Séries temporelles par réseaux de neurones récurrents

Module 8

Objectifs

- savoir appliquer les réseaux de neurones aux séquences
- comprendre les contraintes et avantages des réseaux récurrents dans le cadre des séries temporelles

Problème à résoudre

Traiter une séquence d'inputs :

- séquence d'appels dans le cas GFI
- séquence de mots dans une phrase
- séquence de traces dans un log
- séquence d'actions sur une page web
- séquence de mesures en géologie / océanographie / météorologie
- ...

Première solution : padding

Idée

- utilisation d'un réseau standard
- taille d'input fixée à la taille maximale des séquences d'input
- padding des séquences plus petites avec des inputs neutres

Problèmes

- très couteux en mémoire et temps de calcul.
- l'entraînement des poids n'est pas optimal. Pourquoi ?

Deuxième solution : fenêtre glissante

Idée

- taille d'input fixe
- parcours de la séquence en décalant une fenêtre de la taille de l'input

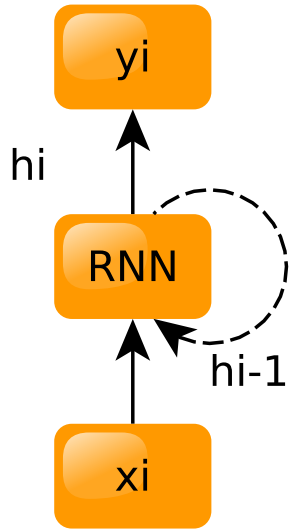
Problème

Modélisation des dépendances longues extrêmement simpliste, parfois non apprise. Comment peut-on les modéliser, même simplement ?

Troisième solution : réseaux récurrents

- taille d'input variable
- utilisation d'un même réseau pour chaque élément de l'input, séquentiellement

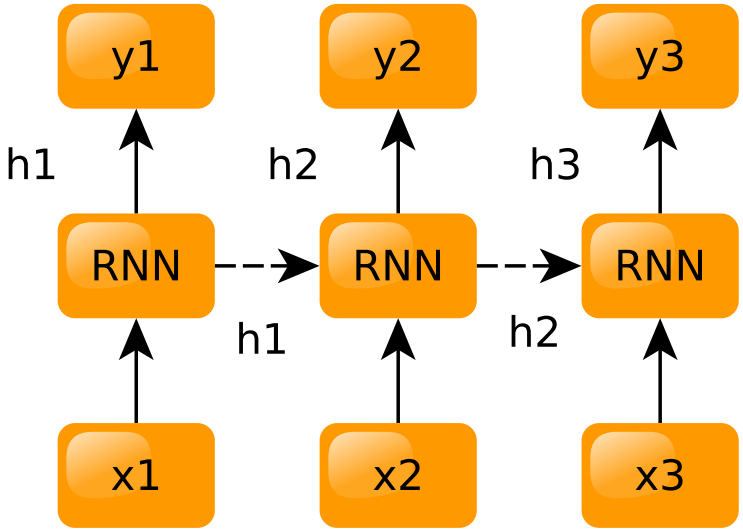
Réseau récurrent enroulé



Pour l'élement i d'une séquence de n inputs :

$$h_i = \text{RNN}(x_i, h_{i-1})$$

Réseau récurrent déroulé



Réseau déroulé très proche d'un réseau standard mais :

- poids partagés par les neurones correspondants des différentes étapes temporelles.
- connections horizontales (calcul pas à pas).

Adaptation de la rétropropagation des gradients

- aucune adaptation nécessaire. Le modèle standard fonctionne.
- seule différence : poids du réseau déroulé modifiés autant de fois qu'il y a d'étapes temporelles.
- de manière équivalente : poids du réseau déroulé modifiés par la somme des modifications des étapes temporelles (règle de chainage).

