TP3 : Découverte des réseaux de neurones RECURRENTS (Rnn)

Les réseaux de neurones récurrents sont une autre grande famille de réseaux de neurones. Ceux-ci sont utilisés pour traiter des données séquentielles. Une donnée séquentielle, ou séquence, est une donnée qui possède une dimension temps. Dans un RNN, un échantillon n’est pas placé en entrée du réseau de neurone d’un coup, mais séquence par séquence. En effet, un réseau de neurone récurrent est composé (uniquement ou non) de neurones possédant une mémoire. Cette mémoire permet au neurone d’apprendre à mémoriser des informations utiles. Il est donc possible de traiter des problèmes de séquence à séquence (exemple : traduction de texte), de séquence à valeur (exemple : classification de son) ou de valeur à séquence (exemple : description textuelle d’une image).

Ce TP se base sur deux réseaux de neurones : un RNN « vanilla », c’est-à-dire basique, et un LSTM, qui est un type de réseau de neurones très populaire. Nous utiliserons ces deux réseaux pour prédire si un e-mail est un spam ou non, et analyserons ce qui se passe lors de leur entrainement. Nous introduirons par la suite le mécanisme d’attention par-dessus le LSTM, et verrons son impact sur les résultats.

Les données utilisées sont une collection de 5572 sms collectés et taggés « ham » s’ils sont véridiques ou « spam » si ce sont des messages indésirables. Nous ferons le TP sur Colaboratory. Il n’est pas nécessaire d’utiliser un GPU si le temps de latence est important, car les entrainements sont assez rapides (peu de données et données textuelles).

# compte-rendu

???

# Questions

## Analysez rapidement les données. Quels phénomènes apparaissent ? Que permet de faire la fonction tokenize\_and\_encode ? Pourquoi y avoir ajouté pad\_sequences ?

Documentation de tensorflow.keras.preprocessing.text : <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer>

Documentation de tensorflow.keras.preprocessing.sequence : <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences>

## Construisez et faites tourner le réseau RNN sur 25 époques. Qu’est-ce qu’une couche d’emdebbing ? Notez les temps d’exécution et les résultats, A partir de quand y a-t-il du sur-apprentissage ?

Documentation de la couche d’emdebbing : <https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding>

1. **Enlevez ‘mask=True’ de la couche d’emdebbing du RNN. Que se passe-t-il à l’exécution ? QU’est-ce que le vanishing (ou exploding gradient) et quelles sont ses conséquences ?**
2. **Faites tourner le réseau LSTM sur 20 époques. Que constatez-vous au niveau des résultats ?**
3. **Lorsque vous retirez « mask=True » de la couche d’emdebbing, les problèmes sont-ils toujours aussi immédiats ?**

Le LSTM permet de mitiger des problèmes de Vaishing Gradient, mais jusqu’à un certain point seulement. Les problèmes sont plus longs à apparaitre, il est donc plus facile de diagnostiquer un vanishing ou exploding gradient.

## Faites tourner le modèle avec attention sur 20 époques. Que constatez-vous au niveau des résultats et de la rapidité de convergence ? Rajouter l’early stopping dans le modèle. Qu’a permis de faire l’attention dans ce modèle ? Que permet de faire le Callback que nous avons défini, VisualAttentionMap ? Qu’observez-vous au niveau des outputs de VisualAttentionMap ?

## Pensez-vous que sur ce problème spam-or-ham, il est nécéssaire de rajouter de l’attention ? Pourquoi ou pourquoi pas ?