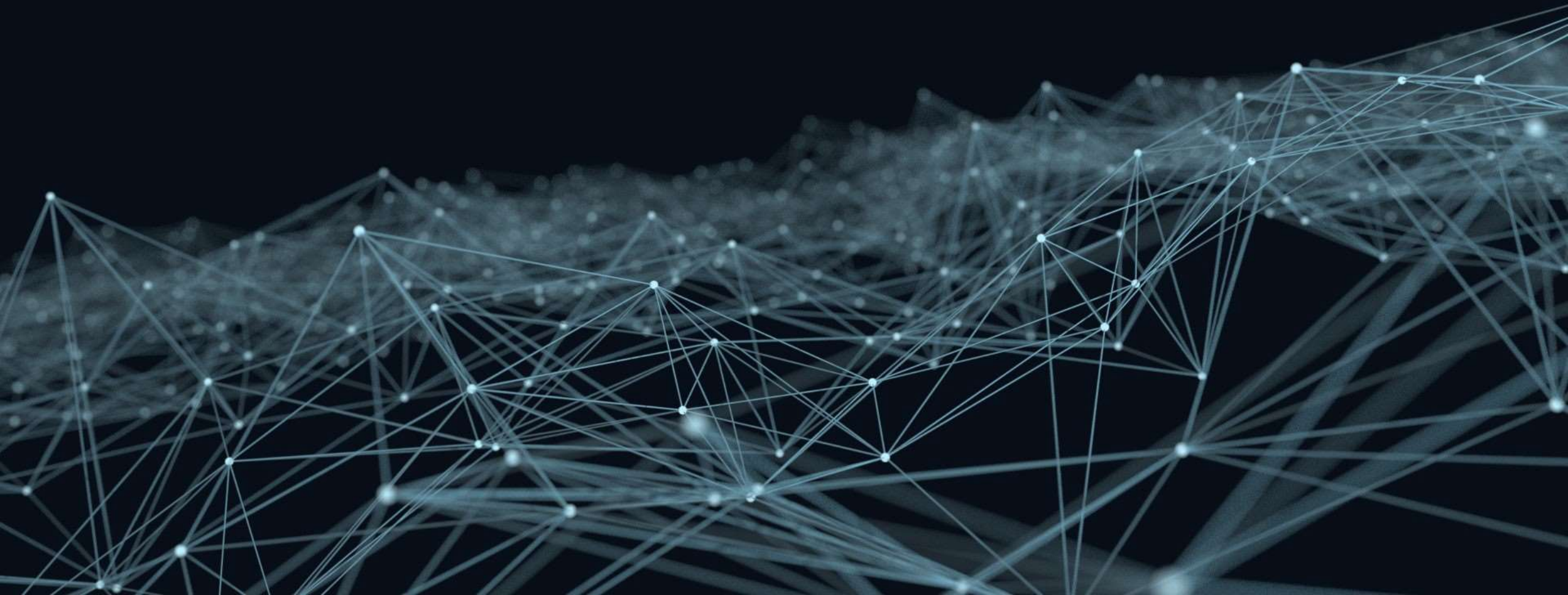
Recurrent Neural Networks

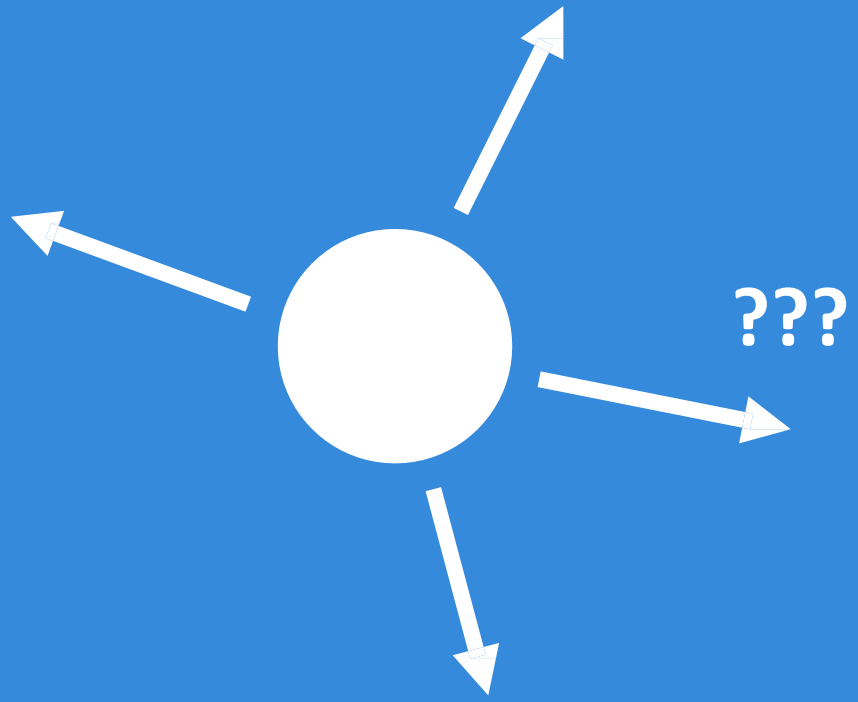
Simplon

2019

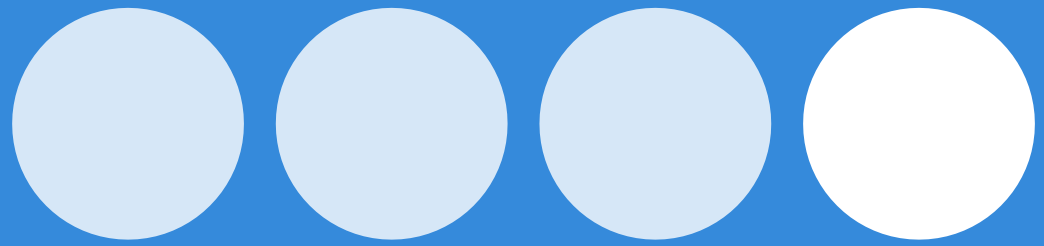


Etant donné l'image d'une balle, pouvez-vous prédire sa prochaine position ?

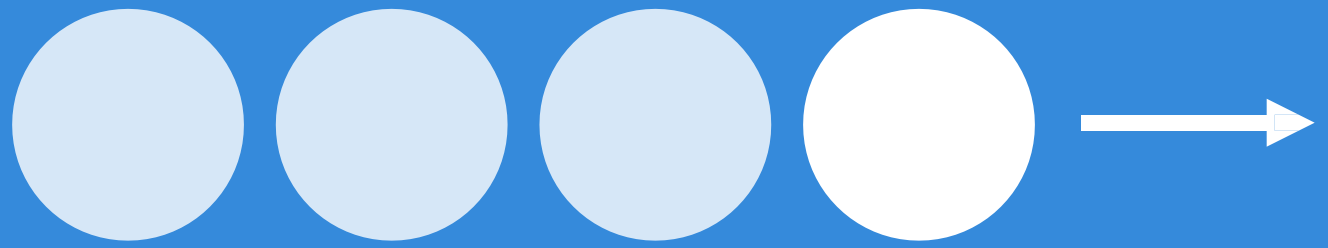
Etant donné l'image d'une balle, pouvez-vous prédire sa prochaine position ?



Etant donné l'image d'une balle, pouvez-vous prédire sa prochaine position ?



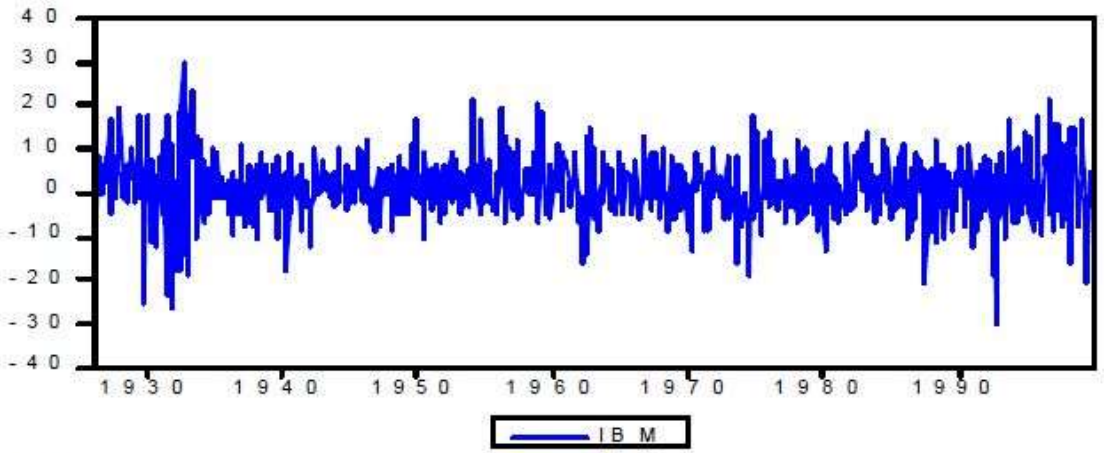
Etant donné l'image d'une balle, pouvez-vous prédire sa prochaine position ?



Types de séquences

Audio

Cours boursier



Types de séquences

character:

Introduction aux réseaux de neurones

word:

Text

Un problème de modélisation séquentielle : Prédire le mot suivant

Modélisation séquentielle : Prédire le mot suivant

"This morning I took my cat for a walk."

Modélisation séquentielle : Prédire le mot suivant

"This morning I took my cat for a walk."

given these words

Modélisation séquentielle : Prédire le mot suivant

"This morning I took my cat for a walk."

À partir de ces mots

Prédire le mot suivant

Idée #1: Fixer une longueur

"This morning I took my cat for a walk."

À partir de ces deux mots

Prédire le mot suivant

Idée #1: Fixer une longueur

"This morning I took my cat for a walk."

À partir de ces deux mots

Prédire le mot suivant

One-hot feature encoding: tells us what each word is

[1 0 0 0 0 0 1 0 0 0]

for a

prediction

Problème #1: on ne peut pas prédire les dépendances à long terme

"France is where I grew up, but I now live in Boston. I speak fluent ____."

On a besoin de l'information du passé proche
pour prédire le mot suivant.

Idée #2: décomposer l'ensemble de la séquence

"This morning I took my cat for a"

"bag of words"

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]

prediction

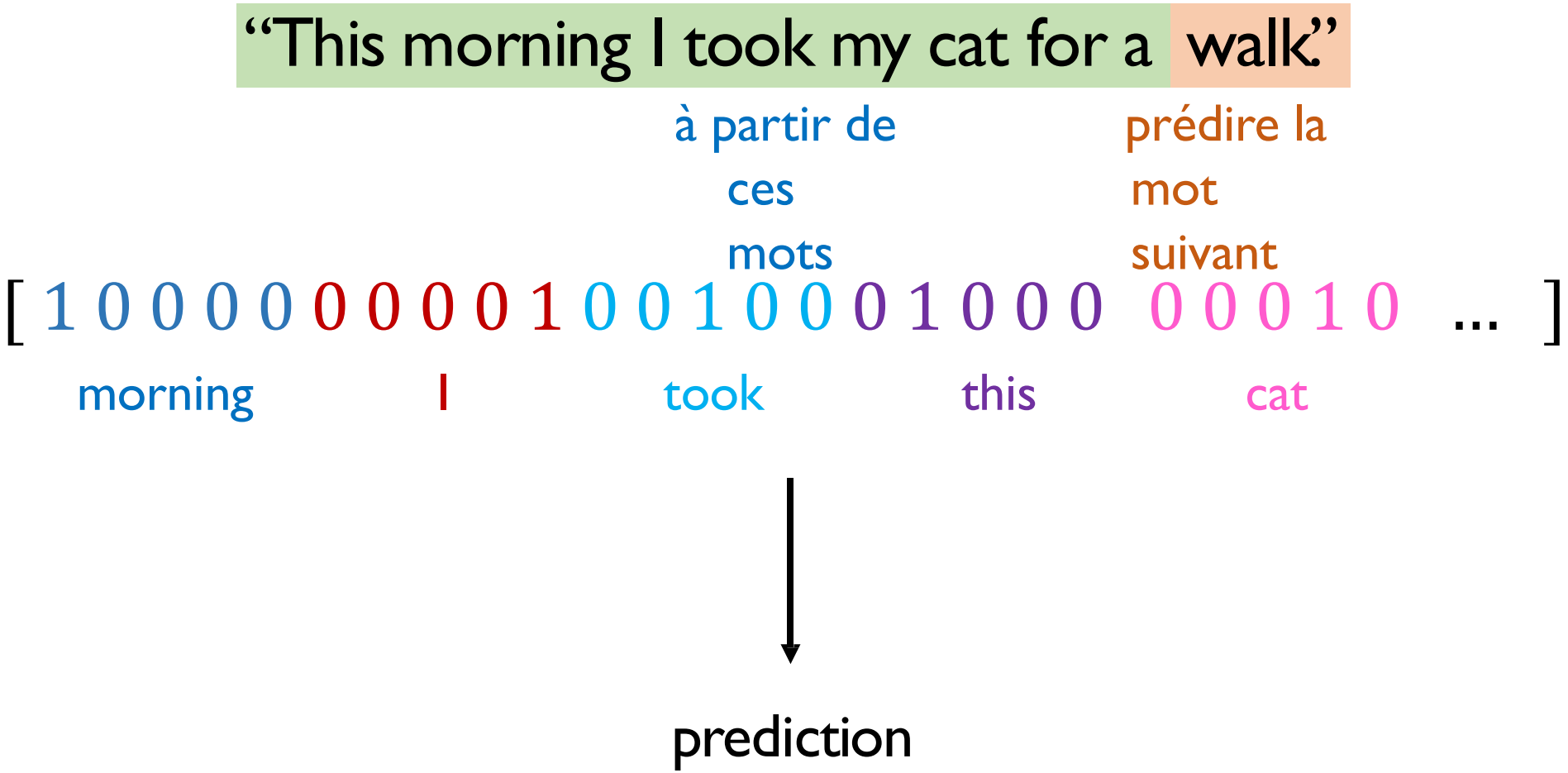
Problème #2: la fréquence ne prend pas en compte l'ordre d'apparition

The food was good, not bad at all.

vs.

The food was bad, not good at all.

Idée #3: utiliser une plus grande longueur



Problème #3: pas de partage de paramètre

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

this morning took the cat

Chacun de ces inputs utilise un paramètre différent:

Problème #3: pas de partage de paramètre

[1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

this morning took the cat

Chacun de ces inputs utilise un paramètre différent:

[0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 ...]

this morning

Problème #3: pas de partage de paramètre

[1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

this morning took the cat

Chacun de ces inputs utilise un paramètre différent:

[0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 ...]

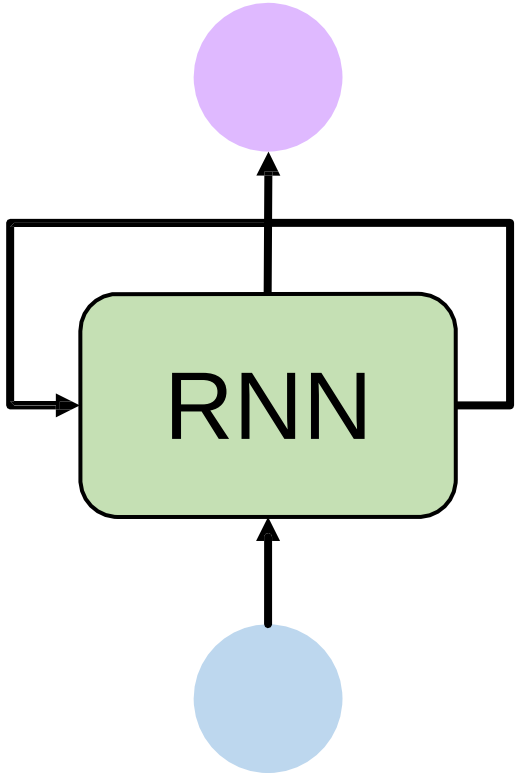
this morning

Il n'y aura pas de transfert si le mot apparait à un endroit différent de la séquence

Modélisation de séquence : critères

Pour modéliser des séquences, on a besoin de :

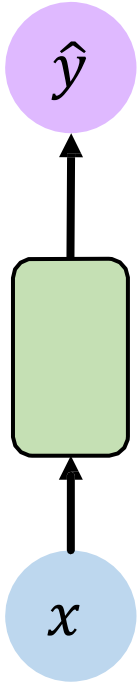
1. Gérer des séquences de longueur variable
2. Prendre en compte les dépendances long terme
3. Prendre en compte l'ordre
4. Partage les paramètres dans toute la séquence



Les Recurrent Neural Networks (RNNs) sont
une approche à la modélisation de séquences.

