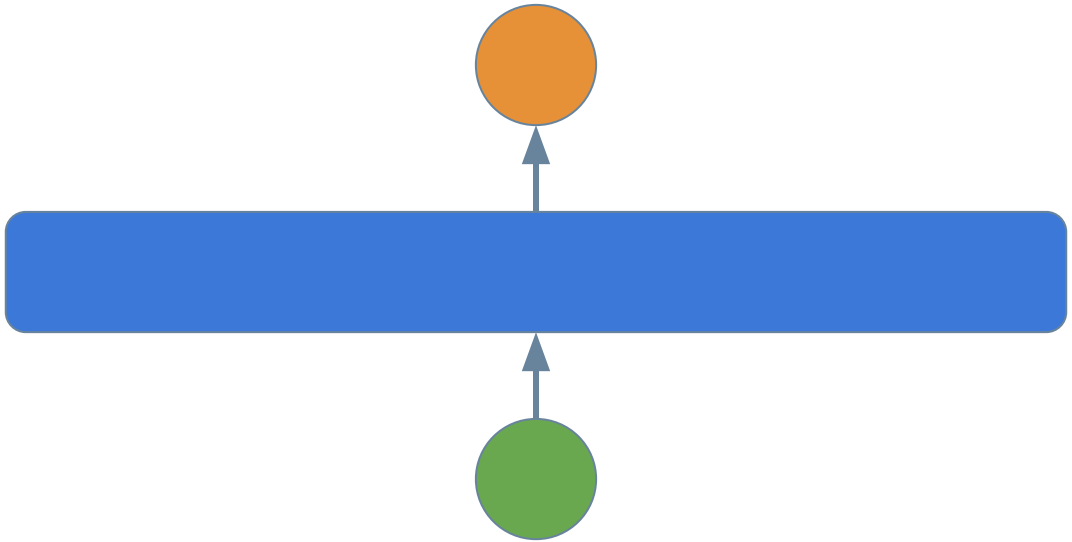
Réseaux Récurrents

Julien Romero

Les séquences

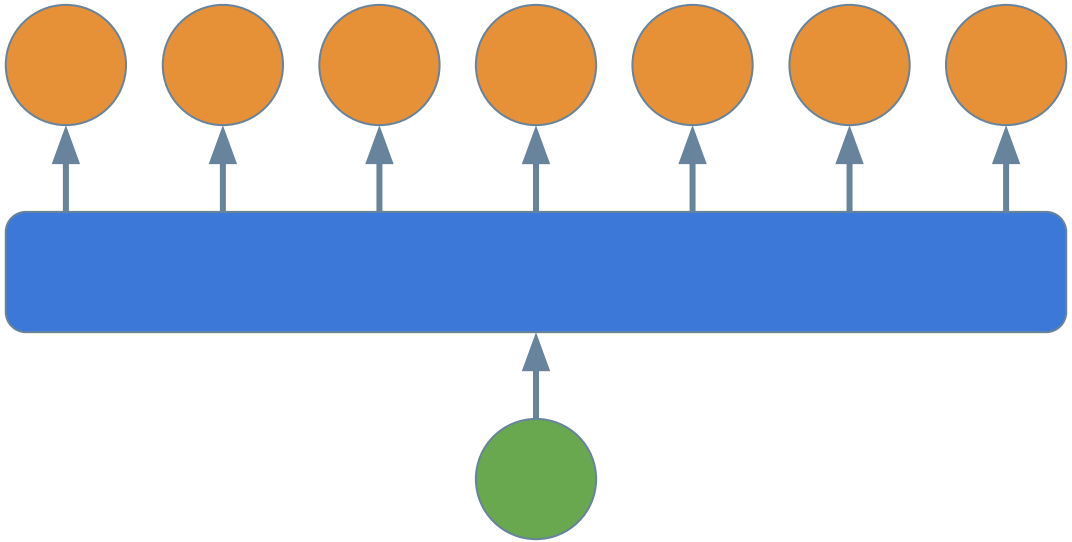
Les differents types de problèmes - One-to-One

- Entrée : Un élément, Sortie : Un élément
  - Ce qu'on a vu jusqu'à présent
- Exemple : Classification d'image



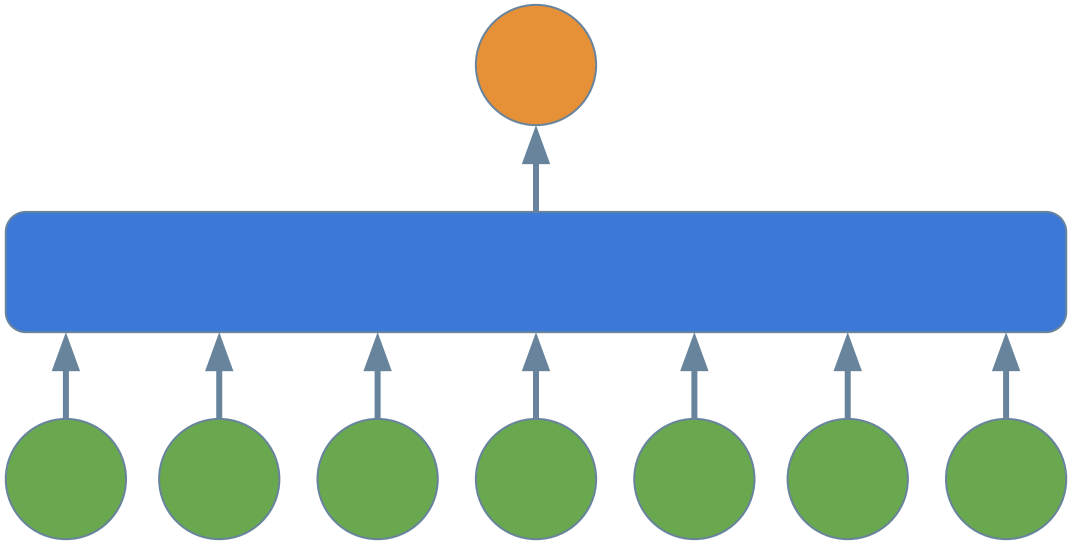
Les differents types de problèmes - One-to-Many

