Modèle Seq2seq

Le **modèle Seq2seq** est un réseau neuronal qui va convertir une séquence de données en entrée vers une nouvelle séquence de données en sortie. Il est utilisé dans de nombreux domaines d’application tels que du traitement automatique du langage naturel (NLP) ou encore pour l’analyse de séries temporelles.

Il sera composé de deux couches de réseaux de neurones récurrents (RNN) : l’encoder et le decoder. La séquence d’entrée et celle de sortie n’ont pas forcément la même longueur.

Andrej Karpathy - CS231n Winter 2016: Lecture 10: Recurrent Neural Networks, Image Captioning, LSTM

<https://www.youtube.com/watch?v=yCC09vCHzF8> 2:09

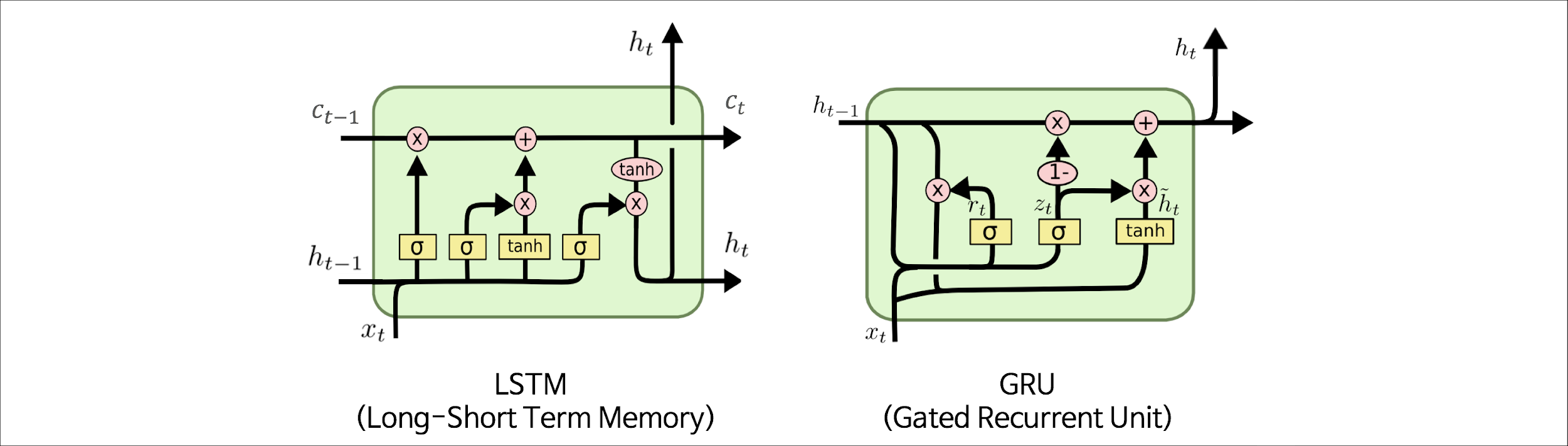
**RNN**

Le réseau de neurones récurrents (RNN) est un réseau d’unités reliées par des arcs possédant des poids. Ce réseau possède des connexions récurrentes puisque l’information est conservée en mémoire en prenant en compte les états précédents. C’est pour cette raison que les RNN sont particulièrement adaptés à des données faisant intervenir un contexte comme des suites de données. Ce réseau ne peut cependant mémoriser qu’une cinquantaine d’itérations et donc rester sur un passé proche, c’est le problème de disparition de gradient. Les réseaux Long short-term memory (LSTM) et les réseaux Gated Recurrent Unit (GRU) vont alors pouvoir répondre à ce problème.