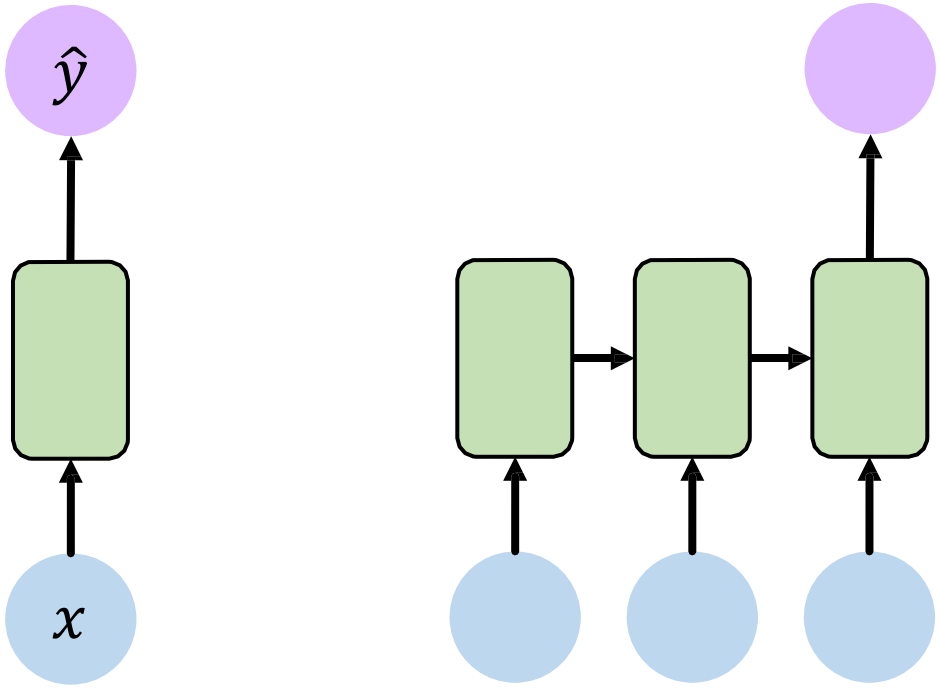
Recurrent Neural Networks (RNNs)

Standard feed-forward neural network



One to One
"Vanilla" neural network

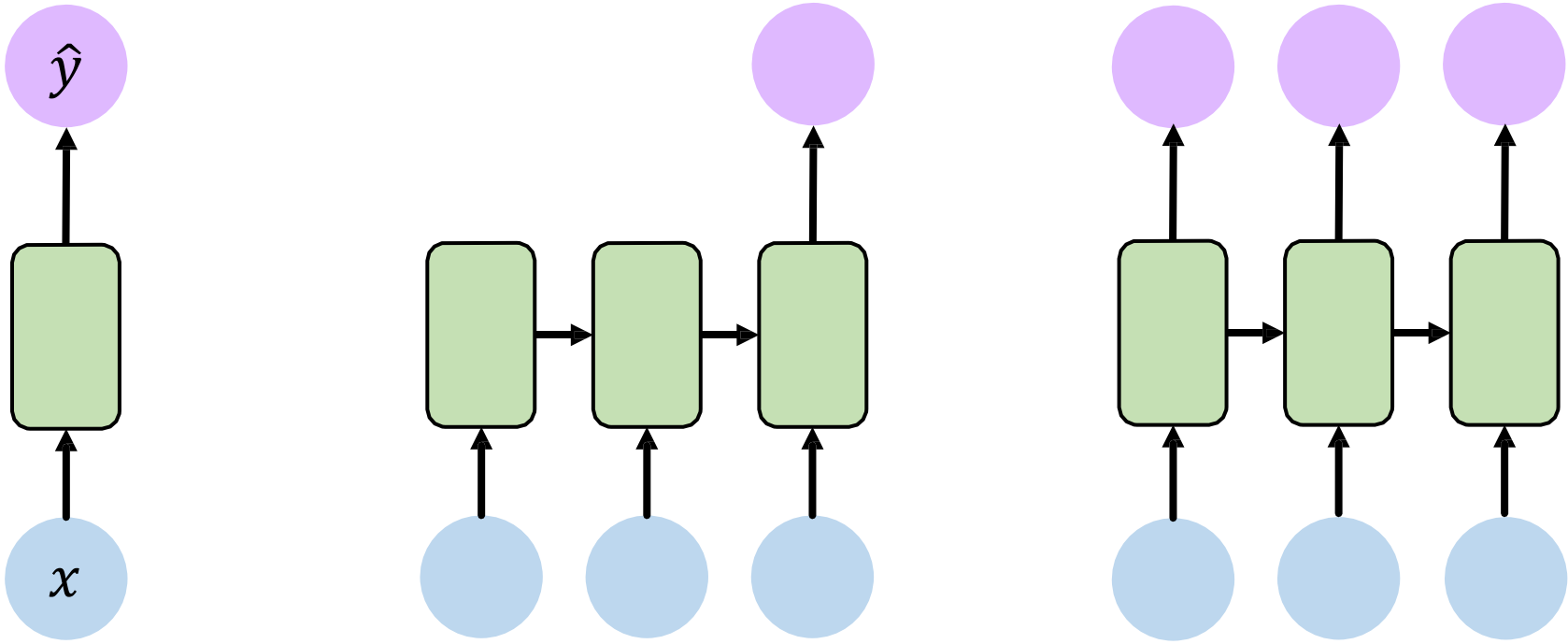
Recurrent neural networks: modélisation de séquences



One to One
"Vanilla" neural network

Many to One
Sentiment Classification

Recurrent neural networks: modélisation de séquences

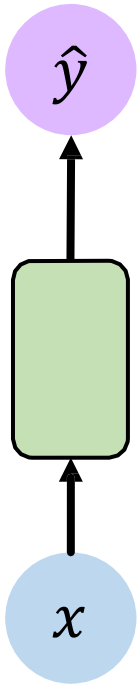


One to One
"Vanilla" neural network

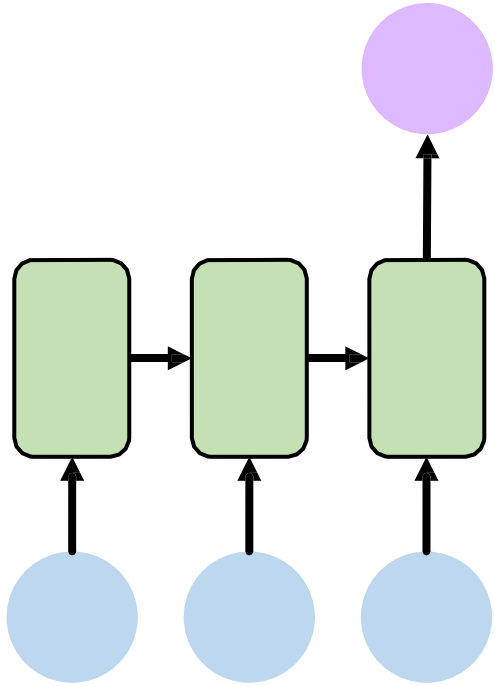
Many to One
Sentiment Classification

Many to Many
Music Generation

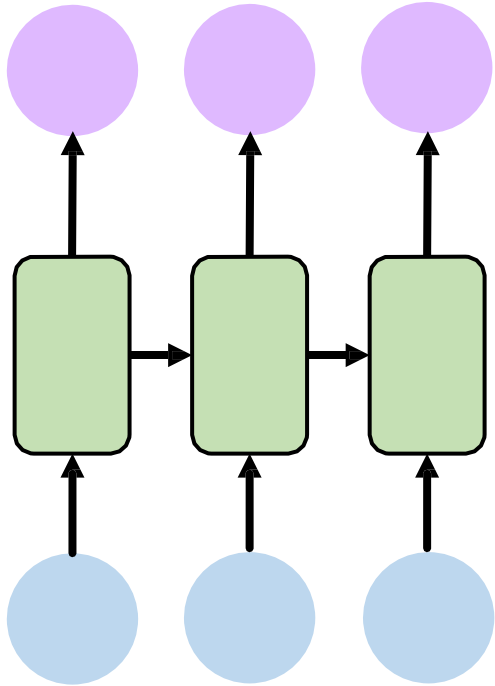
Recurrent neural networks: modélisation de séquences



One to One
"Vanilla" neural network



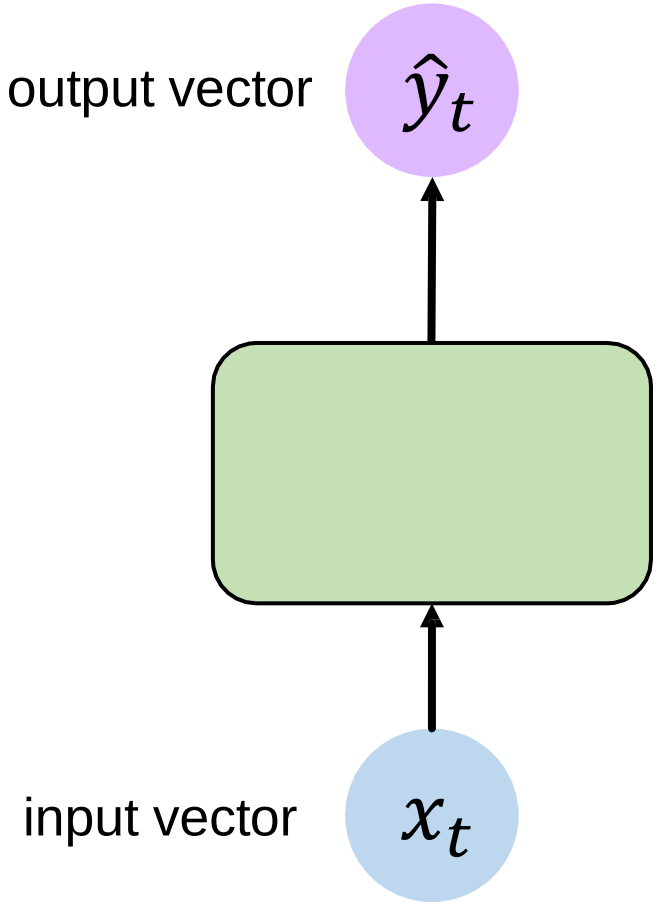
Many to One
Sentiment Classification



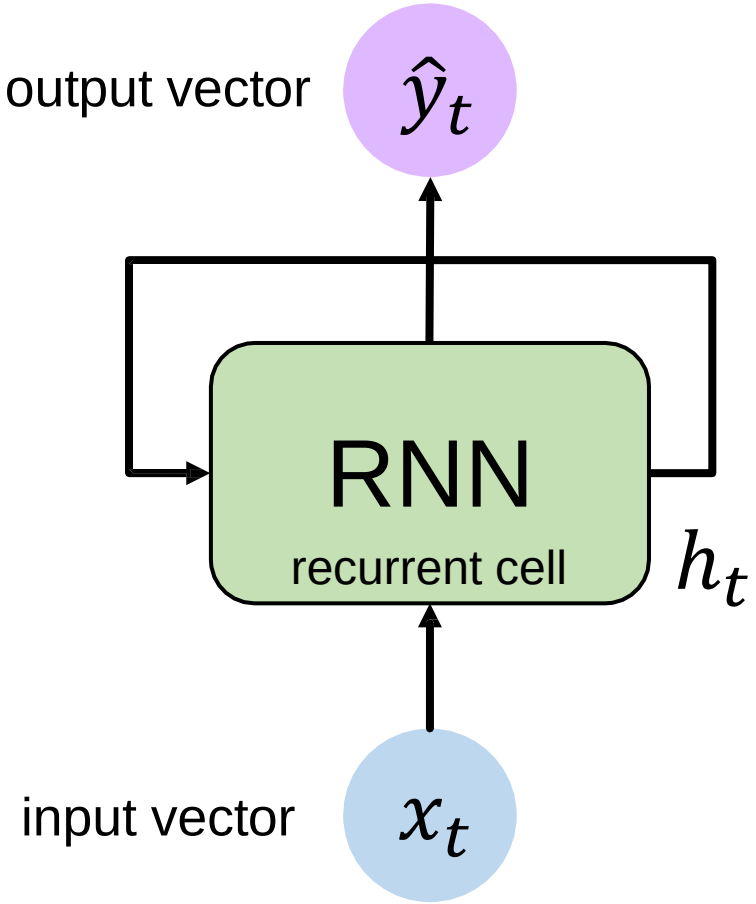
Many to Many
Music Generation

… ainsi que d'autres architectures et applications

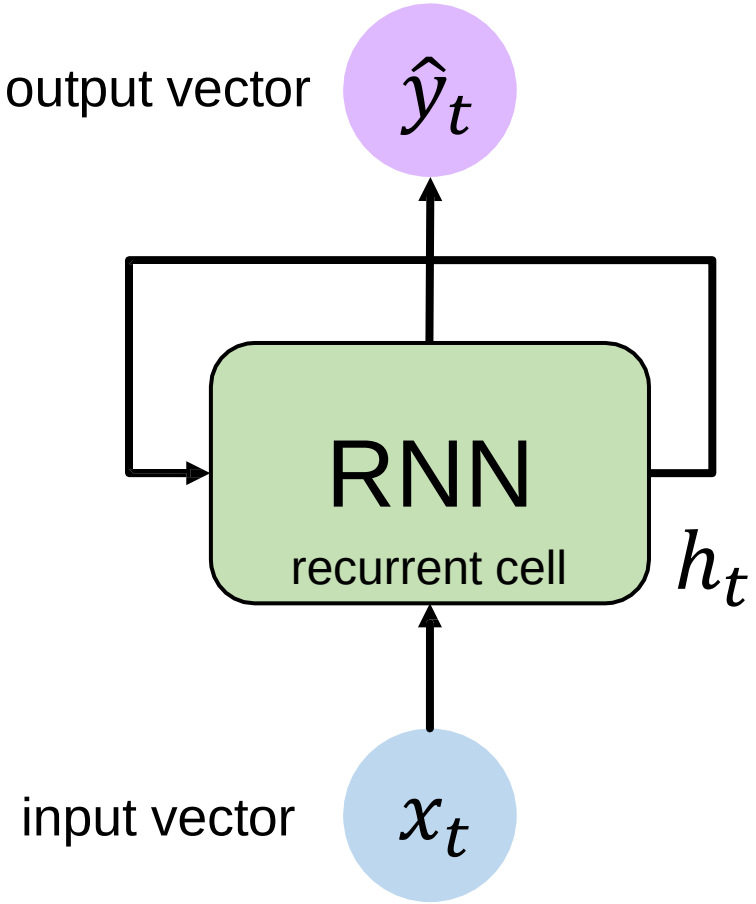
“Vanilla” neural network



Recurrent neural network (RNN)

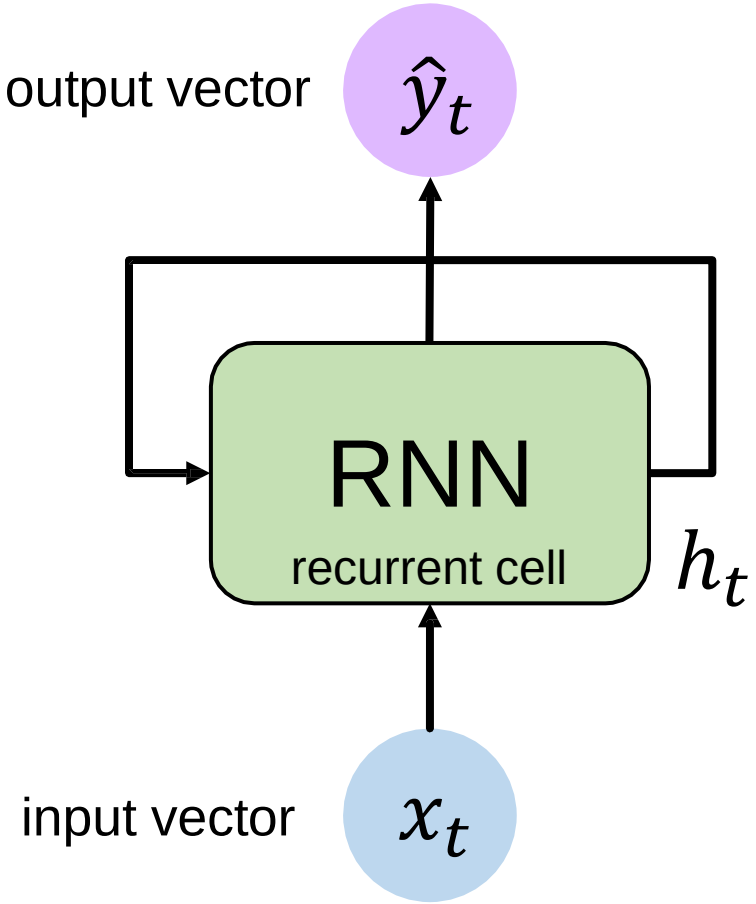


Recurrent neural network (RNN)



Applique une relation de récurrence à chaque intervalle de temps pour traiter la séquence

Recurrent neural network (RNN)

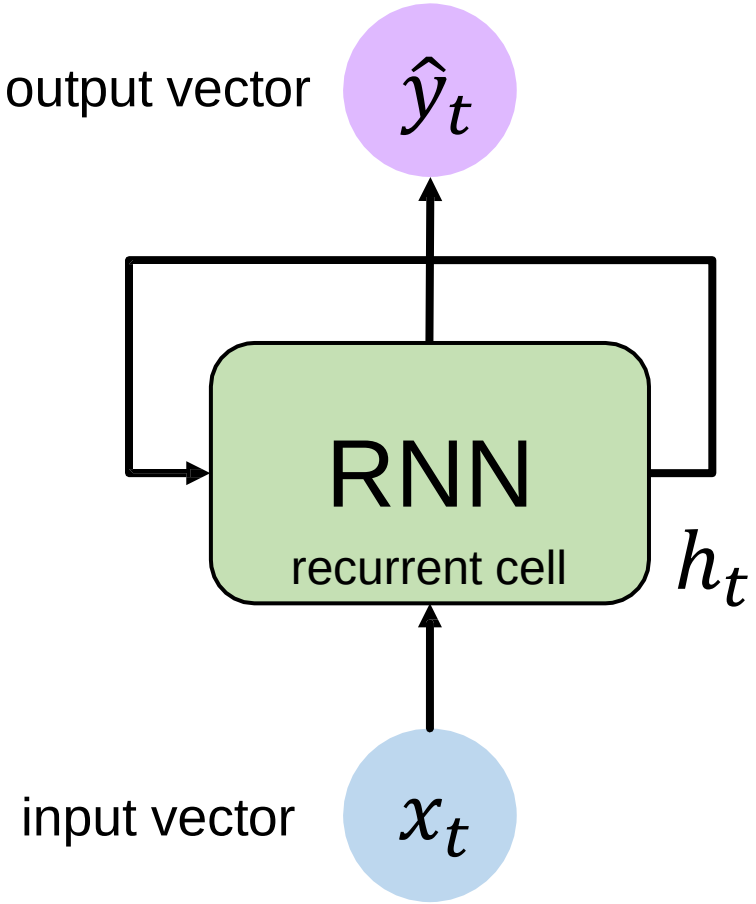


Applique une relation de récurrence à chaque intervalle de temps pour traiter la séquence

