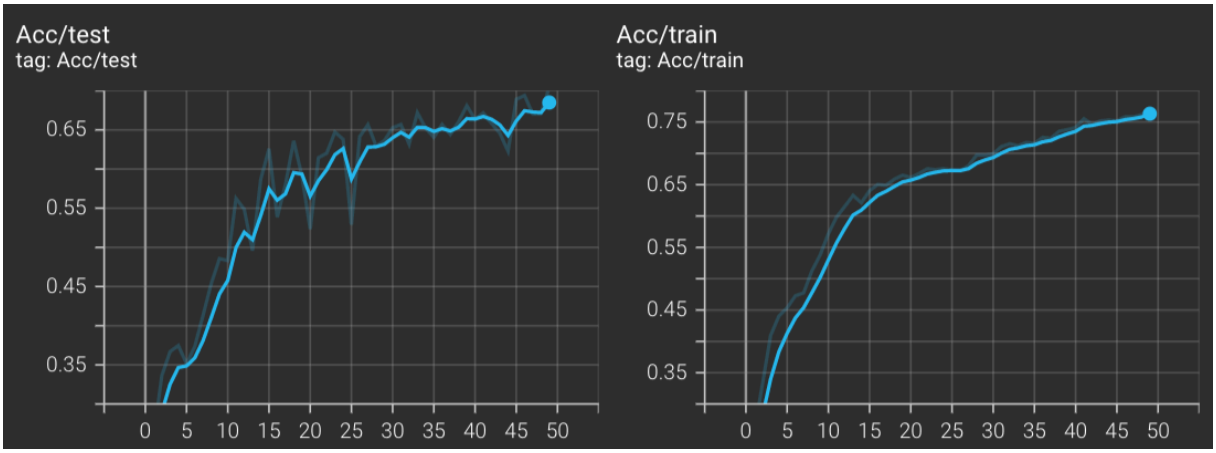
Rapport TPs AMAL 4 à 6

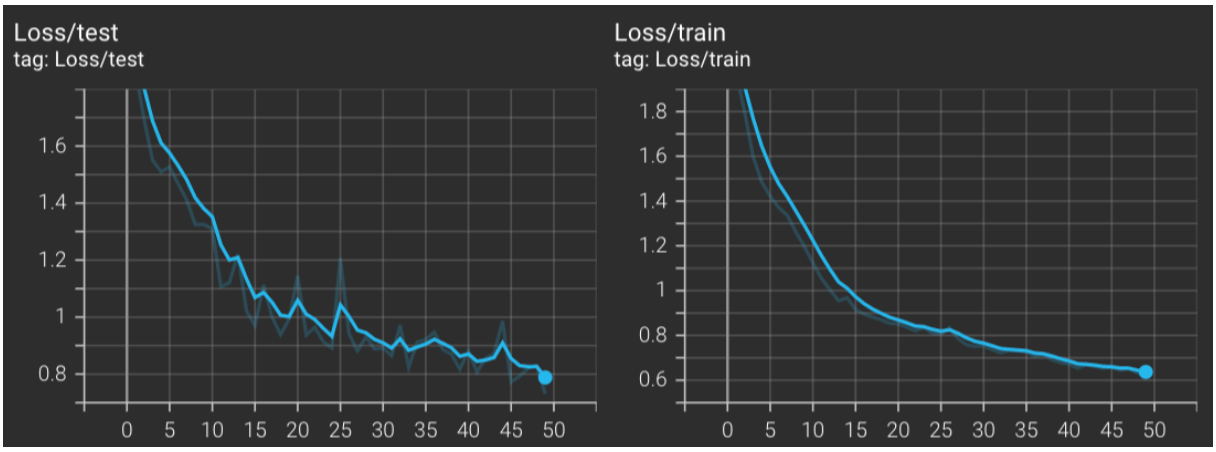
Mathieu Grosso et Thomas Floquet

TP4 :

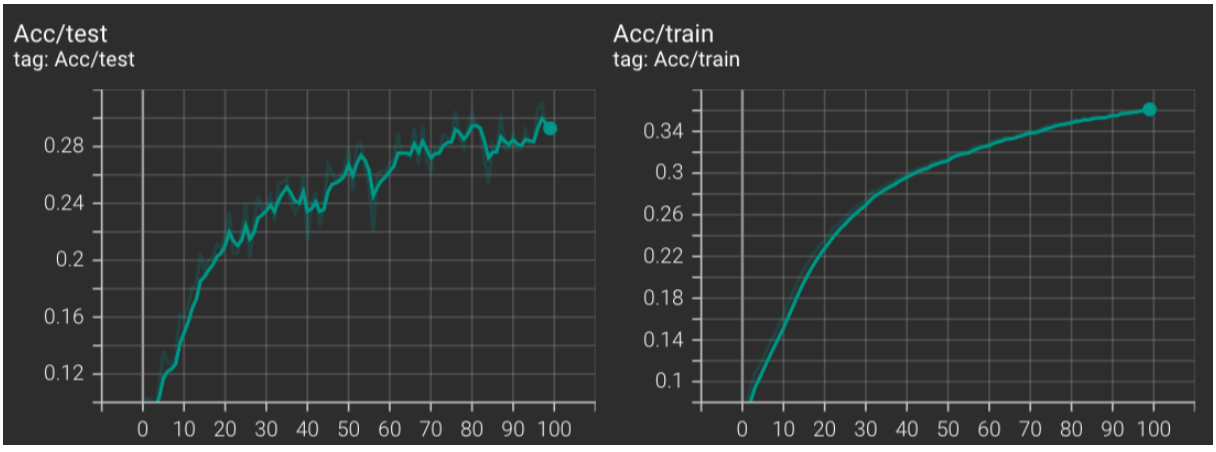
Exo 2 (classification) :

