Réseaux de neurones récurrents

Guillaume Bourmaud

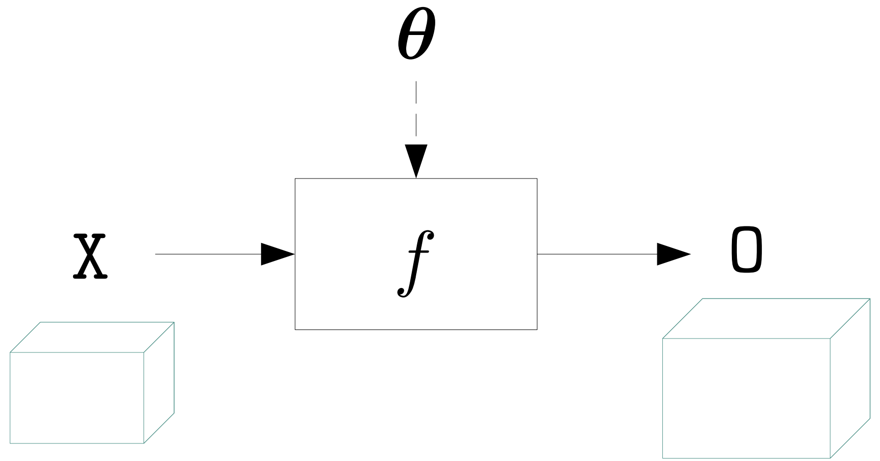
I) Introduction

I)

Rappels : Réseaux de neurones « à propagation avant » pour l’apprentissage supervisé

En anglais « Feed-forward Neural Network » (FNN)

Inférence



$$\mathtt{O} = f\left(\mathtt{X}; \boldsymbol{\theta}\right)$$

Apprentissage

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} l\left(\mathtt{Y}^{(i)}, f\left(\mathtt{X}^{(i)}; \boldsymbol{\theta}\right)\right)$$

Réseaux de neurones récurrents

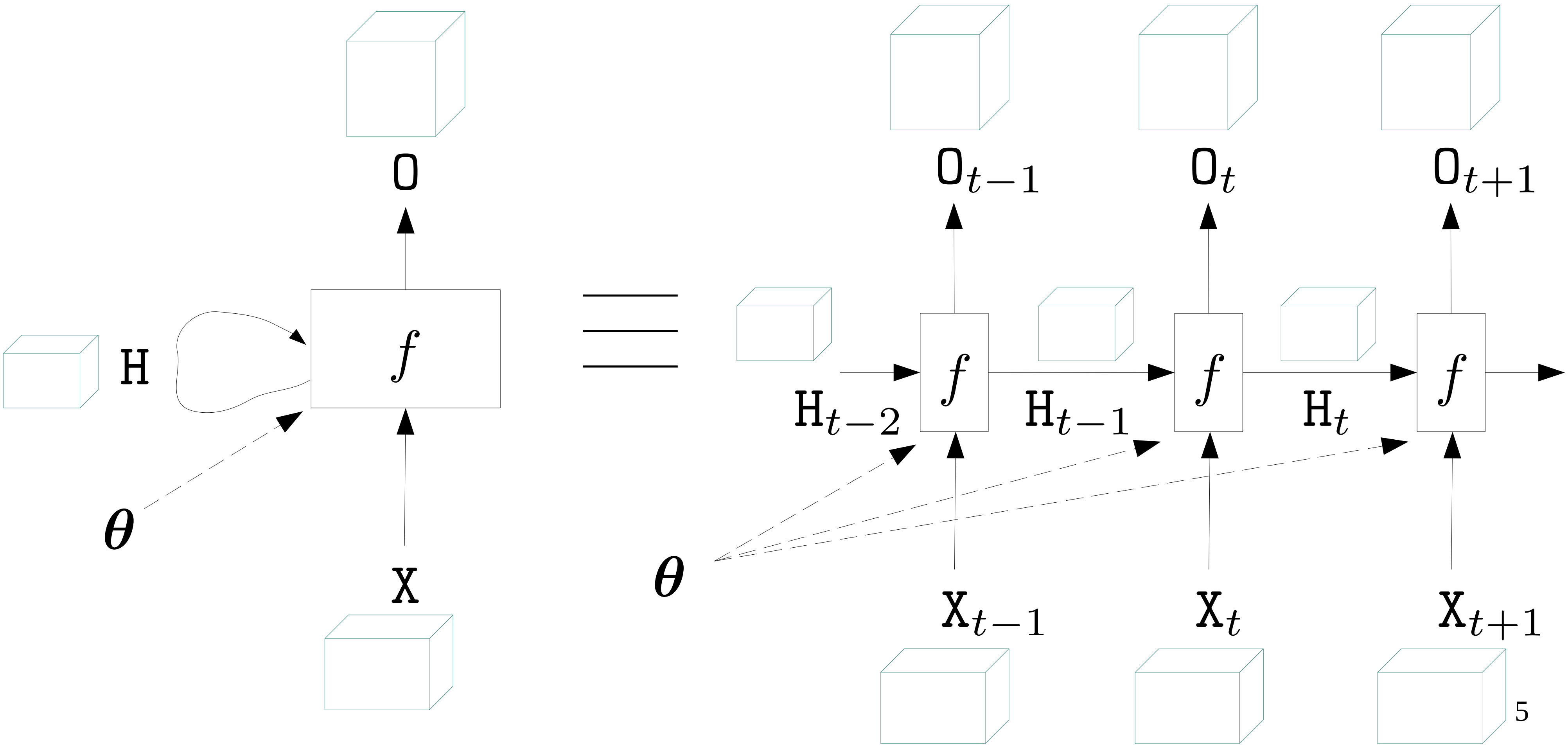
- En anglais « Recurrent Neural Network » (RNN)

- Permettent historiquement de gérer facilement les cas où l'entrée et/ou la sortie du réseau ont une taille variable.

```
"the cat sat on the mat" -> [Seq2Seq model] -> "le chat etait assis sur le tapis"
```

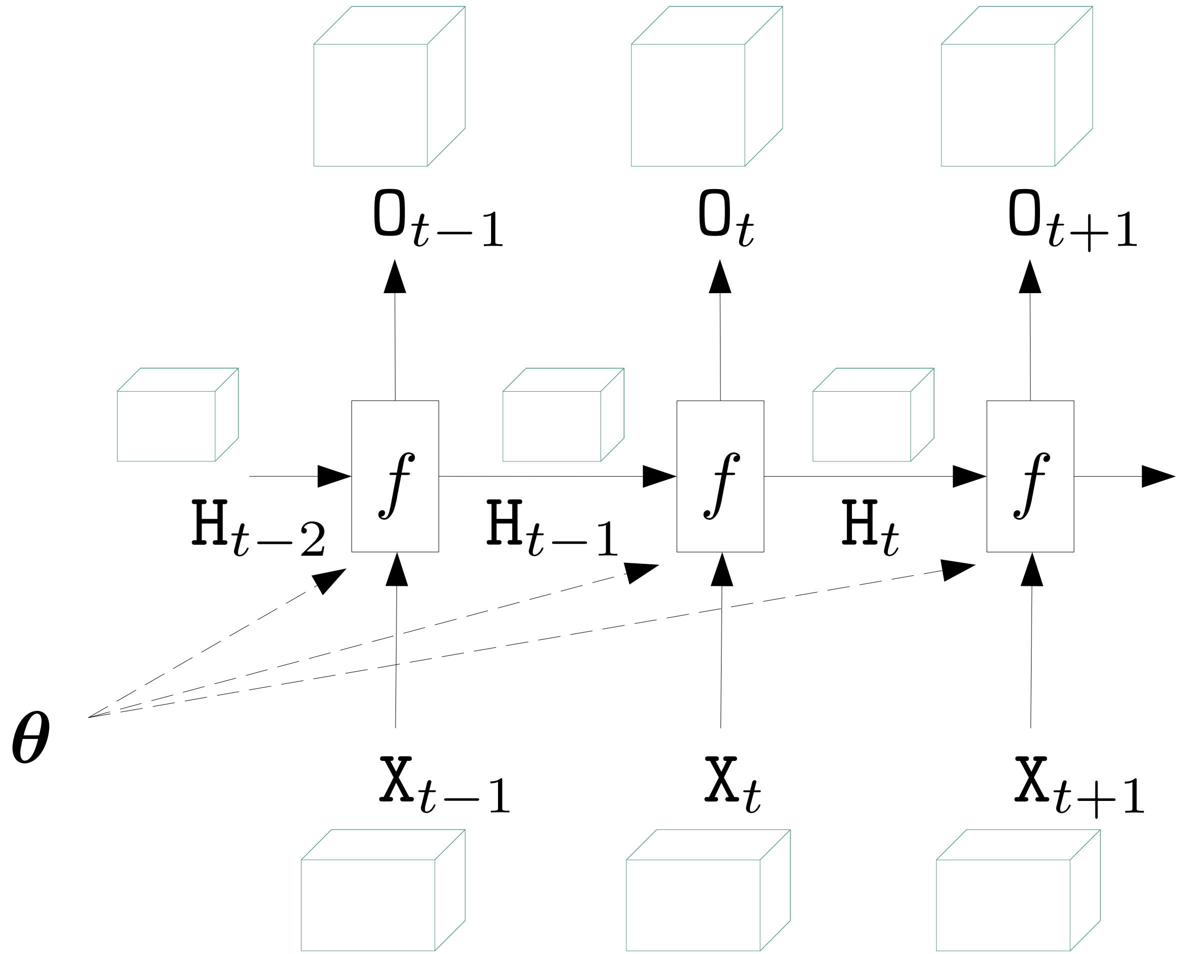
I)