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

- h_t : cell state
- f_W : fonction paramétrée par W
- h_{t-1} : ancien état
- x_t : input vector à l'intervalle t

Recurrent neural network (RNN)



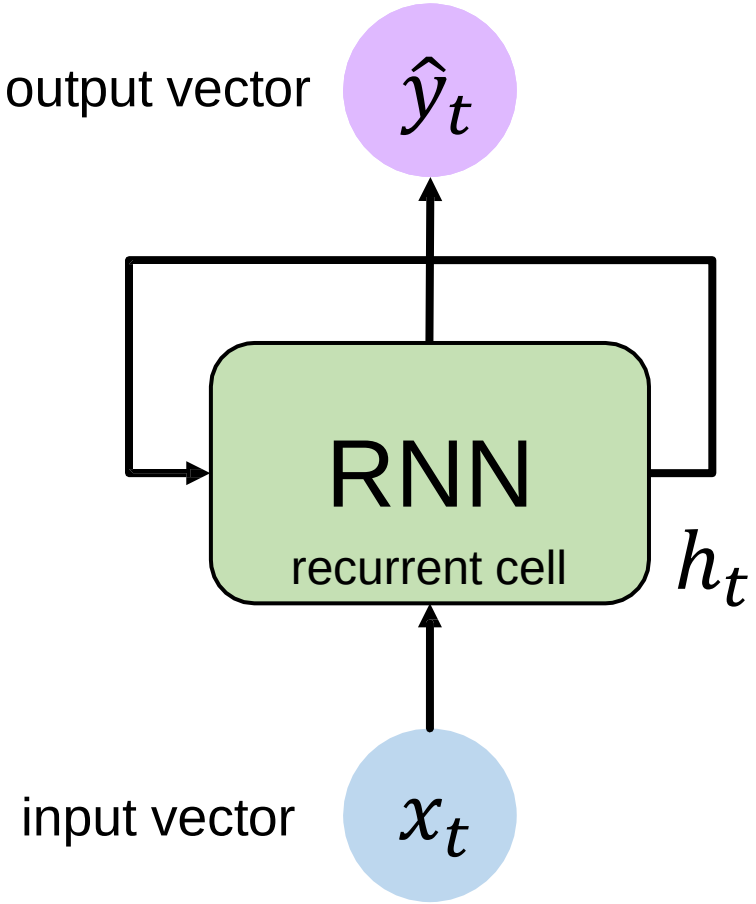
Applique une relation de récurrence à chaque intervalle de temps pour traiter la séquence

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

- h_t : nouvel état
- f_W : Fonction paramétrée par W
- h_{t-1} : ancien état
- x_t : input vector à l'intervalle t

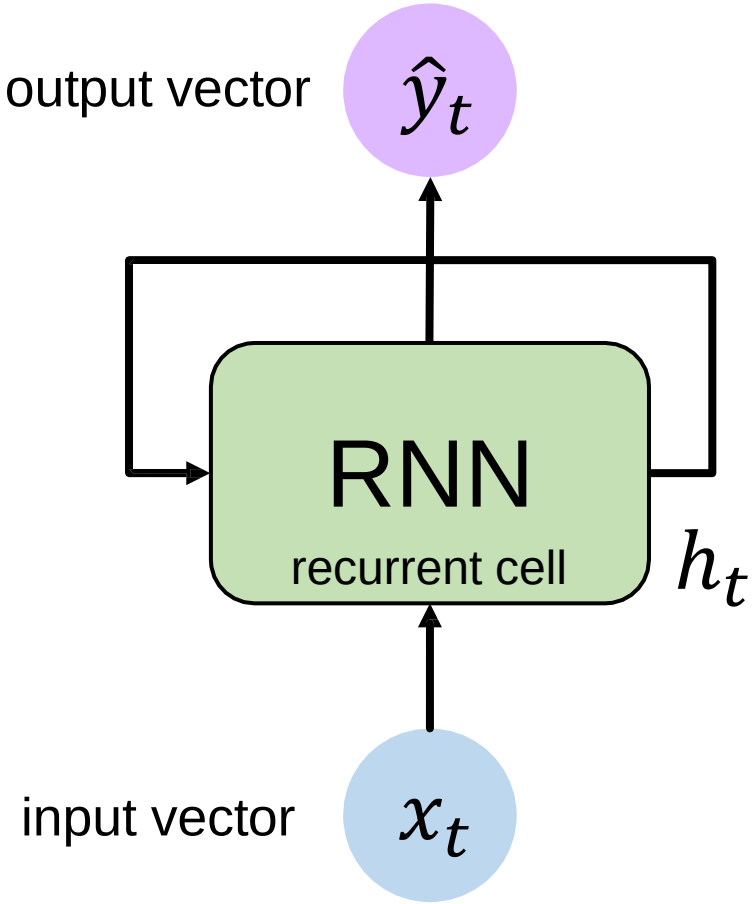
Attention : la même fonction et ensemble de paramètre sont utilisés à chaque intervalle de temps

