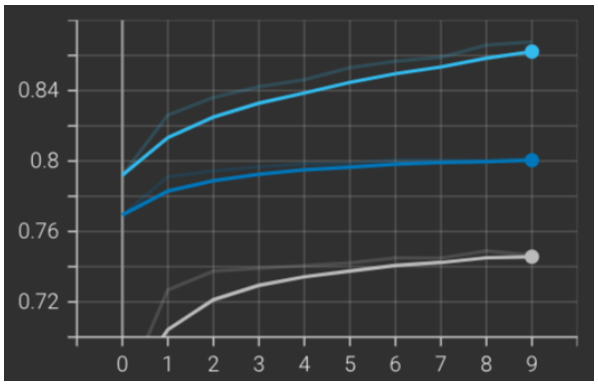

Figure 15: Accuracy of the 3 models of TP9

Légende : En gris le modèle BasicNet, en bleu foncé le modèle SimpleAttention et en bleu clair le modèle ComplexAttention.

Comme prévu, ComplexAttention obtient de meilleurs résultats que SimpleAttention qui lui-même obtient de meilleurs résultats que BasicNet (Figure 16). Cela s'explique par le fait que le modèle ComplexAttention apprend la query de manière à la faire dépendre du texte lui-même, ce qui permet d'avoir une query plus intéressante pour l'analyse de sentiment.

On remarque que le modèle s'intéresse bien aux mots les plus importants pour l'analyse de sentiments. En effet, les mots avec les valeurs d'attention les plus élevées sont hopelessly et gooey, et les mots avec les valeurs les moins importantes sont hard et correctness.

4 TP10 - Attention globale

4.1 1 - Modèle de base

Nous avons d'abord étudié un modèle simple de transformers avec L = 3. (3 modules de self Attention). Nous avons utilisé 100 pour la dimension des embeddings, un learning rate de 0.001, 10 époques et une batch size de 128. Le modèle apprend vite et converge autour de 0.75 ce qui est meilleur que ce que nous avions trouvé dans les TP précédents (cf TP09 au dessus). Les hyperparamètres pour les dimensions sont les suivants et sont disponible dans le fichier config du code:

Ensuite nous avons testé en rajoutant toute les options d'un transformers

```
input phrase: ['somewhere', 'in', 'the', 'dark', 'recesses', 'of', 'my', 'brain', 'cells', 'a', 'song', 'plays', 'in', 'my', '__OOV__',
'i', '__OOV__', 'forget', 'it', 'no', 'matter', 'how', 'hard', 'i', '__OOV__', '__OOV__', 'midnight', 'madness', 'and', '__OOV__',
'gonna', 'get', 'to', '__OOV__', 'wish', 'i', 'could', 'find', 'a', 'copy', 'of', 'this', 'on', 'a', '45rpm', '__OOV__', 'five',
'disparate', 'teams', 'head', 'out', 'one', 'night', 'in', 'l.a.', 'for', 'a', 'scavenger', 'hunt', 'for', 'clues', 'instead', 'of',
'physical', '__OOV__', 'an', 'unkempt', '__OOV__', 'with', 'two', 'gorgeous', 'assistants', 'is', 'the', 'mastermind', 'of', 'all',
'this', 'insanity', '__OOV__', 'about', 'to', 'be', 'unleashed', 'on', 'l.a.', 'all', 'the', 'teams', 'are', 'stereotypes', '__OOV__',
'movie', 'being', 'from', '__OOV__', 'before', 'political', 'correctness', 'screwed', 'everything', '__OOV__', 'the', '__OOV__',
'__OOV__', 'the', '__OOV__', 'led', 'by', 'eddie', '__OOV__', 'the', 'dumb', '__OOV__', '__OOV__', 'the', '__OOV__', '__OOV__',
'especially', 'the', '__OOV__', 'the', 'giggling', 'twins', 'are', 'a', '__OOV__', '__OOV__', 'and', '__OOV__', 'the', '__OOV__',
'__OOV__', 'with', 'stephen', 'furst', 'as', 'the', '__OOV__', 'furst', 'is', 'hilarious', 'as', 'the', 'overweight', 'slob', '__OOV__',
'whose', 'attempt', 'to', 'use', 'a', 'computer', 'to', 'decipher', 'the', 'various', 'clues', 'leads', 'to', 'a', 'gooey', '__OOV__',
'movies', 'like', 'this', '__OOV__', 'made', '__OOV__', 'these', '__OOV__', 'movies', 'have', 'to', 'have', 'an', '__OOV__', 'to',
'them', 'with', 'some', 'dark', 'characters', 'and', 'other', '__OOV__', 'go', 'back', 'to', 'the', 'days', 'when', 'the', '__OOV__',
'__OOV__', 'led', 'by', 'david', 'naughton', 'were', 'still', 'good', 'and', 'not', 'hopelessly', '__OOV__', 'so', 'dump', 'all',
'serious', 'pretensions', 'and', 'go', 'back', 'to', '__OOV__', '__OOV__', 'midnight', 'madness', '.', '.', '.']
word: hopelessly, attention value: 0.040796324610710144
word: gooey, attention value: 0.03663202375173569
word: slob, attention value: 0.022535987198352814
word: serious, attention value: 0.020592158660292625
word: hilarious, attention value: 0.019024258479475975
word: unleashed, attention value: 0.016735348850488663
word: good, attention value: 0.016594644635915756
word: gorgeous, attention value: 0.015835488215088844
word: especially, attention value: 0.01349448412656784
word: pretensions, attention value: 0.012700004503130913
word: dumb, attention value: 0.012213854119181633
word: forget, attention value: 0.012088504619896412
word: unkempt, attention value: 0.011397838592529297
word: hard, attention value: 0.010123910382390022
word: correctness, attention value: 0.010118484497070312
```

Figure 16: Attention values du TP9

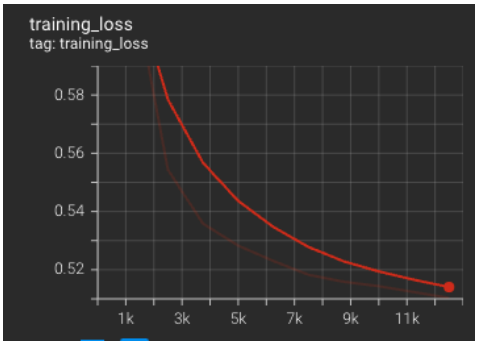

Figure 17: loss en train TP10

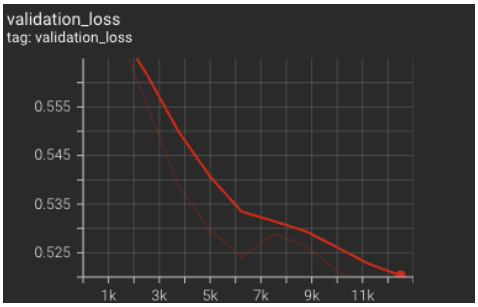

Figure 18: loss en val TP10

4.2 2 - Ajout d'un modèle Résiduel, d'un token de CLS et d'un positional encoding

Dans cette section, nous avons rajouté des connections résiduel, un token CLS et le positional encoding. Les hyperparamètres des dimensions sont les mêmes que précédemment ainsi que les paramètres d'entrainement. En revanche nous avons fait plus d'époques car au bout de 10 époques les résultats étaient moins bon que précedemment. En effet l'ajout d'un modèle pour le token peut ralentir le

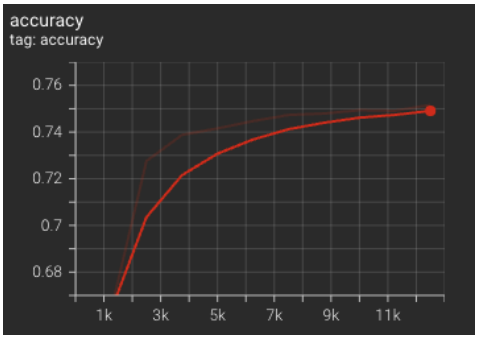

Figure 19: Accuracy en train.