Où trouver des RNNs implémentés

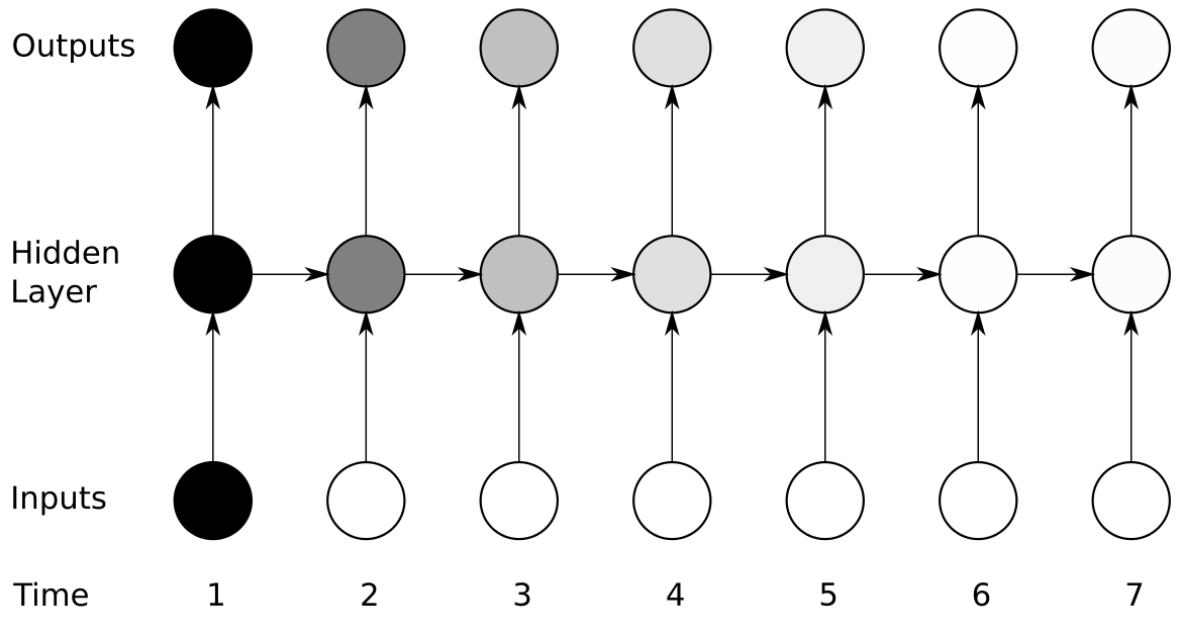
- keras (de loin le plus facile)
- tensorflow

Première conclusion

Une adaptation simple des réseaux de neurones permet de manipuler des séquences d'input