RNN state maj et output



Input Vector

RNN state maj et output

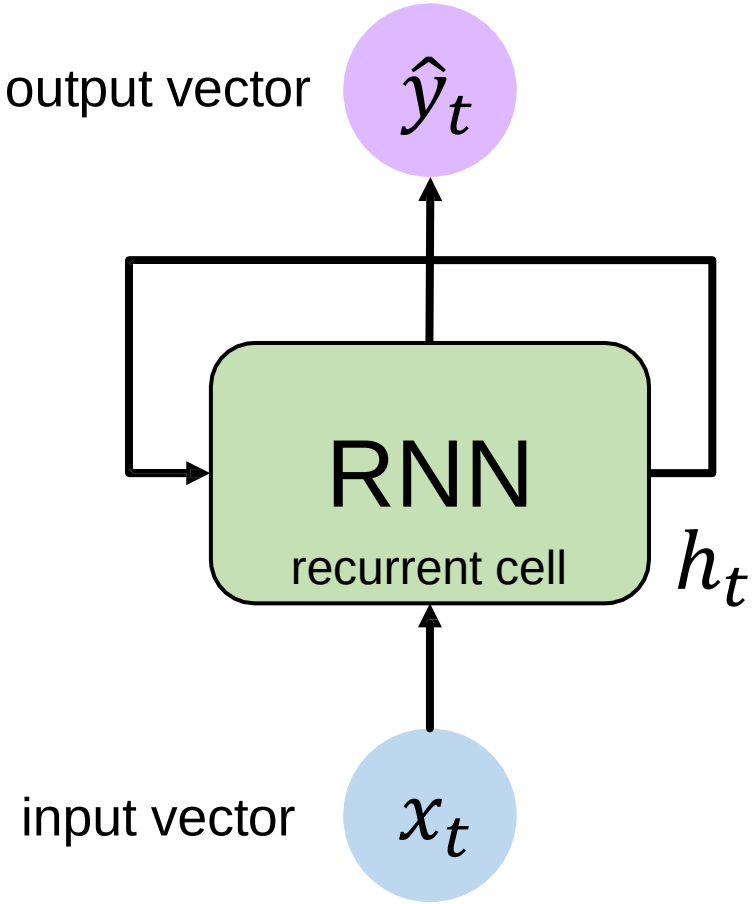


M.a.j. Hidden State

$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

Input Vector

RNN state maj et output



OutputVector

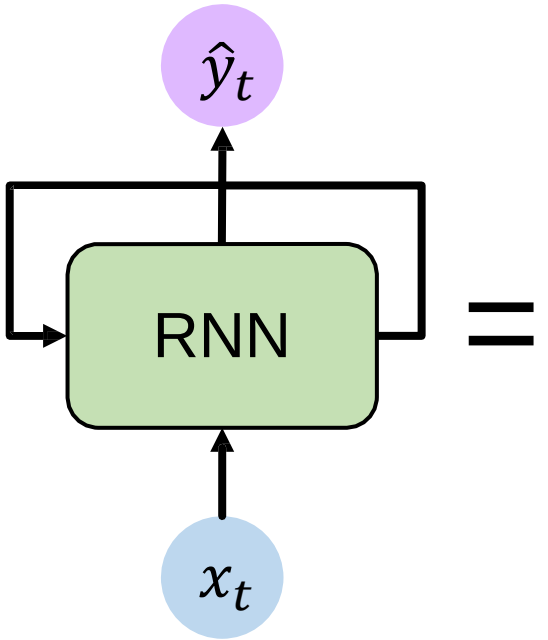
$$\hat{y}_t = \boldsymbol{W_{hy}} h_t$$

M.a.j. Hidden State

$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

Input Vector

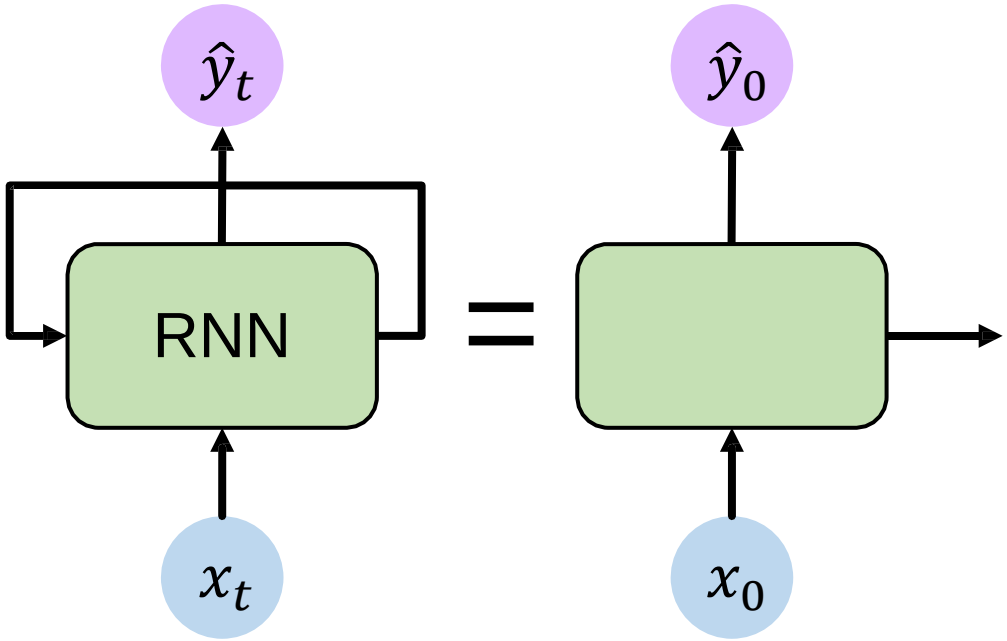
RNNs: computational graph across time



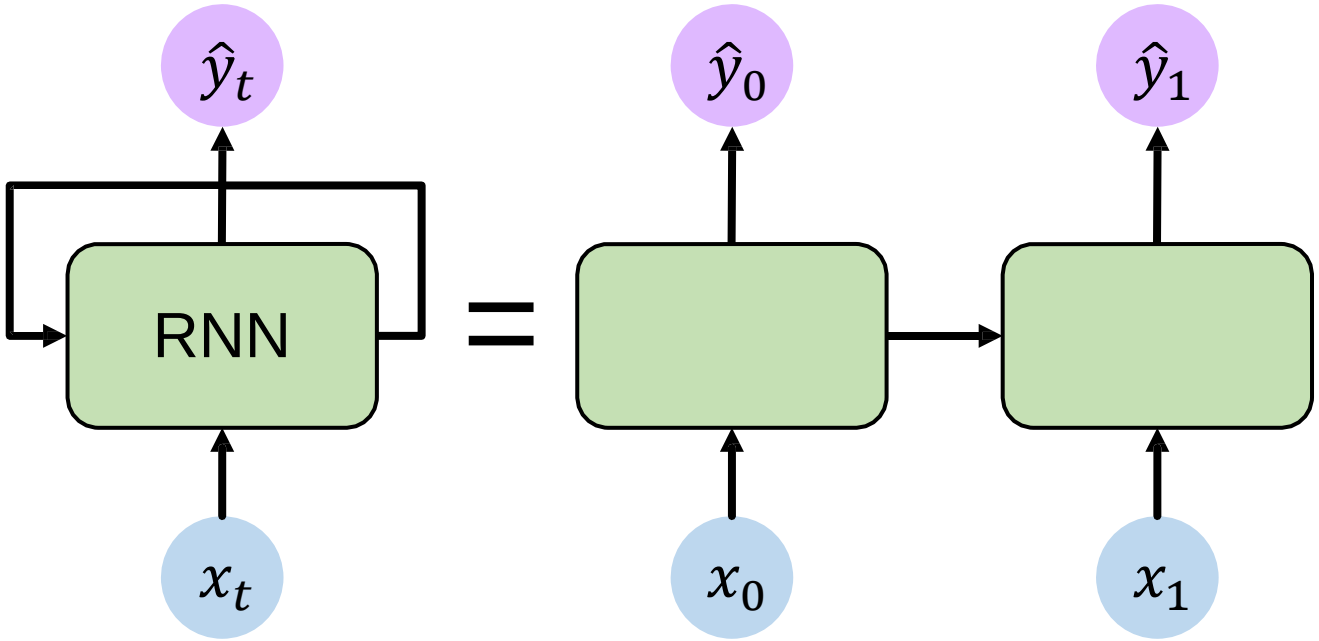
=

Correspond à un graph étiqueté que l'on déroule à travers le temps

RNNs: computational graph across time

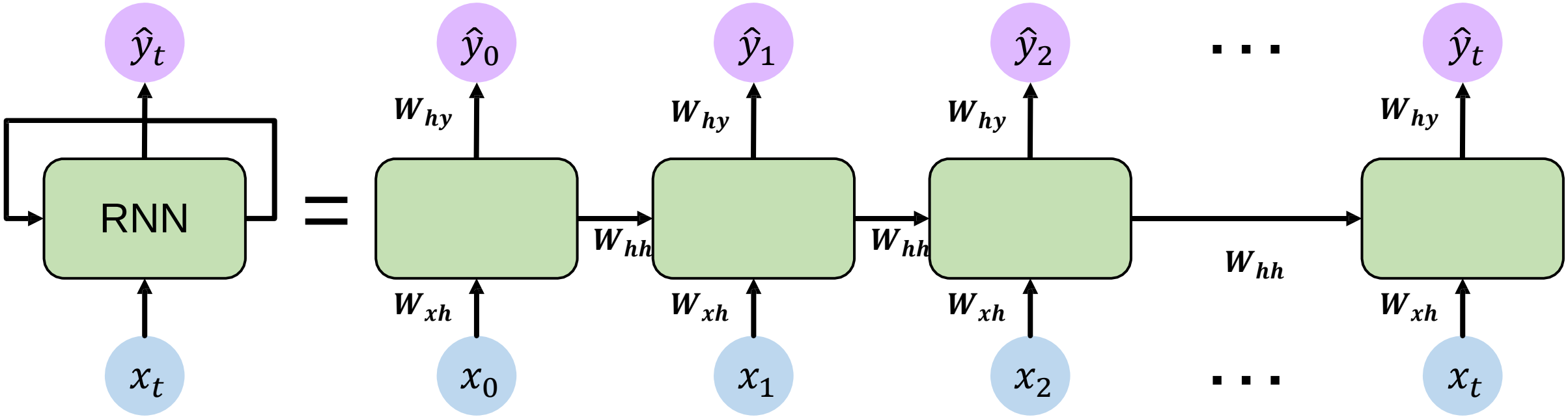


RNNs: computational graph across time

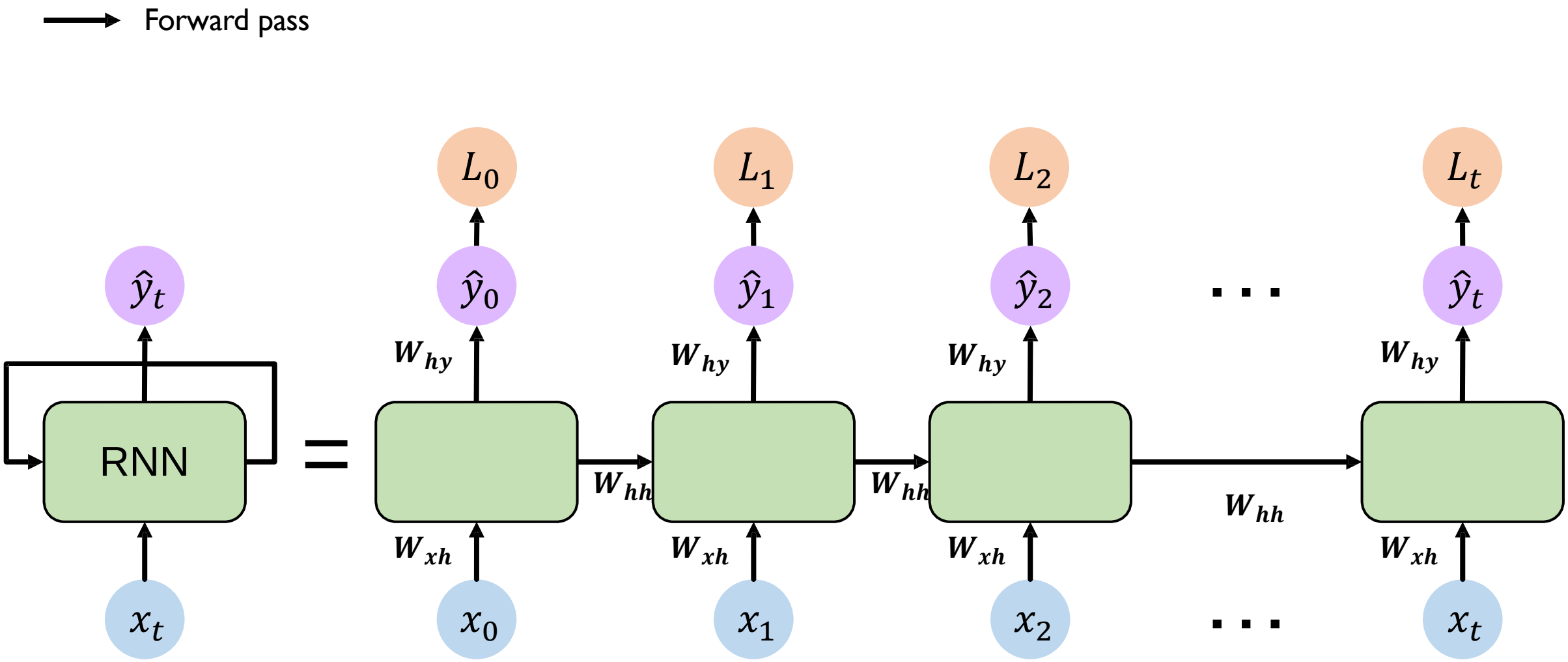


RNNs: computational graph across time

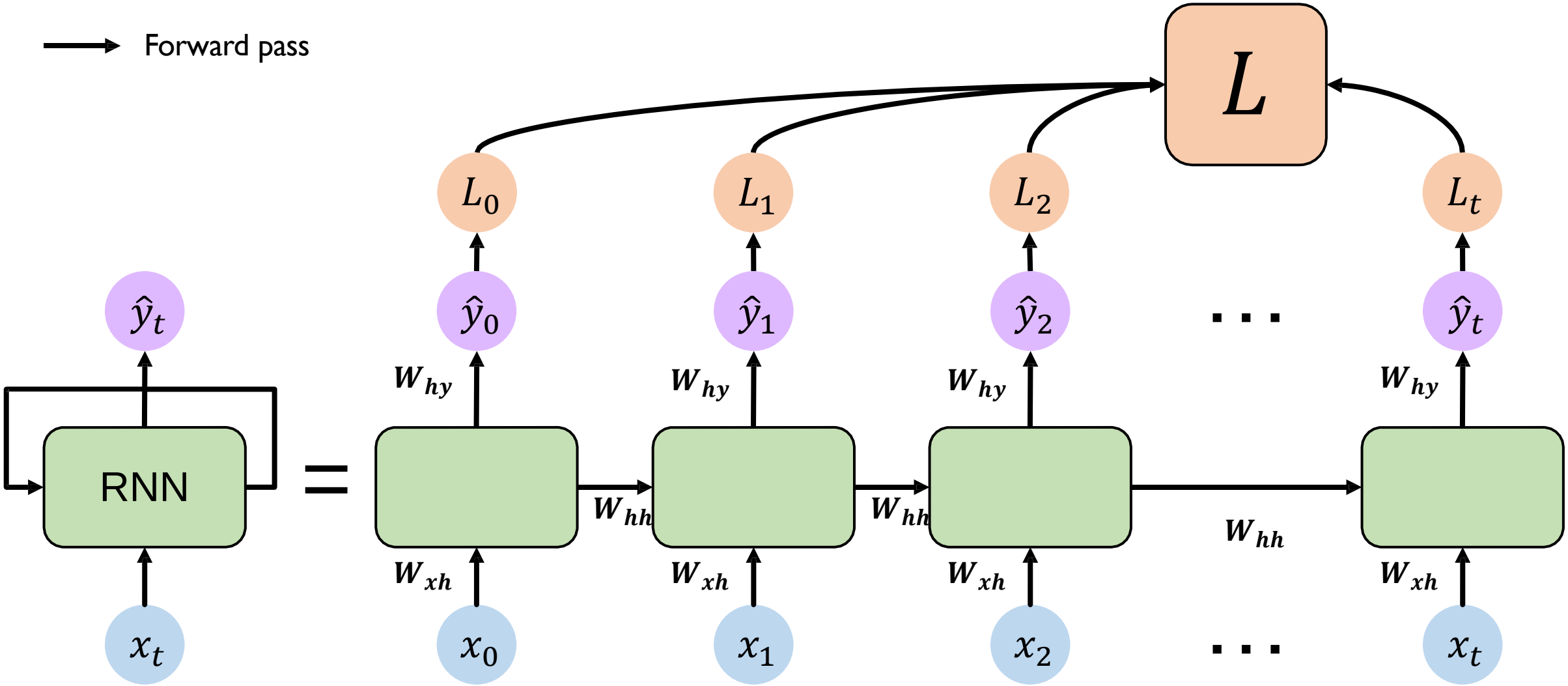
On ré-utilise la même matrice de poids à chaque intervalle



RNNs: computational graph across time

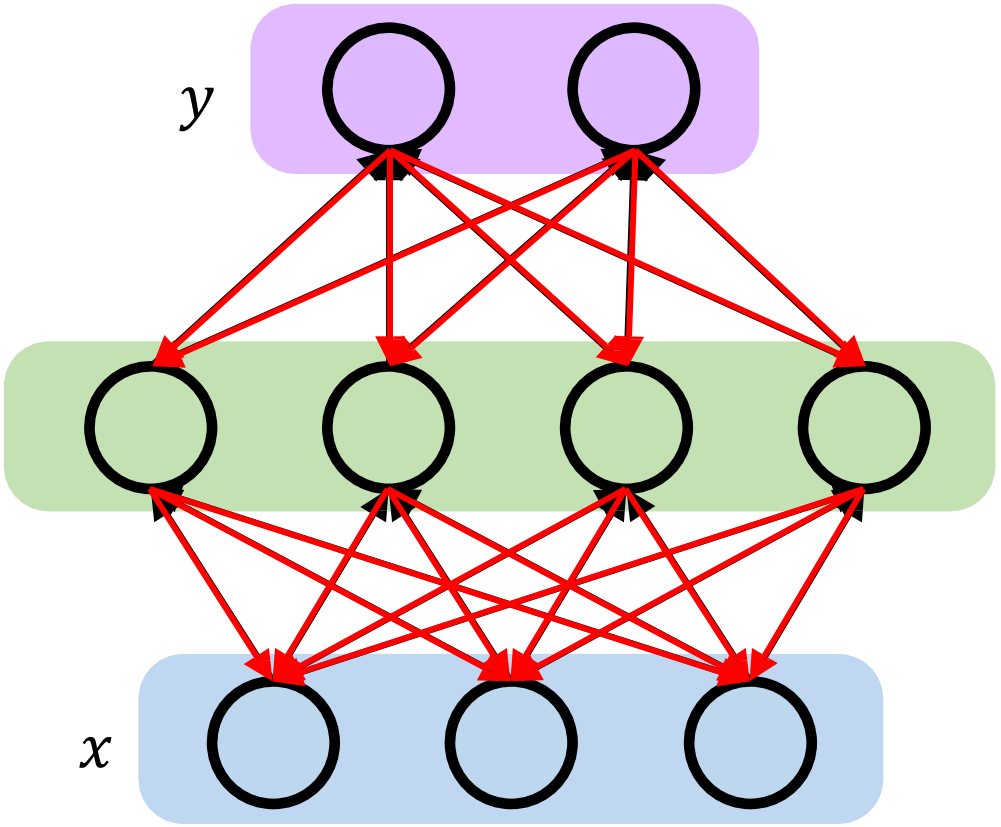


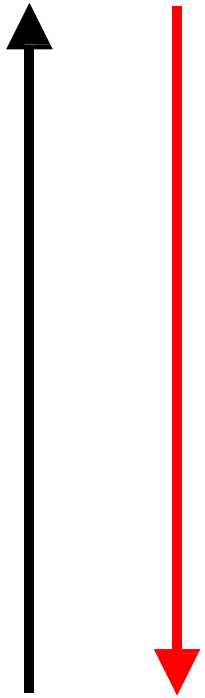
RNNs: computational graph across time



Backpropagation Through Time (BPTT)

Recall: backpropagation in feed forward models

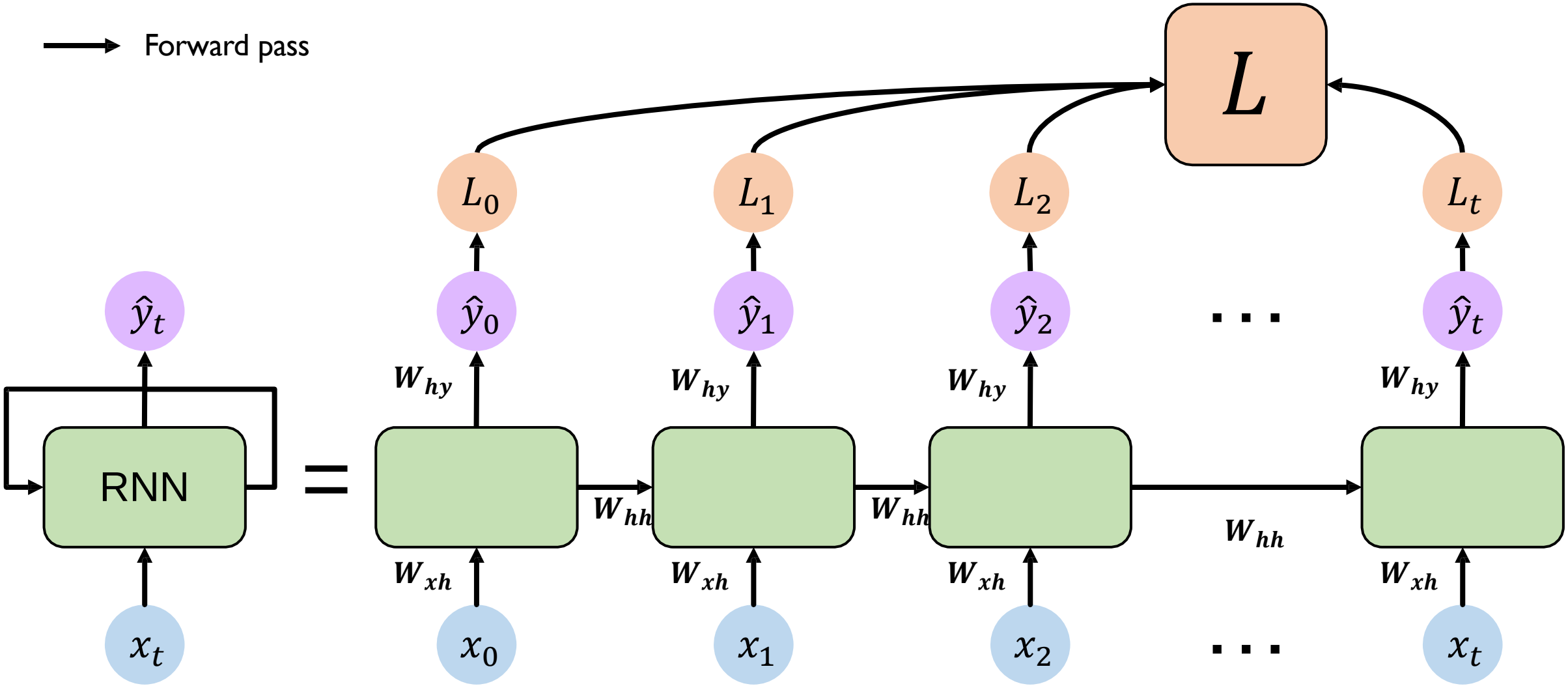




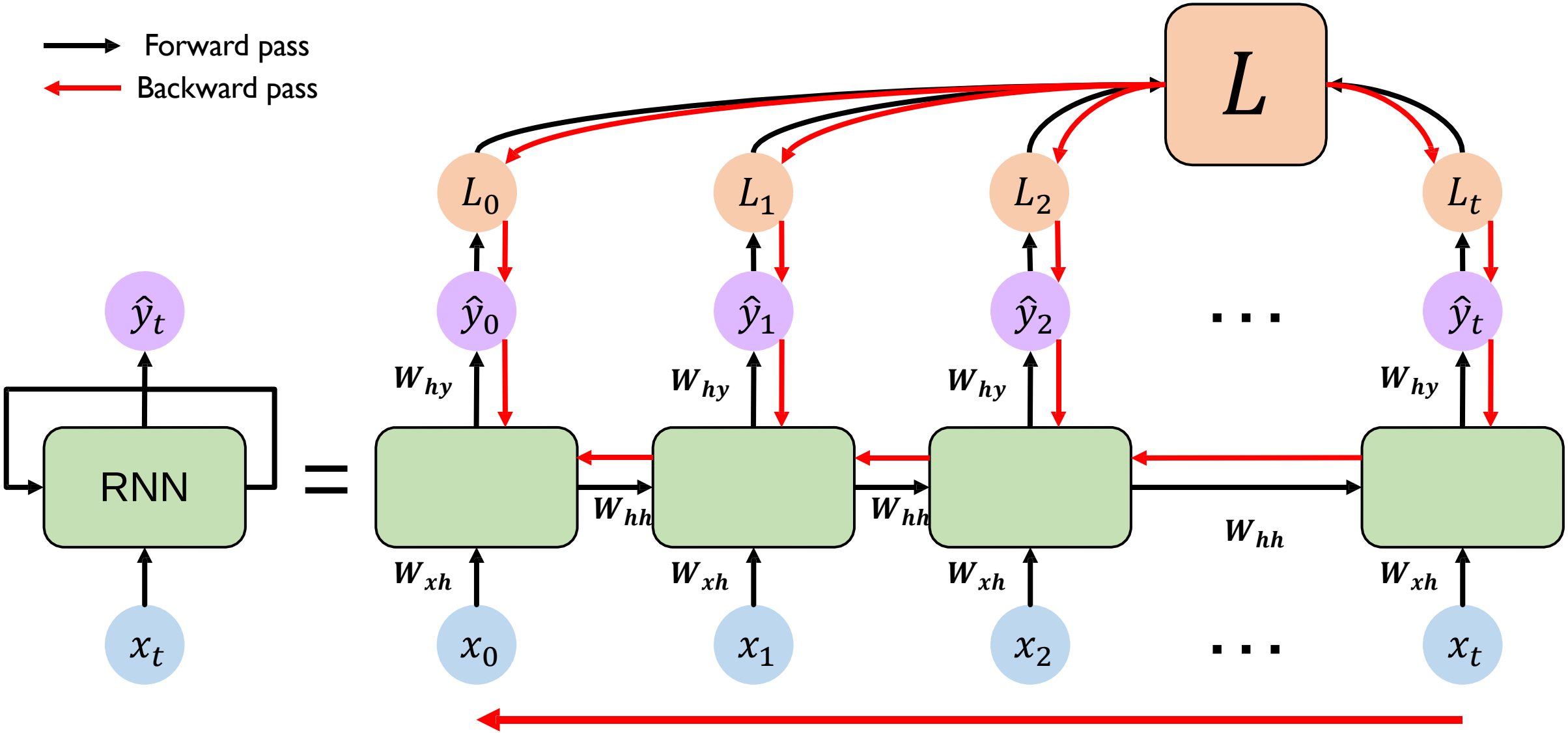
Algorithme de Backpropagation:

1. On calcule la dérivée (gradient) de la perte par rapport à chaque paramètre
2. On ajuste les paramètres de sorte à minimiser la perte

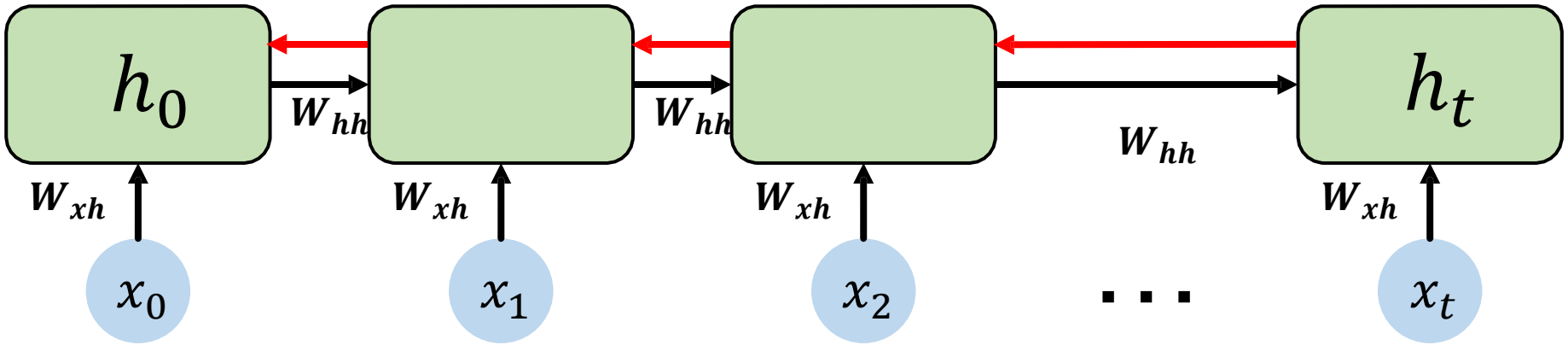
RNNs: backpropagation through time



RNNs: backpropagation through time

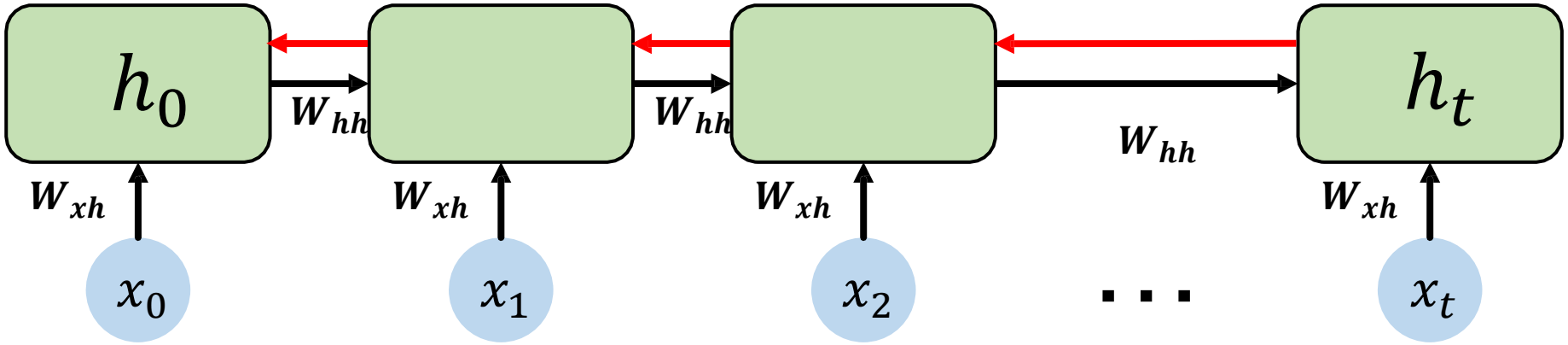


Standard RNN gradient flow



Calculer le gradient par rapport à h_0 implique de nombreuses interactions avec la matrice $\boldsymbol{W_{hh}}$ (et $\boldsymbol{f'}$!)

