Séries temporelles
Méthodes ML et DL
Modèles Deep Learning

Intervenant
Guillaume Hochard

5.1. Introduction 1/2

Promesses du deep learning

- Les réseaux de neurones (NN) peuvent ne pas avoir besoin d'une série temporelle normalisée ou stationnaire comme entrée
- Les NN prennent en charge les entrées multivariées et prennent en charge les sorties multi-étapes
- Les réseaux LSTM permettent un apprentissage efficace des dépendances temporelles
- Les réseaux neuronaux convolutionnels (CNN) permettent un apprentissage efficace des features
- Ces capacités peuvent également être combinées, comme dans l'utilisation de modèles hybrides tels que les CNN-LSTM et les ConvLSTM
- Les modèles hybrides combinent efficacement les diverses capacités de ces différentes architectures

5.1. Introduction 2/2

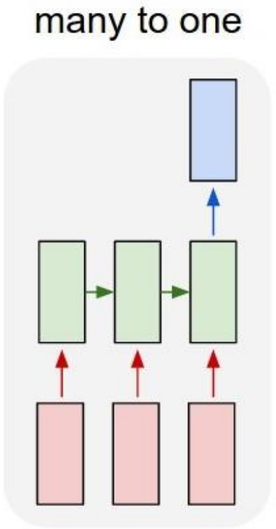
Promesses du deep learning

- Même si cela ne s'avère pas absolument nécessaire, une bonne pratique est d'identifier et de supprimer manuellement les structures systématiques des données de séries temporelles pour faciliter la modélisation du problème (par exemple, rendre la série stationnaire)
- Bonne pratique mais pas une exigence : techniquement, le contexte disponible de la séquence fournie en entrée peut permettre aux modèles de réseaux neuronaux d'apprendre directement la tendance et la saisonnalité.
- Bien que les MLP puissent opérer directement sur des observations brutes, les CNN offrent une efficacité et des performances bien supérieures pour apprendre automatiquement à identifier, extraire et traiter les variables utiles à partir de données brutes.

5.2 Deep Learning : rappels 1/6

L'approche de traitement de séries temporelles avec des modèles de deep learning est fondée sur l'analyse de séquences et de patterns. En cela, de nombreuses méthodes sont partagées avec le NLP (RNN) et le traitement d'images (CNN).

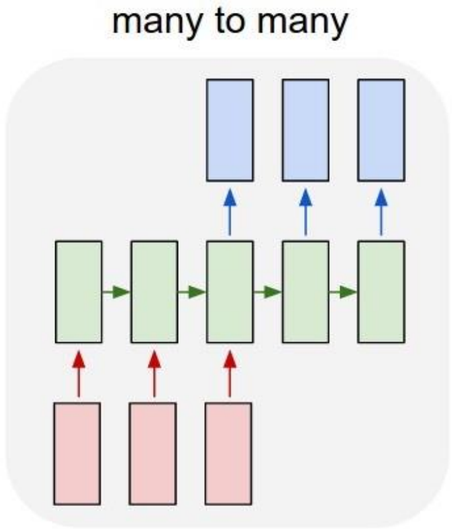
Différentes approches pour traiter une séquence en séries temporelles (prévision)



Prévision one-step

Exemple : prévoir 1 pas de temps à t+1, t+10, …

La sortie est un scalaire ou un vecteur (cas multivarié)



Prévision multi-step

Exemple : prévoir plusieurs pas de temps, de t+1 à t+7, …

La sortie est un vecteur ou une matrice (cas multivarié)

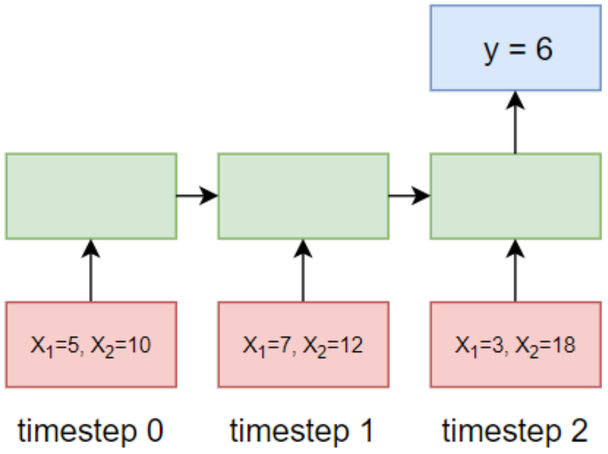
5.2 Deep Learning : rappels 2/6

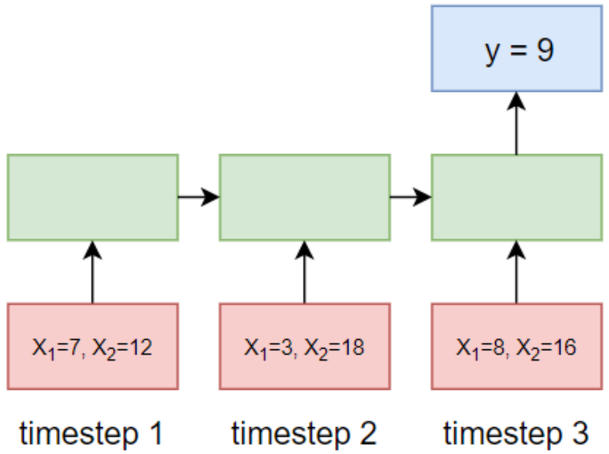
Many to one

Un RNN est utilisé pour ingérer des séquences (features sur une suite de plusieurs timestamps).

Exemple sur un RNN ayant 3 cellules (hidden cells) :

	features		target
timesteps	X_1	X_2	y
timestep 0	5	10	7
timestep 1	7	12	4
timestep 2	3	18	6
timestep 3	8	16	9
timestep 4	2	11	2
timestep 5	9	21	1

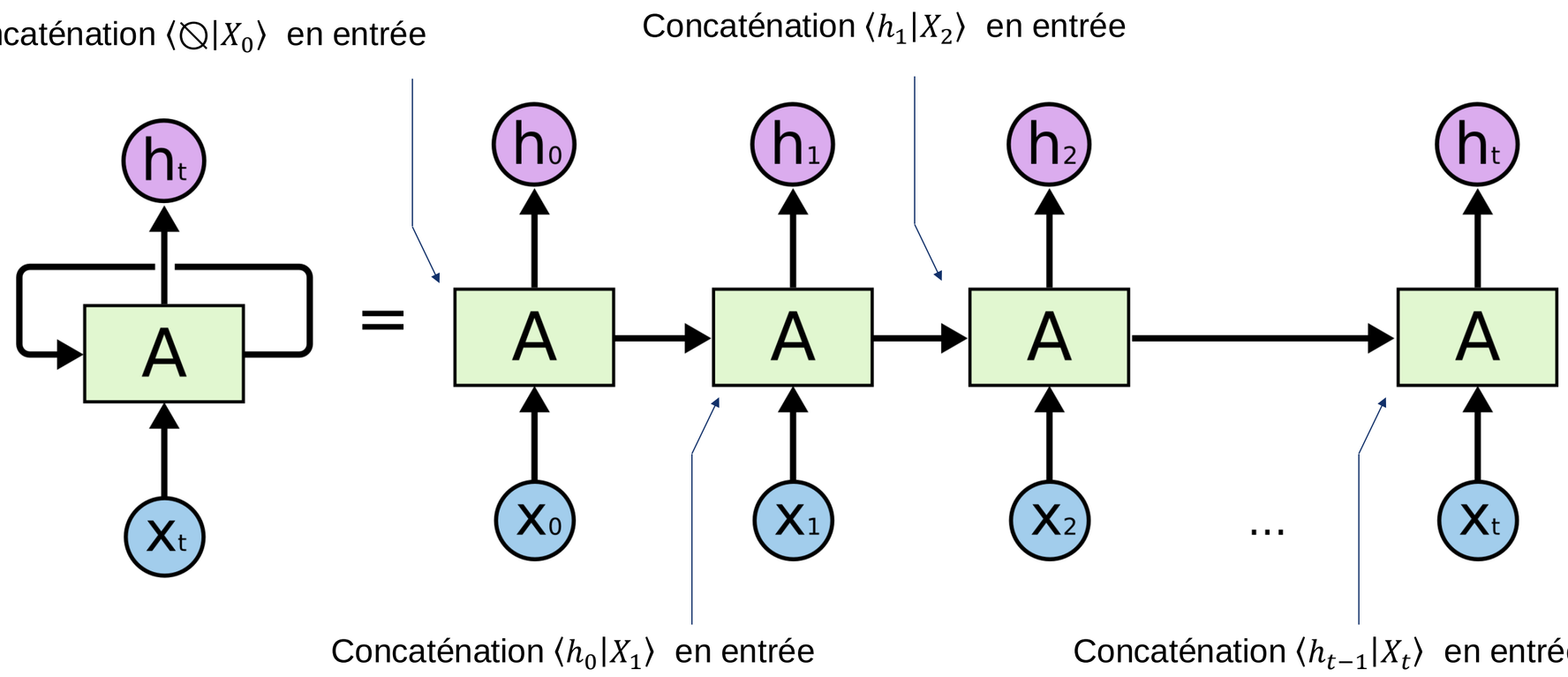




5.2 Deep Learning : rappels 3/6

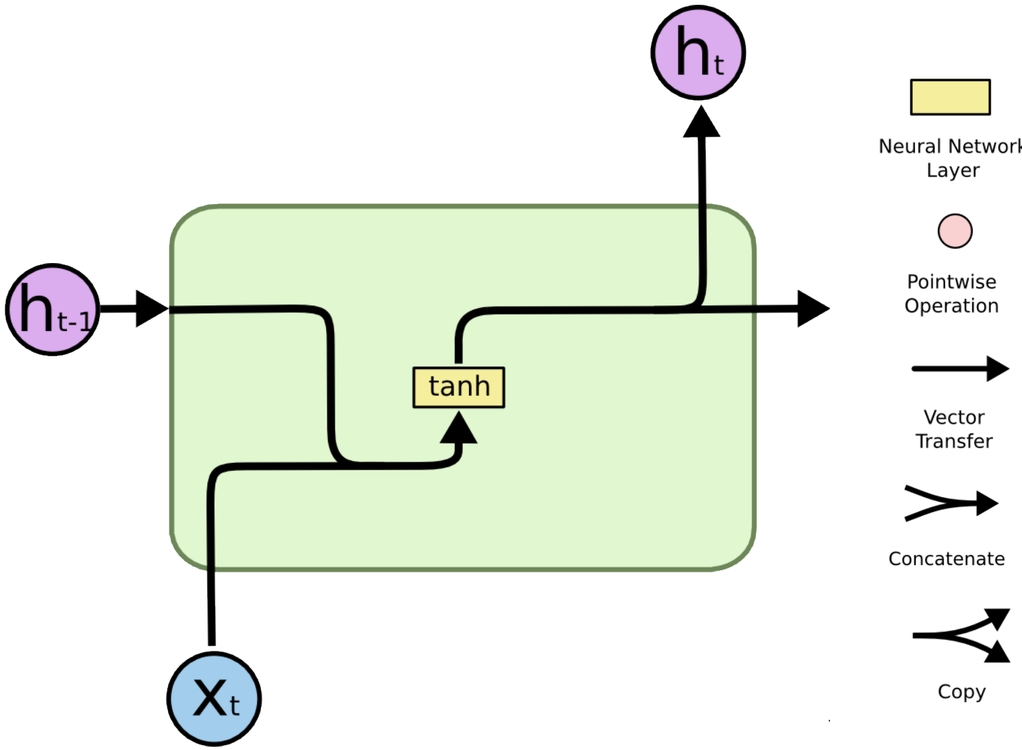
Qu’est-ce qu’une architecture récurrente ?