Source pour LSTM et GRU:

<http://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/>

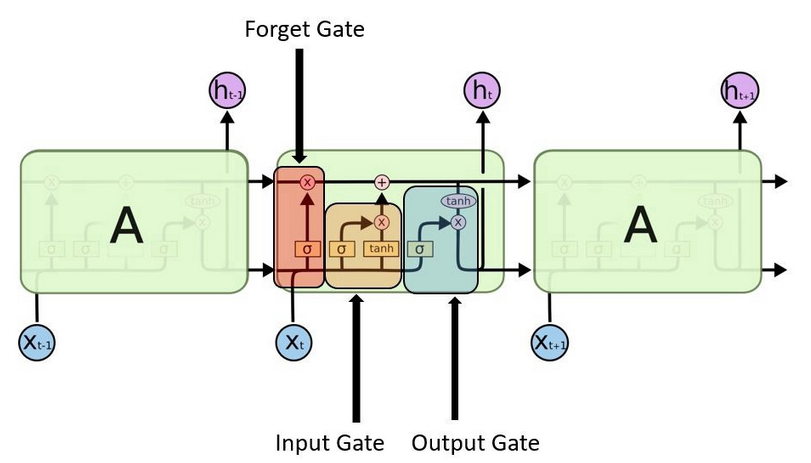
Long short-term memory (LSTM)



<https://docs.chainer.org/en/stable/examples/seq2seq.html>

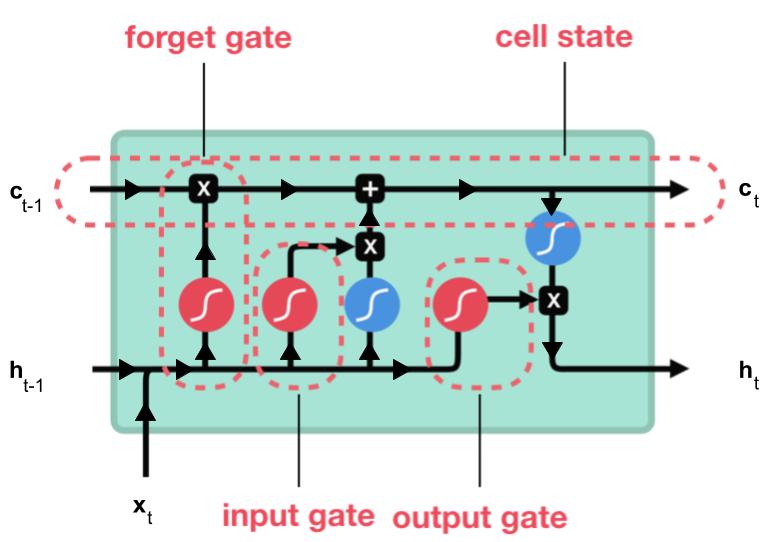
2 layers encoder : embedded, recurrent

3 layers decoder : embedded, recurrent, output



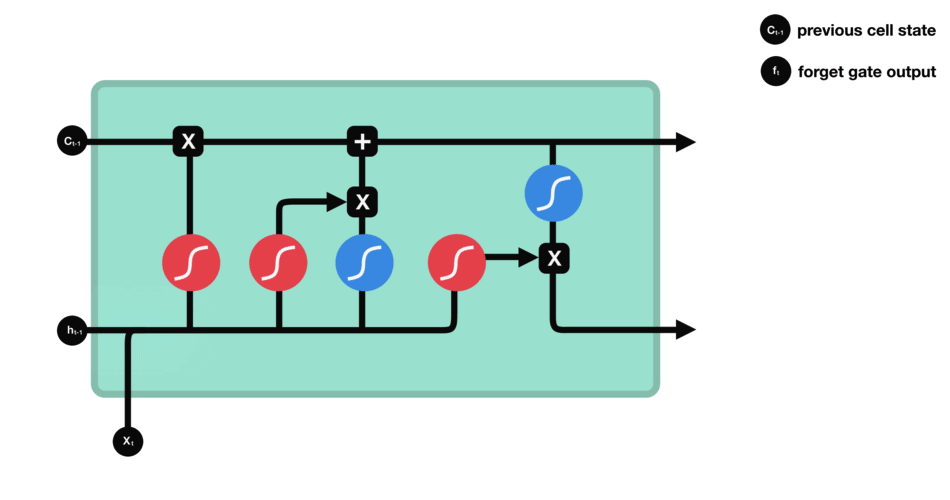
<https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e>

Un LSTM est composé de trois portes (forget gate, input gate et outputgate) et de deux sorties (hidden state, cell state). Les portes peuvent être vus comme des zones de calculs.