- Entrée : Un élément, Sortie : Une séquence d'éléments
- Exemple : Génération de la description d'une image



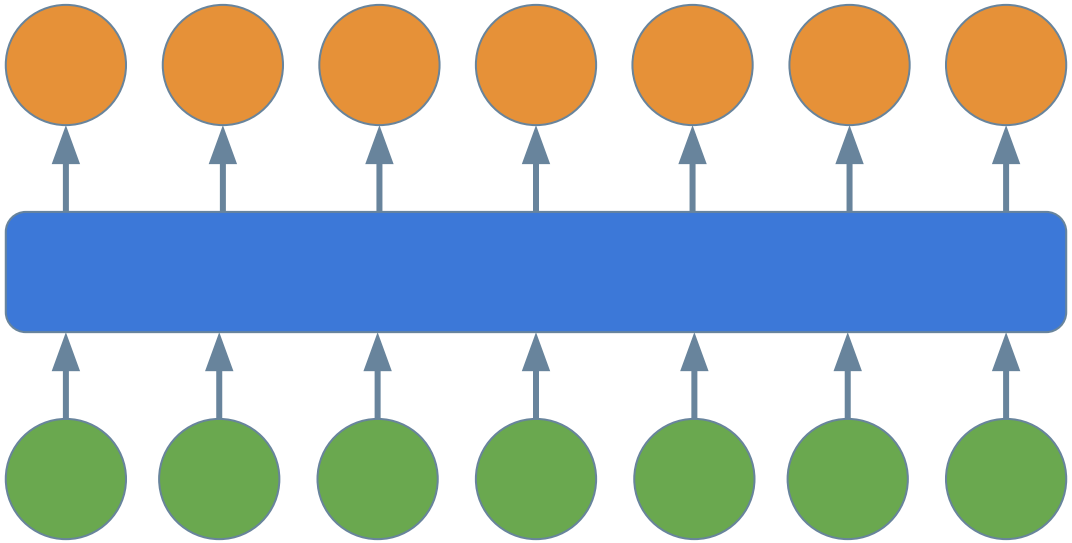
Les differents types de problèmes - Many-to-One

- Entrée : Une séquence d'éléments, Sortie : Un seul élément
- Exemple : Classification du sentiment d'un texte



Les differents types de problèmes - Many-to-Many

- Entrée : Une séquence d'éléments, Sortie : Une séquence d'éléments
- Exemple : Traduction d'un texte



Les réseaux récurrents

Définition du problème

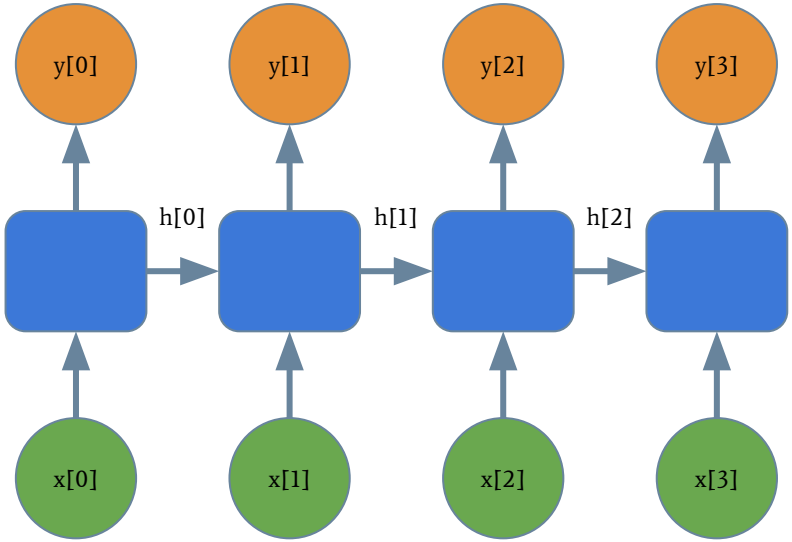
- Nous considérons le problème many-to-many
  - One-to-many et many-to-one peut être déduit en cachant des entrées/sorties
- Nous avons une séquence de longueur L.
- Nous avons une séquence d'éléments **ordonnés** en entrée et en sortie
  - x[1], x[2], ..., x[L]
  - y[1], y[2], ..., y[L]
- En général, on ne peut prédire la sortie y[k] qu'en utilisant les entrées x[i], i <= k précédentes.