Définition et dépliement d'un réseau de neurones récurrents



I)

Dépliement d’un réseau de neurones récurrents (suite)



Mêmes fonction et paramètres à chaque pas de temps

$$(O_t, H_t) = f\,(H_{t-1}, X_t; \boldsymbol{\theta})$$

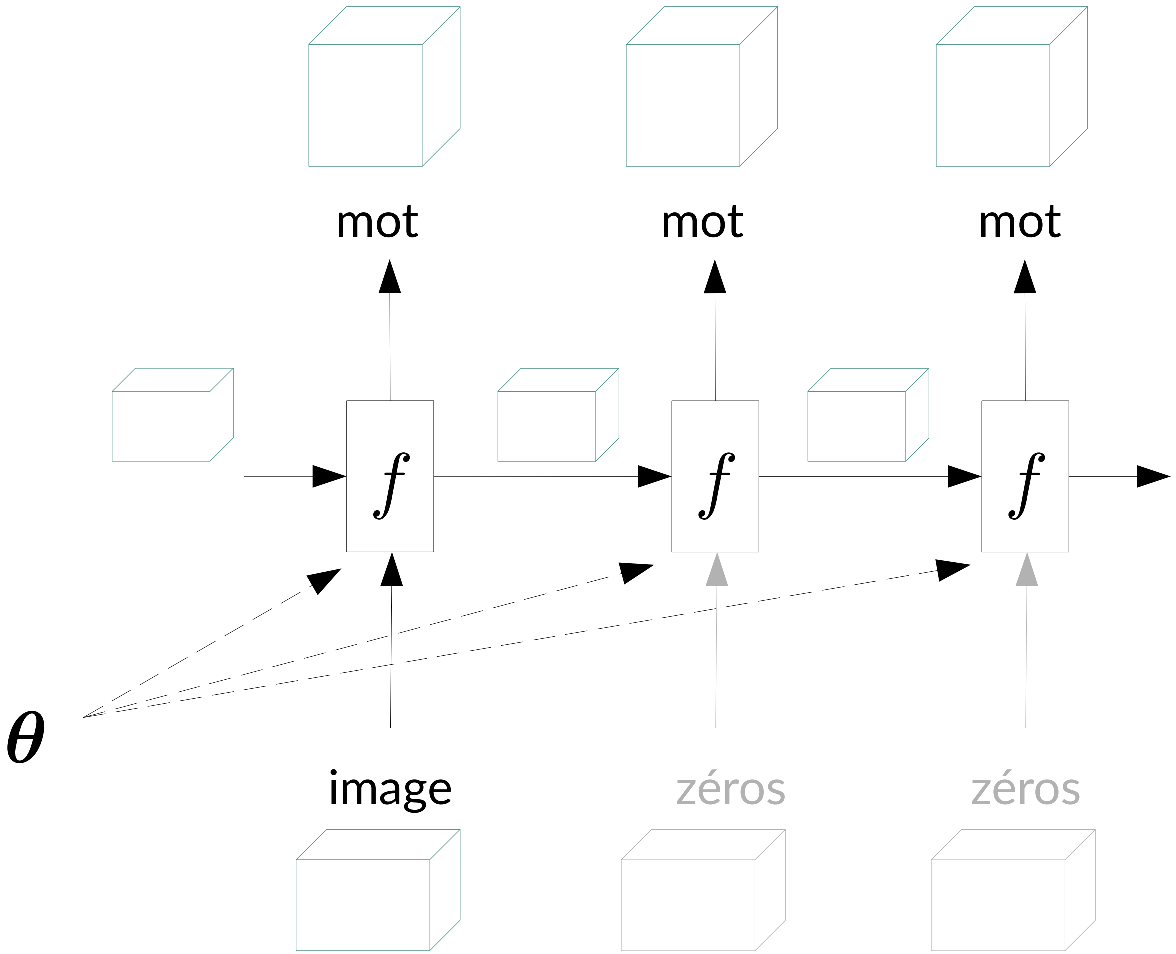
sortie

état caché (joue le rôle de mémoire)

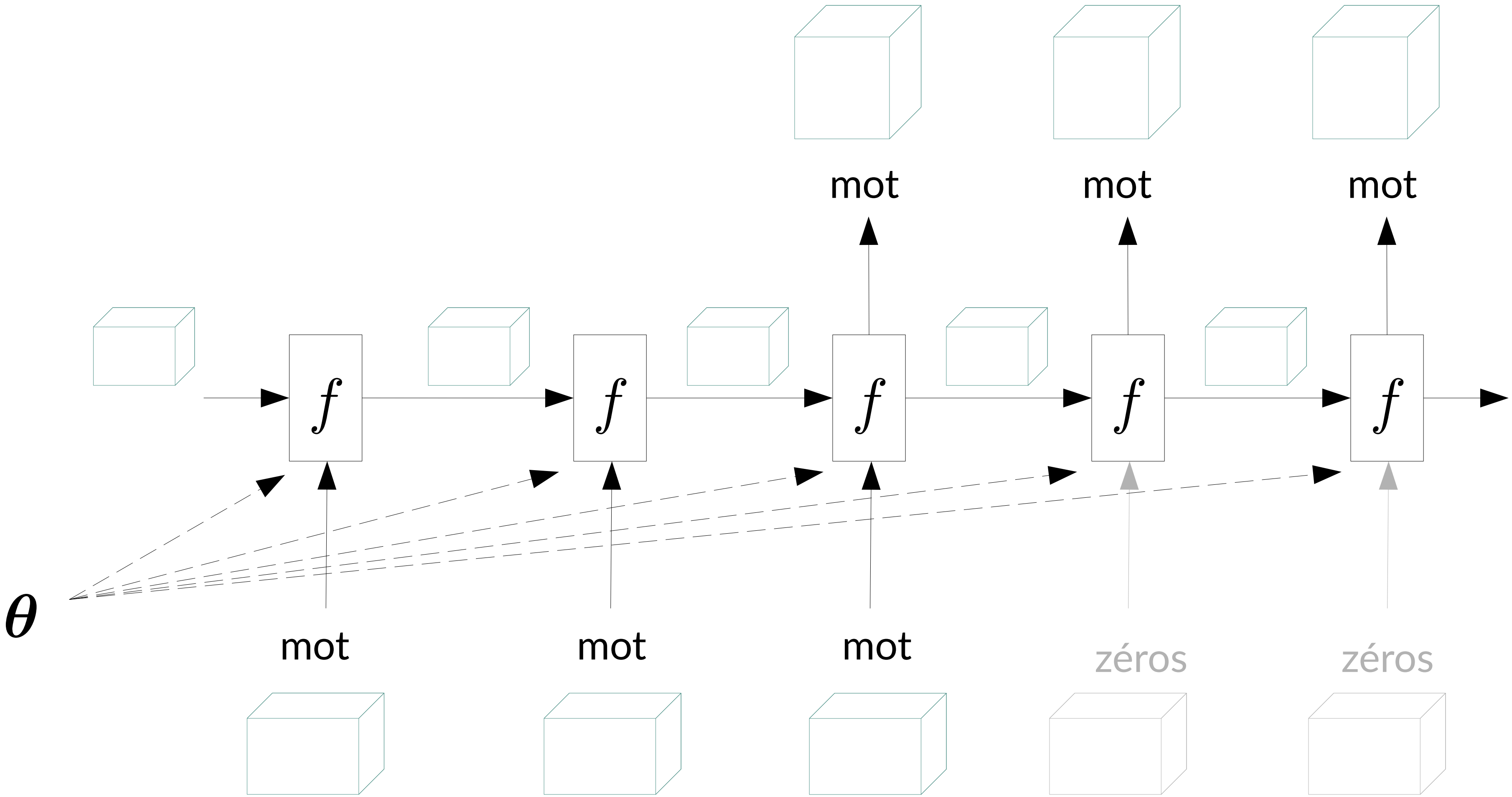
entrée

Longue dépendance : La sortie à l’instant t dépend de **toutes** les entrées passées.