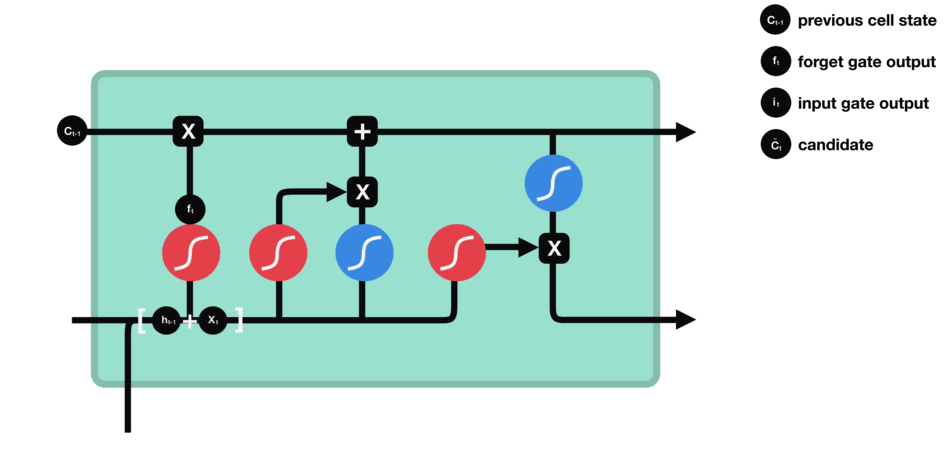
*Cellule LSTM*

La porte d’oubli (**forget gate**) décide de quelle information doit être conservée ou jetée. Il s’agit de l’information de l’état caché précédent concaténée à la donnée en entrée, à laquelle on lui applique une sigmoïde (en rouge) afin de normaliser les valeurs entre 0 et 1. Quand la valeur est proche de 0, l’information doit être oubliée, tandis que lorsqu’elle est proche de 1, l’information doit être mémorisée.



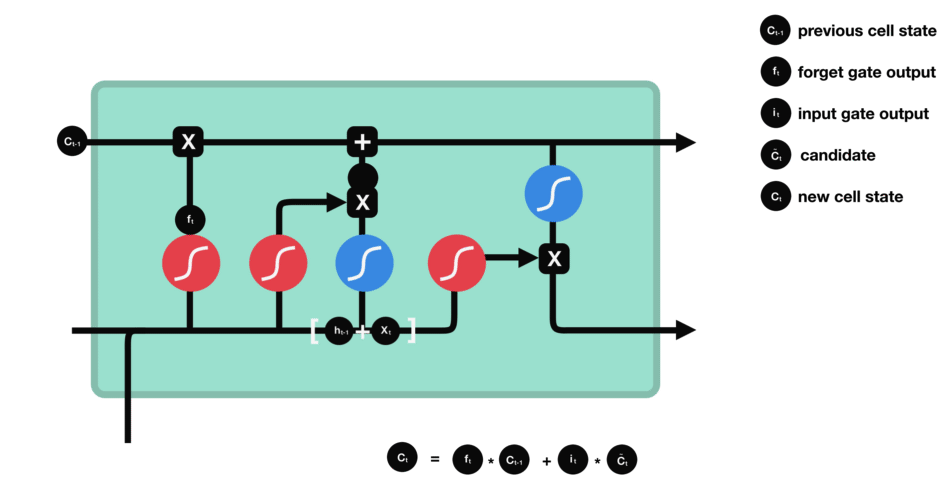
*Fonctionnement d’une porte d’oubli d’un LSTM*