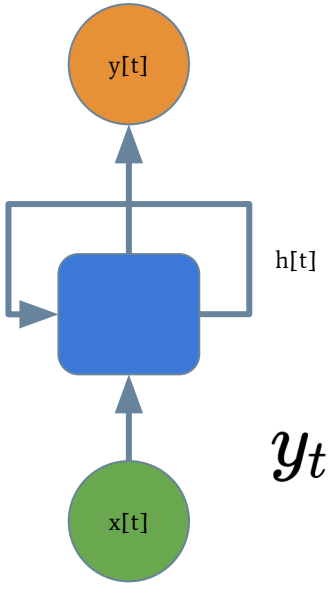
Solutions potentielles

- Concaténer les entrées jusqu'à x[k] pour prédire y[k]
  - Entrée de taille variable difficile à gérer (mais pas impossible, c.f. Transformers)
- Agréger toutes les entrées jusqu'à x[k] pour prédire y[k] avec une moyenne
  - On perd trop d'information
- Utiliser des réseaux récurrents

Neurones avec récurrence

Idée : Apprendre à encoder l'historique précédent dans un vecteur de taille fixe.





$$y_t = f(x_t, h_{t-1})$$

Réseau de neurones récurrents (RNN)

- À chaque étape, on applique la même fonction, avec **les mêmes paramètres** .

$$h_t = f_W(x_t, h_{t-1})$$

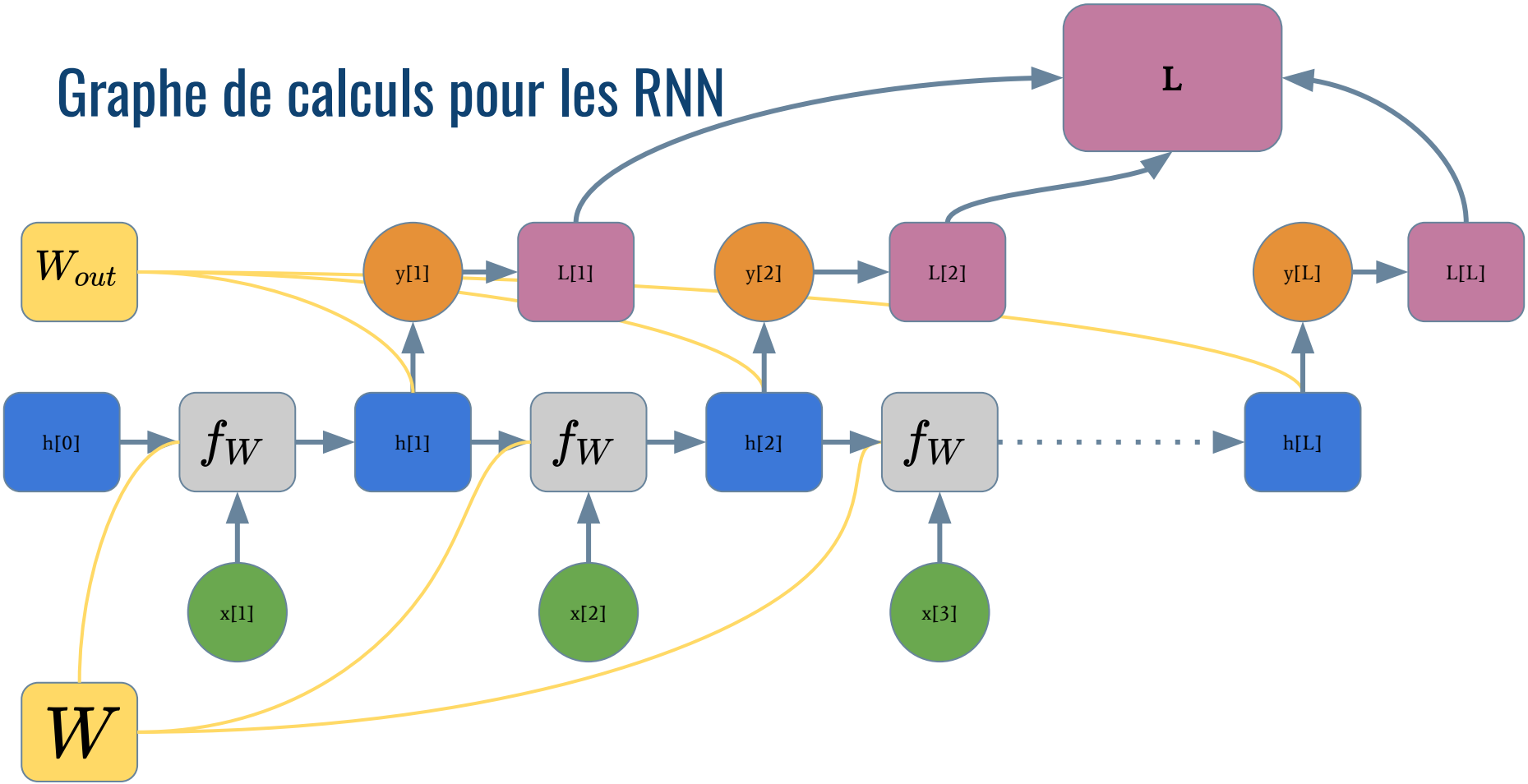
- On déduit ensuite la sortie :

$$y_t = f_{W_{out}}(h_t)$$

- Par exemple :

$$h_t = tanh(W_1^T h_{t-1} + W_2^T x_t) \qquad y_t = W_{out}^T h_t$$

Graphe de calculs pour les RNN



Backpropagation dans le temps

Problème : Le réseau devient vite très profond (on doit déployer le réseau, comme si c'était un réseau de neurone normal)

- Vanishing/exploding gradient très probable
- h[0] est très difficile à apprendre
- Tendance à oublier le début des séquences
  - Pour calculer le gradient par rapport à W, on doit sommer sur tous endroits où W est utilisé, en passant par les h[t]
  - Si on a du vanishing gradient, les contributions ne sont pas égales