C’est un réseau de neurones qui boucle sur lui-même : sa sortie est utilisée en entrée plusieurs fois de suite.



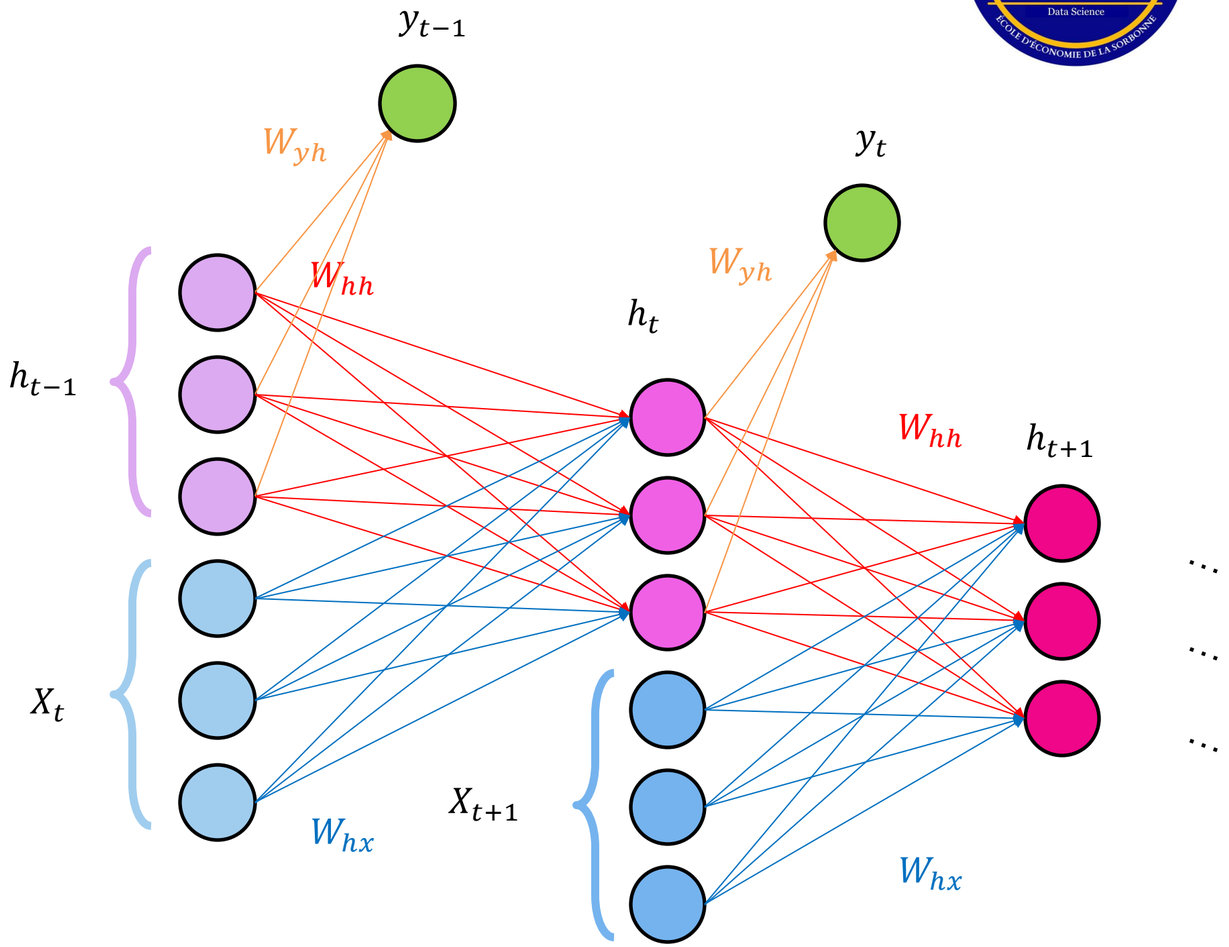
5.2 Deep Learning : rappels 4/6

Principe de la cellule récurrente (vanilla)



$$h_t = g_a(W_{hh}h_{t-1} + W_{hx}X_t + b_h)$$

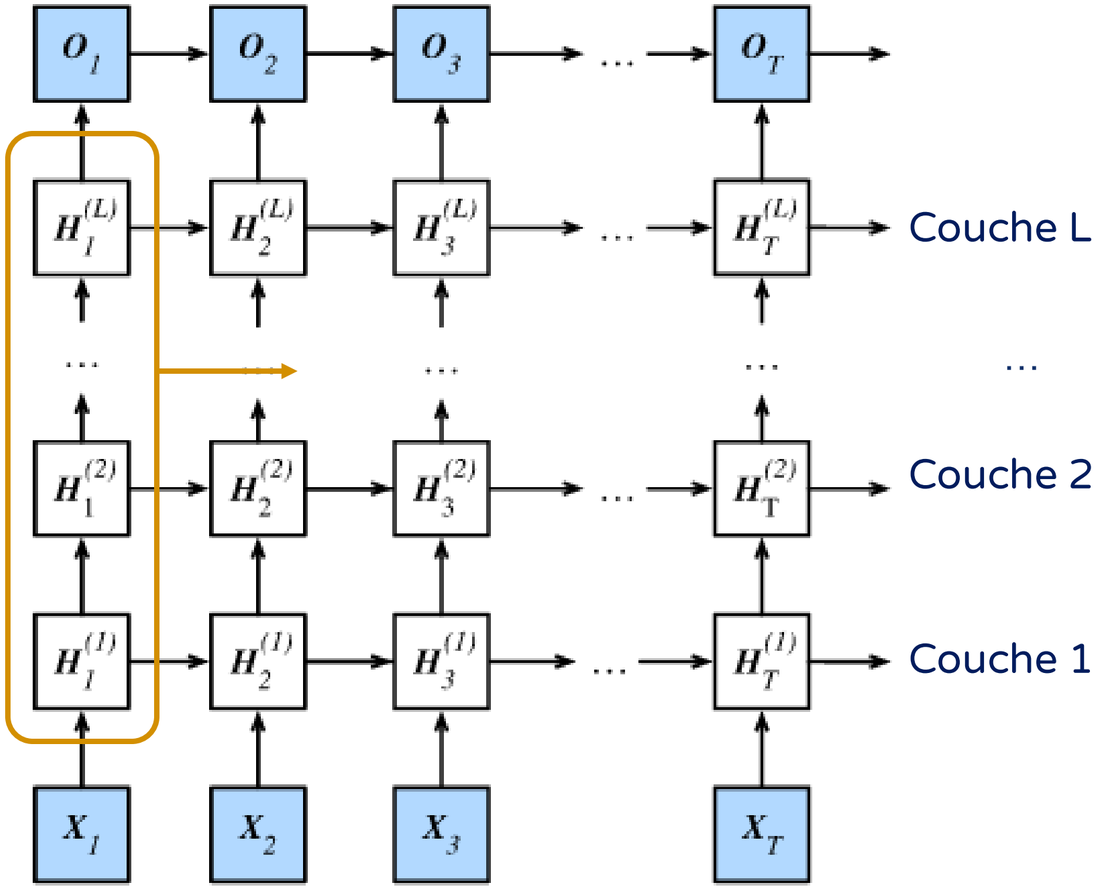
$$\hat{y}_t = g_y(W_{yh}h_t + b_y)$$



Partage des mêmes coefficients à travers les différents éléments de la séquence

5.2 Deep Learning : rappels 5/6

RNN profond



- Un réseau récurrent profond est un réseau où les couches récurrentes s'empilent les unes sur les autres
- La séquence de prédictions de la couche n alimente la séquence d'entrée de la couche $n + 1$
- Les problèmes d'extinction ou d'explosion de gradient s'en trouvent augmentés
- Les réseaux récurrents profonds sont souvent combinés à des méthodes de gradient clipping

5.2. Deep Learning : rappels 6/6

Explosion et extinction du gradient

Lors de l'entrainement d'un RNN profond, via l'algorithme de backpropagation, 2 problèmes se présentent fréquemment

L'explosion du gradient

- La contribution à la dérivé de l'erreur par rapport aux paramètres (gradient) peut grandir exponentiellement avec le nombre de couches et devenir très grand pour les cellules lointaines.
- Identification : les paramètres du réseau deviennent très grands (des NAN peuvent apparaitre)
- Résolution : Gradient clipping (redimmensionnement du vecteur de gradient)