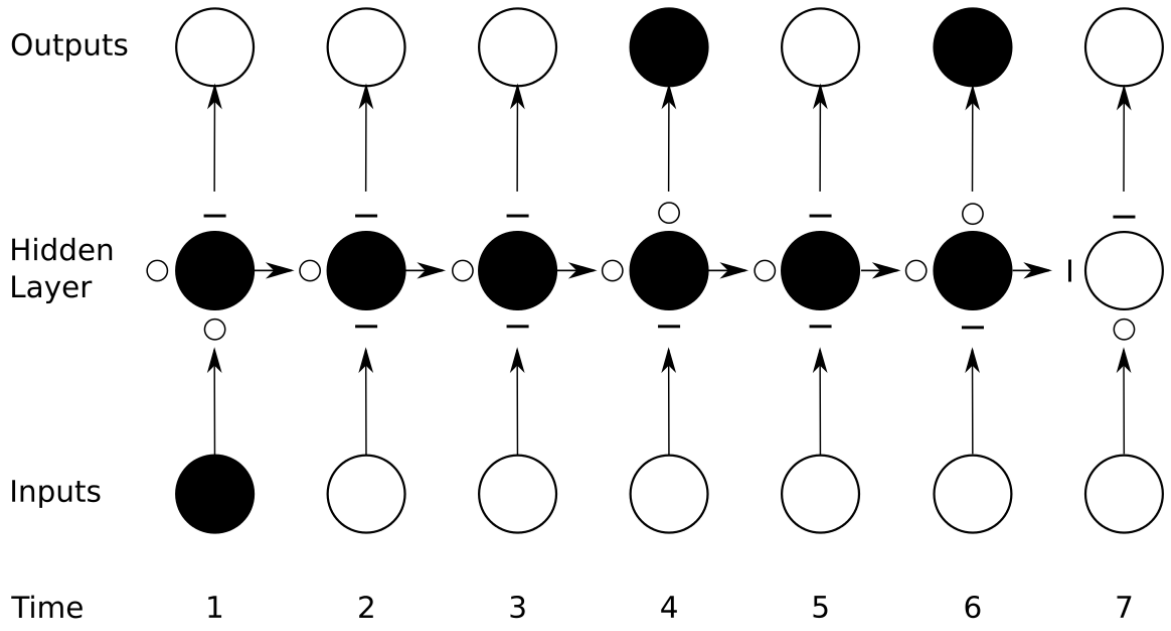
Problème principal

Modèle simple dysfonctionnel en pratique : les gradients ne permettent pas l'apprentissage de dépendances longues :