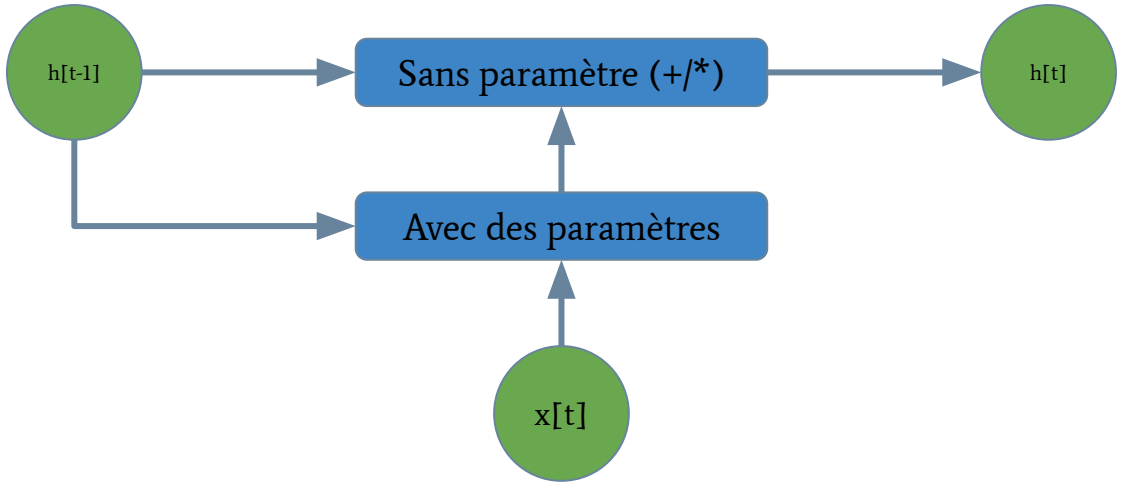
Peut-on adapter les réseaux résiduels ?

Gated Networks

Gated Network

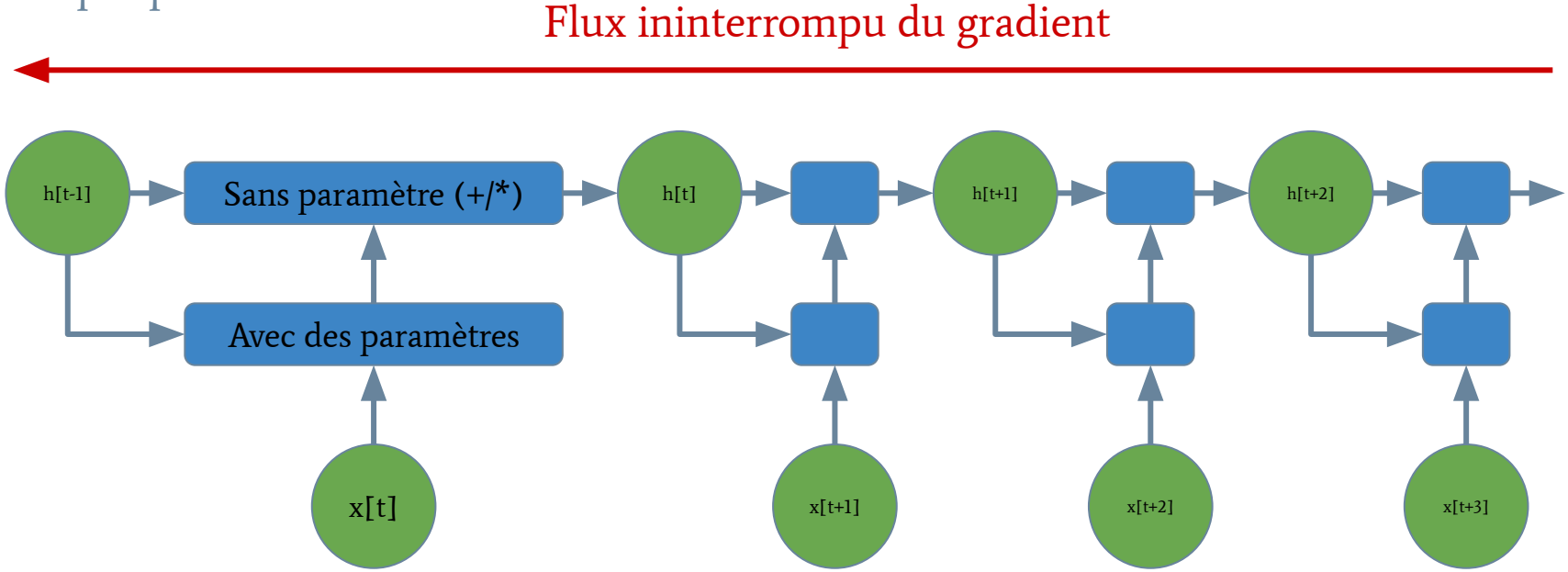
Idée : On va utiliser des portes pour contrôles les informations à faire passer depuis les étapes précédentes.

- Ces portes vont permettre de faire plus facilement circuler le gradient.



Gated Network

Idée : On va utiliser des portes pour contrôles les informations à faire passer depuis les étapes précédentes.



Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

Update gate : Combien passer de l'état précédent ?