Exemple d'utilisation d'un RNN : description d'une image



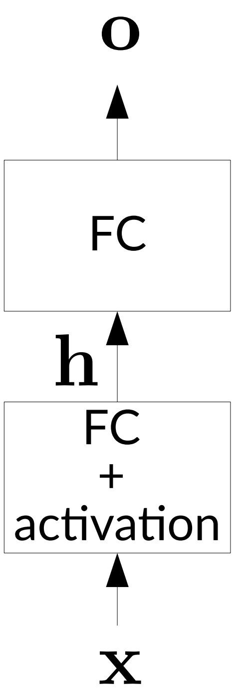
Exemple d’utilisation d’un RNN : traduction de texte

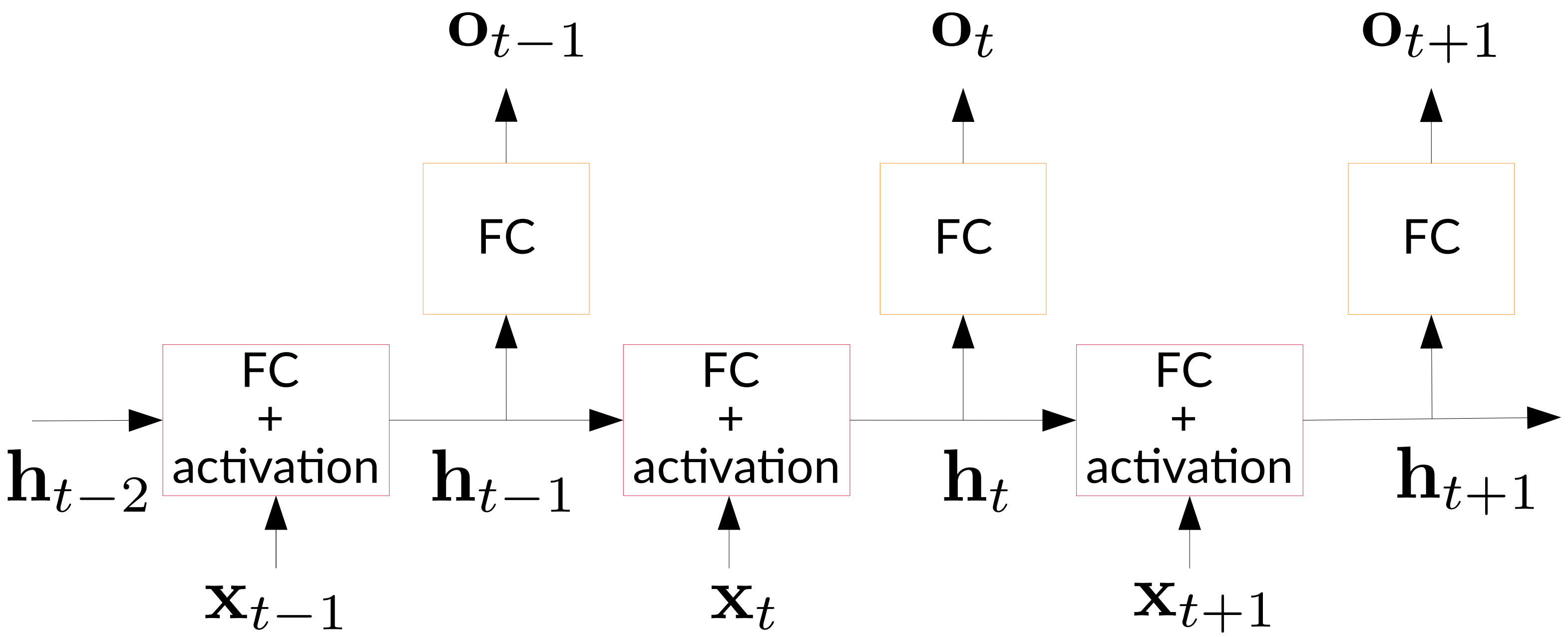


II) Du MLP à une couche cachée au RNN « standard »

II)

Du MLP à une couche cachée au RNN « standard »



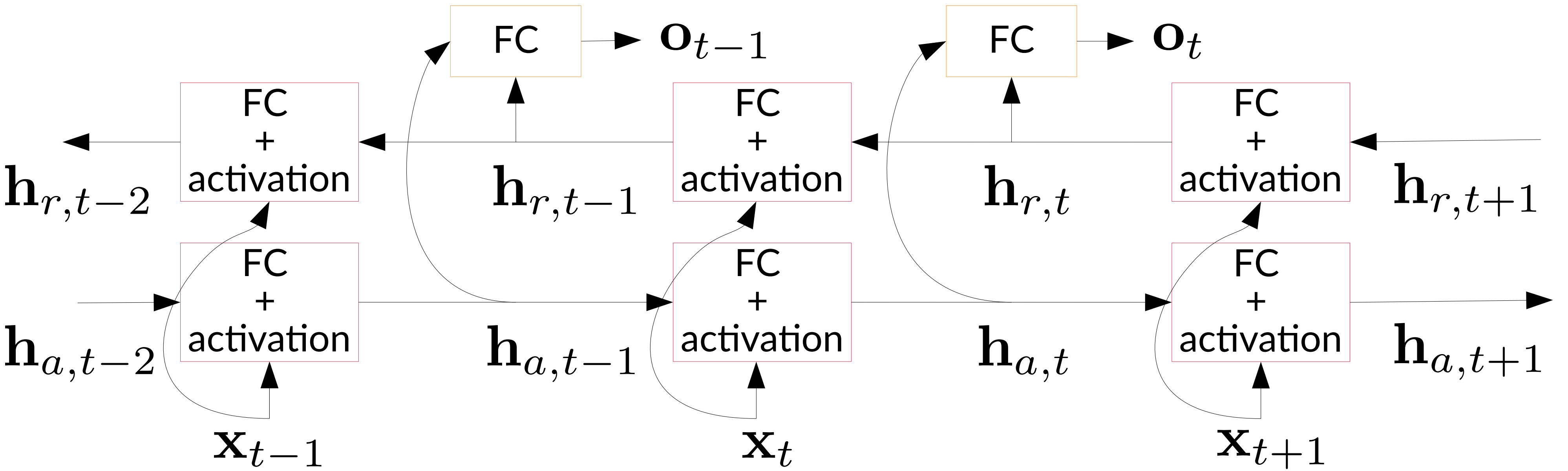


$$\mathbf{h} = \sigma_h\left(\mathtt{W}_h\mathbf{x} + \mathbf{b}_h\right)$$
$$\mathbf{o} = \mathtt{W}_o\mathbf{h} + \mathbf{b}_o$$

$$\mathbf{h}_t = \sigma_h\left(\mathtt{W}_h\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_h\right) := f_h\left(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}_h\right)$$
$$\mathbf{o}_t = \mathtt{W}_o\mathbf{h}_t + \mathbf{b}_o := f_o\left(\mathbf{h}_t; \boldsymbol{\theta}_o\right)$$

II)

RNN « standard » bidirectionnel



$$\mathbf{h}_{a,t} = \sigma_h\left(\mathtt{W}_{a,h}\begin{bmatrix}\mathbf{h}_{a,t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_{a,h}\right) \qquad \mathbf{h}_{r,t} = \sigma_h\left(\mathtt{W}_{r,h}\begin{bmatrix}\mathbf{h}_{r,t+1}\\ \mathbf{x}_{t+1}\end{bmatrix} + \mathbf{b}_{r,h}\right)$$

$$\mathbf{o}_t = \mathtt{W}_o\begin{bmatrix}\mathbf{h}_{a,t}\\ \mathbf{h}_{r,t}\end{bmatrix} + \mathbf{b}_o$$

Exemple jouet : génération du mot « bille » avec un RNN

Vocabulaire de 4 lettres : i = $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ b = $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ e = $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ l = $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$
(« one hot encoding »)

Paramètres fixés : $\boldsymbol{\theta}_h = (\mathtt{W}_h, \mathbf{b}_h)$ $\boldsymbol{\theta}_o = (\mathtt{W}_o, \mathbf{b}_o)$

Dimension de l'espace caché : 3

$$\mathbf{h}_0 = \begin{bmatrix}0\\0\\0\end{bmatrix} \qquad \mathbf{x}_1 = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

1) $\mathbf{h}_t = \sigma_h\left(\mathtt{W}_h \begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_h\right)$