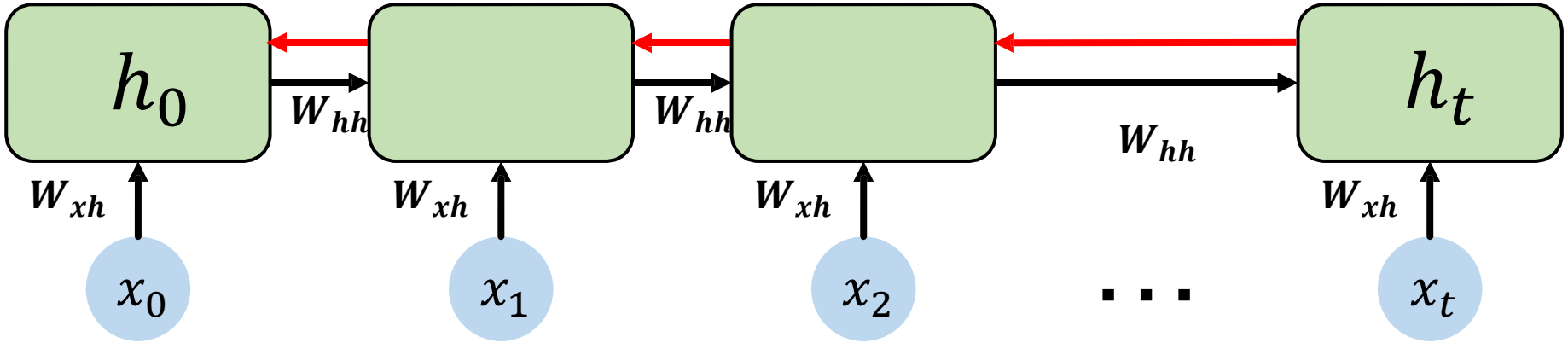
Standard RNN gradient flow: exploding gradients



Calculer le gradient par rapport à h_0 implique de nombreuses interactions avec la matrice $\boldsymbol{W_{hh}}$ (et $\boldsymbol{f'}$!)

Plusieurs valeurs > 1: exploding gradients

Standard RNN gradient flow: exploding gradients

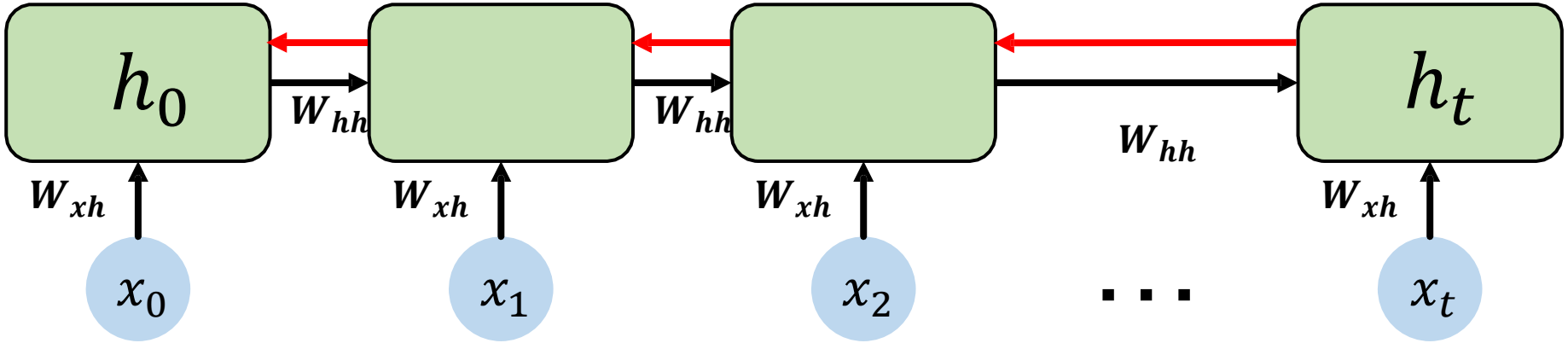


Calculer le gradient par rapport à h_0 implique de nombreuses interactions avec la matrice $\boldsymbol{W_{hh}}$ (et $\boldsymbol{f'}$!)

Plusieurs valeurs> 1: exploding gradients

On peut utiliser le "gradient clipping" pour scaler

Standard RNN gradient flow: vanishing gradients



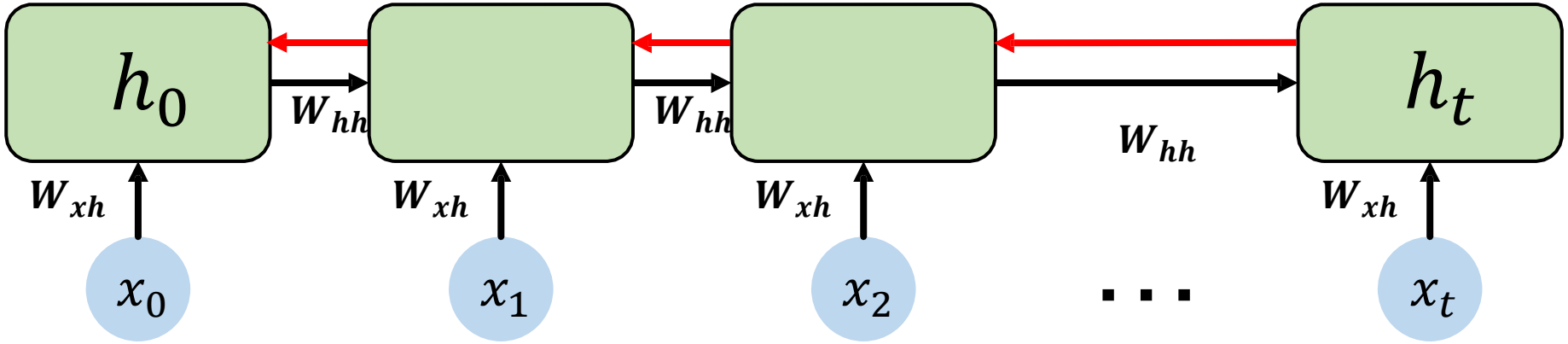
Calculer le gradient par rapport à h_0 implique de nombreuses interactions avec la matrice $\boldsymbol{W_{hh}}$ (et $\boldsymbol{f'}$!)

Plusieurs valeurs> 1: exploding gradients

On peut utiliser le "gradient clipping" pour scaler

Plusieurs valeurs < 1: vanishing gradients

Standard RNN gradient flow: vanishing gradients



Calculer le gradient par rapport à h_0 implique de nombreuses interactions avec la matrice $\boldsymbol{W_{hh}}$ (et $\boldsymbol{f'}$!)

Plusieurs valeurs> 1: exploding gradients

On peut utiliser le "gradient clipping" pour scaler

Plusieurs valeurs < 1: vanishing gradients

1. Fonction d'activation
2. Initialisation des poids
3. Changer l'architecture

Le problème des dépendances long terme

Pourquoi les vanishing gradients sont problématiques ?

On multiplie plusieurs petits nombres ensemble

Le problème des dépendances long terme

Pourquoi les vanishing gradients sont problématiques ?

On multiplie plusieurs petits nombres ensemble

↓

Les erreurs des intervalles les plus éloignés ont des gradients de plus en plus petits

Le problème des dépendances long terme

Pourquoi les vanishing gradients sont problématiques ?

On multiplie plusieurs petits nombres ensemble

↓

Les erreurs des intervalles les plus éloignés ont des gradients de plus en plus petits

↓

On biaise alors les paramètres pour capturer les dépendances à court terme

Le problème des dépendances long terme

Pourquoi les vanishing gradients sont problématiques ?

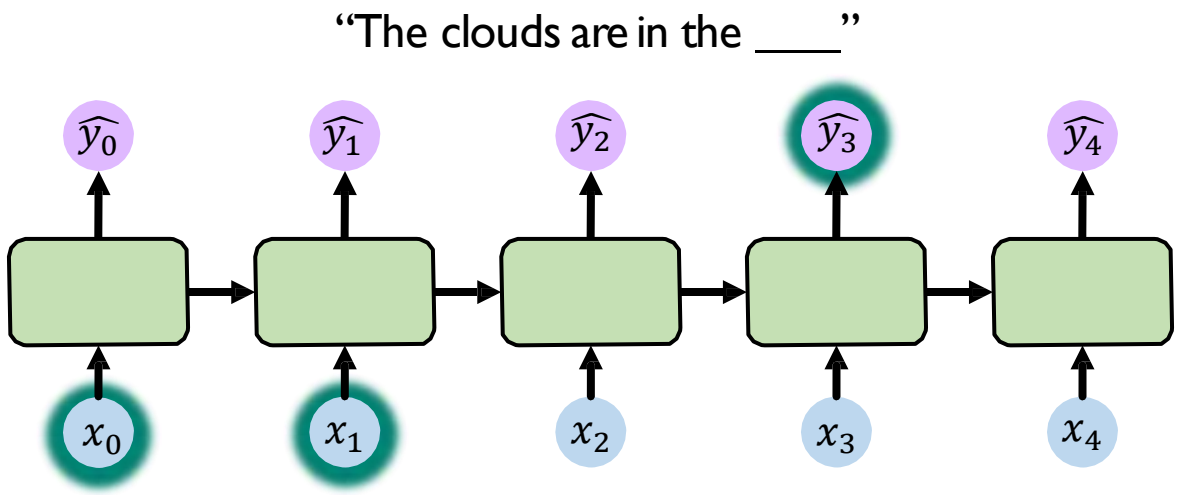
On multiplie plusieurs petits nombres ensemble

↓

Les erreurs des intervalles les plus éloignés ont des gradients de plus en plus petits

↓

On biaise alors les paramètres pour capturer les dépendances à court terme



Le problème des dépendances long terme

Pourquoi les vanishing gradients sont problématiques ?

On multiplie plusieurs petits nombres ensemble

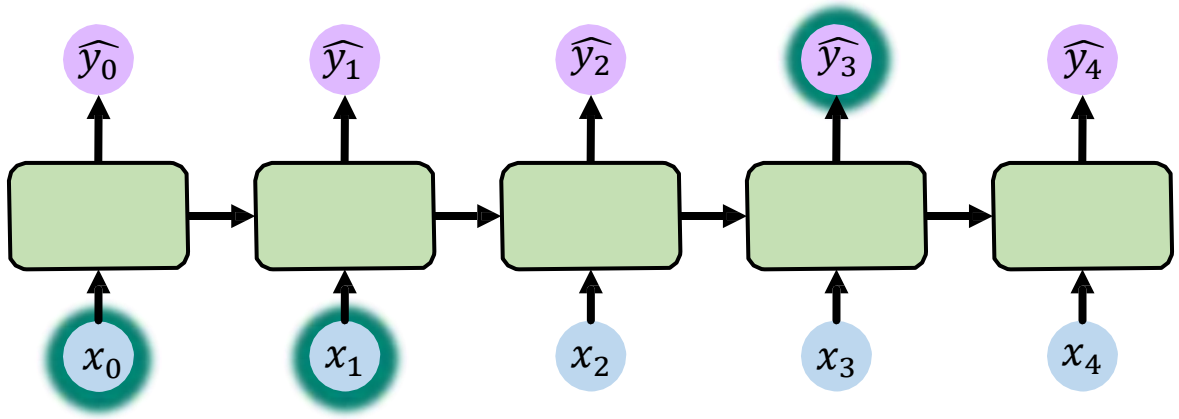
↓

Les erreurs des intervalles les plus éloignés ont des gradients de plus en plus petits

↓

On biaise alors les paramètres pour capturer les dépendances à court terme

"The clouds are in the ____"



"I grew up in France, … and I l speak fluent____"

Le problème des dépendances long terme

Pourquoi les vanishing gradients sont problématiques ?

On multiplie plusieurs petits nombres ensemble

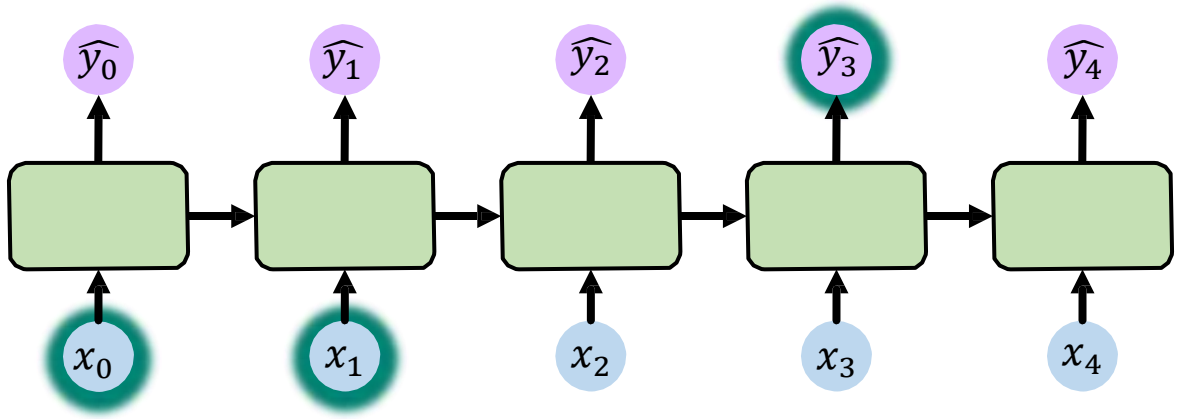
↓

Les erreurs des intervalles les plus éloignés ont des gradients de plus en plus petits

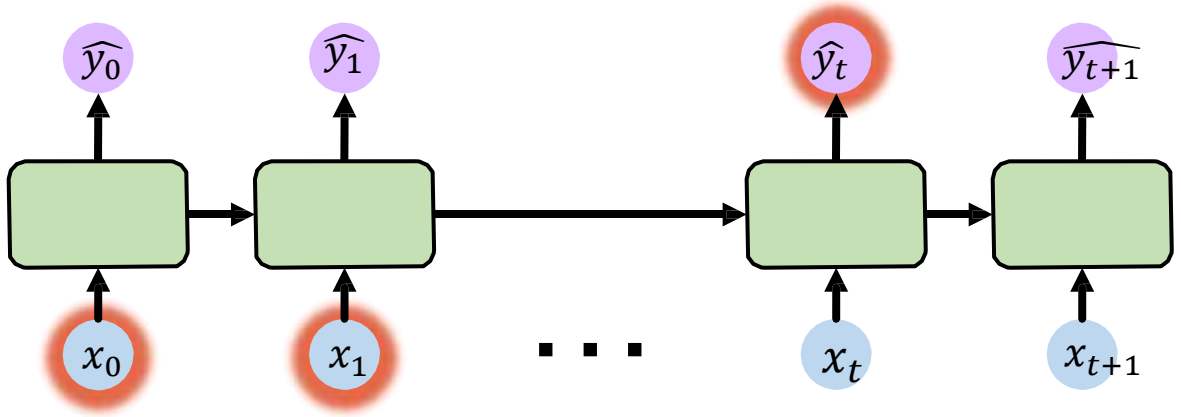
↓

On biaise alors les paramètres pour capturer les dépendances à court terme

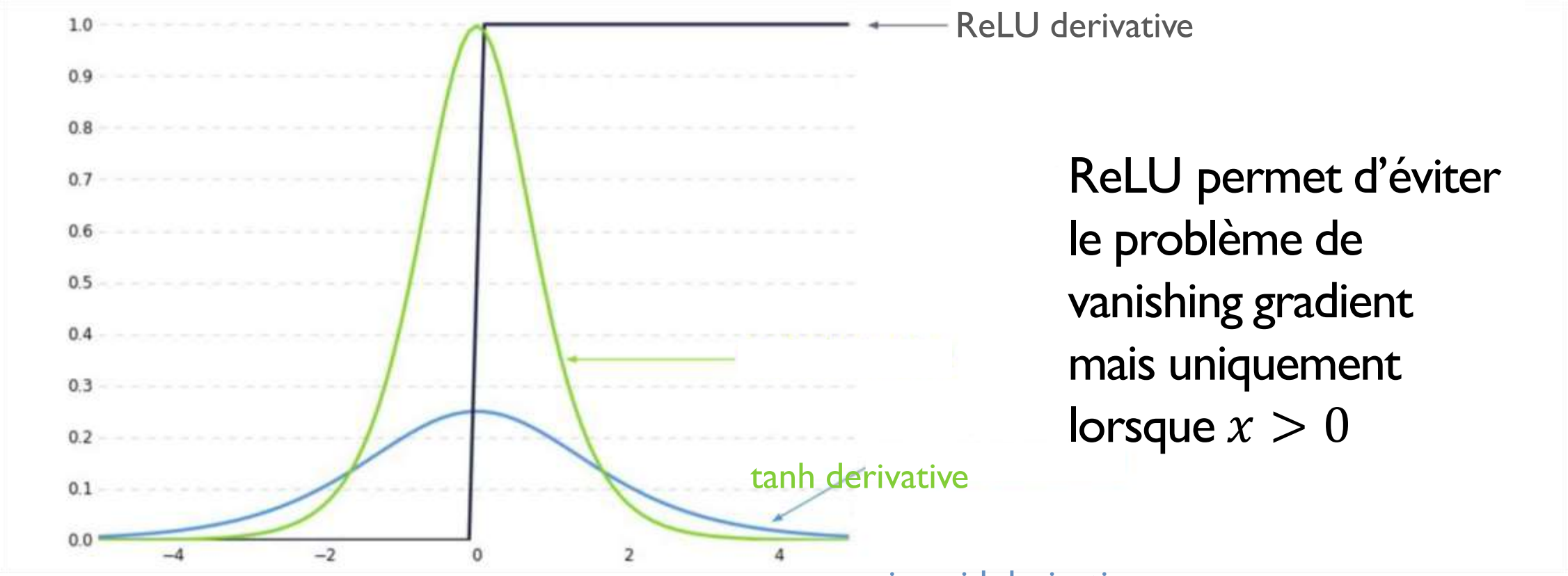
"The clouds are in the ____"