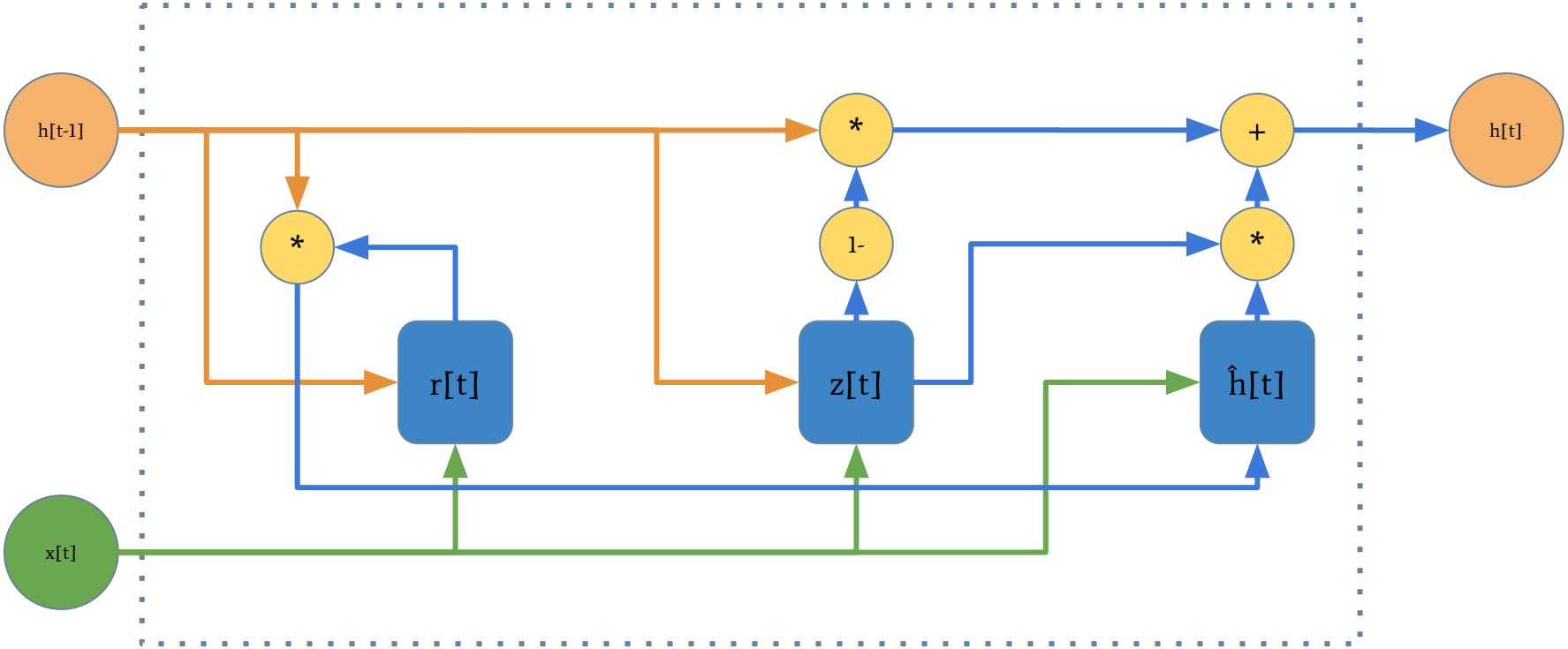
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

Reset gate : De quoi a-t-on besoin dans l'état précédent ?

$$\hat{h}_t = \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$

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

Gated Recurrent Unit (GRU)



Long Short-Term Memory

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$ Forget gate : Que garder de l'état précédent pour le prochain ?

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$ Input gate : Que garder de l'état actuel pour le prochain ?

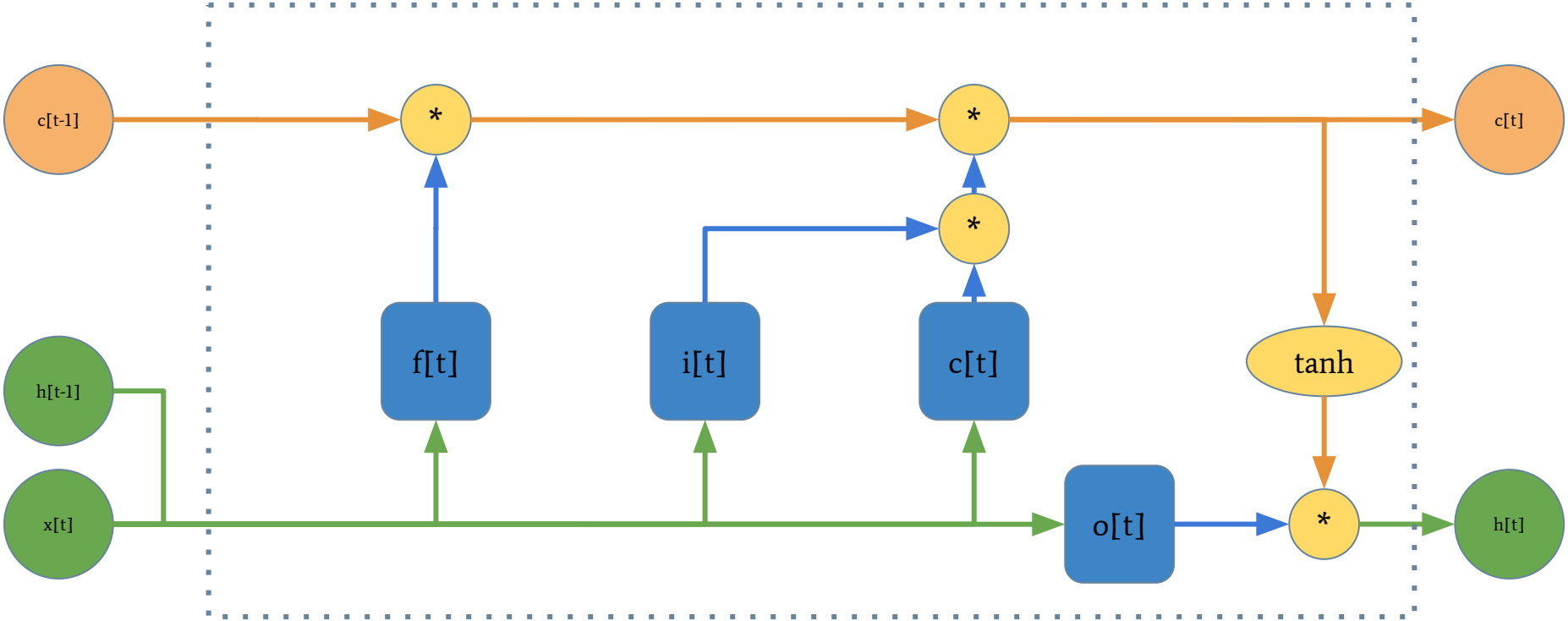
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$ Output gate : Que garder de l'état actuel pour la sortie ?

$$\hat{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$ Output gate : Quelle modification de l'état ?