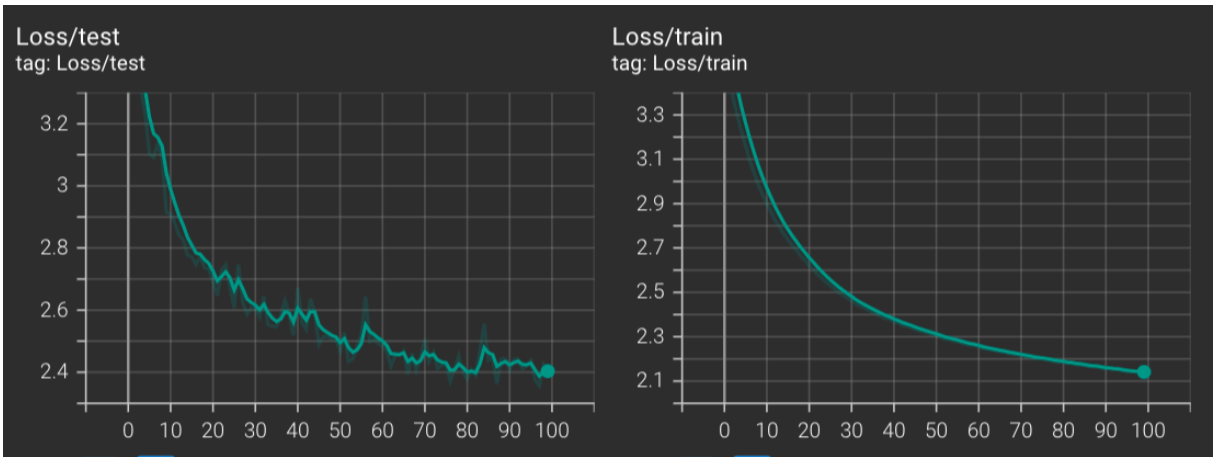
Nb stations = 10 :





Avec toutes les stations (nb stations = 80) :



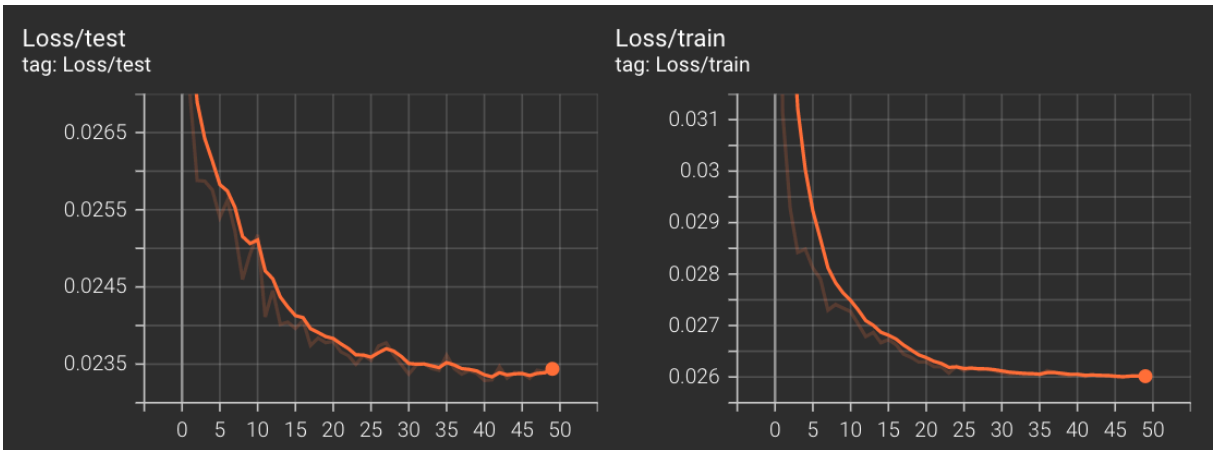


L'accuracy est donc moins bonne quand on augmente le nombre de stations, ce qui est logique car il est plus difficile de bien classifier parmi 80 que parmi 10.

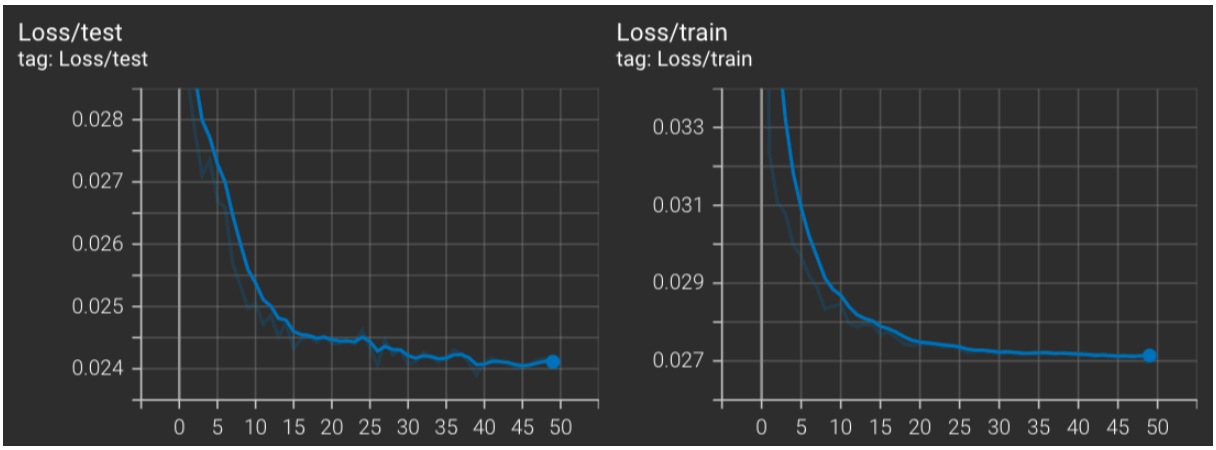
Exo 3 :