"I grew up in France,… and I l speak fluent____"



Solution #1: Fonctions d'activation



ReLU permet d'éviter le problème de vanishing gradient mais uniquement lorsque $x > 0$

Solution #2: Initialisation des paramètres

On peut utiliser la matrice identité

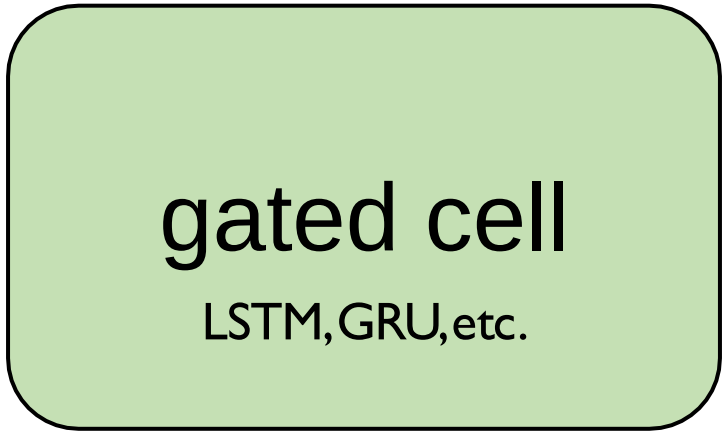
Les biais sont traités par les zéros

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Permet d'éviter que les poids tendent vers zéro

Solution #3: les gated cells

On utilise une cellule récurrente plus complexe avec des « portes » pour filtrer l'information qui y circule

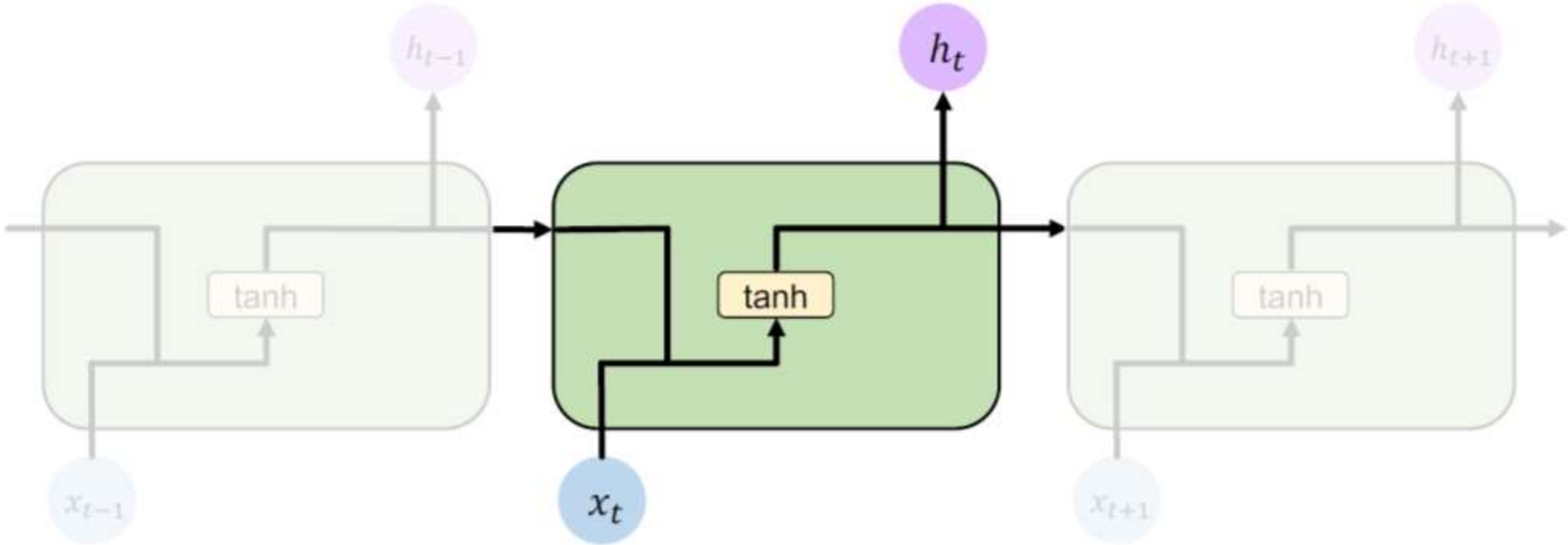


Les réseaux Long Short Term Memory (LSTMs) utilisent des gated cells pour traiter l'information à chaque intervalle de temps.

Long Short Term Memory (LSTM) Networks

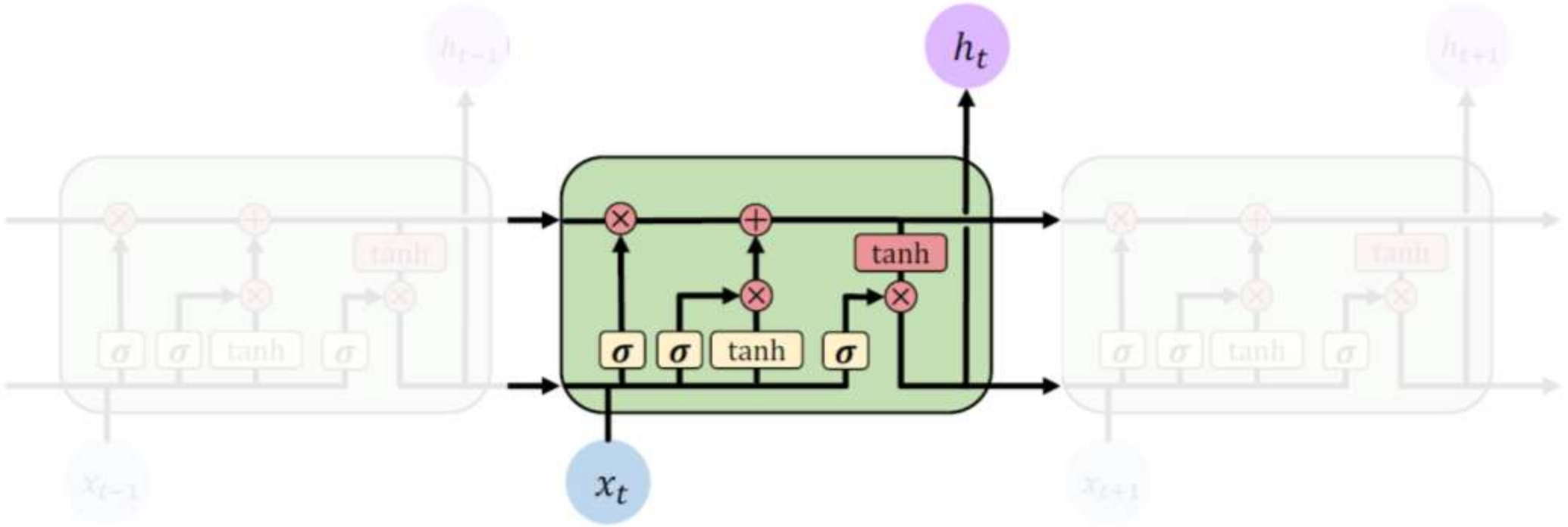
Standard RNN

Dans un RNN classique, un simple calcul intervient dans chaque module.



Long Short Term Memory (LSTMs)

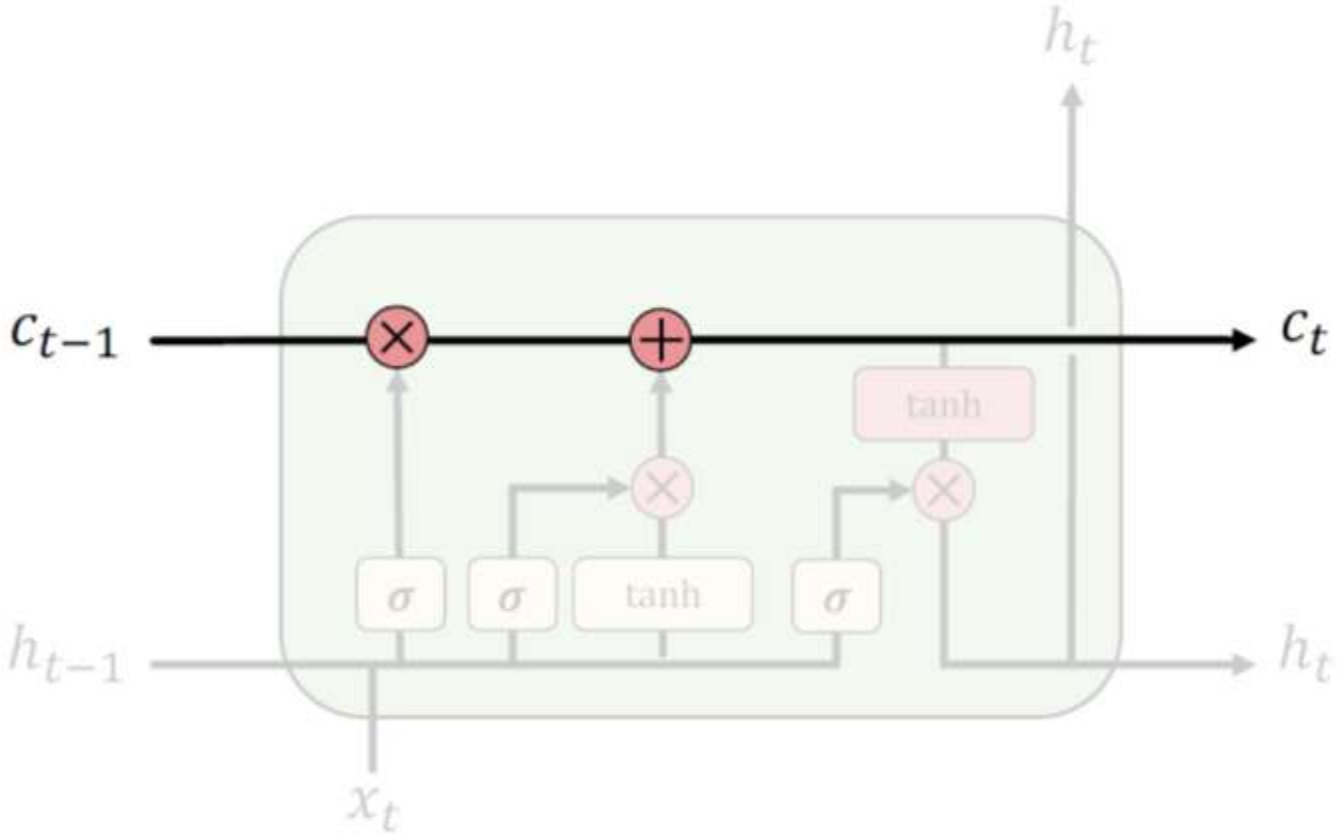
Dans un LSTM, de nombreuses couches filtrant l'information interviennent dans chaque module



Les cellules d'un LSTM sont capables de filtrer l'information

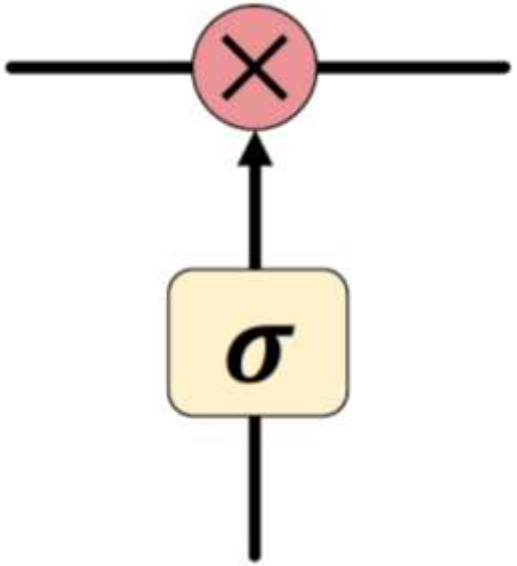
Long Short Term Memory (LSTMs)

Un nouveau cell state c_t est maintenu tout au long du processus



Long Short Term Memory (LSTMs)

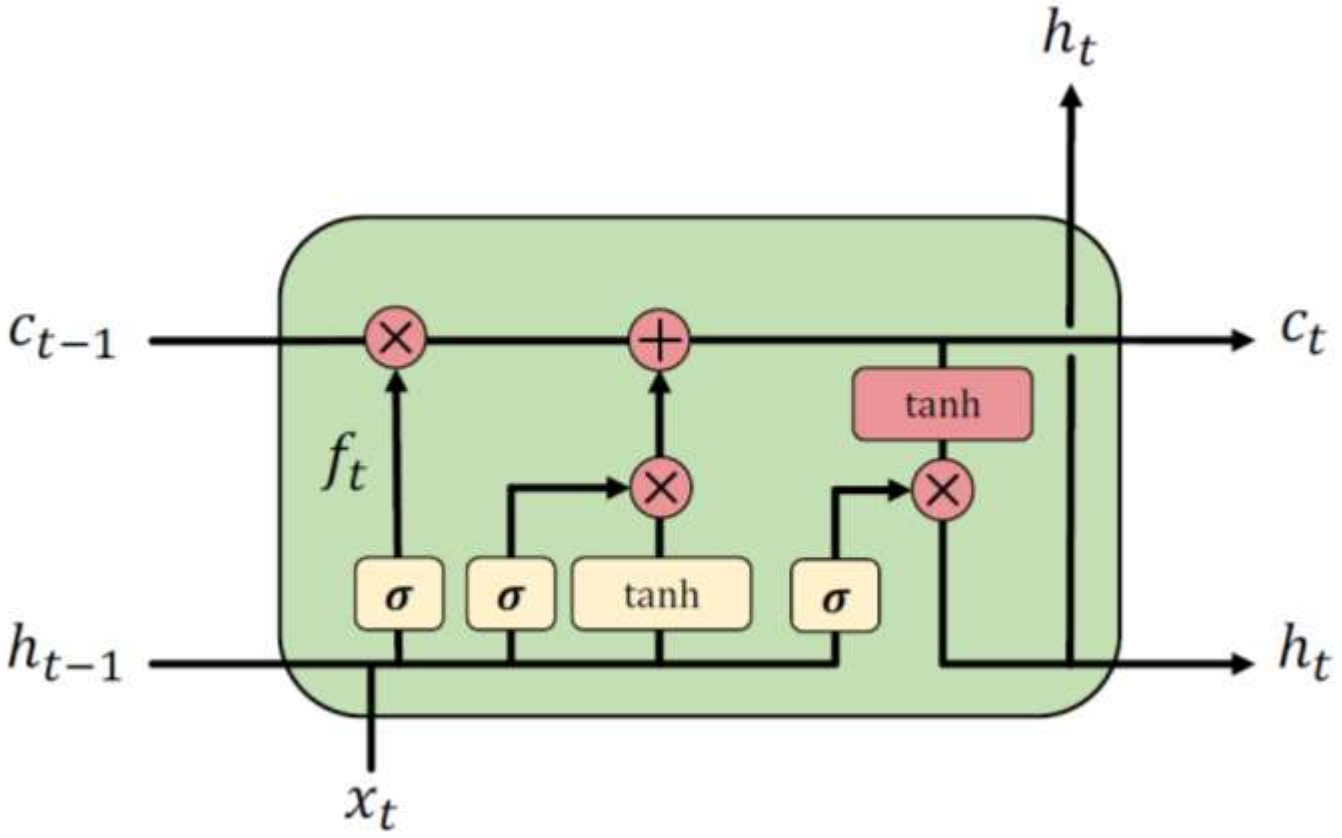
L'information est supprimée ou ajoutée au cell state c_t grâce à des gates



Les gates traitent l'information grâce à une fonction sigmoïde et une multiplication