L'extinction du gradient

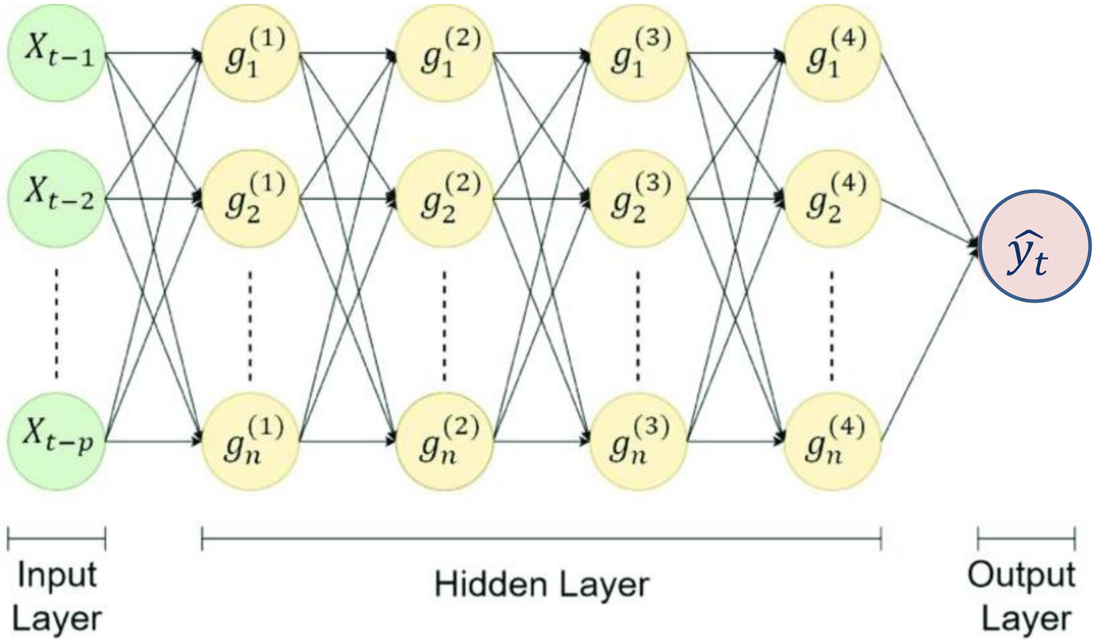
- La contribution au gradient peut réduire exponentiellement avec le nombre de couches et devenir très petit pour les cellules lointaines (~0)
- Ce problème est bien plus génant que l'explosion du gradient

Un RNN classique n'est pas performant pour apprendre les dépendances entre des inputs lointains dans la séquence

5.3. MLP

Multi Layer Perceptron

- Un réseau de neurones non récurrent de type MLP peut également être utilisé pour des tâches de prévision
- Les features et les pas de temps doivent être concaténés dans un vecteur d'entrée qui est ensuite présenté au réseau
- Moins couteux en temps de calcul
- Pas adapté pour capturer des dépendances temporelles



5.4. RNN : LSTM & GRU 1/10

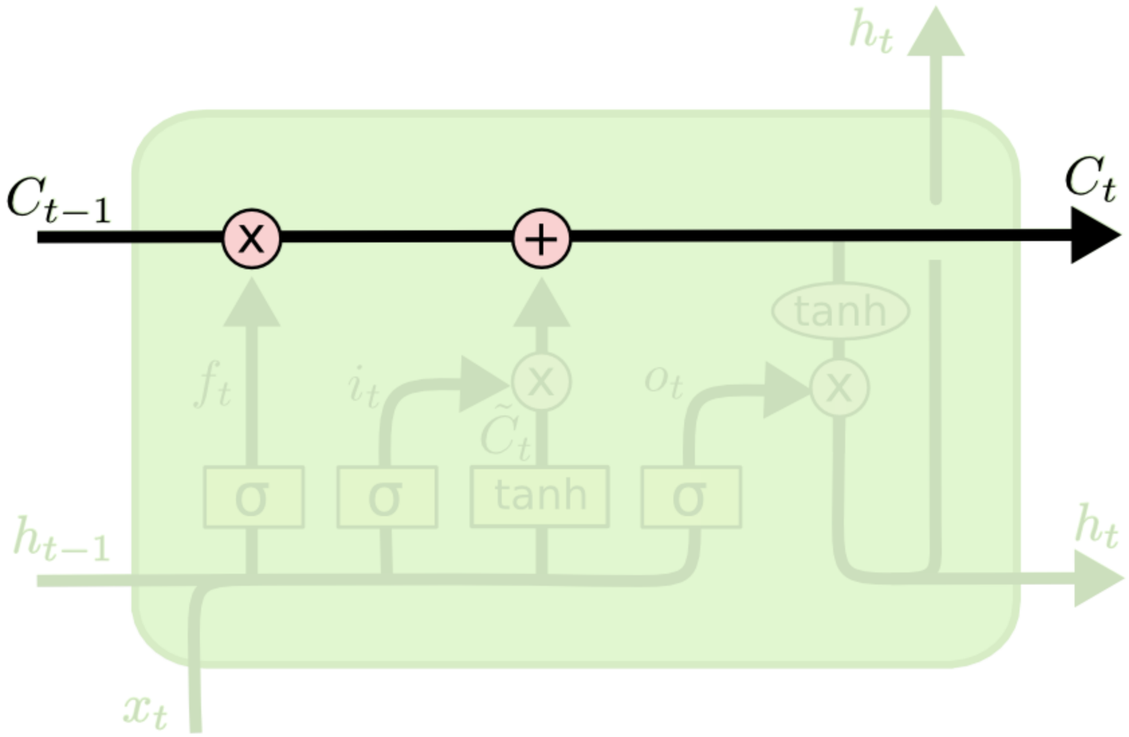
- Les LSTMs (Hochreiter & Schmidhuber, 1997) et les GRUs (Cho et al., 2014a) sont deux types de réseaux de neurones récurrents qui permettent de répondre aux problèmes d'extinction ou d'explosion du gradient.

Les GRUs est une variante légèrement plus simplifiée qui offre souvent des performances comparables et qui est nettement plus rapide à calculer.

- Les portes de réinitialisation (reset gate) aident à capturer les dépendances à court terme dans les séquences
- Les portes de mise à jour (update gate) aident à capturer les dépendances à long terme dans les séquences

5.4. RNN : LSTM & GRU 2/10

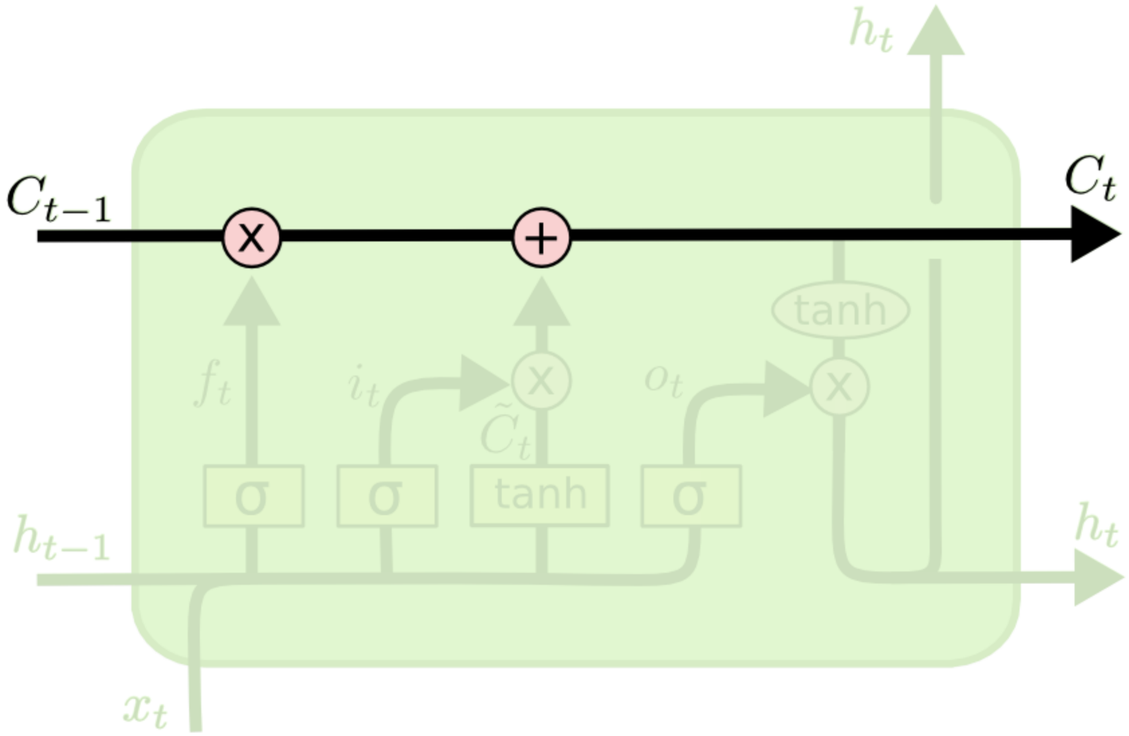
LSTM (Long Short Term Memory)



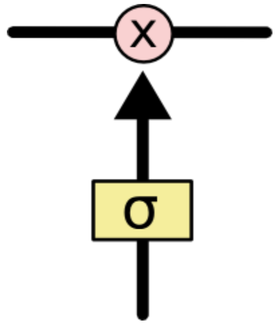
- Cell state C : au cœur du concept des LSTM
- Permet de transférer de l'information d'une cellule à l'autre

5.4. RNN : LSTM & GRU 3/10

LSTM (Long Short Term Memory)

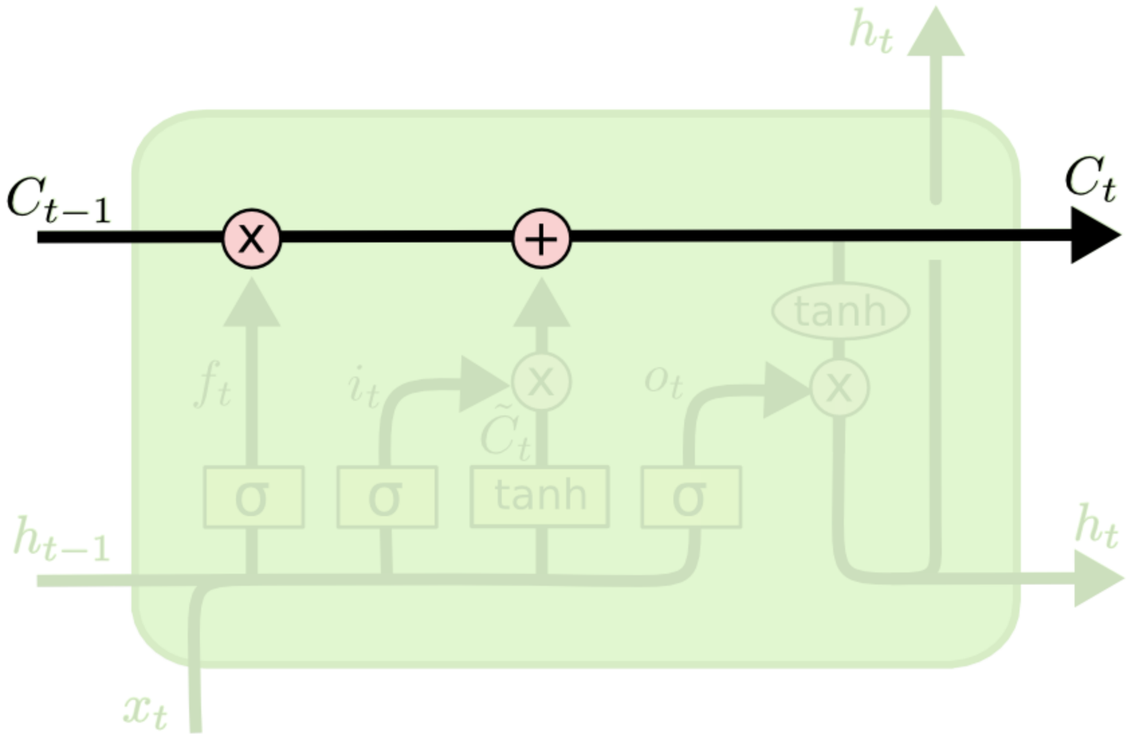


- Gates (ou portes) : moyen de laisser passer (éventuellement) des informations. Elles sont composées d'une sigmoïde et d'une opération de multiplication ponctuelle.