Par rapport au modèle précédent, on ne fait plus une prédiction à la fin de la séquence mais on décode à chaque étape de la séquence. Donc pour décoder on ne prend plus le dernier état caché de chaque séquence mais l'ensemble des états cachés de chaque séquence Puisqu'on ne fait plus de la classification mais de la prédiction, la MSE loss est plus adaptée que la cross-entropy.

Avec taille des séquences = 20 et flux entrants + sortants :

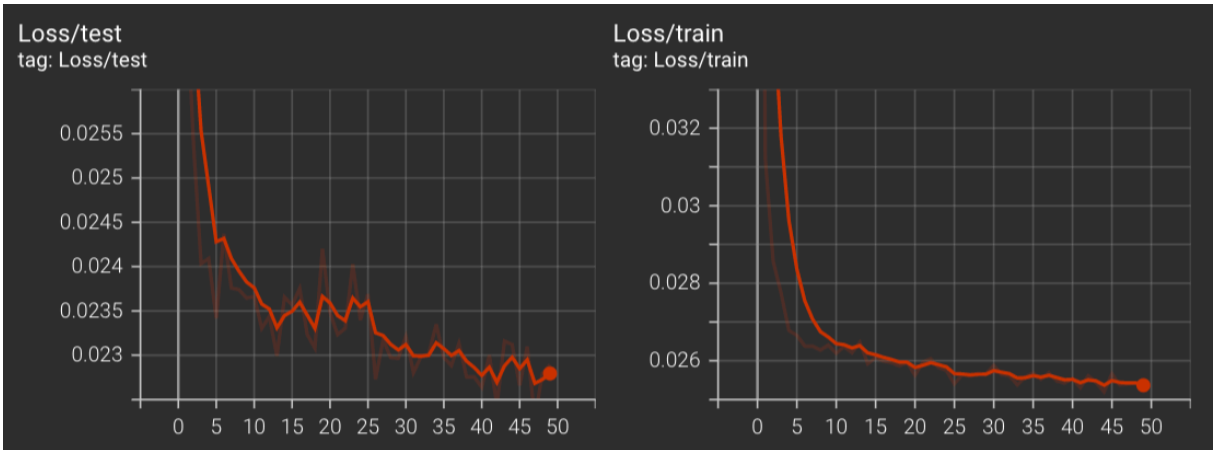


Avec taille des séquences = 20 mais uniquement flux entrants :



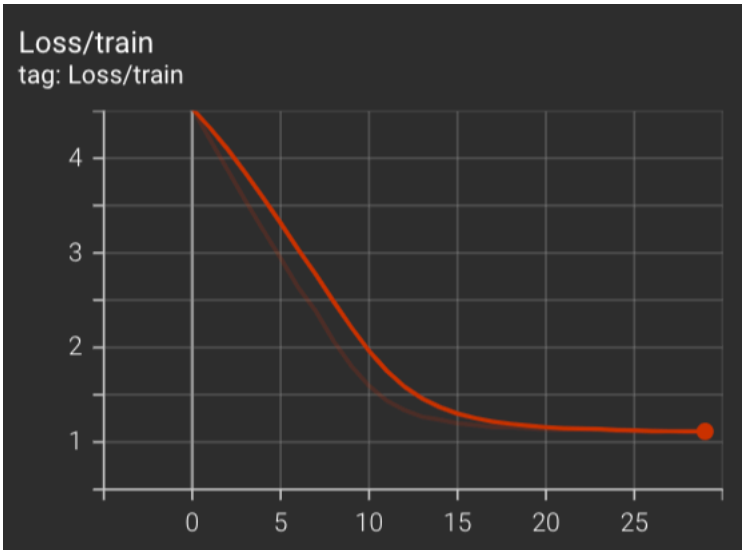
Les résultats sont très légèrement inférieurs lorsqu'on ne considère plus que les flux entrants, ce qui est logique car on a moins de données pour classifier

Avec taille des séquences = 50 et flux entrants + sortants :



Les résultats sont meilleurs quand on augmente la taille des séquences, ce qui est logique car on a des séquences plus longues donc on peut faire de meilleures prédictions

Exo 4 :