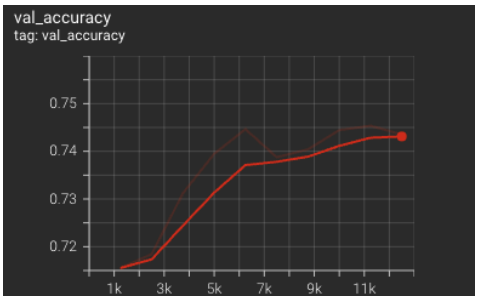

Figure 20: Accuracy en val.

training.
Au bout de 30 epoques voilà les résultats: sur les graphiques en rouge ce sont les résultats pour le modèle basique de transformers sans les ajouts de connections résiduelles, de token et d'encoding, et en vert avec:

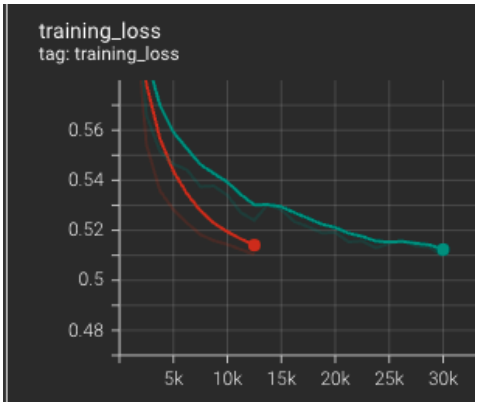

Figure 21: loss en train.

On observe que la courbe rouge obtient de meilleur résultats et semble aussi plus stable. Nous avons aussi tester avec plus d'époques pour l'encoder basique et en effet il semble mieux fonctionner. Nous avons aussi essayer de retirer le token cls et on obtient des résultats comparables. On ne met pas les courbes car ils sont très proches.

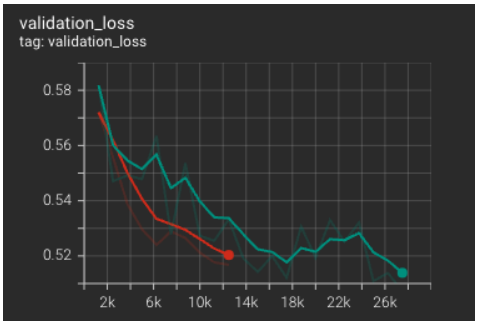

Figure 22: loss en val.

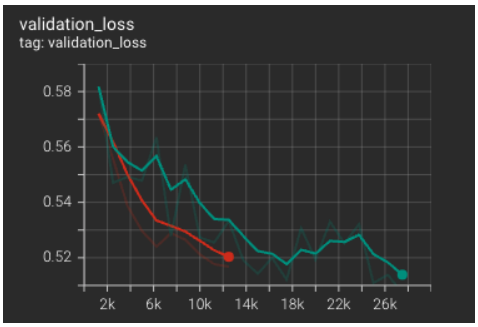

Figure 23: Accuracy en train.

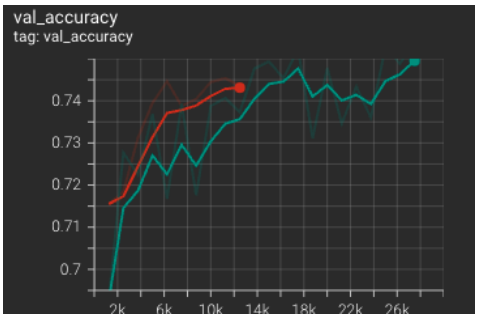

Figure 24: Accuracy en val.