5.4. RNN : LSTM & GRU 4/10

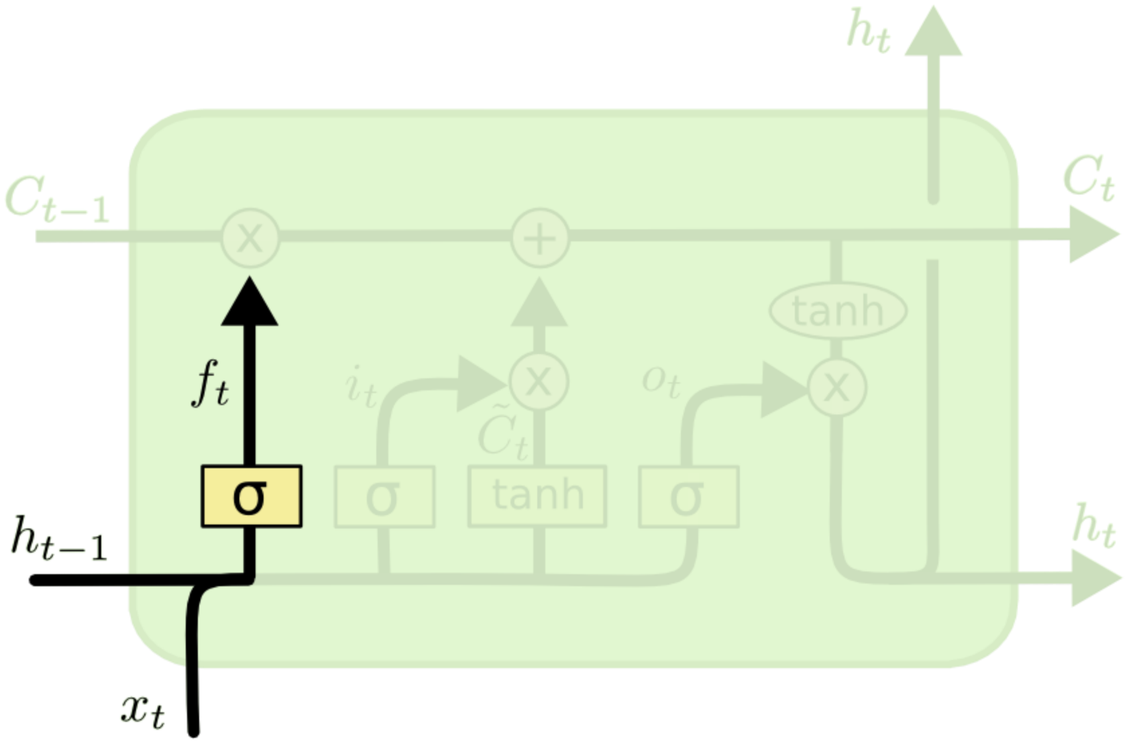
LSTM (Long Short Term Memory)



- Gates
- LSTM possède 3 portes, pour protéger et contrôler l'état de la cellule
- Forget gate : que veut-on garder de la cellule précédente ?
- Input gate : que veut-on garder de la nouvelle cellule ?
- Output gate : comment mettre à jour l'état caché ?

5.4. RNN : LSTM & GRU 5/10

LSTM (Long Short Term Memory)

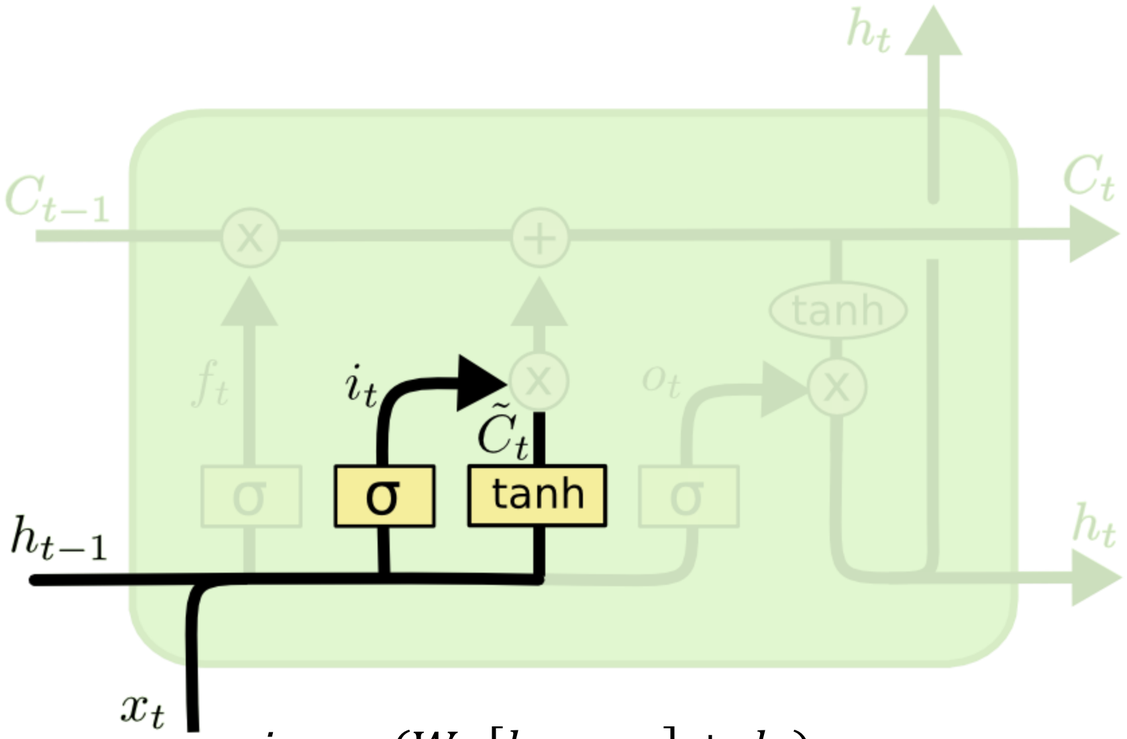


$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f)$$

- Forget gate : que veut-on garder de la cellule précédente ?
- Cette décision est prise par une couche sigmoïde appelée « forget gate layer »
- Examine ht-1 et xt, et produit un nombre entre 0 et 1 pour chaque nombre dans la cellule Ct-1 (cell state).
- Un 1 équivaut à "garder complètement ceci" tandis qu'un 0 représente "se débarrasser complètement de ceci".

5.4. RNN : LSTM & GRU 6/10

LSTM (Long Short Term Memory)



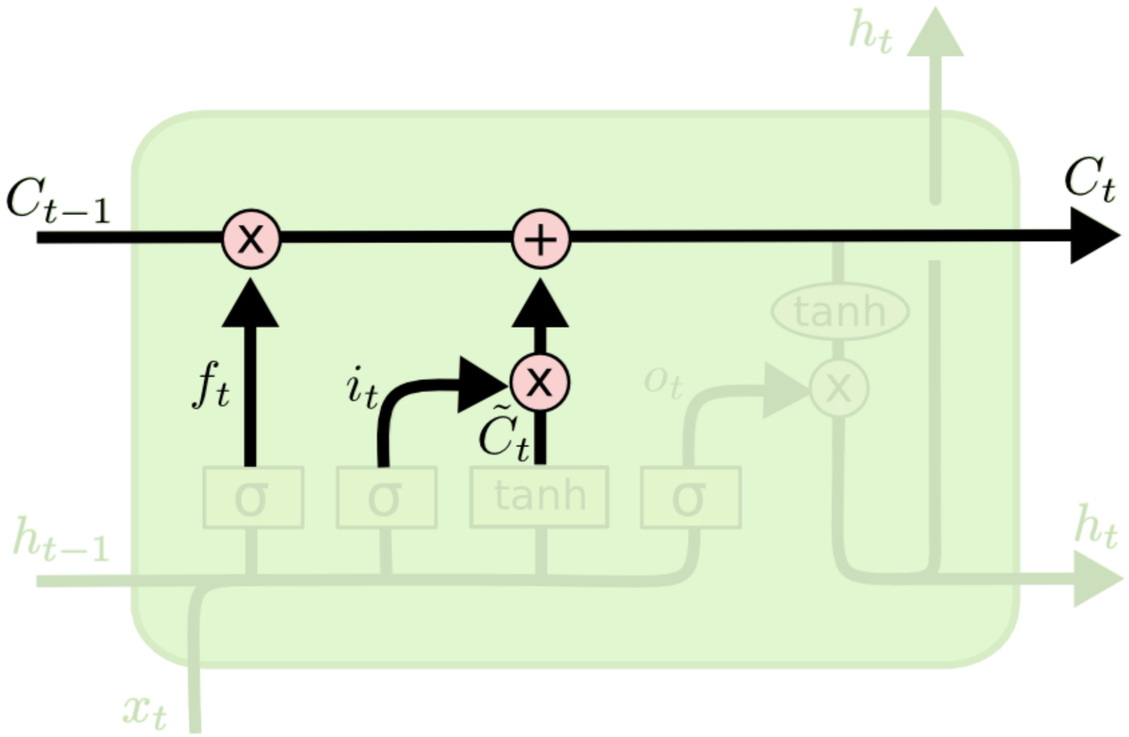
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C)$$