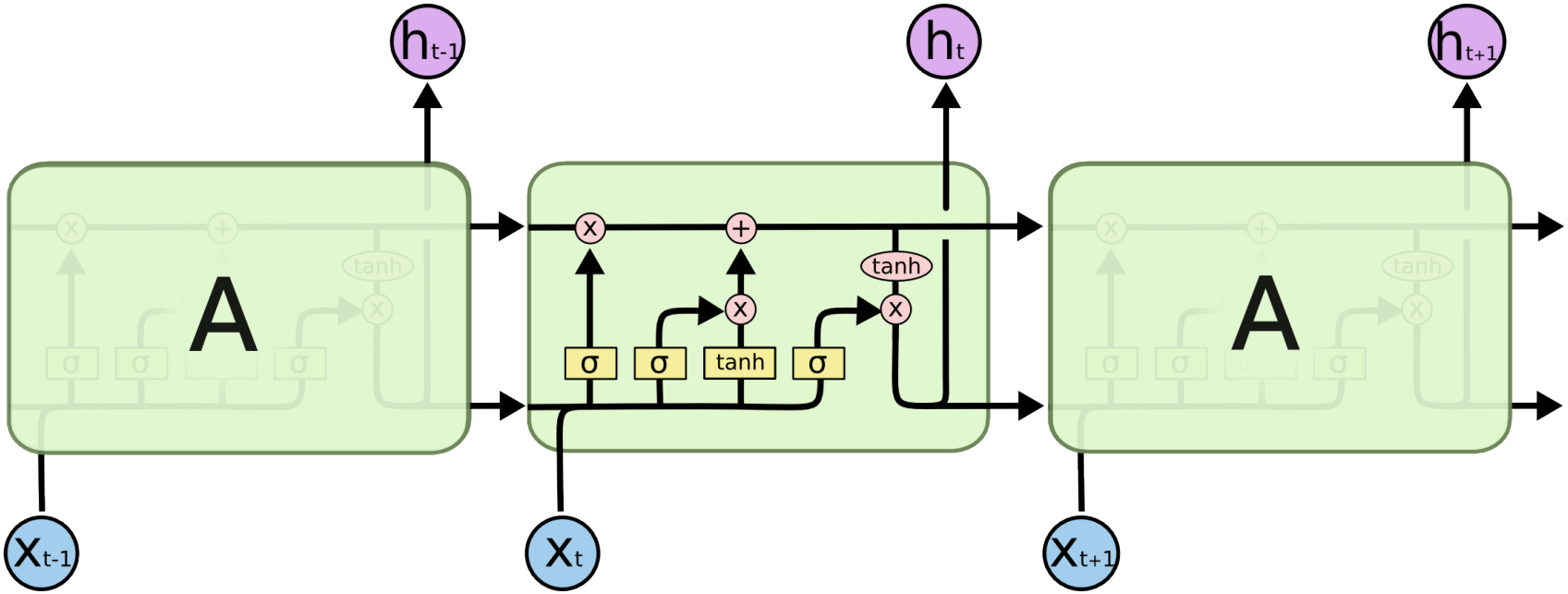
Solution : LSTM

Ajouter un mécanisme de mémoire et de portes pour protéger le flot d'information (et de gradient) :



Approche appelée Long Short-Term Memory Networks.

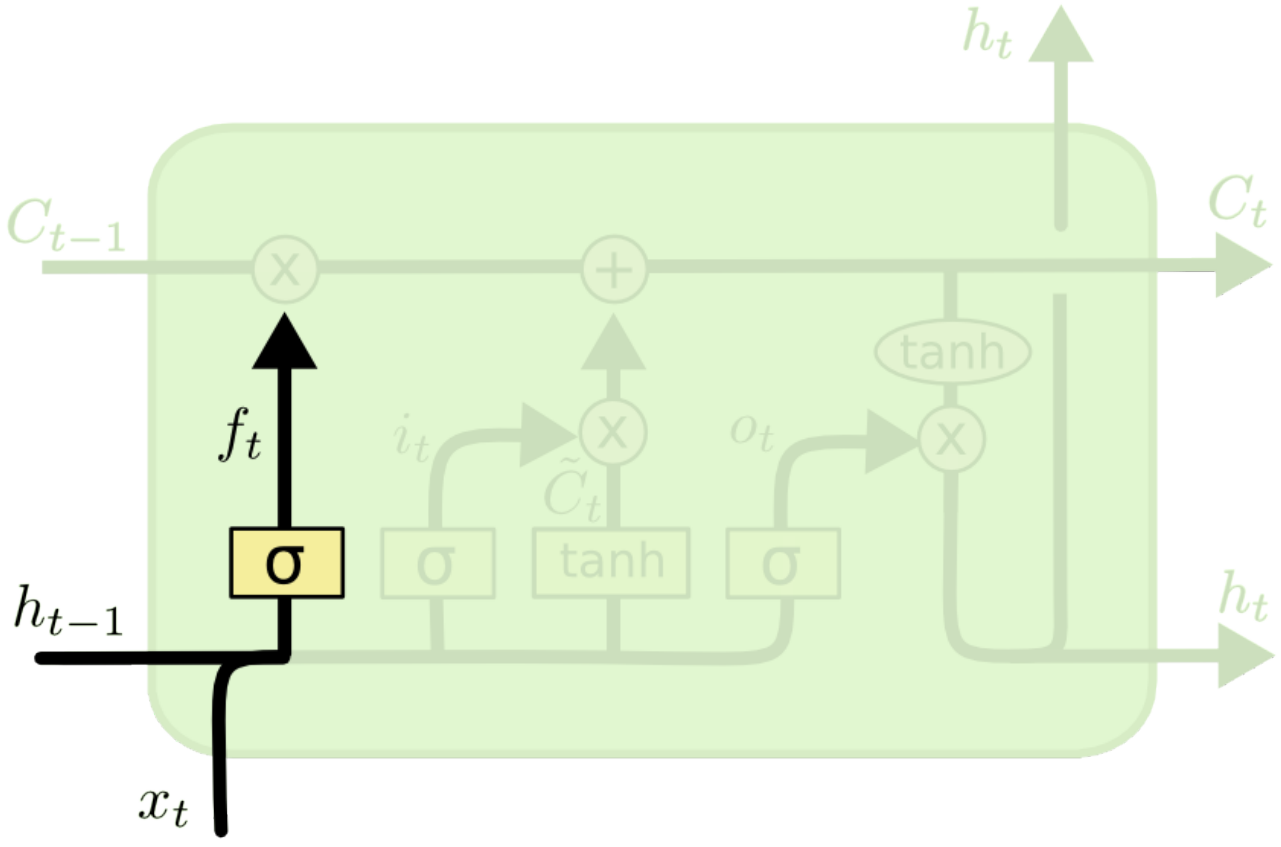
Cellule LSTM



Des portes sont multipliées à l'input et à l'ouput pour limiter leur impact sur le flot d'information au strict nécessaire.