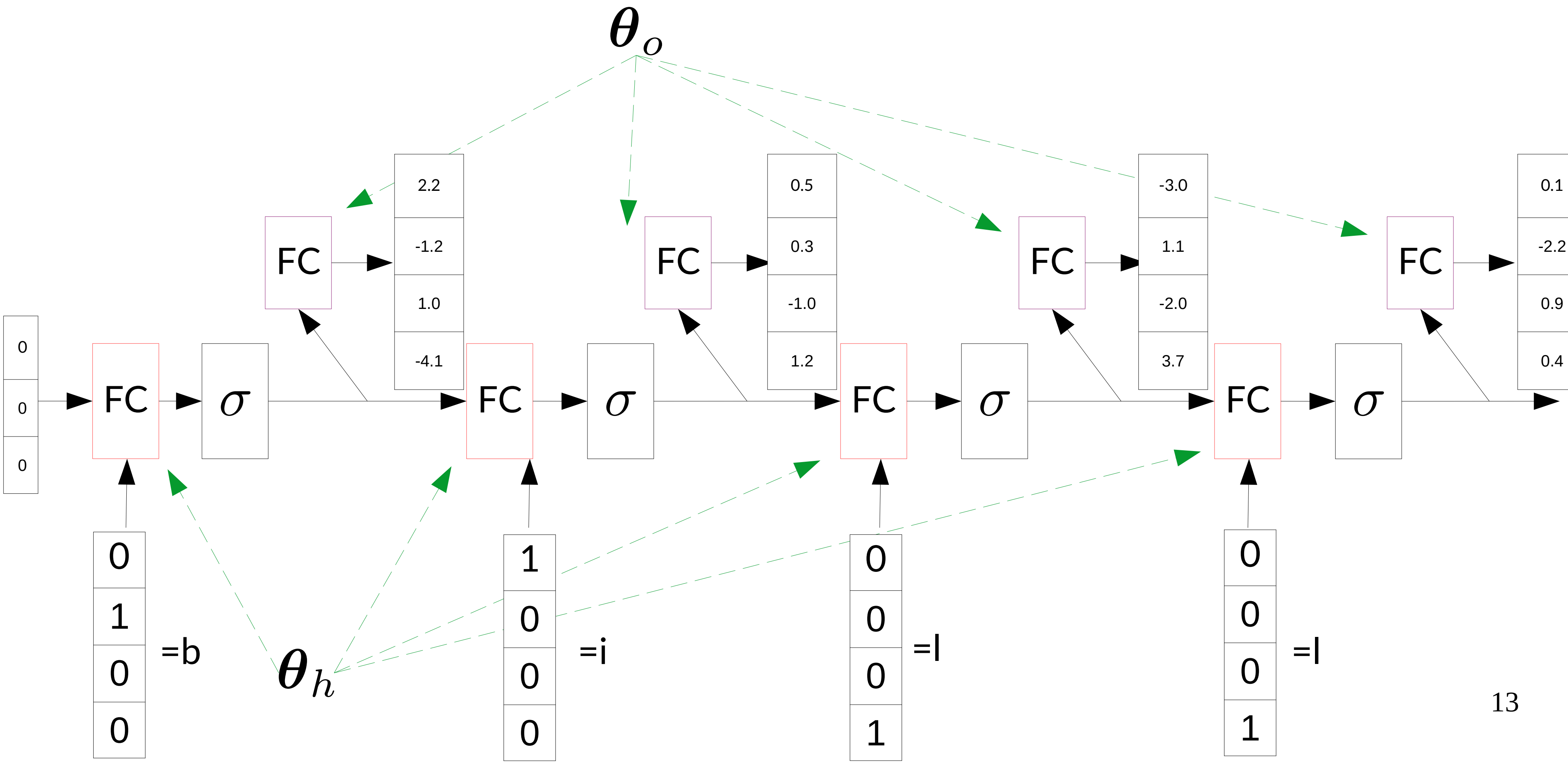
$\mathbf{o}_t = \mathtt{W}_o \mathbf{h}_t + \mathbf{b}_o$

2) $t = t+1,\ \mathbf{x}_t = \arg\max \mathbf{o}_{t-1}$

3) Aller à 1)

II)

Exemple jouet : génération du mot « bille » avec un RNN (suite)

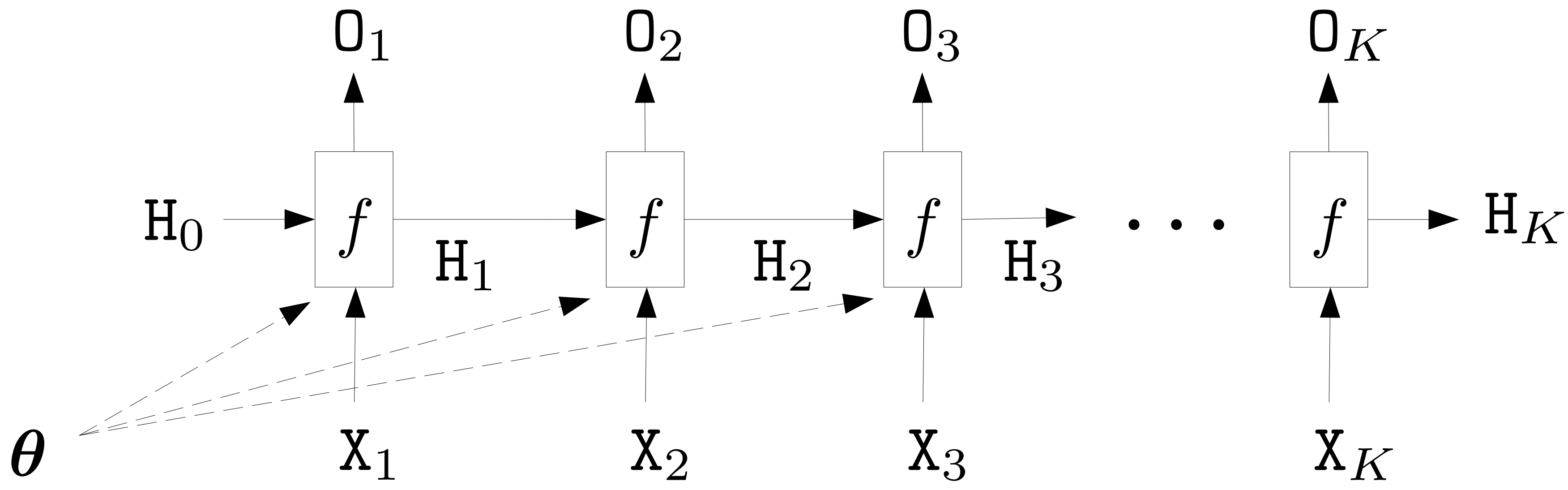


III) Optimisation des paramètres d’un RNN

III)