La porte d’entrée (**input gate**) va extraire l’information de la donnée courante. Pour cela, on applique une sigmoïde et un tanh aux deux données concaténées. D’un côté, on a la sigmoïde (en rouge) qui va nous dire si une information est importante (quand la valeur d’une coordonnée d’un vecteur est proche de 1) ou non (proche de 0), et d’un autre côté, on a la fonction tanh (en bleu) qui va normaliser les valeurs entre -1 et 1 pour éviter d’avoir de trop gros chiffres et surcharger l’ordinateur en calculs. On fait ensuite le produit des deux vecteurs, qui nous permettra de garder uniquement les informations importantes (car multipliée par des valeurs proches de 1 si importante, et par des valeurs proches de 0 sinon)



*Fonctionnement de la porte d’entrée d’un LSTM*

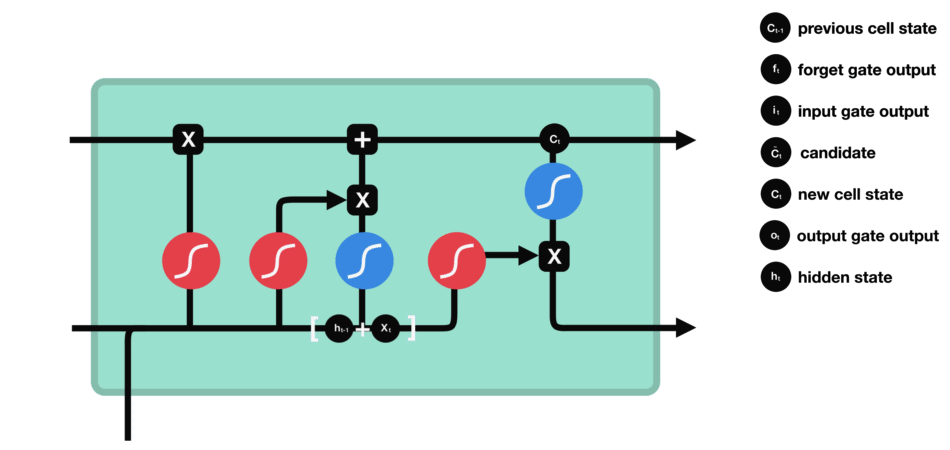
L’état de la cellule (**cell state**) se calcul assez facilement. On part tout d’abord de la porte d’oubli et de l’ancien état de la cellule, en faisant une multiplication de ces deux vecteurs. Ceci permet d’oublier certaines informations. On récupère ensuite ce résultat, et faisons une addition avec la sortie de la porte d’entrée, permettant d'enregistrer dans l’état de la cellule l’information importante.



*Mise à jour de l’état de la cellule d’un LSTM*

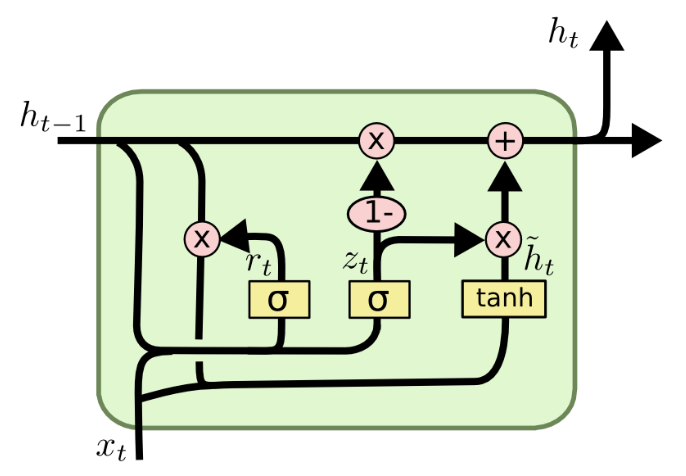
Enfin, la porte de sortie (**output gate**) décide du prochain état caché (hidden state), information qui sert aux prochaines prédictions du modèle.