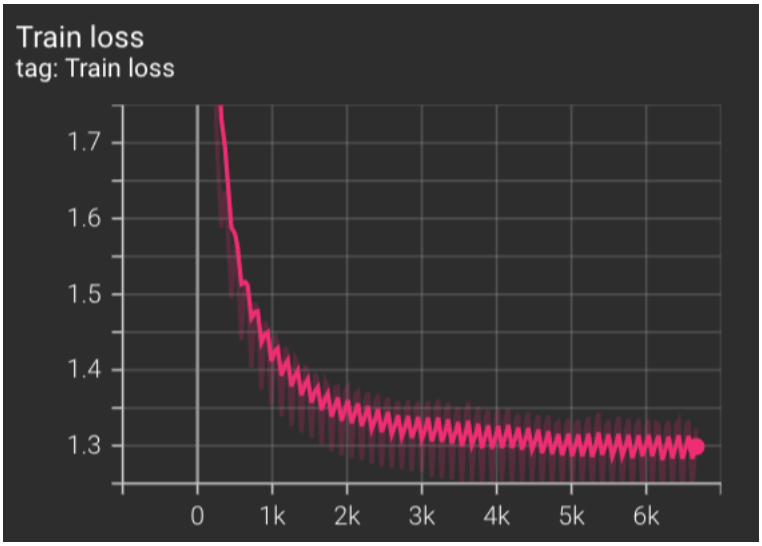
En mode greedy, seulement des caractères vides sont prédits. Il doit y avoir une erreur dans le décodage car on obtient de meilleurs résultats avec un RNN simple dans le TP5…

TP5 :

Hyperparamètres : nombre d'epochs = 50, learning rate = 0.001, batch size 128, k=5 pour la beam search, start = "American people want ", maxlen = 200

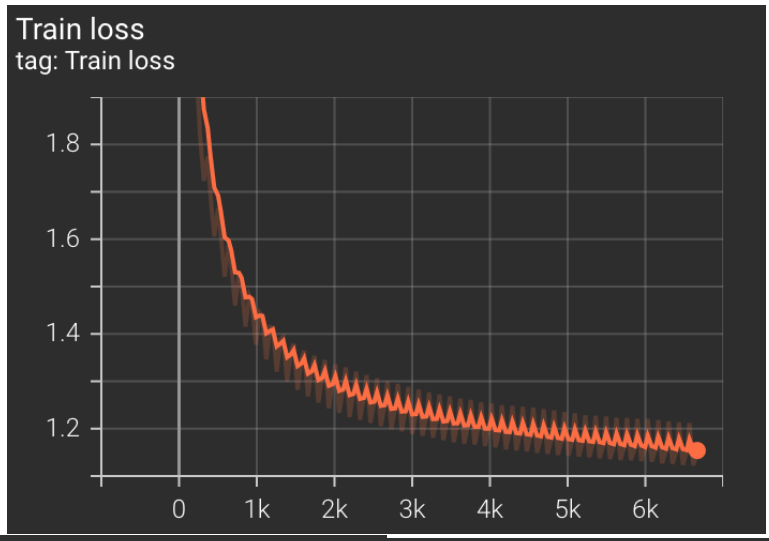
RNN :

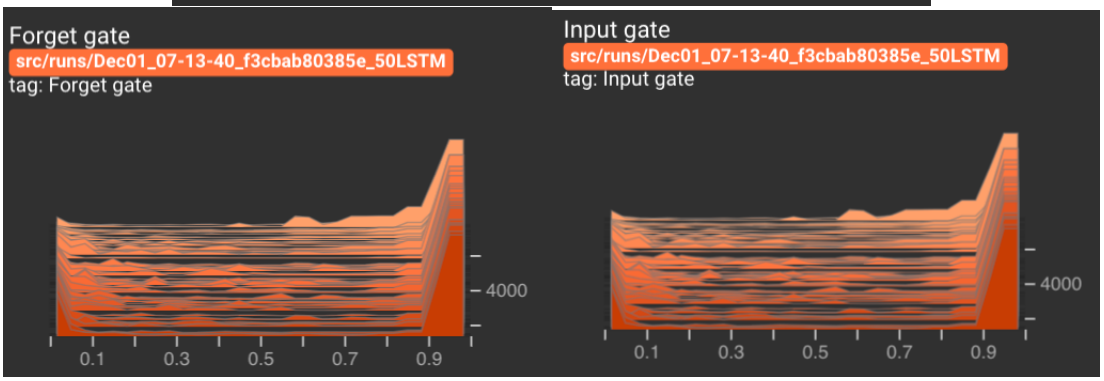
Loss après 50 epochs : 1.39
Greedy: 'American people want to be a going to be a going to be a going to be a going to be a going to be a going to be a going to be a going to be a going to be a going to be a going to be a going to be a goi<EOS>'
Multinomial: 'American people want our campay state the only copon? Blorts aread to beliepeler 'ndinning heard of Missia of right? One is to vote he support and Think of us.<EOS>'
Beam search : 'American people want taomwith the wolithave ay.<EOS>'
Nucleus sampling : 'American people want t+-,/thone of the people and they want to be a government to the people and they want to be a government to the people and they want to be a government to the people and they want to b<EOS>'

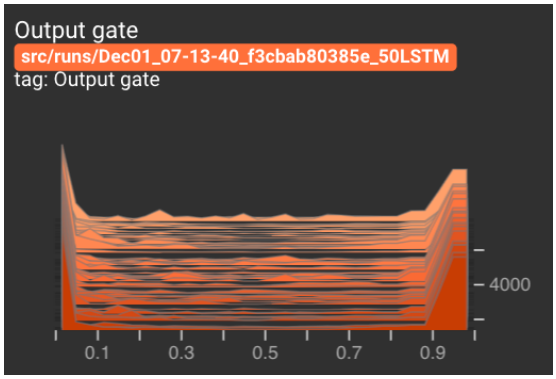


LSTM :