Optimisation des paramètres d'un RNN

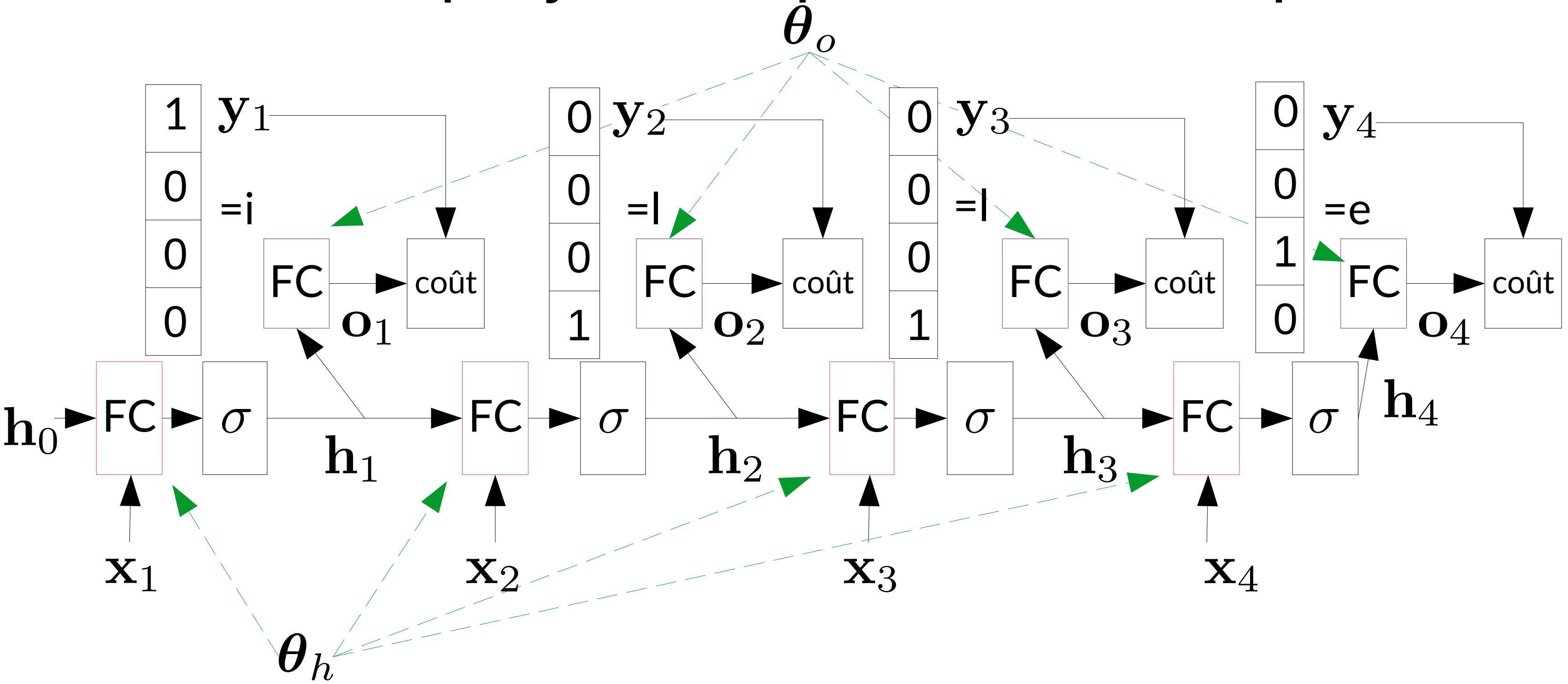
RNN déplié sur K pas de temps = FNN



Optimisation des paramètres du RNN = Descente de gradient sur le RNN déplié

En anglais « BackPropagation Through Time » (BPTT)

Retour à l'exemple jouet : optimisation des paramètres



$$
\begin{aligned}
\boldsymbol{\theta}_o^*, \boldsymbol{\theta}_h^* = \arg\min_{\boldsymbol{\theta}_o, \boldsymbol{\theta}_h} \quad & l(\mathbf{y}_1, f_o(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o)) \\
& + l(\mathbf{y}_2, f_o(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o)) \\
& + l(\mathbf{y}_3, f_o(f_h(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h), \mathbf{x}_3; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o)) \\
& + l(\mathbf{y}_4, f_o(f_h(f_h(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h), \mathbf{x}_3; \boldsymbol{\theta}_h), \mathbf{x}_4; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o))
\end{aligned}
$$

III)

Retour à l'exemple jouet : optimisation des paramètres (suite)

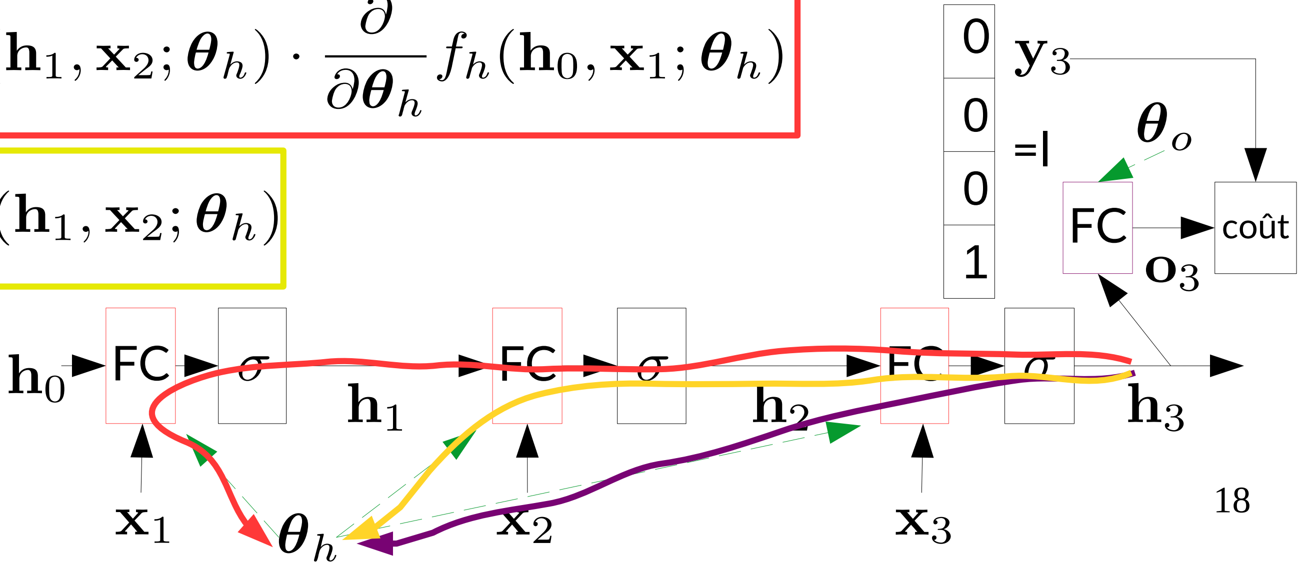
$$\boldsymbol{\theta}_o^*, \boldsymbol{\theta}_h^* = \arg\min_{\boldsymbol{\theta}_o, \boldsymbol{\theta}_h} \begin{aligned}[t] & l(\mathbf{y}_1, f_o(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o)) \\ +& l(\mathbf{y}_2, f_o(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o)) \\ +& l(\mathbf{y}_3, f_o(\boxed{f_h(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h), \mathbf{x}_3; \boldsymbol{\theta}_h)}; \boldsymbol{\theta}_o)) \\ +& l(\mathbf{y}_4, f_o(f_h(f_h(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h), \mathbf{x}_3; \boldsymbol{\theta}_h), \mathbf{x}_4; \boldsymbol{\theta}_h); \boldsymbol{\theta}_o)) \end{aligned}$$

$$\frac{\partial}{\partial \boldsymbol{\theta}_h} f_h(f_h(f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h), \mathbf{x}_2; \boldsymbol{\theta}_h), \mathbf{x}_3; \boldsymbol{\theta}_h) =$$

$$\boxed{\frac{\partial}{\partial \mathbf{h}_2} f_h(\mathbf{h}_2, \mathbf{x}_3; \boldsymbol{\theta}_h) \cdot \frac{\partial}{\partial \mathbf{h}_1} f_h(\mathbf{h}_1, \mathbf{x}_2; \boldsymbol{\theta}_h) \cdot \frac{\partial}{\partial \boldsymbol{\theta}_h} f_h(\mathbf{h}_0, \mathbf{x}_1; \boldsymbol{\theta}_h)}$$

$$+ \boxed{\frac{\partial}{\partial \mathbf{h}_2} f_h(\mathbf{h}_2, \mathbf{x}_3; \boldsymbol{\theta}_h) \frac{\partial}{\partial \boldsymbol{\theta}_h} f_h(\mathbf{h}_1, \mathbf{x}_2; \boldsymbol{\theta}_h)}$$

$$+ \boxed{\frac{\partial}{\partial \boldsymbol{\theta}_h} f_h(\mathbf{h}_2, \mathbf{x}_3; \boldsymbol{\theta}_h)}$$