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$ Cell state : État caché interne

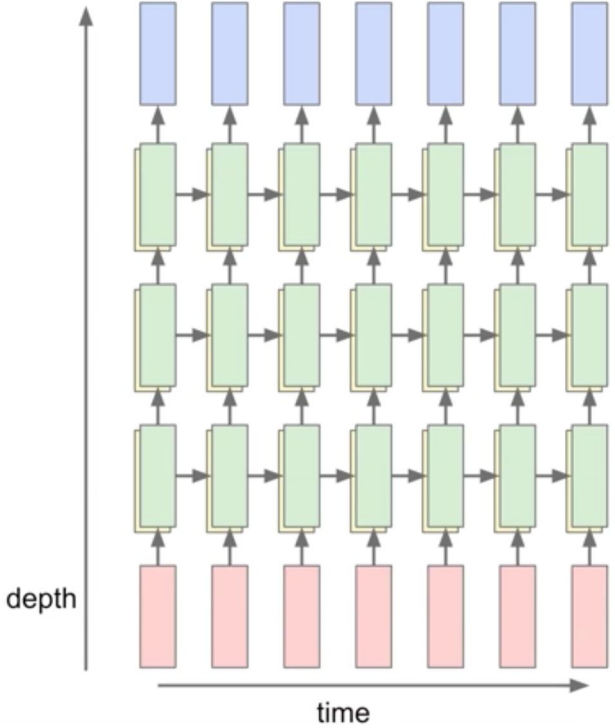
$$h_t = o_t \odot tanh(c_t)$$ Hidden state : Sortie du LSTM permettant de calculer la sortie du réseau

Long Short-Term Memory



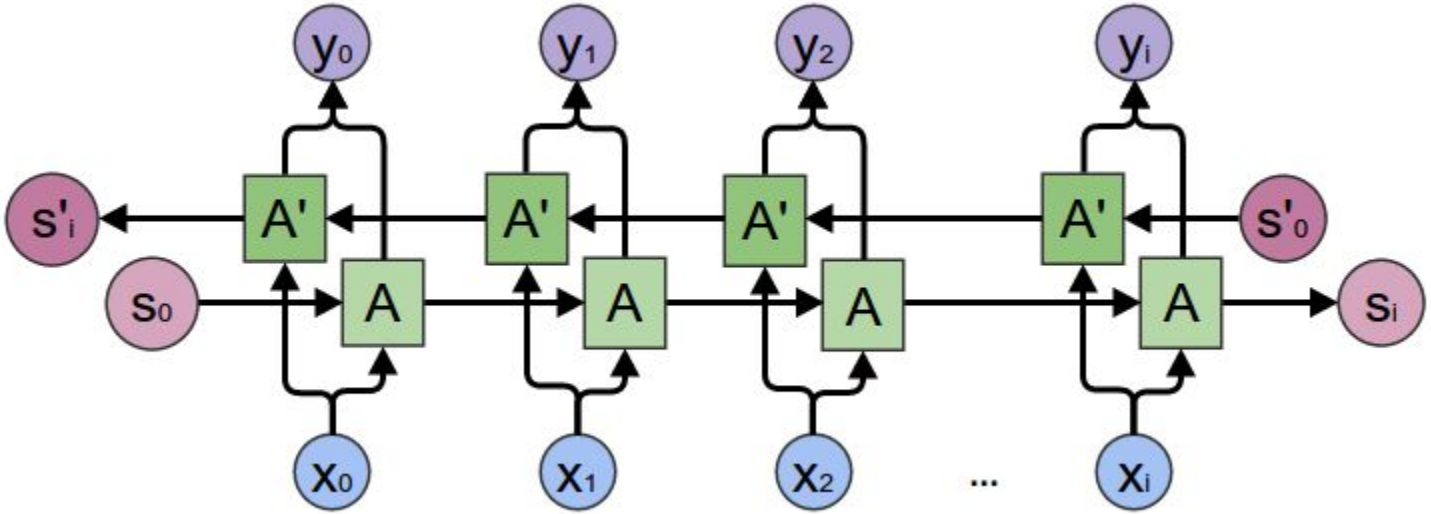
Variations sur les RNN : le RNN multi-couche

- On peut mettre plusieurs RNN l'un sur l'autre



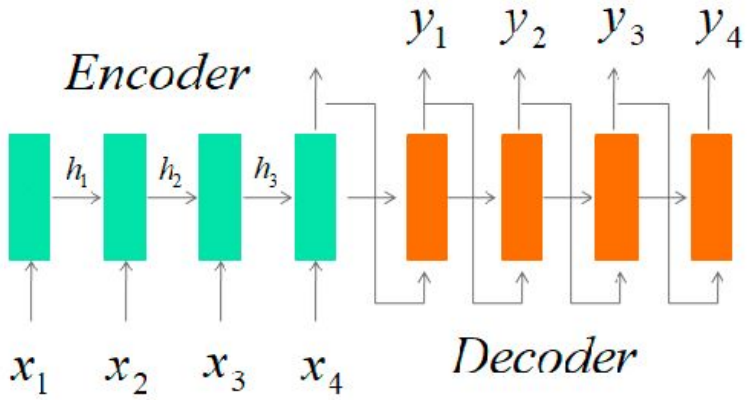
Variations sur les RNN : le RNN bidirectionnel

- Quand on connaît déjà toute la séquence d'entrée, on peut ajouter un RNN en sens inverse.