Dans un premier temps, on récupère le nouvel état de la cellule (cell state) sur lequel on applique la fonction tanh (en bleu) pour normaliser les valeurs entre -1 et 1. On applique ensuite la fonction sigmoïde (en rouge) au vecteur concaténé de l’entrée courante avec l’état caché précédent qui nous permet de décider des informations à conserver (proche de 0 on oublie, proche de 1 on conserve dans l’état de la cellule).

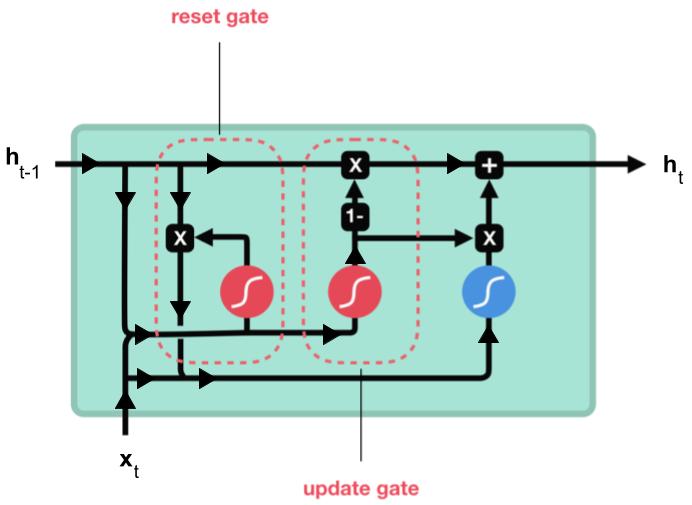


*Fonctionnement de la porte de sortie d’un LSTM*

Gated Recurrent Unit (GRU)



Un GRU est composé de deux portes (reset gate et update gate) et d’une sortie (cell state).