LSTM : étape 1/4

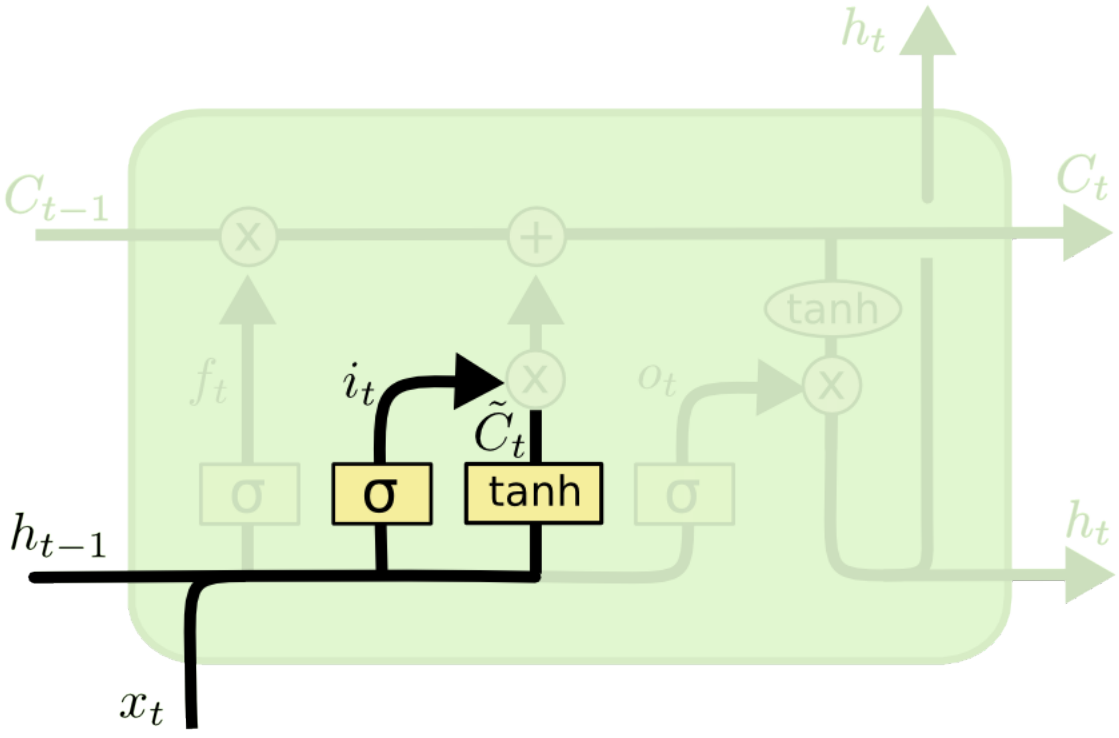
Conserver ou non les informations en mémoire :