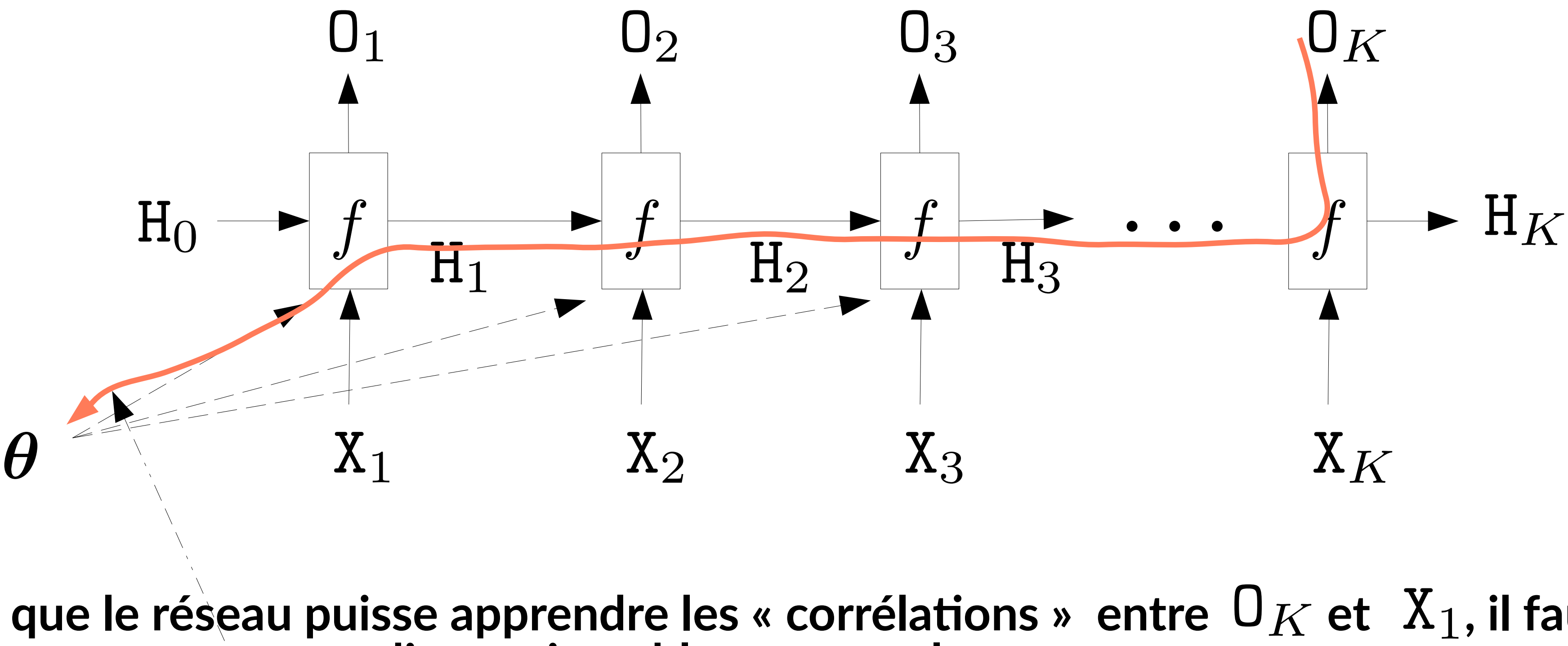


IV) Le problème de l’apprentissage de longues dépendances

IV)

Le problème de l'apprentissage de longues dépendances

Longue dépendance = la sortie à un instant dépend de toutes les entrées précédentes

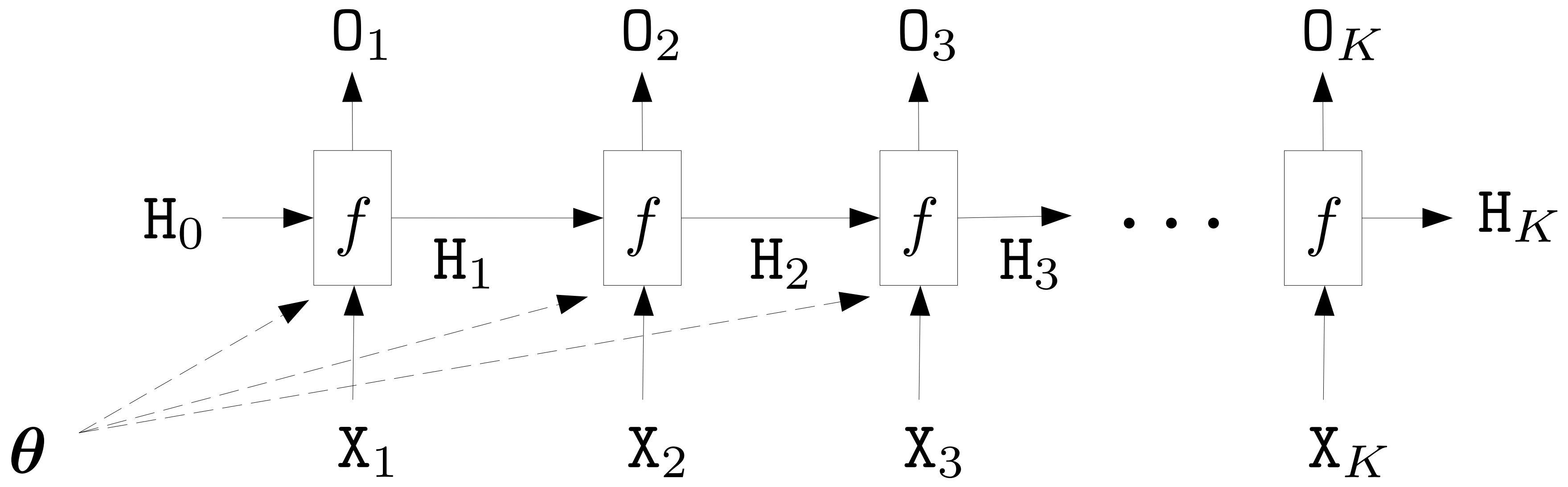


Pour que le réseau puisse apprendre les « corrélations » entre O_K et X_1, il faut notamment que ce gradient soit stable et non nul.

IV)

Le problème de l'apprentissage de longues dépendances (suite)

RNN déplié sur K pas de temps = FNN à K couches



K grand = FNN profond → risque d'évanouissement ou d'explosion du gradient

Si explosion → aucun apprentissage
Si évanouissement → apprentissage uniquement de courtes dépendances

Évanouissement/explosion du gradient dans un RNN

RNN « standard » : fonction d'activation = tanh

Problème : Gradient de tanh entre 0 et 1

→ multiplication du gradient par un facteur positif inférieur à 1 lors de la rétropropagation

→ affaiblissement systématique du gradient à chaque pas de temps

Solution 1 : Remplacer tanh par ReLu

Lors de la rétroprogation, le gradient est multiplié exactement soit par 0 soit par 1

Solution 2 : Changement d'architecture → LSTM / GRU

Évanouissement/explosion du gradient dans un RNN (suite)

En supposant que la fonction d'activation est l'identité, on a :

$$\mathbf{h}_t = \begin{bmatrix} \mathtt{W}_{hh} & \mathtt{W}_{hx} \end{bmatrix} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_h \qquad \frac{\partial}{\partial \mathbf{h}_{t-1}} \mathbf{h}_t = \mathtt{W}_{hh}$$

$$\frac{\partial}{\partial \mathbf{h}_{t-K}} \mathbf{h}_t = \mathtt{W}_{hh}^K$$

Matrice à la puissance K

→ évanouissement ou explosion du gradient

Solution 1 : Initialisation $\mathtt{W}_{hh} = \mathtt{I} \quad \mathbf{b}_h = \mathbf{0}$

Variante : Initialisation en « contrôlant » les valeurs propres (« Echo-State Network »)

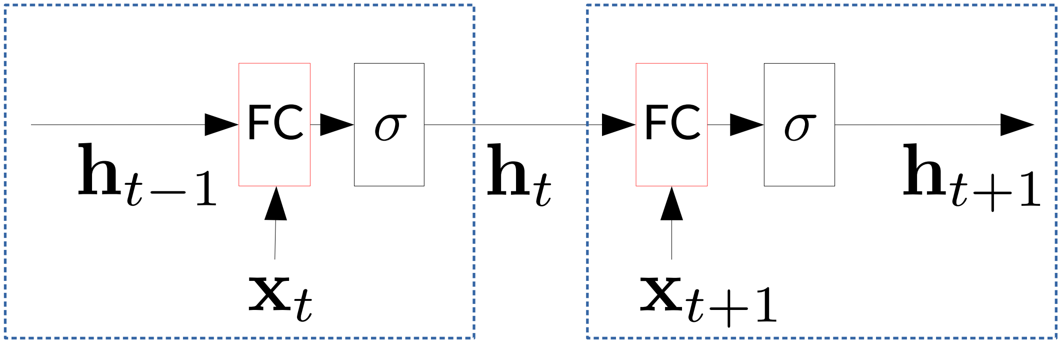
Solution 2 : Saturation du gradient (« Gradient Clipping)

Solution 3 : Changement d'architecture → LSTM / GRU

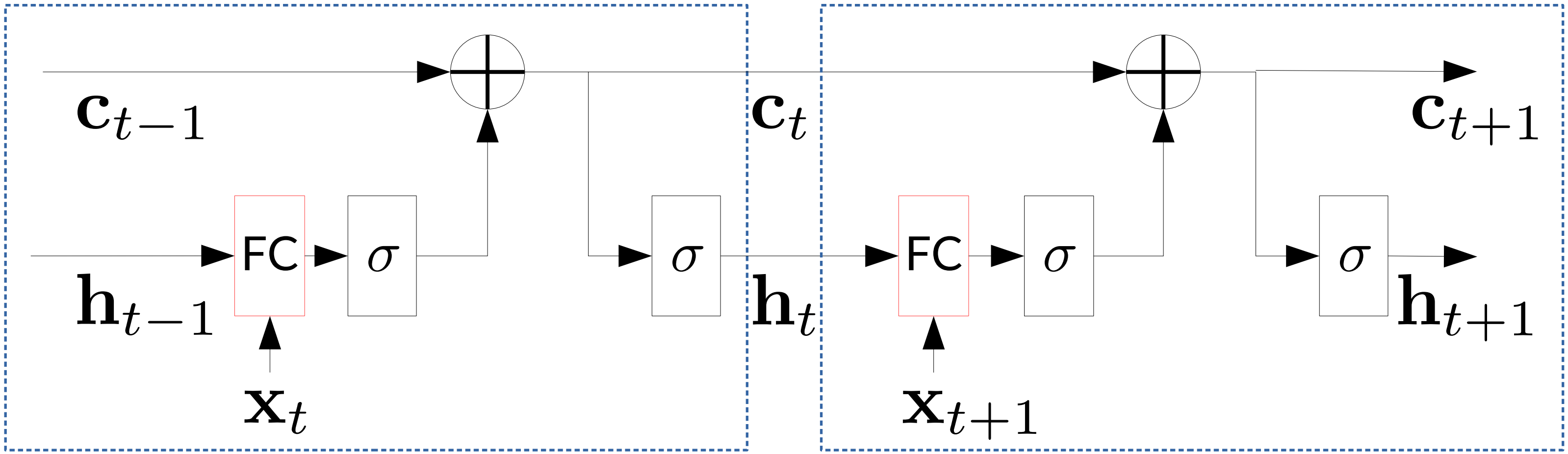
V) Architectures LSTM / GRU

V)