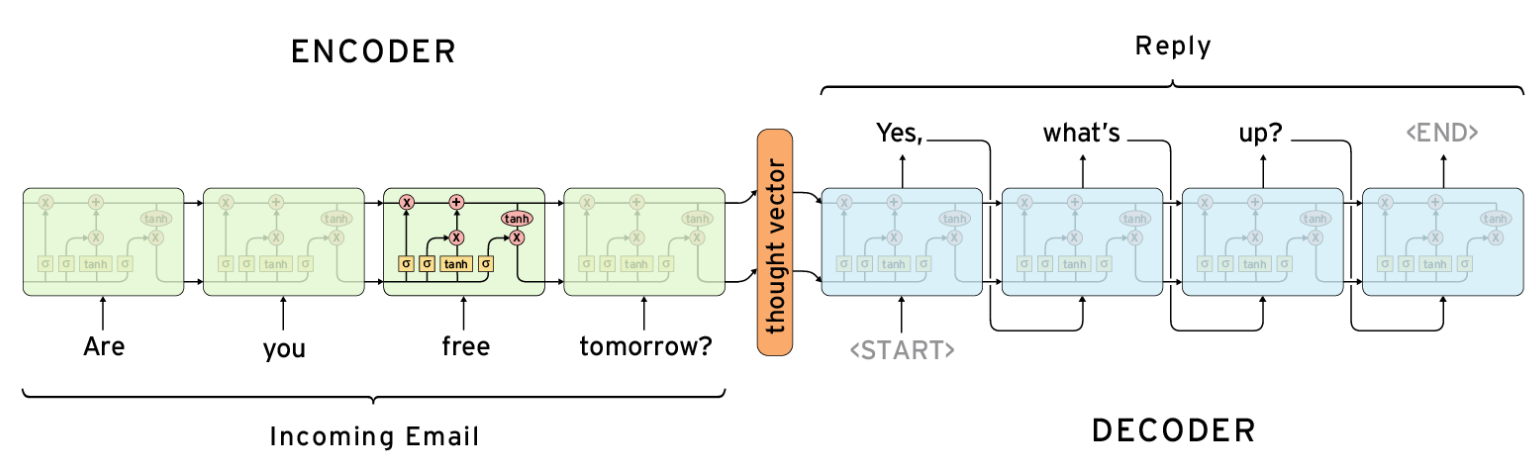


*Cellule GRU*

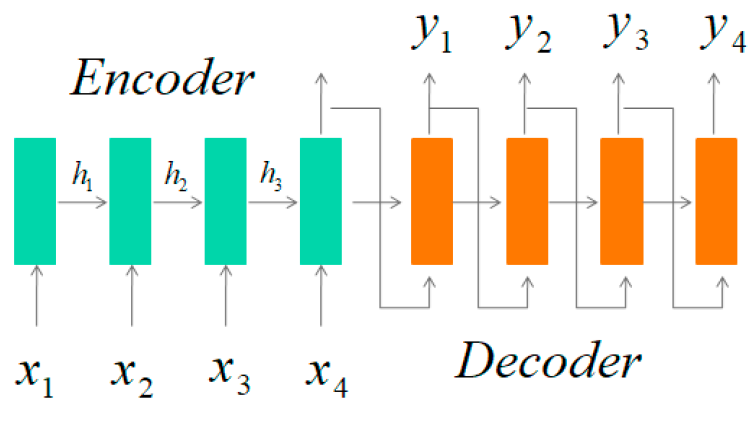
On retrouve une porte de reset (**reset gate**) qui sert à contrôler combien d’information passée le réseau doit oublier. L’état caché (hidden state) précédent, concaténée avec les données d’entrées, passent par une sigmoïde (pour ne conserver que les données intéressante), puis cette valeur est multipliée avec l’état caché précédent. On conserve donc uniquement les données importantes de l’état précédent.

La porte de mise à jour (**update gate**) décide des informations à oublier et à conserver, comme dans un LSTM. L’ancien état caché (hidden state) concaténée avec les données d’entrées passent par une sigmoïde qui détermine les informations importantes.

L’état caché précédent (partiellement effacé par la porte de reset) est concaténée à l’entrée du réseau et normalisée entre -1 et 1 par une fonction tanh. Depuis la sortie de la porte de mise à jour, on va annuler les coordonnées jugées inutiles pour les prédictions, et on y ajoute les coordonnées de l’état caché précédent jugées inutiles.



<https://towardsdatascience.com/nlp-sequence-to-sequence-networks-part-2-seq2seq-model-encoderdecoder-model-6c22e29fd7e1>

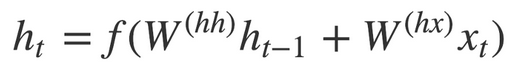


**Encoder**

L’encoder va récupérer le *contexte* de la séquence en entrée sous la forme d’un *vecteur d’état caché*, ainsi que la valeur d’entrée pour générer un nouveau vecteur d’état caché et l’envoyer au decoder pour produire la séquence en sortie.

En général il sera composé de réseaux de type RNN comme LSTM (Long Short-Term Memory) ou GRU (Gated Recurrent Units).

Chaque état caché sera calculé avec la formule :



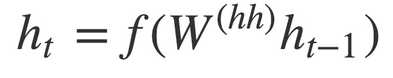
avec ht l’état caché, W le poids et xt le vecteur d’entrée

L’état caché final en sortie de l’encoder est donc un vecteur qui sera en même temps état caché initial du decoder. Ce vecteur va encapsuler les informations de l’ensemble des éléments en entrée pour permettre au decoder de faire ses prédictions.

**Decoder**

La couche decoder prend en entrée le vecteur généré à la sortie de l’encoder. Elle sera également créée avec les architectures RNN (LSTM, GRU).