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

LSTM : étape 2/4

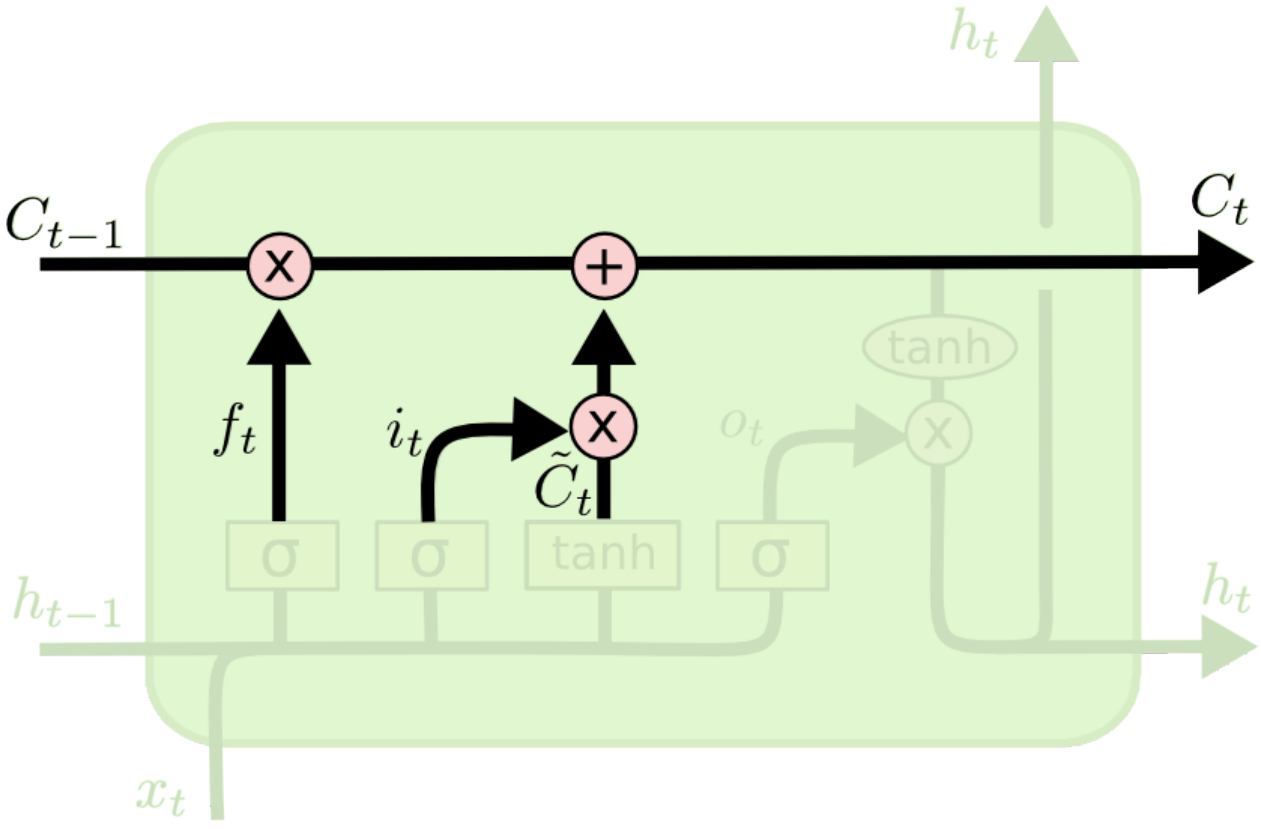
Prendre en compte ou non l'input :



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh\left(W_C \cdot [ht-1, x_t] + b_C\right)$$