« Long-Short Term Memory » (LSTM) = Réseau à mémoire court-terme et à mémoire long-terme



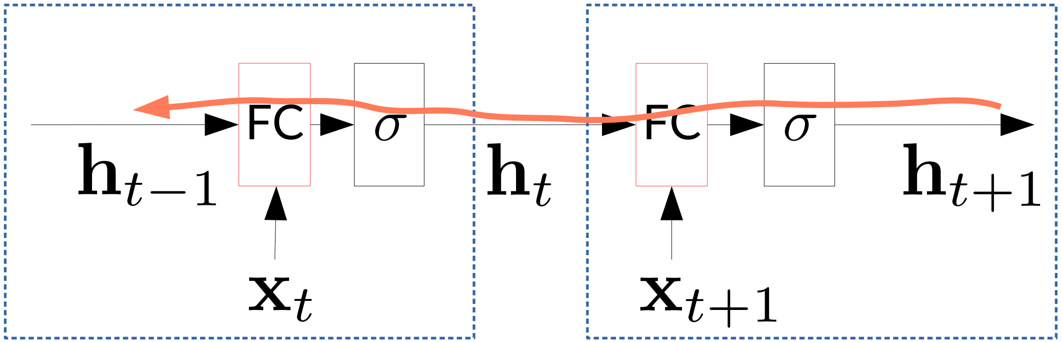
RNN « standard »



LSTM « simple »

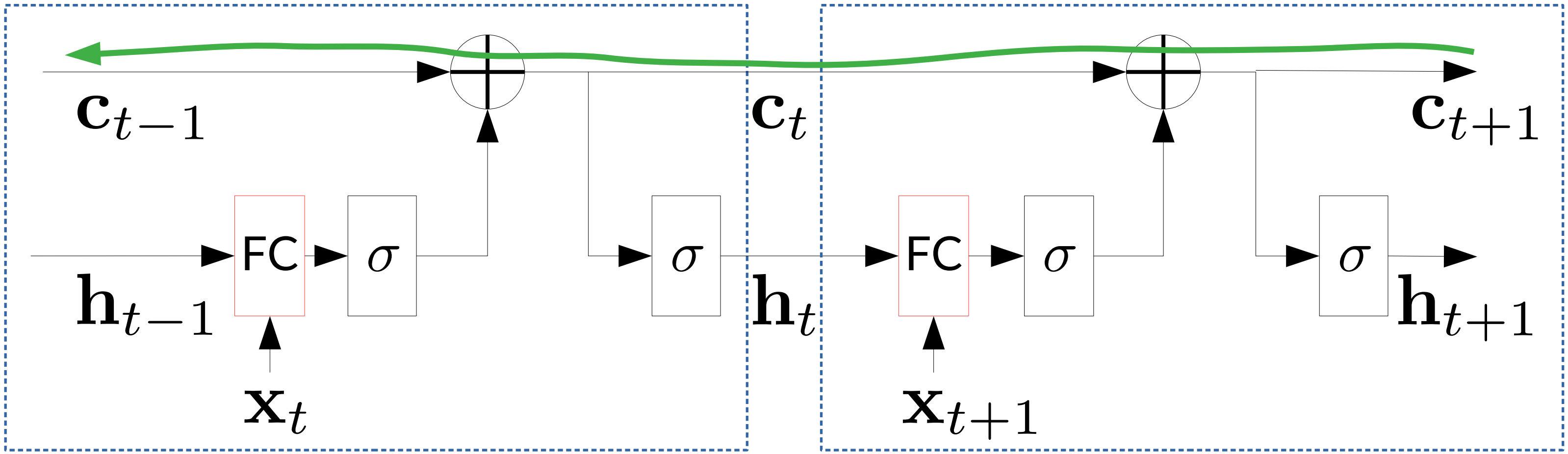
V)

« Long-Short Term Memory » (LSTM) = Réseau à mémoire court-terme et à mémoire long-terme



RNN « standard »

Propagation du gradient difficile



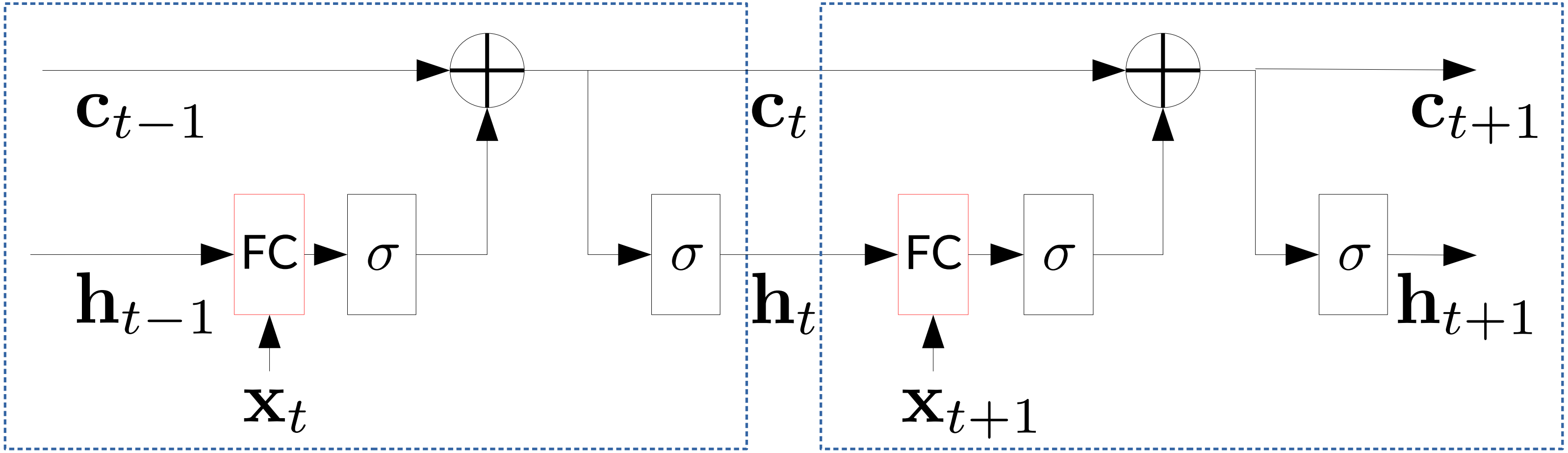
LSTM « simple »

Propagation du gradient plus simple

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathrm{I}$$

V)

« Long-Short Term Memory » (LSTM) = Réseau à mémoire court-terme et à mémoire long-terme



LSTM « simple »

Cellule (joue le rôle de mémoire) → $$\mathbf{c}_t = \mathbf{c}_{t-1} + tanh(\mathtt{W}_h \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_h)$$

État caché → $$\mathbf{h}_t = tanh(\mathbf{c}_t)$$

Connexion résiduelle, comme dans ResNet (2015)
Rappel : LSTM (1998)

V)

LSTM « complet »

Sigmoïde (sortie dans [0, 1])

$$\mathbf{g}_{f_t} = \sigma\left(\mathtt{W}_{g_f} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{g_f}\right)$$

« forget gate » pour effacer la mémoire

$$\mathbf{g}_{i_t} = \sigma\left(\mathtt{W}_{g_i} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{g_i}\right)$$

« input gate » pour écrire dans la mémoire

$$\mathbf{g}_{o_t} = \sigma\left(\mathtt{W}_{g_o} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{g_o}\right)$$