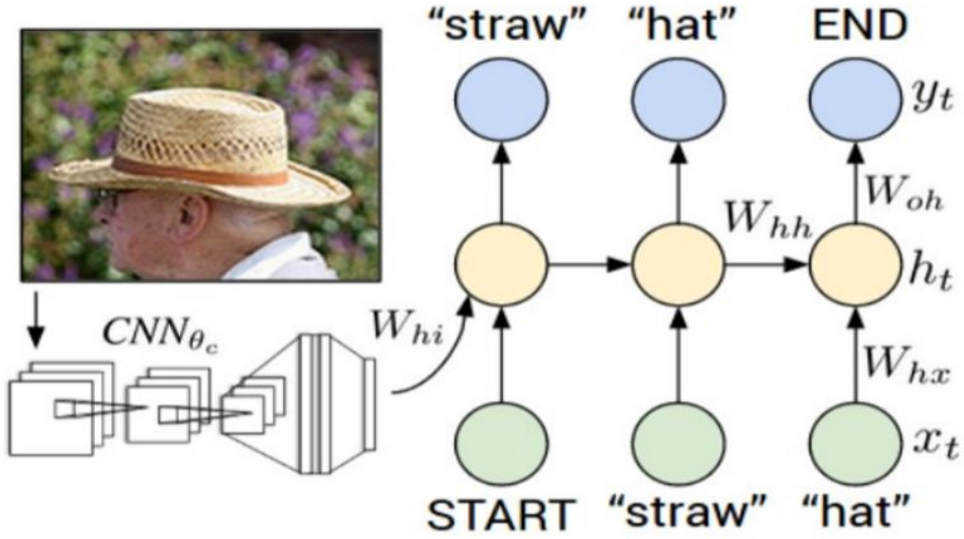


Variations sur les RNN : Le Sequence to Sequence (seq2seq)

- **Encodeur** : On utilise un premier RNN pour encoder une séquence de longueur quelconque en un seul vecteur : le dernier état caché (many-to-one)
- **Décodeur** : On utilise ce dernier état caché au début d'un autre RNN pour générer une séquence de longueur quelconque (one-to-many)



Exemple : One-to-Many - Image Captioning



En Pytorch

RNN

CLASS torch.nn.RNN(*input_size*, *hidden_size*, *num_layers=1*, *nonlinearity='tanh'*, *bias=True*, *batch_first=False*, *dropout=0.0*, *bidirectional=False*, *device=None*, *dtype=None*) [SOURCE]

Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

En entrée : La séquence d'entrée + h0 (souvent que des 0)
En sortie : L'ensemble des états cachés + le dernier état caché

En Pytorch

`nn.RNN`	Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence.
`nn.LSTM`	Apply a multi-layer long short-term memory (LSTM) RNN to an input sequence.
`nn.GRU`	Apply a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

Avantages des RNN

- Peuvent traiter des séquences de longueurs variables
- Peuvent encoder des dépendances sur le long terme à travers le context
- Peuvent maintenir des informations sur l'ordre des éléments
- Paramètres partagés : réduit le nombre de paramètres, indépendants de la position dans la séquence