Multinomial : 'American people want to get a hurtate some quartody.<EOS>'
Beam search : 'American people want toiafor themoranineved.<EOS>'
Nucleus sampling : 'American people want t+-,/onour the most to be a great people and the same new some of the people and the same new some of the people and the same new some of the people and the same new some of the people<EOS>'

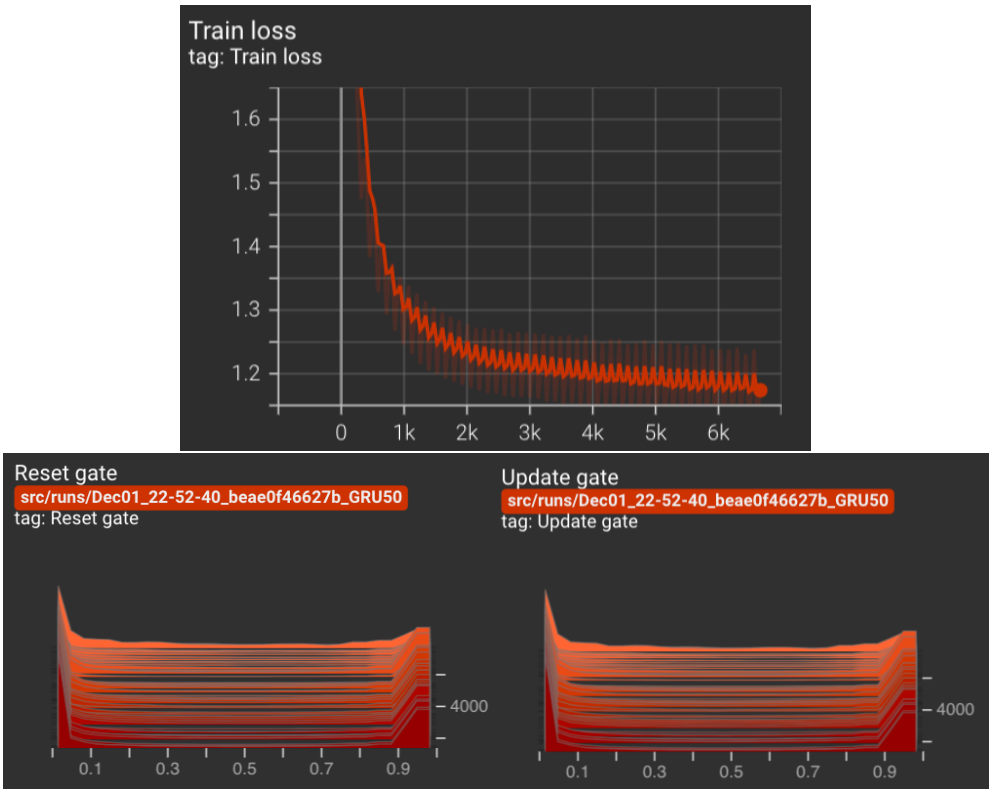






GRU :

Loss après 50 epochs : 1.29
Multinomial : 'American people wants thous, of our put vote.<EOS>'
Beam search : 'American people want tioayou.<EOS>'
Nucleus sampling : 'American people want t+-,/22 do and we will go to the people and we will go to the people and we will go to the people and we will go to the people and we will go to the people<EOS>'



Analyses :

Comme on pouvait s'y attendre, si on compare les loss, le GRU et le LSTM performent mieux que le RNN simple grâce à la prise en compte des dépendances lointaines grâce aux portes. On remarque que pour le LSTM la forget gate et l'input gate (similaires à l'update gate du GRU) sont très souvent activées (donc on garde l'information de l'état précédent et cette information est considérée comme importante pour la mise à jour de la cellule), l'ouput gate oscille entre 0 et 1.
Si on regarde les phrases générées, la méthode greedy n'est pas très efficace car on répète toujours la même chose. De manière assez surprenante cependant, la version multinomiale semble générer des phrases plus réalistes que la version avec le beam search. Peut-être faudrait-il entraîner davantage les modèles afin que le beam search performe mieux. Enfin, avec le nucleus sampling on a encore une répétition de mots et la génération commence toujours par les caractères *t+-,/*, peut-être y a-t-il une erreur d'implémentation…

TP6 :

1 Introduction

Dans ce troisième TP sur les RNNs, nous avons étudié deux tâches de type Seq2seq. Tout d'abord du tagging, puis de la traduction en générant un texte à partir d'un état latent. L'état latent represente la donnée en entrée et dépend de la tâche.

2 Partie 1 - Tagging

On entraine un réseau récurrent pour la tâche de tagging. En l'occurence on entraine un LSTM à apprendre la classification de chaque mot. En input on donne une phrase segmentée en token, et on retourne la classe de chacun des mots dans la phrase. L'output est une liste de classes grammaticaux (19 classes en tout : Noun, Cconj, verb, punct...).

2.1 Modèle et hyperparamètres:

Le modèle: Tout d'abord on utilise un embedding calculé grace à la classe torch.nn.embedding, puis on passe la sortie de l'embedding dans un LSTM et enfin on utilise une couche linéaire qui fait office de décodeur ici. La couche linéaire prend en entrée la sortie du LSTM et ressort un output de la taille du nombre de classes dans le dictionnaire (19).