Chaque unité récurrente prendra un état caché précédent en entrée et produira un autre état caché en sortie, selon la formule :



avec ht-1 l’état caché précédent

Le vecteur de sortie sera alors calculé avec la formule suivante :

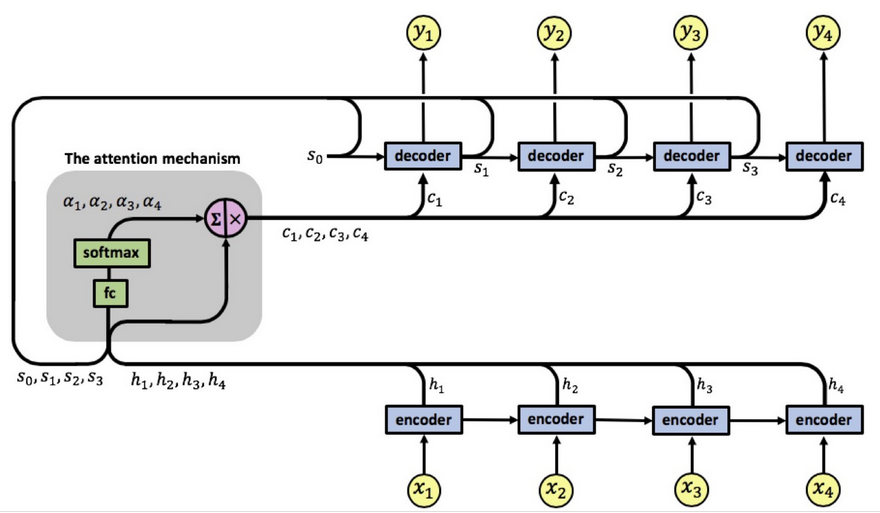


avec Ws leur poids respectif

La séquence en sortie va alors énormément dépendre du contexte défini par l’état caché à la sortie de l’encoder. Il est donc possible de perdre le contexte initial si la séquence devient trop longue. La solution du mécanisme de l’*Attention* a été introduite par les papiers de [Bahdanau et al., 2014](https://arxiv.org/abs/1409.0473) ainsi que [Luong et al., 2015](https://arxiv.org/abs/1508.04025), permettant de conserver le contexte du début jusqu'à la fin.

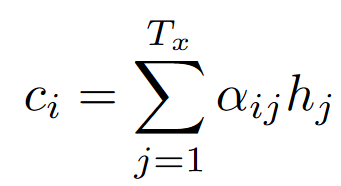
**Attention**

Le mécanisme d’attention permet au modèle de se concentrer sur une partie de la séquence d’entrée à chaque étape de la génération de la séquence de sortie. Au lieu d’alimenter le decoder avec un vecteur contexte de longueur fixe, un vecteur de contexte dynamique sera utilisé, différent sur chaque étape. L’idée est de prendre en compte uniquement les unités les plus pertinentes en entrée selon l’unité à générer en sortie.

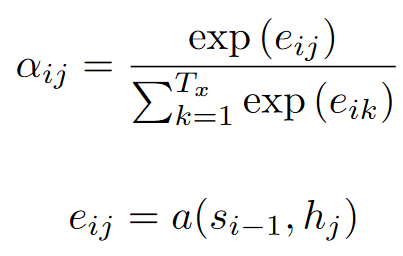


<https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05>

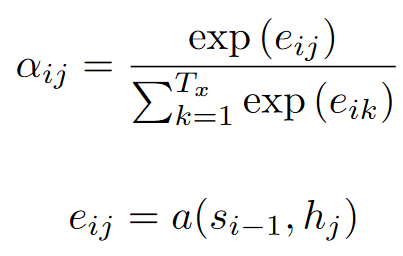
Le vecteur de contexte se calcule :



Chaque poids est calculé par :



avec



Andrew NG - C5W3L08 Attention Model

<https://www.youtube.com/watch?v=quoGRI-1l0A>

papiers sur RNN encoder-decoder :

· Sutskever, I., Vinyals, O., and Le, Q. (2014). Sequence to sequence learning with neural networks.InAdvances in Neural Information Processing Systems (NIPS 2014)<https://arxiv.org/pdf/1409.3215.pdf>

· Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder-decoder for statistical machine translation.InProceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014). to appear.<https://arxiv.org/pdf/1406.1078.pdf>

papier sur le mécanisme de l’Attention :

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate.arXiv preprint arXiv:1409.0473, 2014 <https://arxiv.org/abs/1409.0473>