- Input gate : que veut-on garder de la nouvelle cellule ?
- Stockage dans l'état de la cellule (cell state)
- Processus comporte deux parties
- 1. Une sigmoïde appelée « input gate layer » décide des valeurs à mettre à jour.
- 2. Une tanh crée un vecteur de nouvelles valeurs candidates, C~t, qui pourraient être ajoutées à l'état

5.4. RNN : LSTM & GRU 7/10

LSTM (Long Short Term Memory)

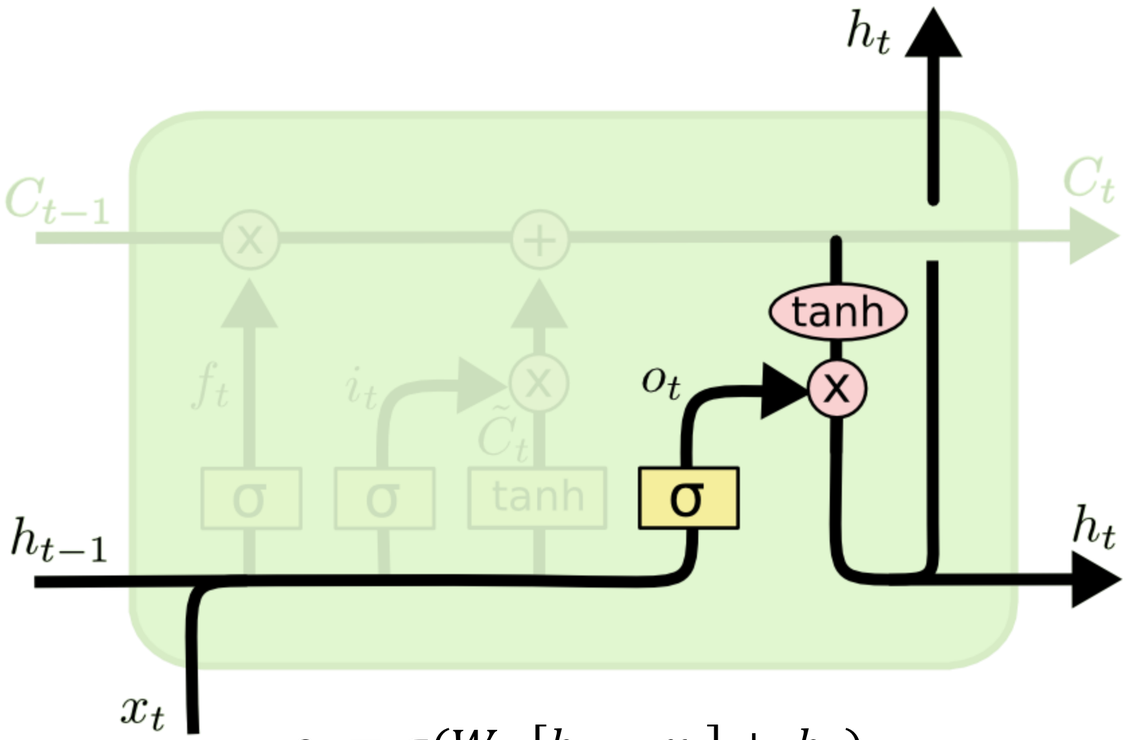


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Mise à jour du cell State C_{t-1} en C_t
- Combinaison des deux étapes précédentes pour créer une mise à jour de l'état
- Multiplication de l'ancien état par f_t, en oubliant les choses que nous avons décidé d'oublier plus tôt
- Puis ajout de $i_t * \tilde{C}_t$: nouvelles valeurs candidates, mises à l'échelle par combien nous avons décidé de mettre à jour chaque valeur d'état

5.4. RNN : LSTM & GRU 8/10

LSTM (Long Short Term Memory)



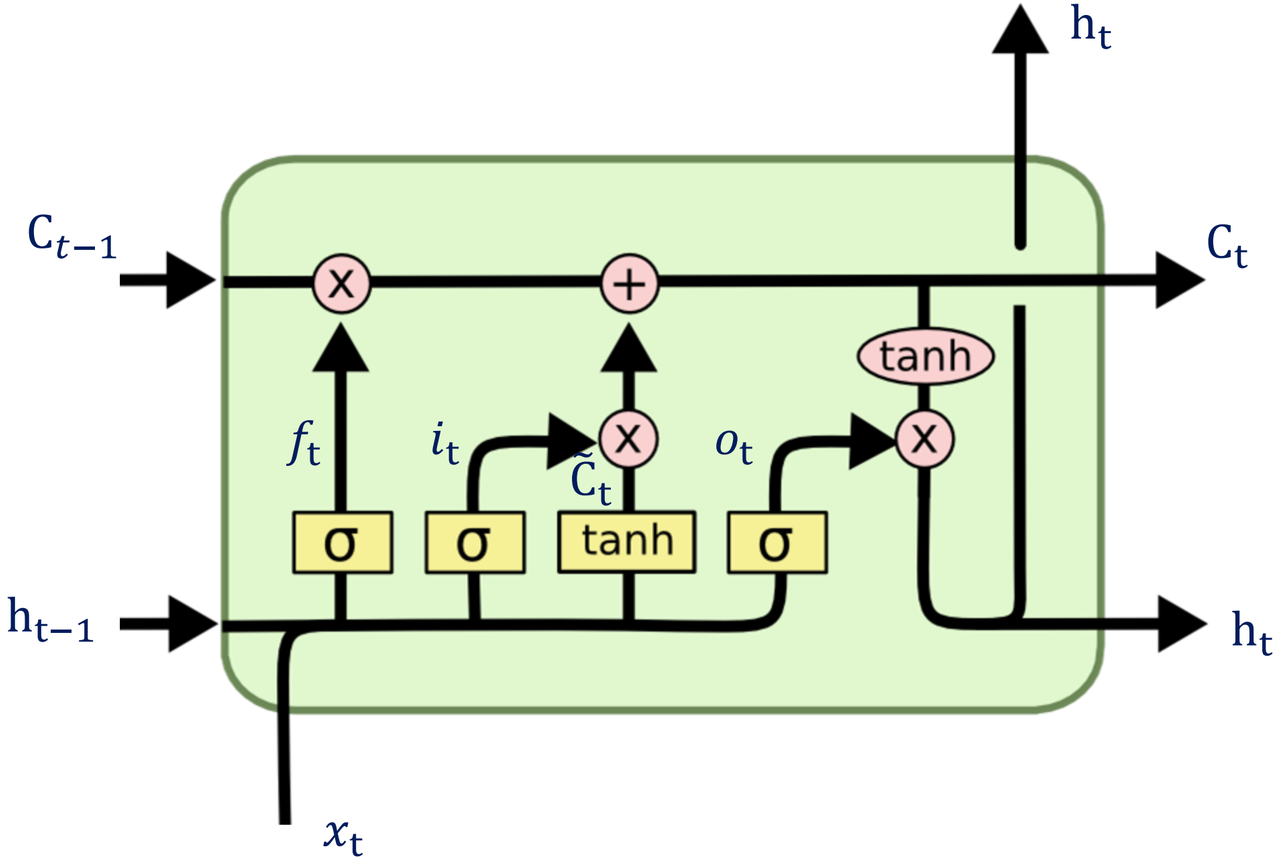
$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

- Génération de la sortie ht
- Basée sur l'état de cellule C_t , mais est une version filtrée
- Couche sigmoïde qui décide des parties de l'état de la cellule que nous allons avoir en sortie
- On fait passer l'état de la cellule par tanh (sortie entre -1 et 1) et on le multiplie par la sortie de la porte sigmoïde, afin de ne sortir que les parties précédemment sélectionnées

5.4. RNN : LSTM & GRU 9/10

LSTM (Long Short Term Memory)



Cellule complète

Récapitulatif des différentes étapes, paramètres et équations

$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f)$ (forget)

$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i)$ (input)

$\tilde{C}_t = \tanh(W_C . [h_{t-1}, x_t] + b_C)$

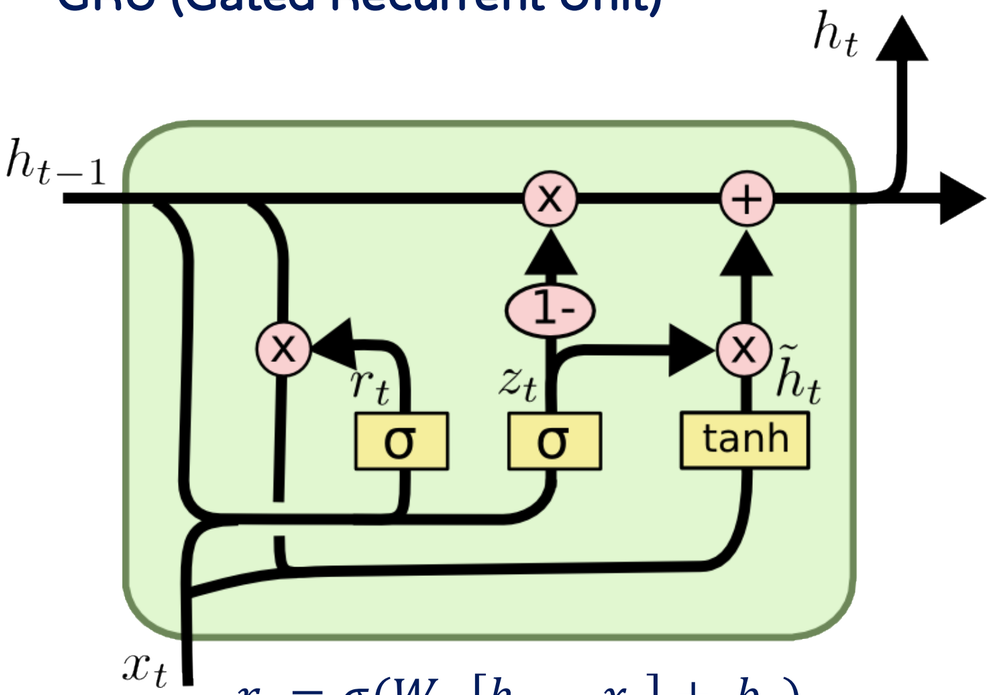
$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$ (cell state)

$o_t = \sigma(W_o . [h_{t-1}, x_t] + b_o)$ (output)

$h_t = o_t * \tanh(C_t)$ (hidden state)

5.4. RNN : LSTM & GRU 10/10

GRU (Gated Recurrent Unit)



$$r_t = \sigma(W_r.[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W.[r_t * h_{t-1}, x_t] + b_h)$$

$$z_t = \sigma(W_z.[h_{t-1}, x_t] + b_z)$$

$$h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$$

Reset gate : que veut-on garder de l'état précédent (court terme) ?