LSTM : étape 3/4

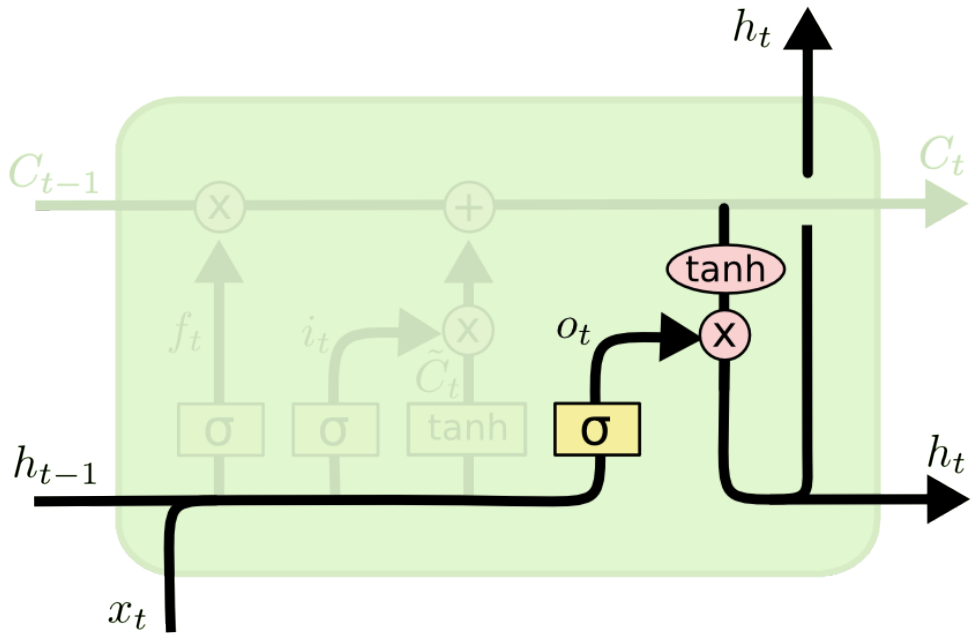
Mettre à jour l'état caché :



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

LSTM : étape 4/4

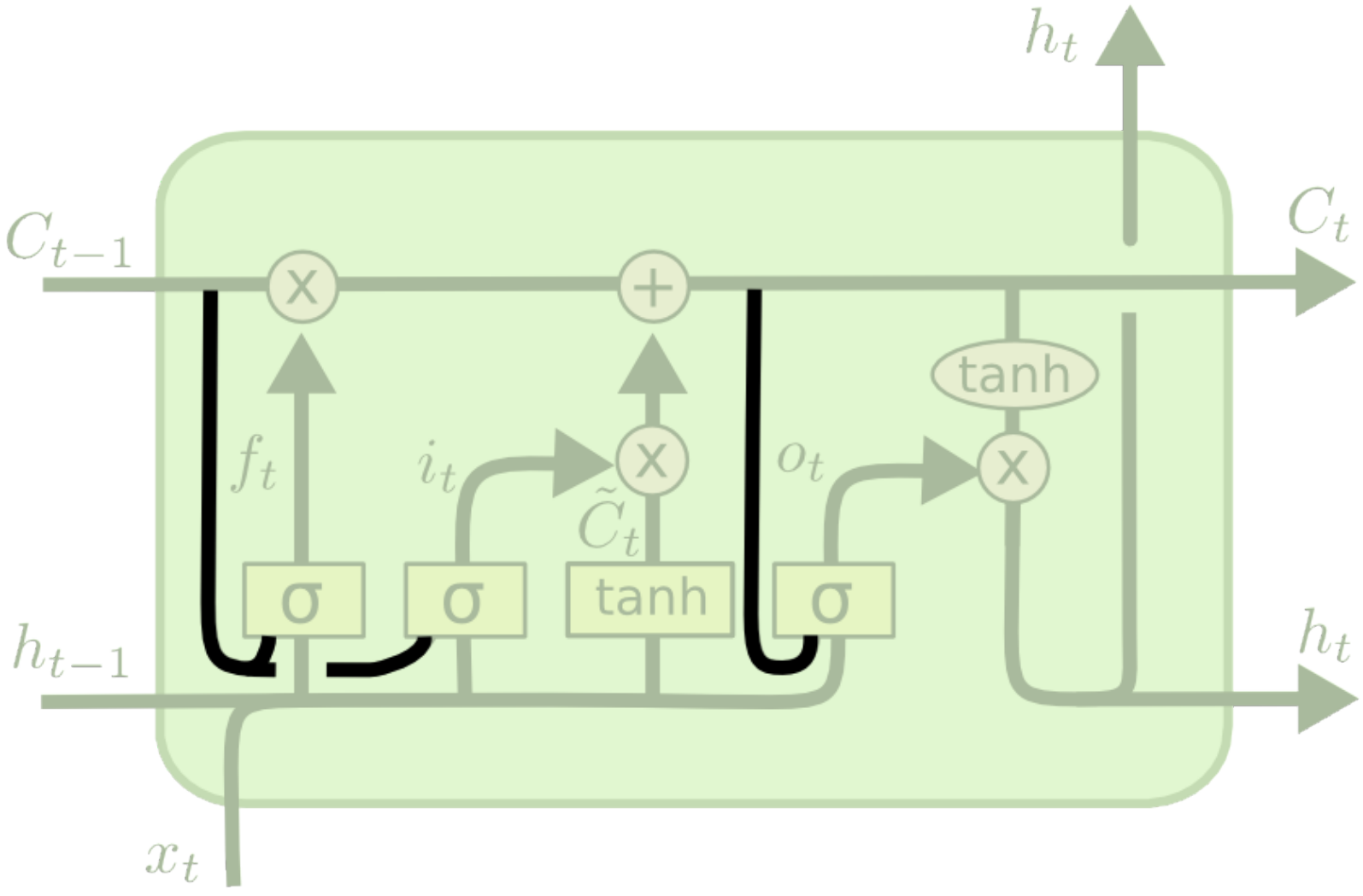
Contrôler si l'on produit ou non une valeur de sortie :