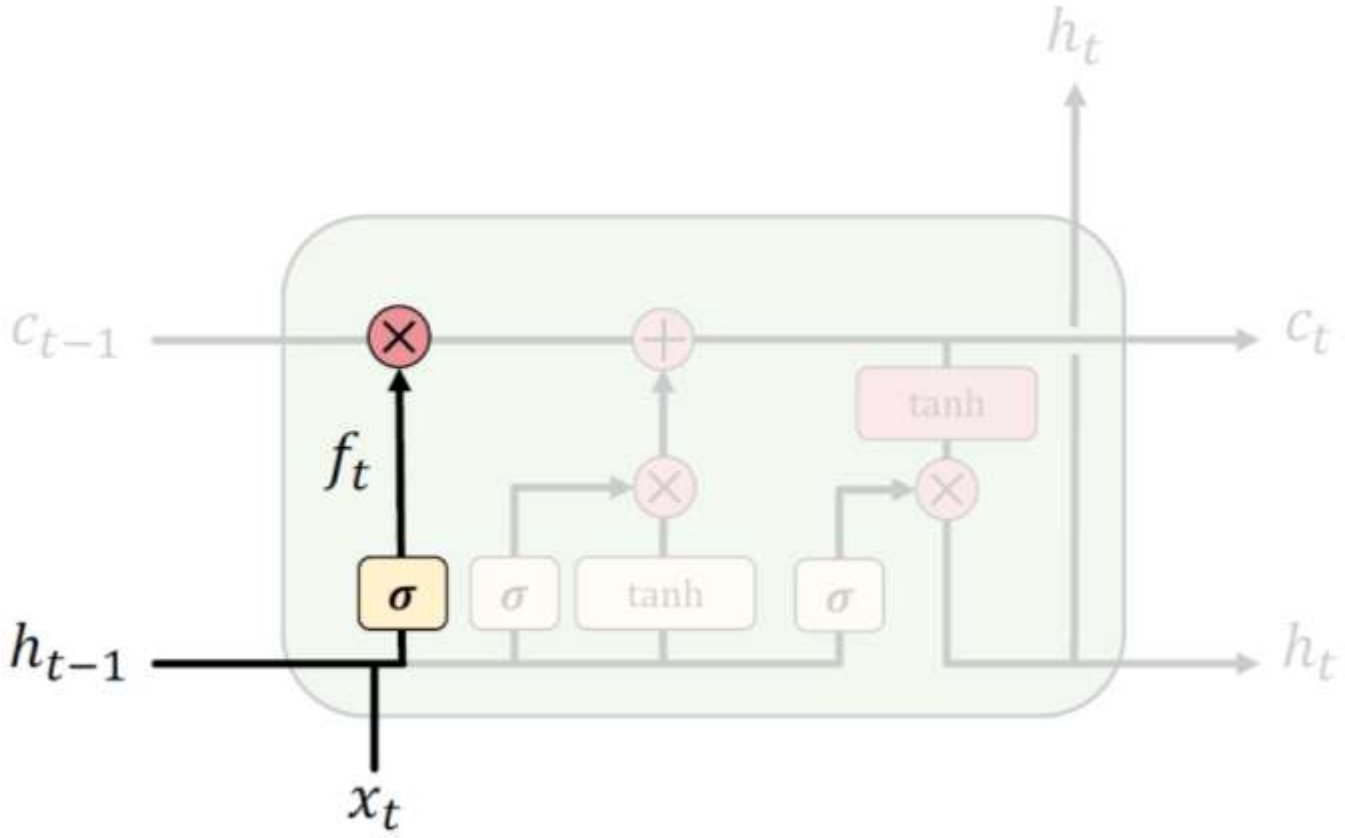
Long Short Term Memory (LSTMs)

Comment fonctionne un LSTM ?



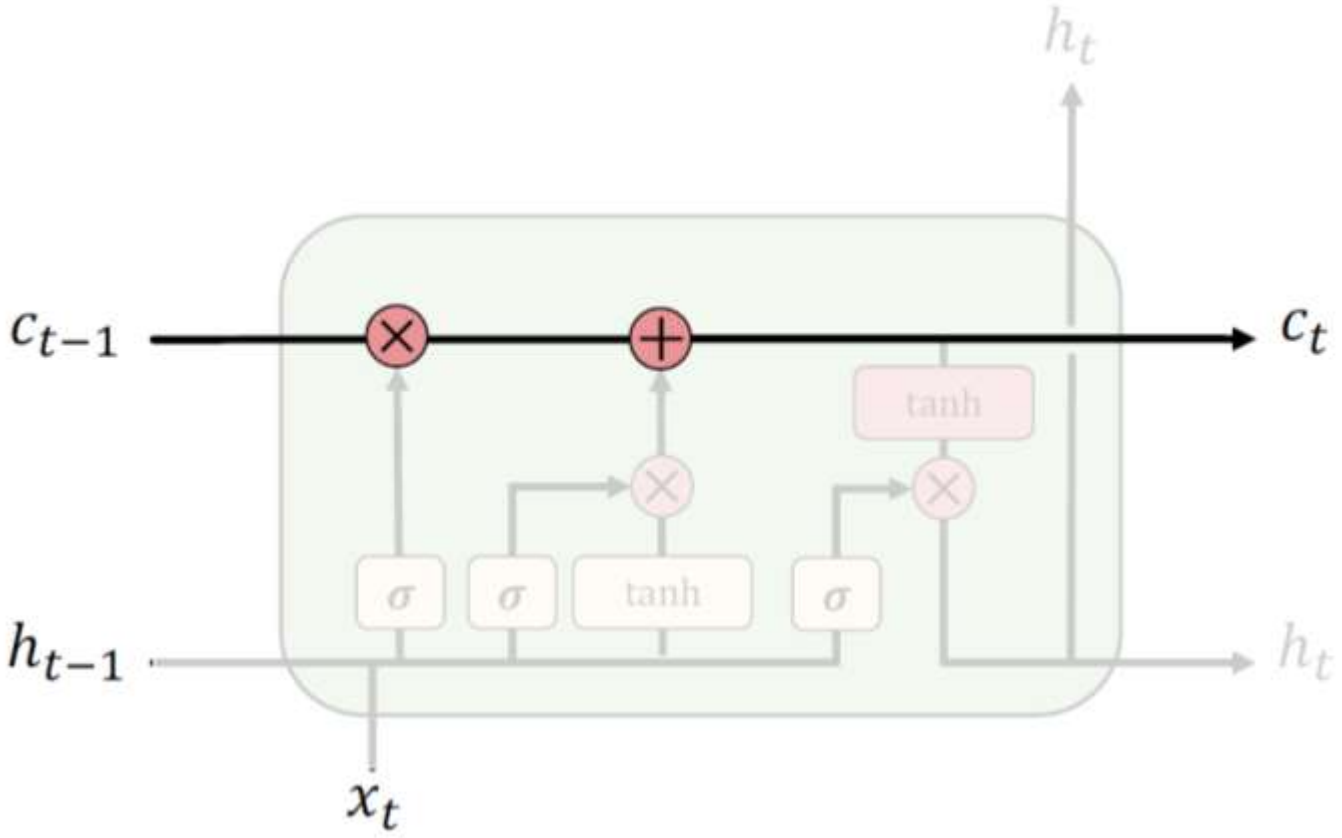
Long Short Term Memory (LSTMs)

Les LSTMs oublient les informations non pertinentes de l'état précédent



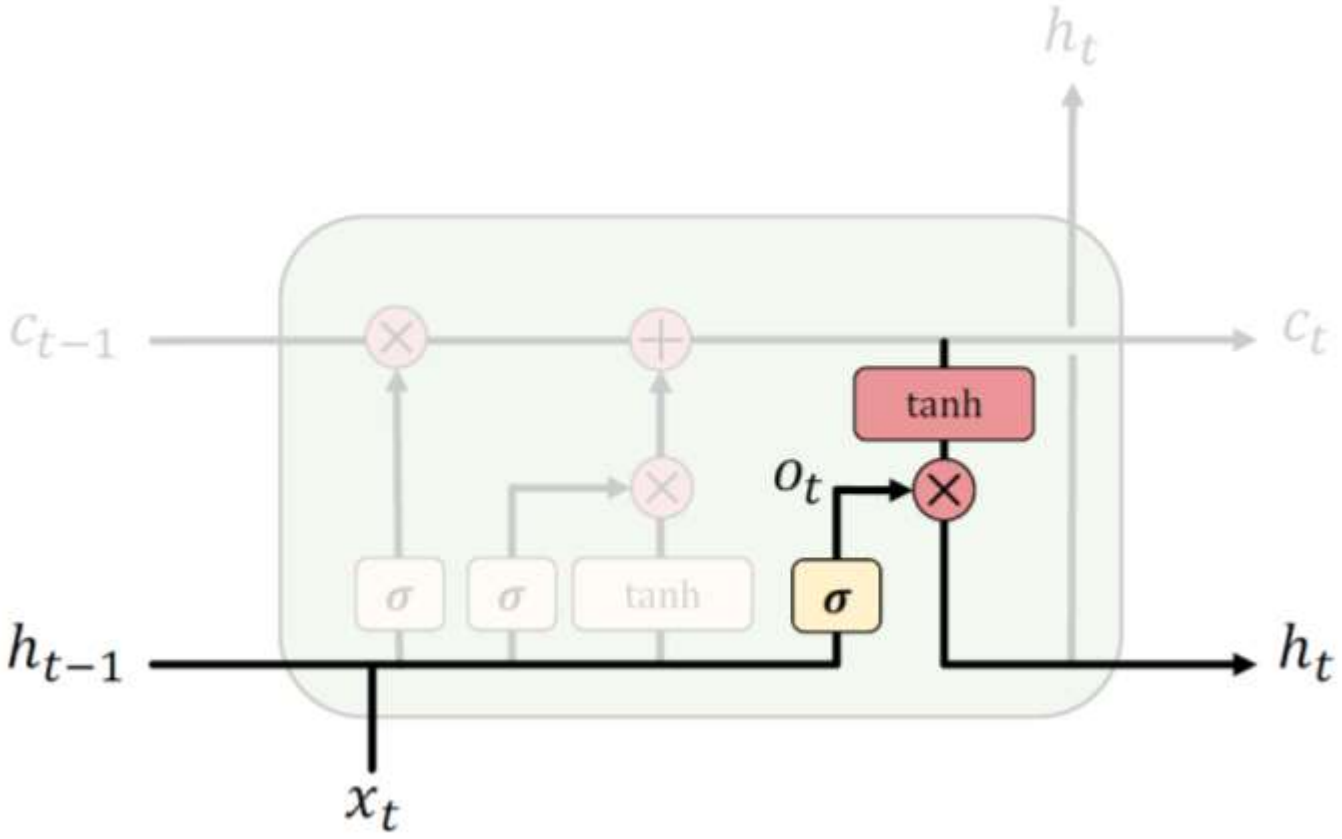
Long Short Term Memory (LSTMs)

Les LSTMs mettent à jour les valeurs du cell state



Long Short Term Memory (LSTMs)

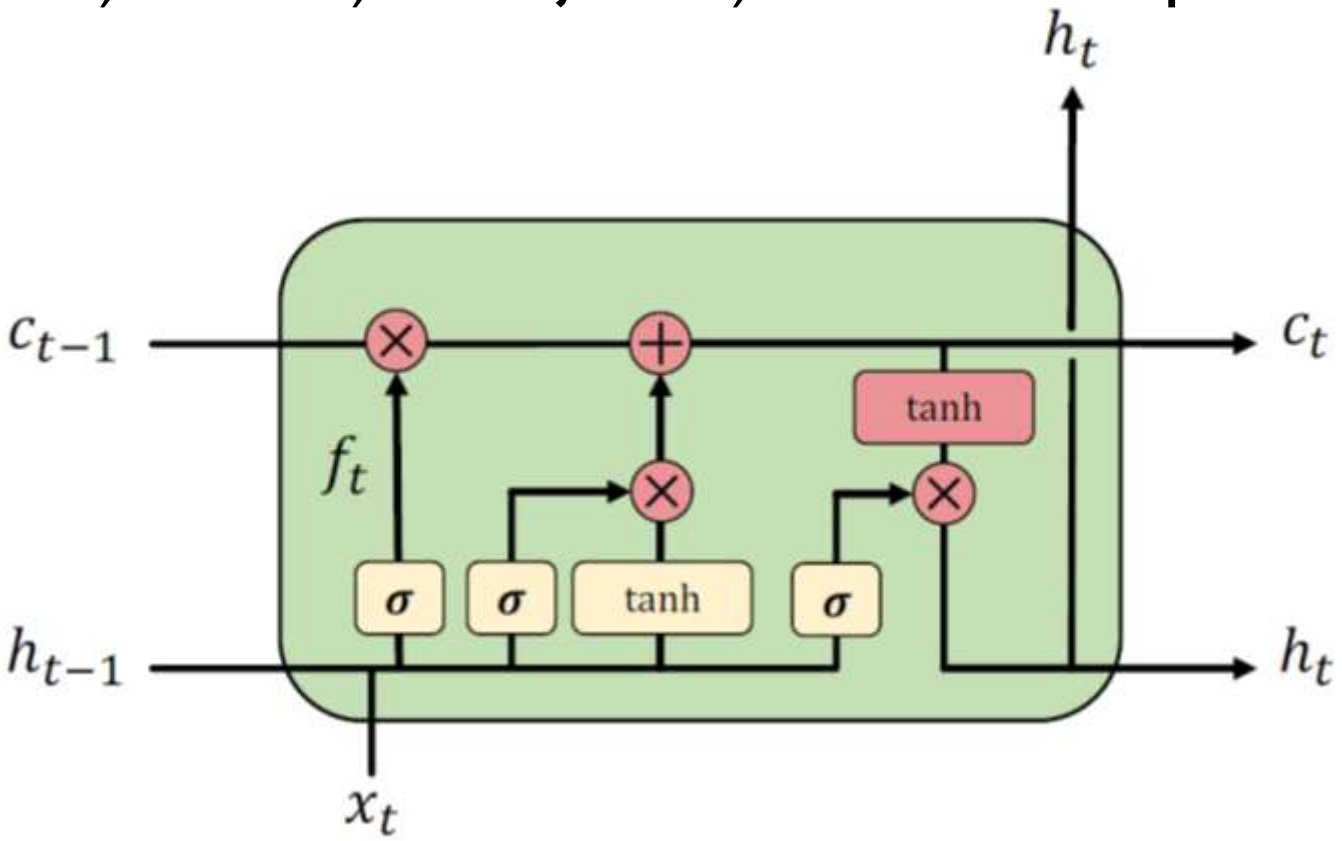
Les LSTMs utilisent un output gate pour générer un output final à partir de certaines parties du cell state



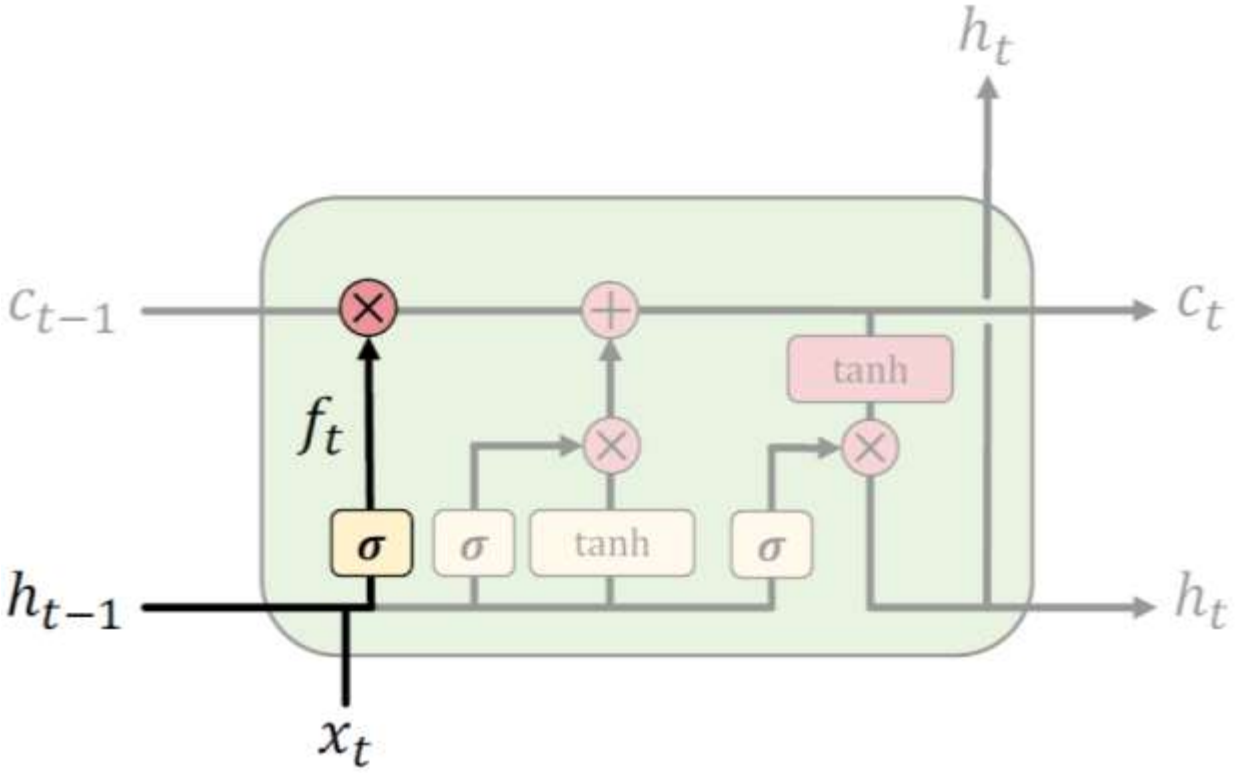
Long Short Term Memory (LSTMs)

Comment fonctionne un LSTM ?