- Quand la sortie est proche de 1, on préserve l'information
- Quand la sortie est proche de 0, on ignore l'information
- Cela permet de moduler l'influence de l'état précédent

Nouvel état : comment mettre à jour l'état de la cellule ?

- On se base sur l'information filtrée par la reset gate r_t
- On se base sur le nouvel élément de la séquence

Update gate z_t : que veut-on garder du nouvel état (long terme) ?

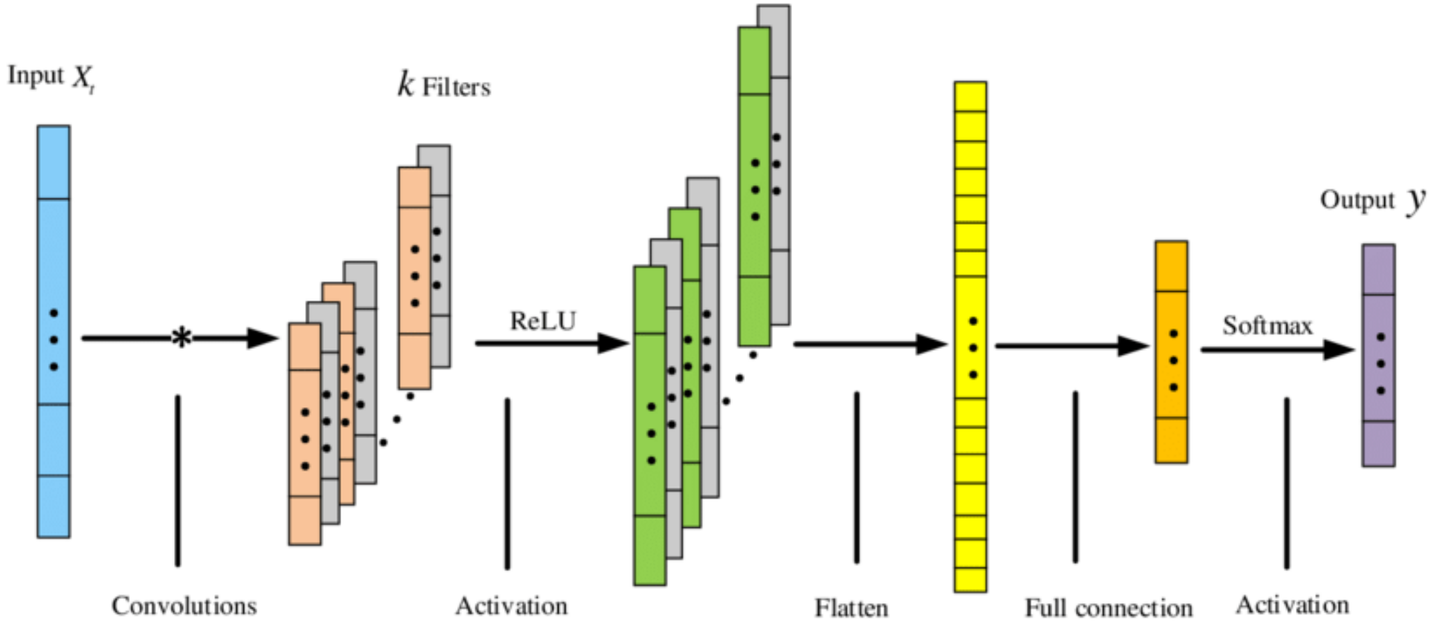
- Quand la sortie est proche de 1, on censure le nouvel état
- Quand la sortie est proche de 0, on tient compte du nouvel état
- Cela permet de moduler l'influence du nouvel état h_t

- Moins de paramètres que pour LSTM (1 porte en moins)
- Performance parfois meilleure

5.5. CNN 1/7

Les réseaux de neurones convolutifs (CNN)

- Développés et sont restés très populaires dans le domaine de la classification d'images
- Application possible à des problèmes unidimensionnels, tels que la prédiction de la valeur suivante dans une séquence (forecast t+1, prochain mot d'une phrase)



5.5. CNN 2/7

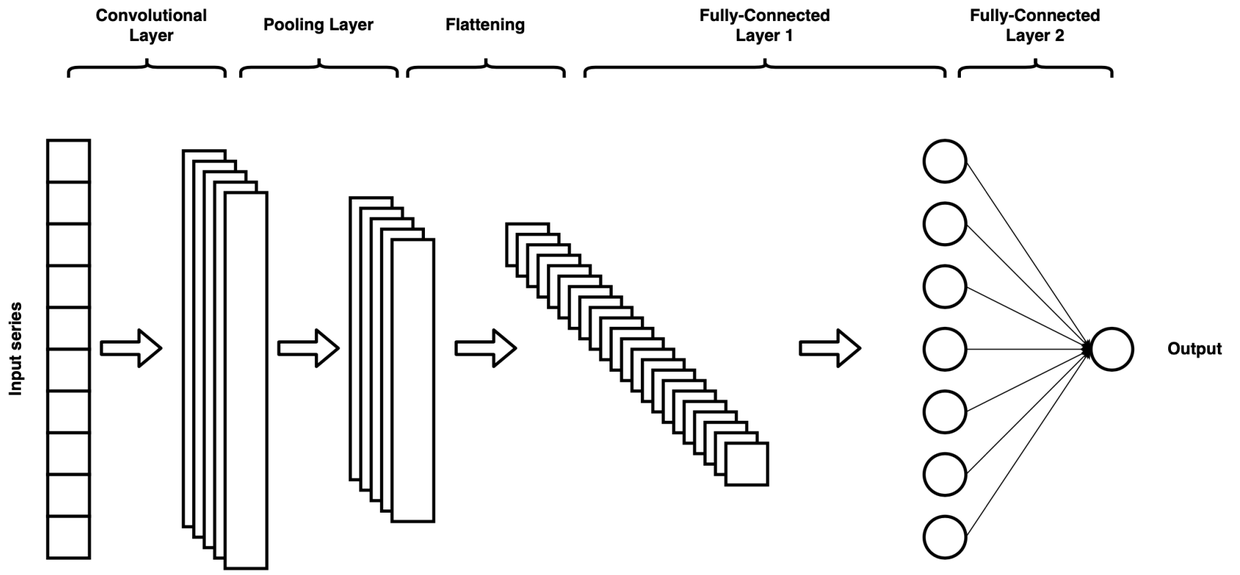
Avantages des CNN

- Efficaces pour découvrir des caractéristiques dans des segments de longueur fixe issus des données (par exemple, prédire la valeur suivante sur la base des n observations précédentes), principalement dans les cas où l'emplacement de la caractéristique dans le segment n'a pas d'importance
- Capables d'extraire des features qui sont indépendantes de la composante temporelle (invariant par translation). Le réseau peut identifier un motif à un endroit de la série (e.g. au début), puis le localiser à un autre endroit (e.g. à la fin de la séquence) et l'utiliser pour faire une prédiction de la cible
- Le processus d'apprentissage des caractéristiques supprime le besoin de connaissances du domaine pour le feature engineering
- Les réseaux 1D permettent des filtres de plus grande taille que dans un cas 2D
- Les CNN sont considérés comme résistants au bruit dans les données
- Les CNN 1D sont moins coûteux en termes de calcul que les RNN et sont parfois plus performants

5.5. CNN 3/7

Les réseaux de neurones convolutifs (CNN)

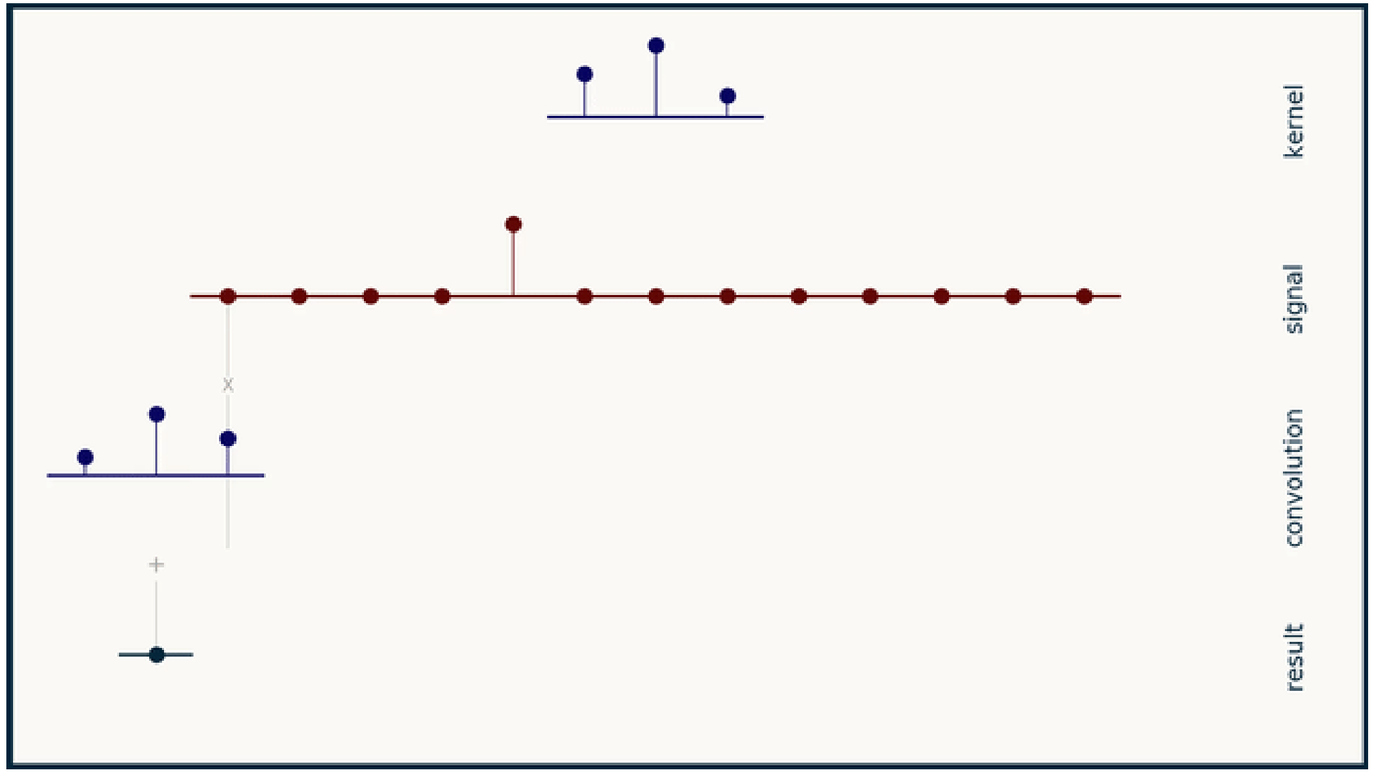
- **Couche convolutive** : Application d'un filtrage convolutif pour extraire des caractéristiques potentielles
- **Couche de mise en commun (Pooling)**: réduit la taille de la série tout en préservant les caractéristiques importantes identifiées par la couche convolutive
- **Couche entièrement connectée** : cette ou ces couches à la fin du réseau ont pour but de mettre en correspondance les caractéristiques extraites par le réseau avec des valeurs à prédire