Les hyperparamètres:

- learning rate: 0.001,
- epochs: 25,
- optimizer: Adam,
- loss: crossentropy,
- dimension de l'espace latent : 100,
- dimension de l'embedding : 100,

Nous avons utilisé plusieurs combinaisons d'hyperparamètres et celle ci est celle qui a le mieux fonctionné. Le nombre d'epoch peut-être réduit car à partir de l'epoch 16 il semble que le modèle stagne et apprennent moins rapidement. L'apprentissage est relativement rapidement même sur cpu; en train on obtient facilement une high accuracy et une loss de 0.04. en test on a une accuracy de 0.85 rapidement puis qui atteint 0.9 plus difficilement. On peut le voir dans les figure 1 et 2.

Ces figures représentent la loss et l'accuracy en test et en train. La loss finale en train est de 0.05 et la loss finale en test est de 0.2. L'accuracy finale en train est de 0.89 et en test elle vaut 0.98.

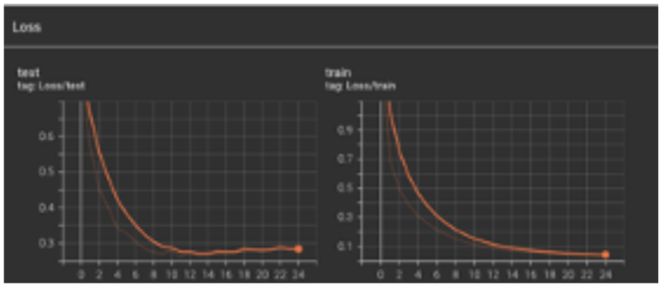


Figure 1: This figure represents the loss during the tagging task.

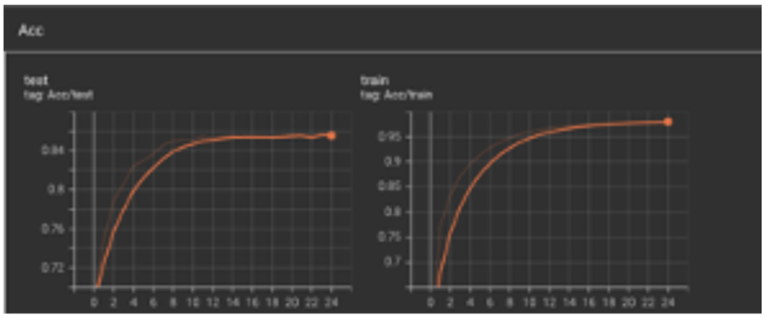


Figure 2: This figure represents the Accuracy during the tagging task.

2.2 Génération

Grace à la génération on vérifie que le modèle fonctionne bien. Nous avons testé le résultat sur différentes phrases et la plupart du temps le modèle avait bon sur tout les mots. Ici, tout les mots sont en effet corrects sur la figure 3 à l'exception du mot OOV. La plupart des erreurs du modèle viennent de ces mots jamais rencontré.
On remarque ici que le mot OOV, donc jamais vu, a été prédit comme étant un déterminant que c'était un PROPN.

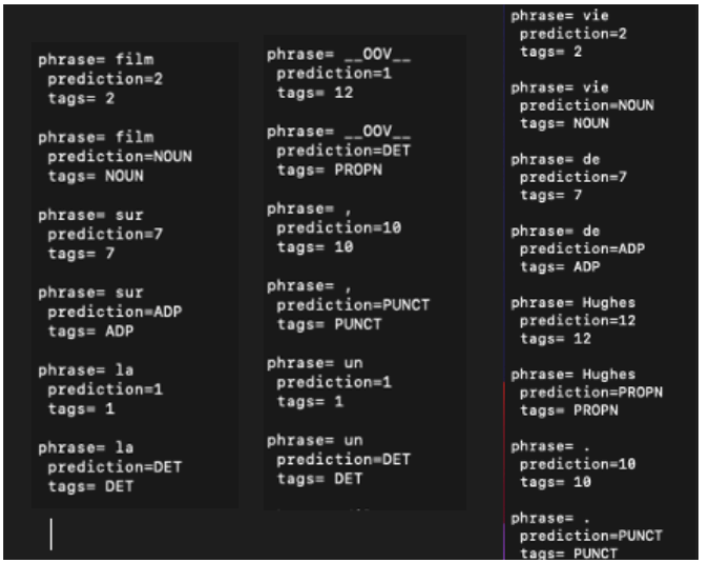


Figure 3: This figure represents the classification during the tagging task.

3 Traduction

3.1 Traduction avec le vocabulaire sans segmentation

Pour cette tâche, nous utilisons deux vocabulaire un en francais et en un anglais. Il y a autant de mots dans chacun des vocabulaires et chaque mots à son équivalent dans l'autre langue. Mais certaisn mots peuvent ne pas apparaitre dans le vocabulaire de train et apparaitre dans celui de test il faut donc aussi y penser lors de la traduction.
Pour entrainer le modèle, on utilise deux méthodes: mode contraint (teacher forcing) et mode non contraint. La probabilité de choisir l'une ou l'autre est un hyperparamètre, et cet hyperparamètre est important puisqu'il fait beaucoup varier les résultats. Le mode contraint est plus facile à apprendre, en effet le mode non contraint peut induire d'importantes erreur, une seule erreur implique une phrase complétement différente. Mais puisqu'en test on ne peut utiliser que le mode non contraint (on a pas accés au label), alors on va devoir se concentrer sur les deux approches pour eviter que le modèle ne généralise mal.