1) Oubli 2) Met à jour 3) Génère un output



LSTMs: Oublier information non pertinente

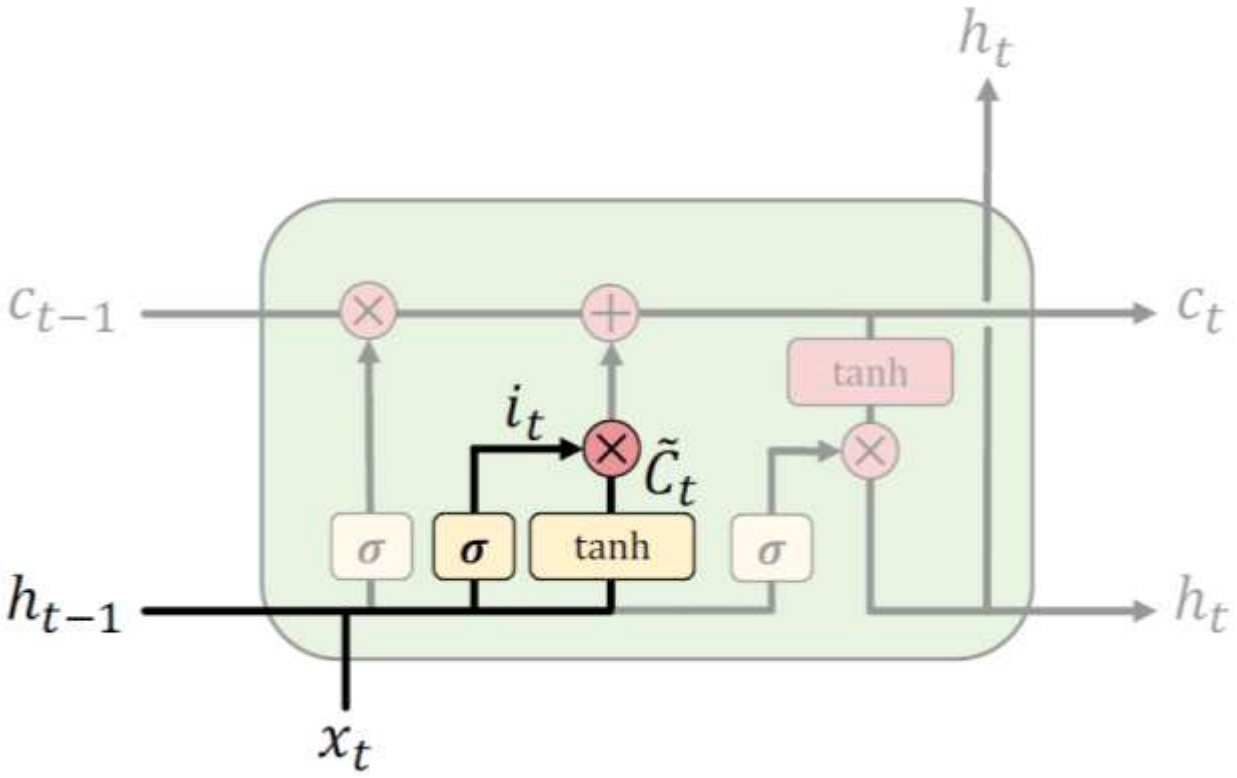


$$f_t = \sigma(\boldsymbol{W_f}[h_{t-1}, x_t] + b_f)$$

- Utilise l'output et l'input précédents
- Couche Sigmoïde : valeur 0 et 1 – "oublie totalement" vs."conserve totalement"

Ex : Ne plus prendre en compte le genre des anciens sujets dans une phrase.

LSTMs: Identifier nouvelle info à conserver



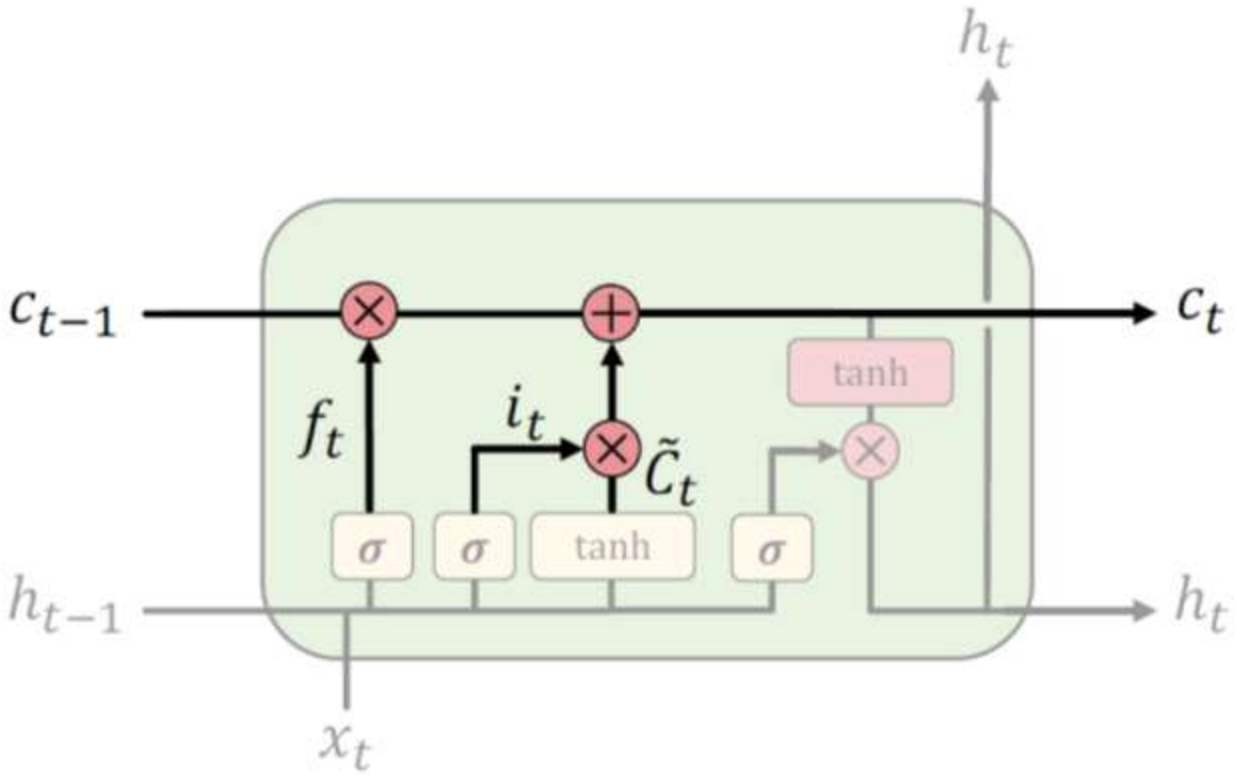
$$i_t = \sigma(\boldsymbol{W_i}[h_{t-1}, x_t] + b_i)$$

$$\widetilde{C}_t = \tanh(\boldsymbol{W_c}[h_{t-1}, x_t] + b_c)$$

- Couche Sigmoïde :choisir quelles valeurs mettre à jour
- Couche Tanh : génère un nouveau vecteur avec les valeurs potentielles qui pourraient être ajoutées au cell state

Ex : Ajouter le genre d'un nouveau sujet qui va remplacer celui de l'ancien sujet

LSTMs: Mettre à jour le cell state

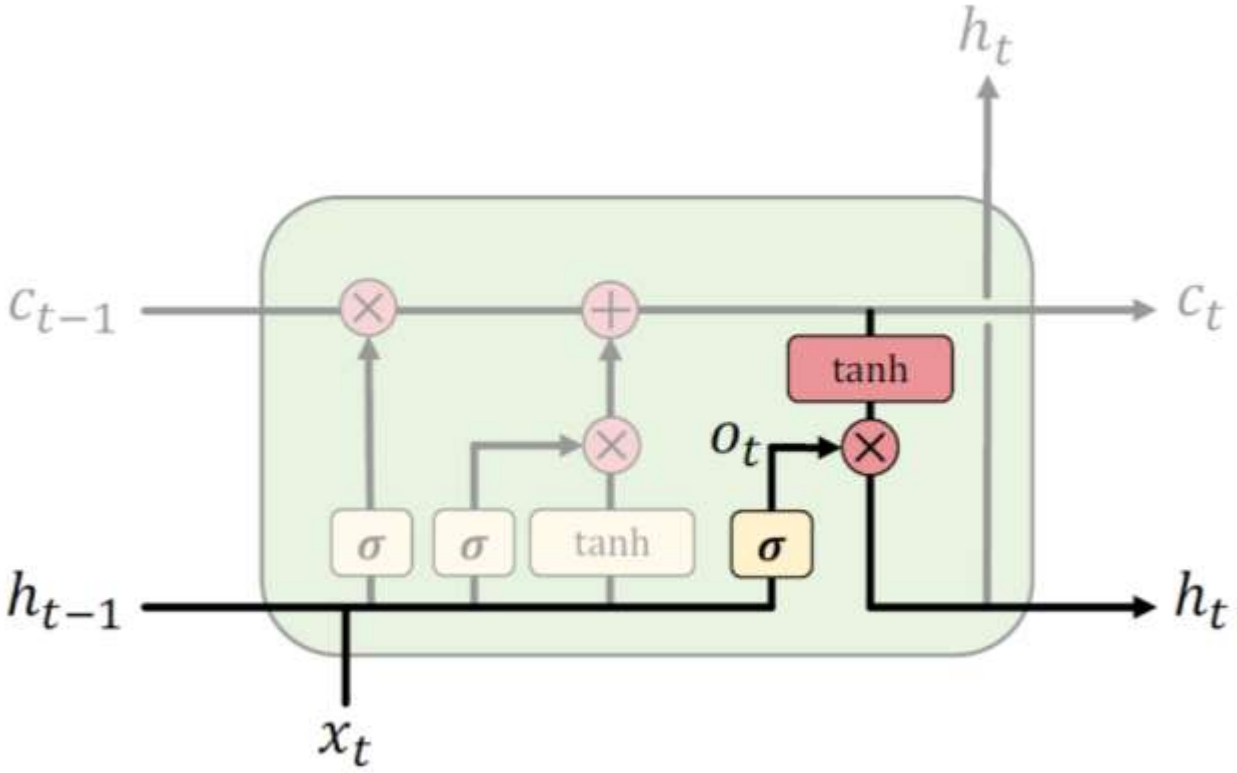


$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t$$

- On applique l'opération d'oubli à le cell state précédente : $f_t * C_{t-1}$
- On ajoute les nouvelles valeurs potentielles, scalées par i_t : $i_t * \widetilde{C}_t$

Ex : Oublier les anciennes infos et ajouter les nouvelles infos du genre

LSTMs: Output filtré



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

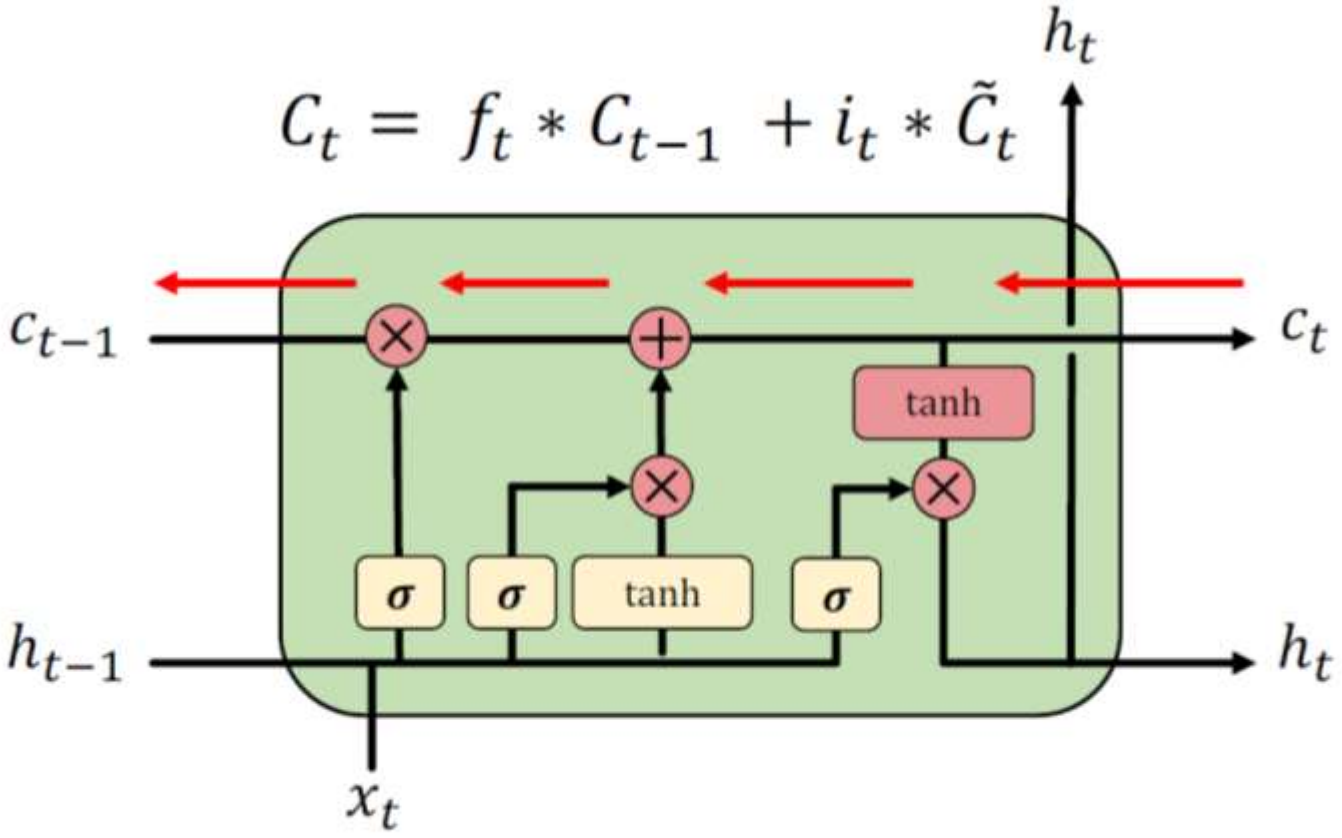
$$h_t = o_t * \tanh(C_t)$$

- Couche Sigmoïde : choisir à partir de quelles parties de l'état générer l'output
- Couche Tanh : Compresse les valeurs entre -1 et 1
- $o_t * \tanh(C_t)$: génère un output filtré de la cell state

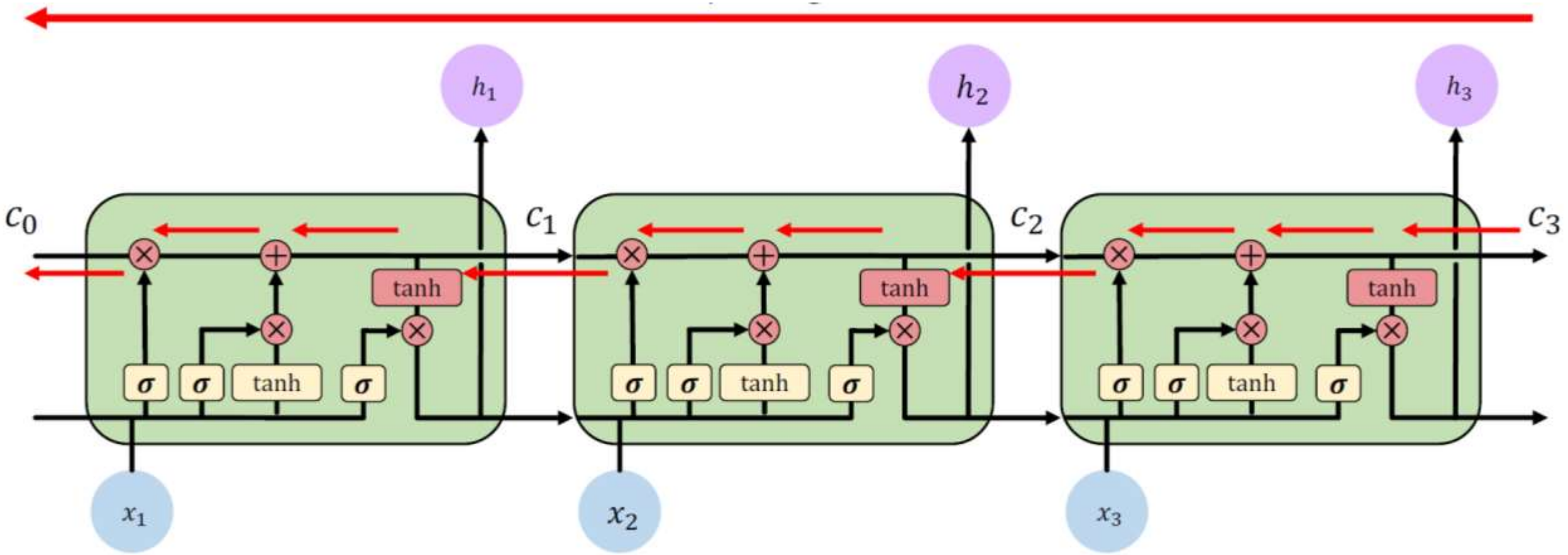
Ex : Génère une information relative à un verbe .

LSTM gradient flow

La Backpropagation s'effectuant de c_t à c_{t-1} ne nécessite plus qu'une multiplication !
Aucune multiplication matricielle : plus de problème de vanishing gradient.



LSTM gradient flow



LSTMs : ce qu'il faut retenir

1. On conserve tout au long du processus un cell state
2. On utilise des gates pour filtrer l'information
   - Les gates d'oubli éliminent les informations non pertinentes
   - On met à jour le cell state
   - L'output gate génère un version filtrée du cell state
3. La Backpropagation de c_t à c_{t-1} ne nécessite plus de multiplication matricielle : il n'y a plus d'interruption dans le calcul du gradient

Applications

Recurrent neural networks (RNNs)

1. Les RNN sont adaptés pour traiter des séquences
2. Relation de récurrence
3. La Backpropagation s'effectue à travers le temps
4. Les LSTMs utilisent des gated cells pour modéliser des dépendances à long terme
5. Sentiment qualification, prédiction cours boursiers, traducteurs

References:
http://introtodeeplearning.com/