5.5. CNN 4/7

Principe d'une convolution 1D

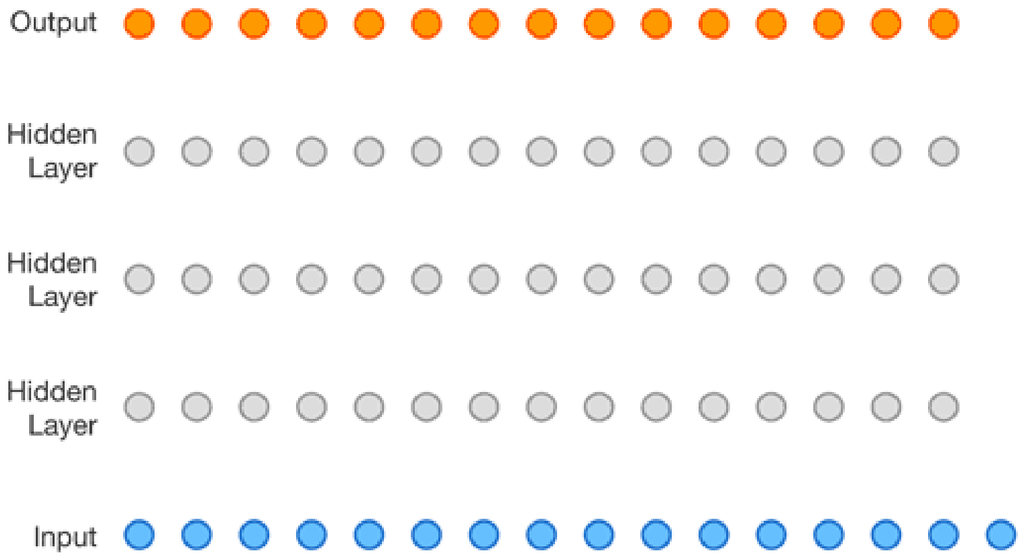
- Un noyau (kernel) de taille fixe (ex 3)
- L'opérateur convolution
- Nombre de filtres
- Une fonction d'activation en sortie (reLu)
- Généralement suivie d'une couche de pooling
- Attention au data leakage : la fenêtre de convolution ne doit pas être centrée sur le timestamp !



5.5. CNN : Wavenet 5/7

Wavenet (2016)

- Algorithme open sourcé par Google Deepmind
- Initialement développé pour des séquences audio
- Repose sur le principe de convolutions causales dilatées

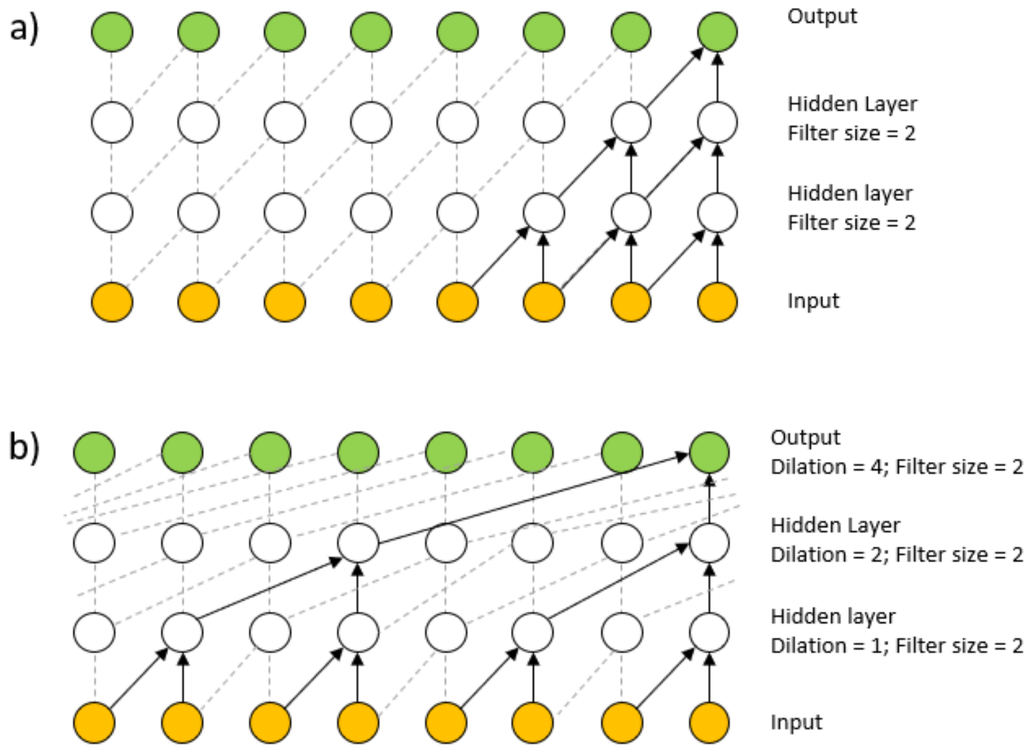


https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

5.5. CNN : Wavenet 6/7

Wavenet (2016)

- Algorithme open sourcé par Google Deepmind
- Initialement développé pour des séquences audio
- Repose sur le principe de convolutions causales dilatées
- Divers facteurs de dilatation (b) qui permettent à son champ réceptif de croître exponentiellement avec la profondeur et de couvrir des milliers de pas de temps par rapport à une convolution classique (a)

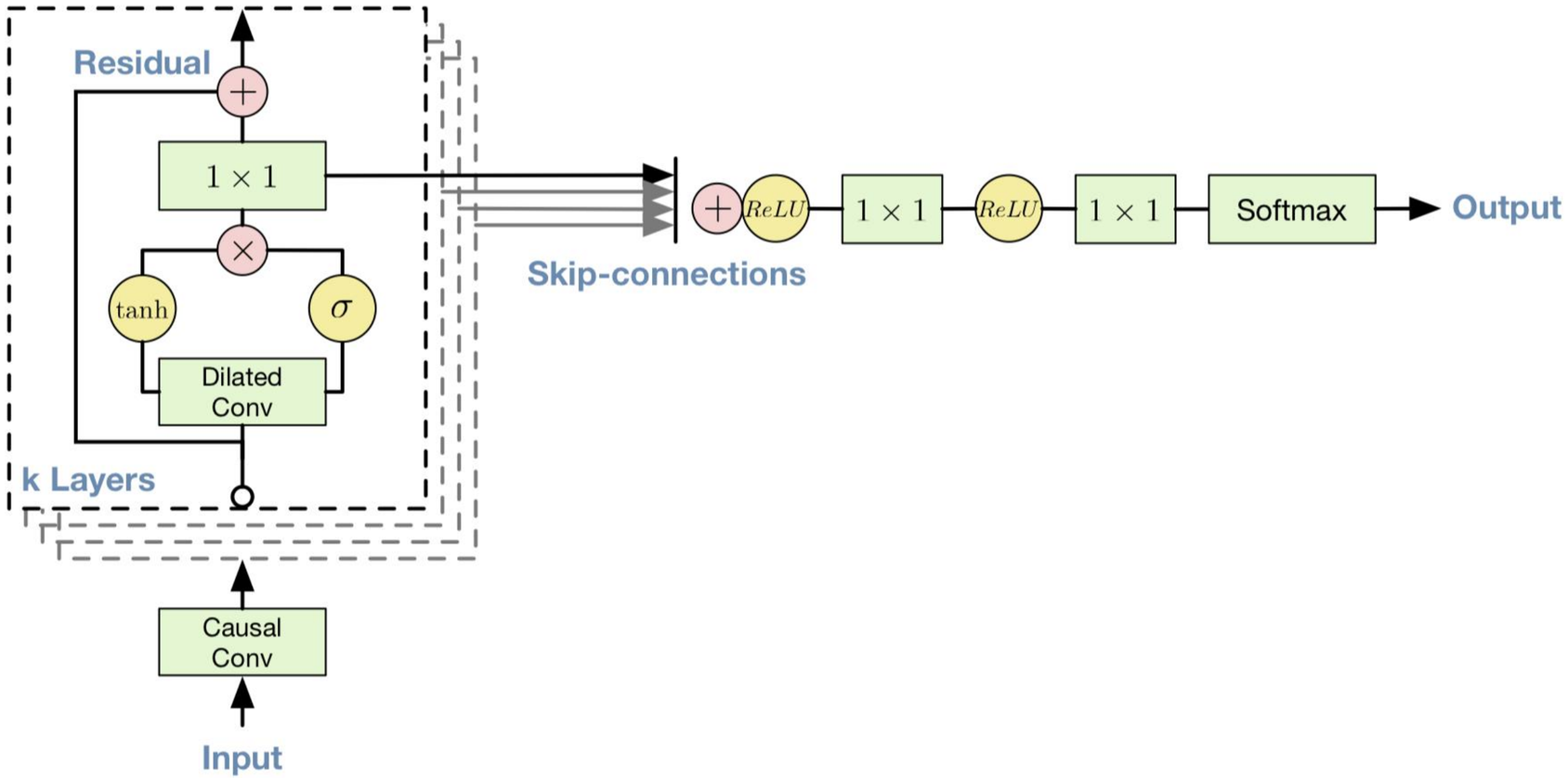


5.5. CNN : Wavenet 7/7 (Bonus)

Architecture Wavenet

3 grands principes :