3.1.1 Modèle et hyperparamètres:

Le modèle: L'architecture est la suivante: un encodeur et un décodeur qui sont tout deux des GRU. L'architecture complète est un embedding, puis un gru pour l'encodeur, et ensuite un autre embedding, un gru et un couche linéaire de décodage pour le décodeur.

Les hyperparamètres:

- learning rate: 0.001,
- epochs: 50,
- batch size : 100,
- coef : proba de choisir entre contraint et non contraint : 0.35 (contraint si la probabilité est inférieure à 0.35),
- optimizer: Adam,
- loss: Crossentropy,
- dimension de l'espace latent : 250,
- dimension de l'embedding : 250,

3.1.2 Résultats et génération :

Nous avons fait varier les hyperparamètres mais les résultats restent toujours très similaires. la figure 4 et 5 montrent les résultats obtenus en accuracy et en train. La meilleure accuracy obtenu en train est de 0.73 en utilisant un coef de 0.35, et de 0.39 en test. La meilleure loss en train est de 0.92 et de 3.9 en test. Les résultats en test sont largement moins bons qu'en train et même en changeant la proba d'être en teacher forcing on atteint difficilement mieux (en utilisant une proba plus faible on réduit les résultats en train sans vraiment amélioer ceux en test).

Génération: La génération fonctionne plutôt bien surtout en greedy. On a eu du mal à ne pas avoir des séquences qui se répetaient. En trichant et en empechant le modèle de répeter les séquences on obtient de meilleurs résultats mais nous avons décidé de ne pas suivre cette technique. Les figure 6,7 et 8 sont des exemples de générations. On observe bien que le modèle arrivent à cerner la phrase. En utilisant une génération Beam search, les résultats ne sont pas meilleurs (surement que cela peut-être optimisé). Dans tout les cas au vu de l'accuracy les résultats restent plutôt encourageant et la méthode de génération pourrait-être optimisé.

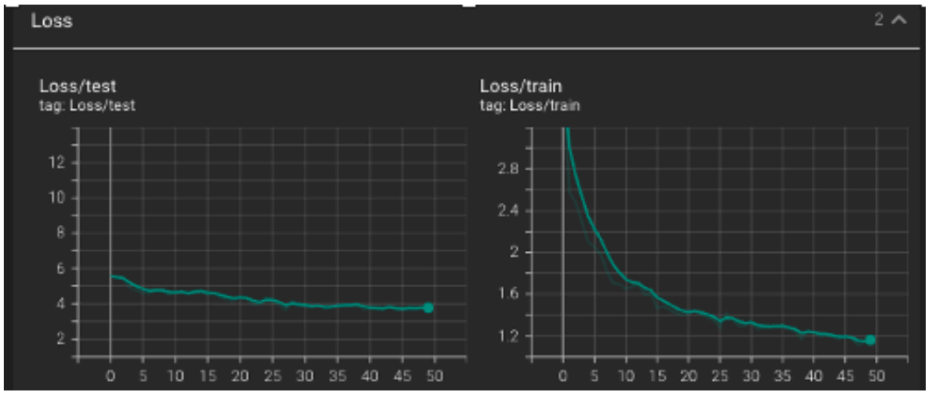


Figure 4: This figure represents the loss during the traduction task without segmentation.

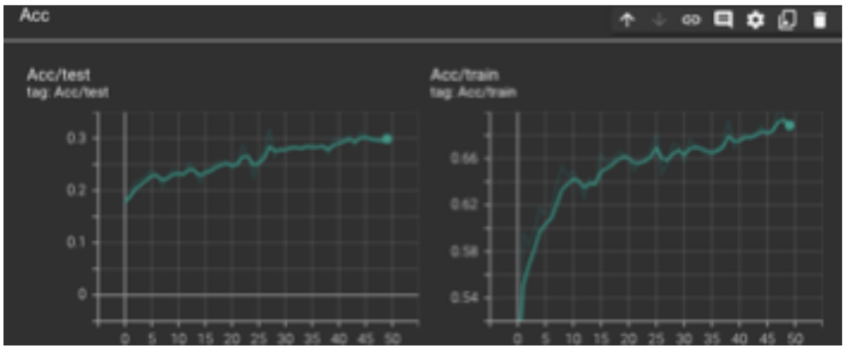


Figure 5: This figure represents the Accuracy during the traduction task without segmentation.

```
Phrase anglais: ['this', 'must', 'be', 'a', 'mistake', 'EOS']
Phrase français (truth): ['ce', 'doit', 'etre', 'une', 'erreur', 'EOS']
Traduction: ['cela', 'cela', 'un', 'etre', 'un', 'EOS']
```

Figure 6: This figure represents the generation using a greedy approach without segmentation.

```
Phrase anglais: ['tom', 'said', 'that', 'he', 'doesn', 't', 'regret', 'his', 'decision', 'EOS']
Phrase français (truth): ['tom', 'a', 'dit', 'qu', 'il', 'ne', 'regrettait', 'pas', 'sa', 'decision', 'EOS']
traduction: ['tom', 'tom', 'dit', 'qu', 'il', 'regrettait', 'EOS']
```

Figure 7: This figure represents the generation using a greedy approach without segmentation.

```
Phrase anglais: ['it', 's', 'not', 'even', 'true', 'EOS']
Phrase français (truth): ['ce', 'n', 'est', 'meme', 'pas', 'vrai', 'EOS']
Traduction: ['ce', 'n', 'n', 'pas', 'pas', 'vrai']
```

Figure 8: This figure represents the generation using a greedy approach without segmentation.