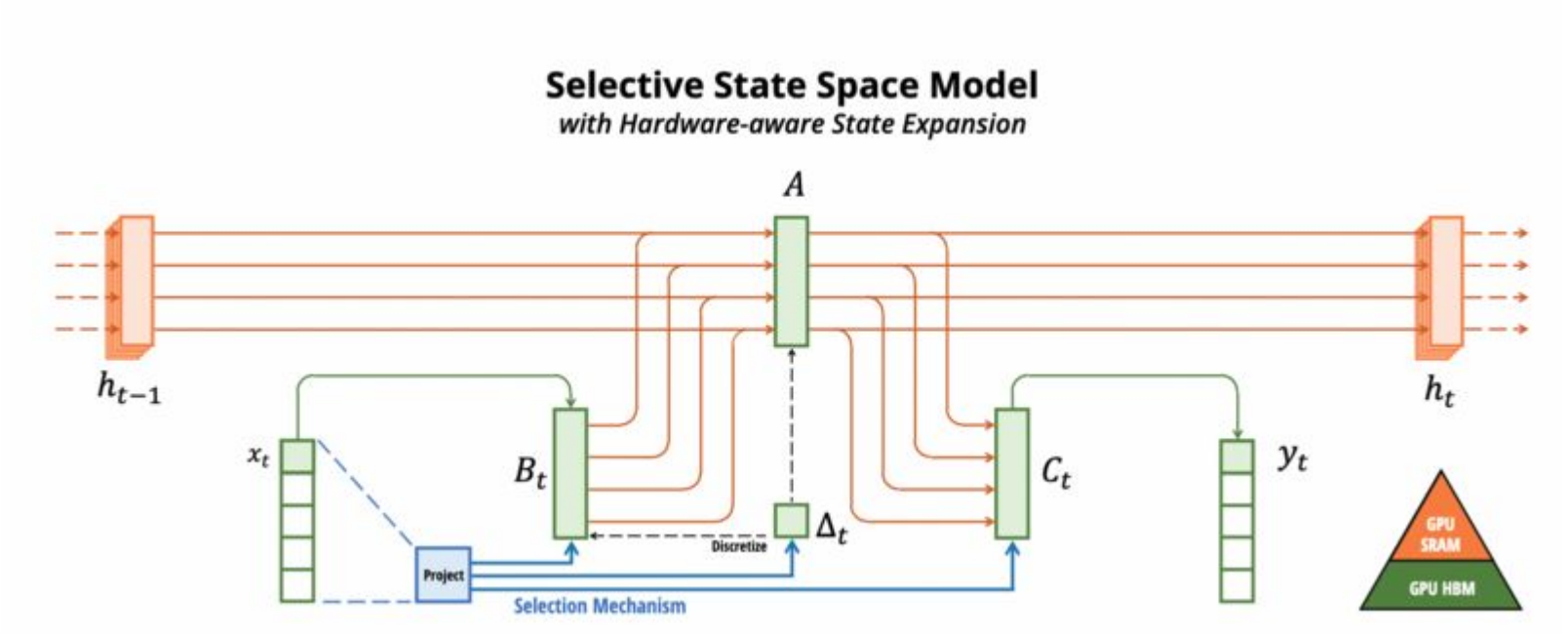
Désavantages des RNN

- Difficiles à entraîner sur les longues séquences à cause du vanishing/exploding gradient
- Mémoire limité et compliquée à contrôler, en particulier sur le long terme
- Lents et difficiles à paralléliser

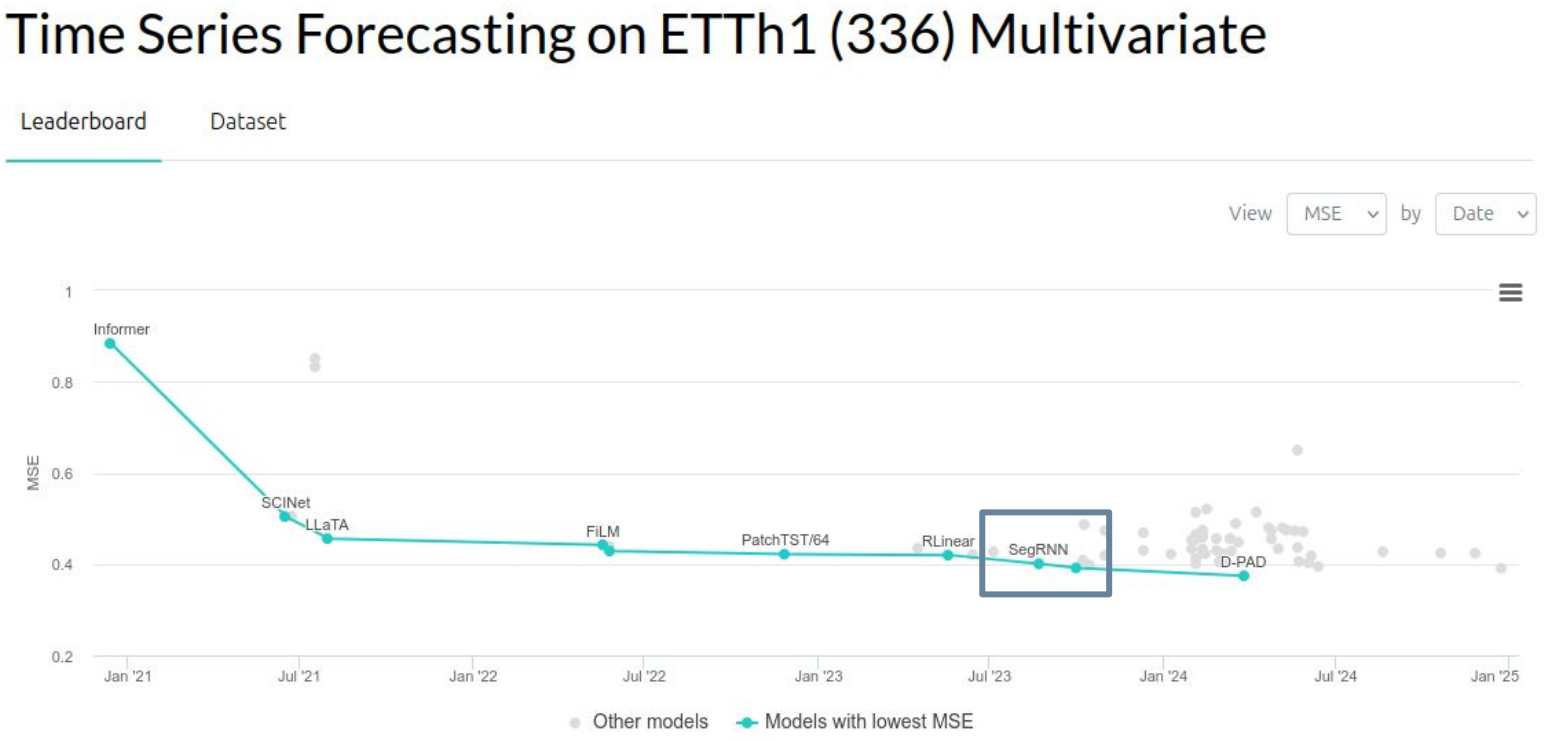
LSTM is not dead : xLSTM (2024)

- Améliorations notables sur LSTM pour résoudre les problèmes précédents
  - Utilisation d'une fonction exponentielle à la place de sigmoids
  - Utilisation de matrices pour les états c[t] à la place de vecteur
    - Plus de parallelisme possible

RNN is not dead : Mamba (2023)



RNN pour les séries temporelles numériques



En résumé

- On peut traiter des séquences avec des réseaux récurrents
- LSTM et GRU souvent préférés
- De nombreux défis
  - Vanishing/exploding gradient
  - Parallélisme
  - Gestion de la mémoire