- Convolutions Dilatées
- Résidus
- Skip connections



5.6. CNN-LSTM, ConvLSTM

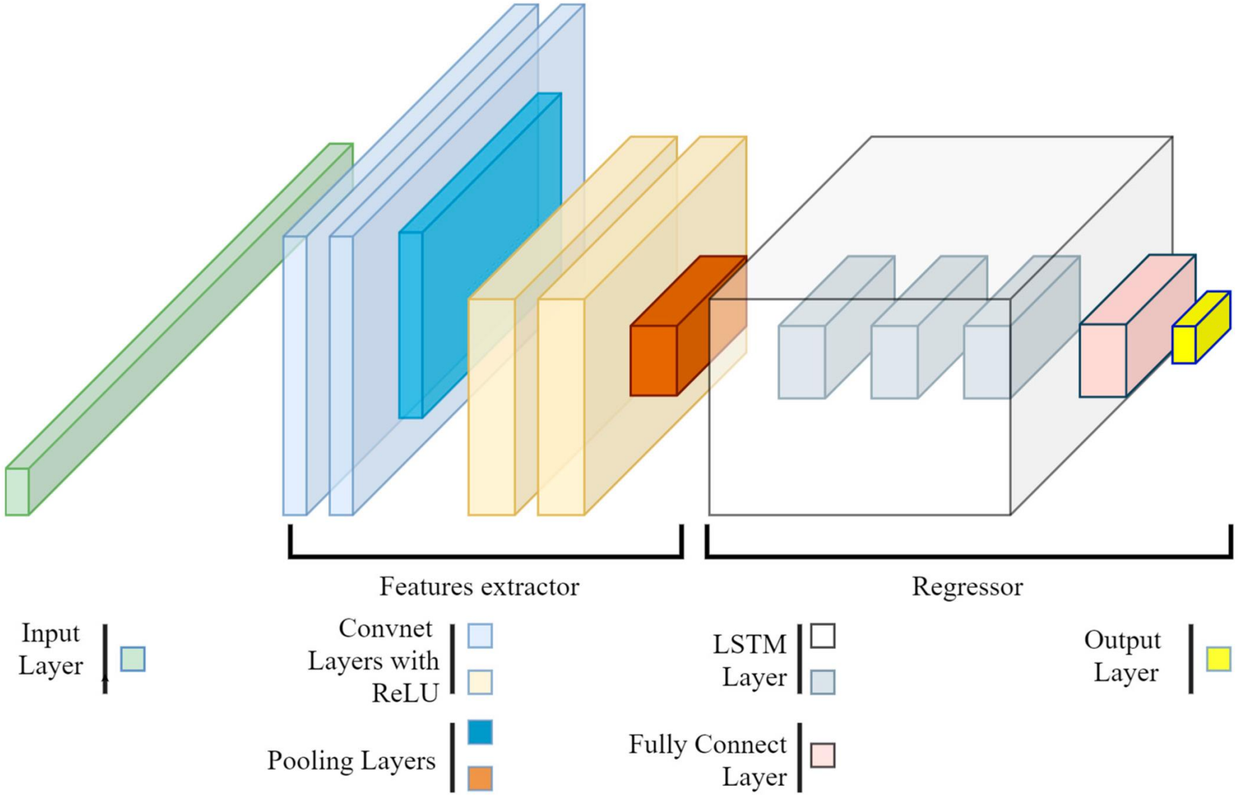
Hybridation des deux méthodes

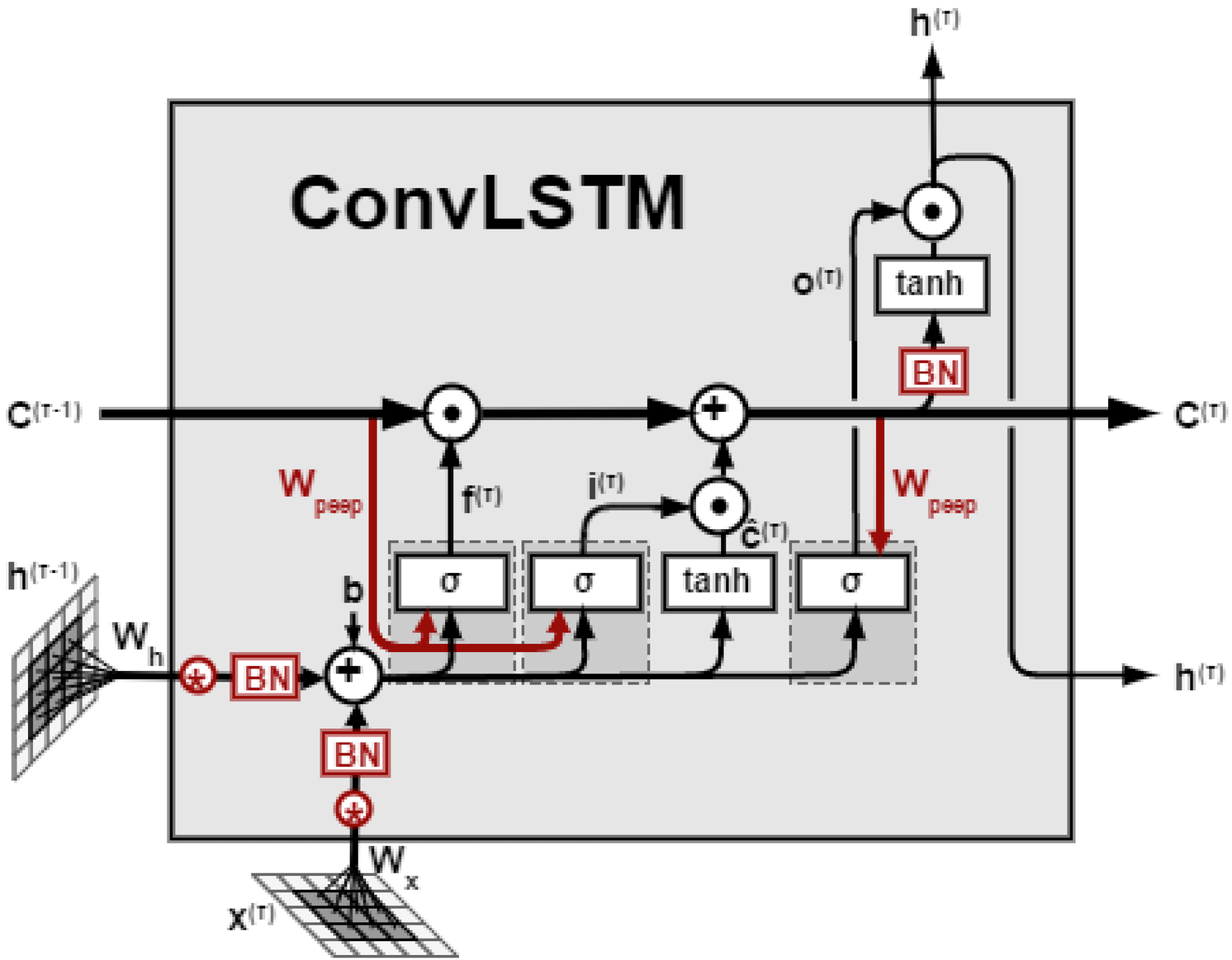
CNN-LSTM

- CNN : feature extractor qui vient alimenter le LSTM
- LTSM : capture les dépendances temporelles
- Méthode intéressante pour simplifier le signal d'entrée pour LSTM et augmenter les performances

Autre variante

- Conv-LSTM : la convolution est directement intégrée dans la cellule du LSTM
- Très utilisé avec des données spatio-temporelles





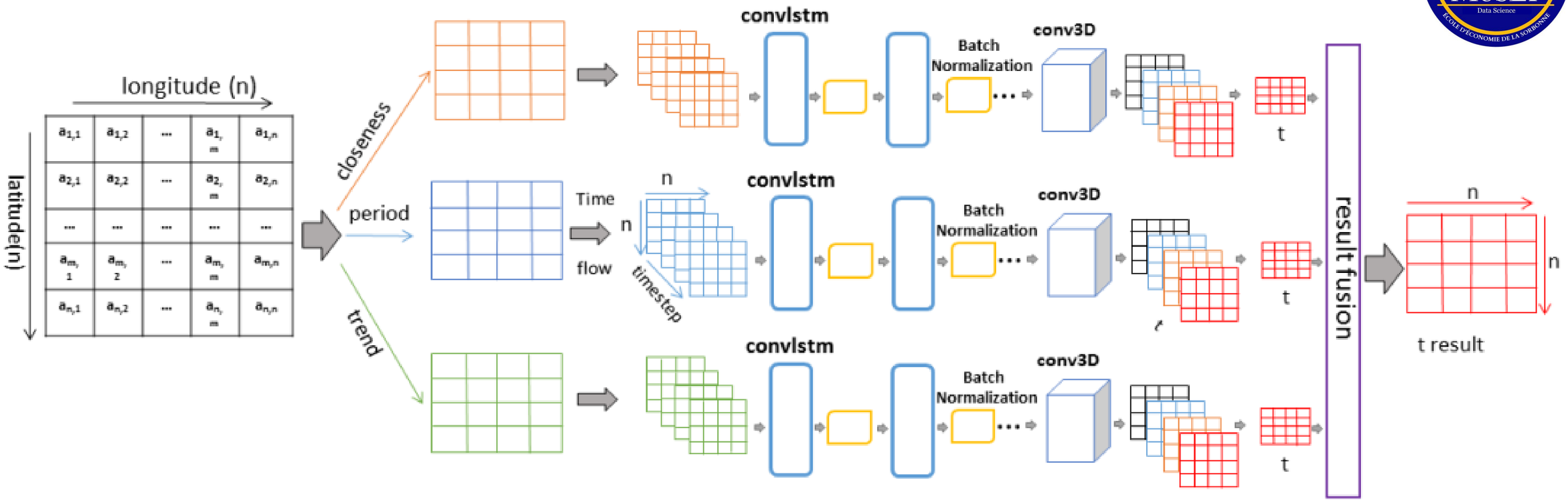


Fig. 3. Deep Spatial-Temporal Convolutional LSTM Network

5.7. Time to practice

Rendez-vous sans plus attendre sur Python pour la mise en pratique !