Dans la figure 6 on observe que la phrase est pratiquement identique mais que le fait que certains caractère se repête ne permet pas d'avoir la fin de la phrase (on utilise ici une len de generation égale à la len de la phrase). Il est fort probable qu'en utilisant une len de generation plus longue la phrase aurait correspondu. Dans la figure 8 on voit que même lorsque le modèle saute un ou plusieurs mots de la phrase en input, il réussit quand même à prédire les mots suivants.

3.2.1 Résultats

On utilise les même hyperparamètres car encore une fois cela ne change pratiquement pas les résultats. on observe que les courbes sont très similaires à ce que nous obtenions avant (cf figure 9 et 10). L'accuracy en test sature vite autour de 0.3 et la loss sature autour de 3.8. En train les résultats sont améliorés on arrive à atteindre une accuracy de 0.79 à l'epoch 45.

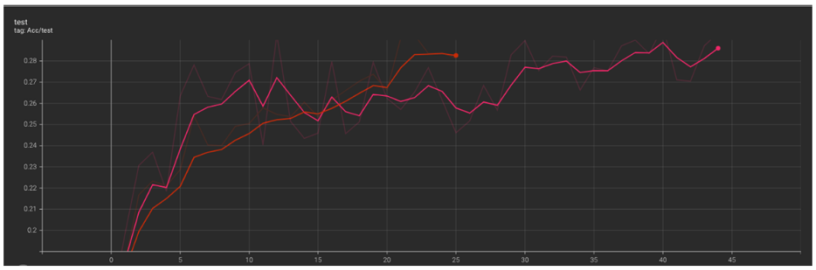


Figure 9: This figure represents the accuracy of the test during traduction with segmentation Vs without.

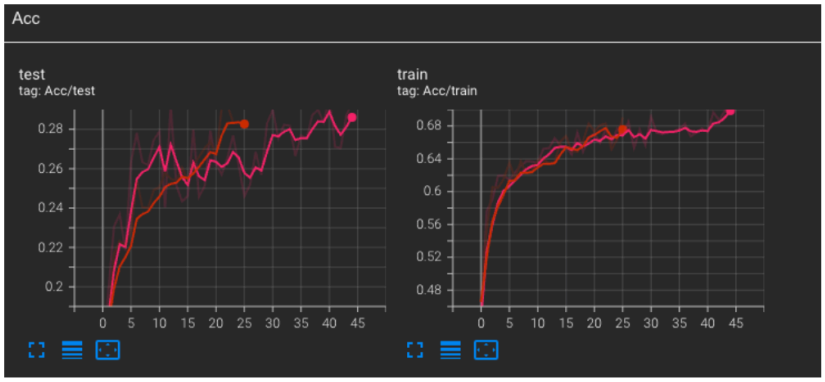


Figure 10: his figure represents the accuracy during traduction with segmentation Vs without.

En rouge, une tache sans segmentation. On voit qu'on atteint une meilleure accuracy en test en seulement 25 epoch, à l'époque 50 seulement la méthode avec segmentation rattrape la méthode sans segmentation. De même, en test la loss est meilleure que celle sans segmentation mais cela prend 10 epoch de plus. Au final il semblerait que les deux approches restent très similaires, il reste à test qualitativement le modèle en faisant de la génération. On va se concentrer sur la génération greedy puisque la beam search n'était pas efficace sur la tâche précédente.

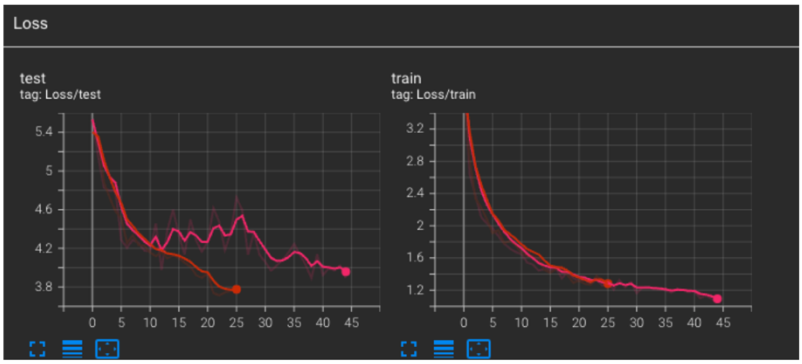


Figure 11: This figure represents the loss during traduction with segmentation Vs without.

3.2.2 Génération

La génération se passe plutôt bien, la phrase est logique mais elle ne correspond pas exactement à la phase voulue. La figure 10 ci dessous est un exemple de phrase traduite. Globalement on trouve les résultats meilleures que sans la segmentation au moment de la génération.

```
Phrase anglais: ['i', 'want', 'you', 'to', 'speak', 'frankly', 'EOS']
Phrase français (truth): ['je', 'veux', 'que', 'vous', 'parliez', 'franchement',
 'EOS']
Traduction: ['je', 'je', 'veux', 'parler', 'francais', 'EOS']
```

Figure 12: This figure represents the generation of a sentenc during the traduction task;