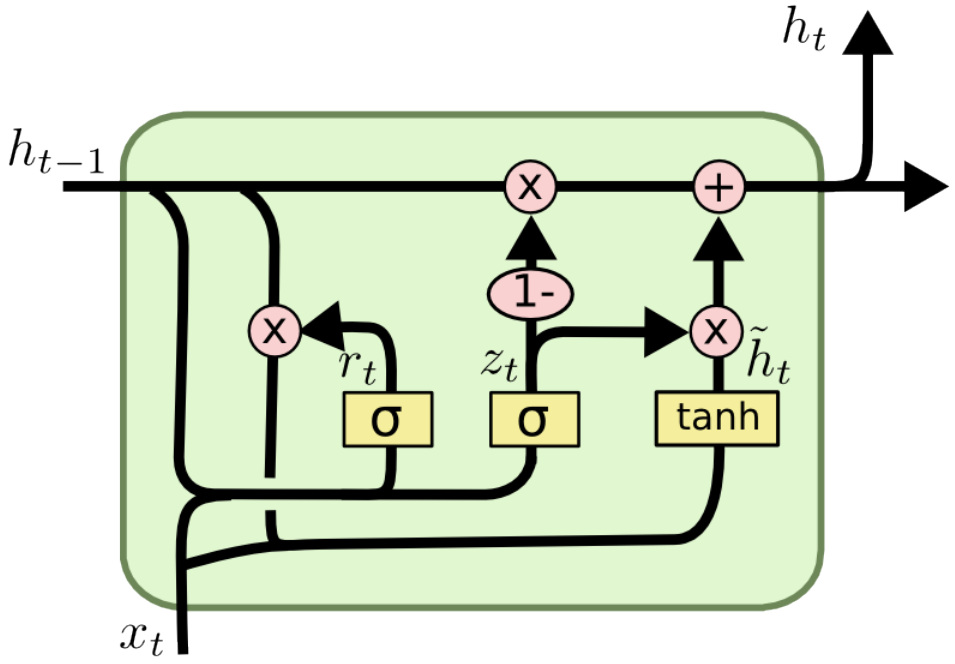
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

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

Extension : judas



Variante : GRU



$$\text{update gate} : z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$\text{reset gate} : r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\text{input candidat} : \tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$\text{mise à jour de l'état} : h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Deuxième conclusion

- les modèles modernes permettent avec des mécanismes simples de gérer les dépendances longues
- les séquences sont donc des citoyens de premier ordre dans les modèles de machine learning