« output gate » pour lire dans la mémoire

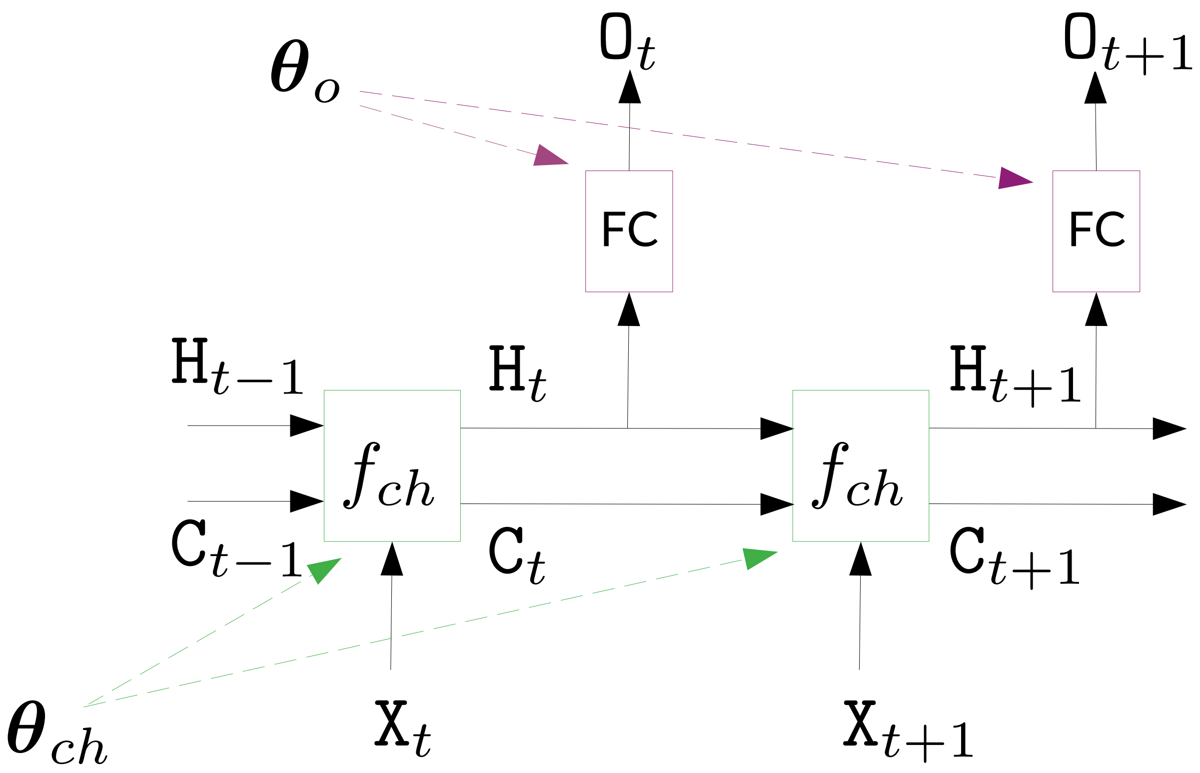
$$\mathbf{c}_t = \mathbf{g}_{f_t} \cdot \mathbf{c}_{t-1} + \mathbf{g}_{i_t} \cdot tanh\left(\mathtt{W}_h \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_h\right)$$

$$\mathbf{h}_t = \mathbf{g}_{o_t} \cdot tanh(\mathbf{c}_t)$$

$$\boxed{(\mathbf{h}_t, \mathbf{c}_t) := f_{ch}\left(\mathbf{h}_{t-1}, \mathbf{c}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}_{ch}\right)}$$

V)

LSTM : vue globale



Remarque : LSTM bidirectionnel = même chose que pour le RNN bidirectionnel

V)

« Gated Recurrent Unit » (GRU)

$$\mathbf{g}_{r_t} = \sigma(\mathtt{W}_{g_r} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{g_r})$$ «reset gate »

$$\mathbf{g}_{z_t} = \sigma(\mathtt{W}_{g_z} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{g_z})$$ «update gate »

$$\tilde{\mathbf{h}}_t = tanh(\mathtt{W}_h \begin{bmatrix} \mathbf{g}_{r_t} \cdot \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_h)$$

$$\boxed{\mathbf{h}_t := f_h\left(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}_h\right)}$$

$$\mathbf{h}_t = (1 - \mathbf{g}_{z_t}) \cdot \tilde{\mathbf{h}}_t + \mathbf{g}_{z_t} \cdot \mathbf{h}_{t-1}$$

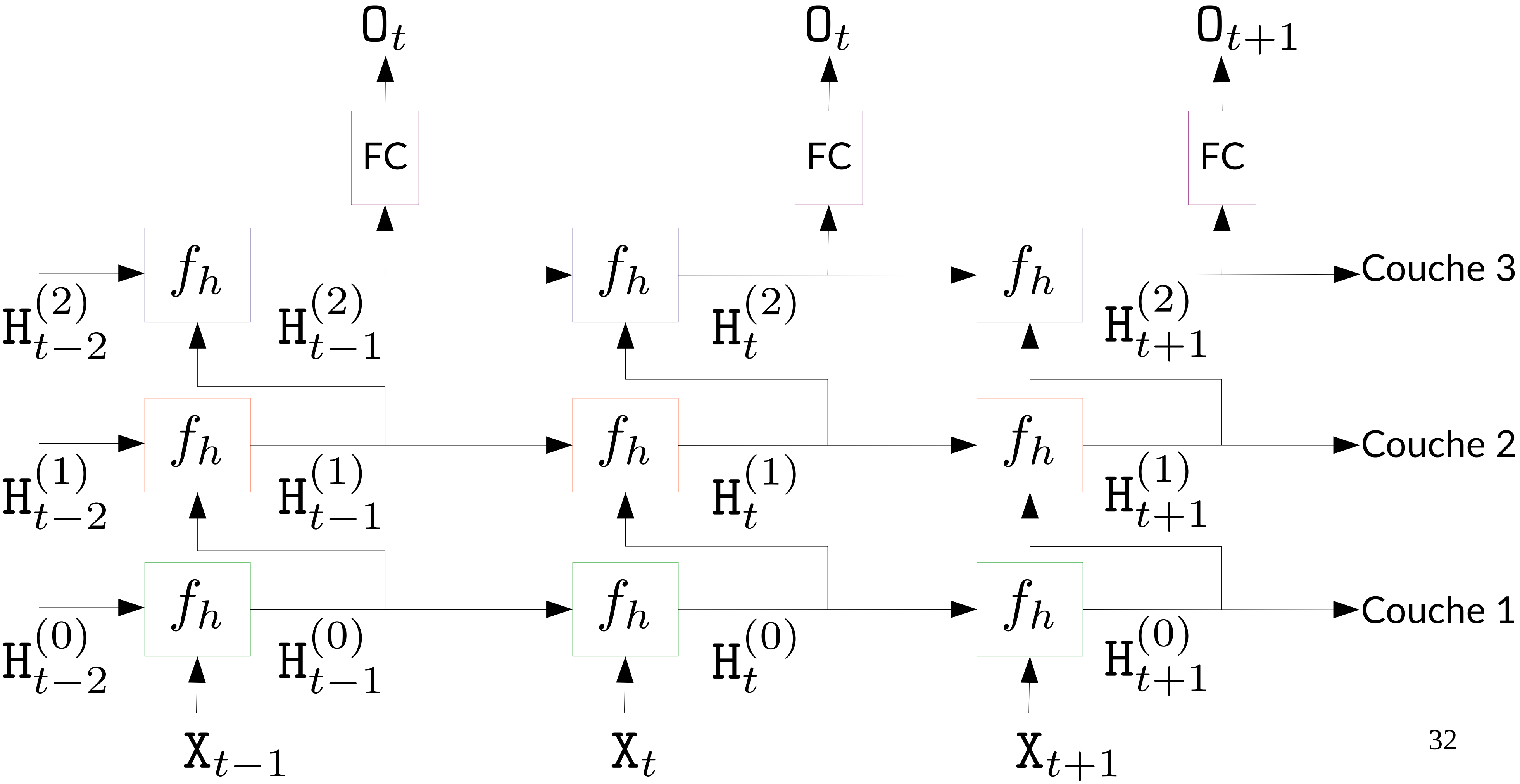
- Plus « léger », pas de cellule, et moins de paramètres que LSTM

- Généralise le RNN « standard »
  Si tous les éléments de la « reset gate » valent 1
  Si tous les éléments de la « update gate » valent 0
  → RNN « standard »

VI) Réseaux de neurones récurrents profonds (« Deep RNN »)

VI)

Réseaux de neurones récurrents profonds (« Deep RNN »)



VII) Limites et conclusion

VII)

Limites et conclusion

Avantages du RNN

- Peut s'appliquer à beaucoup de problèmes
- Peut théoriquement apprendre de très longues dépendances

Inconvénients du RNN

- Séquentiel par nature

  → Peu parallélisable

  → Apprentissage long

RNN souvent remplacés par les CNN et dernièrement par les Transformers

TP : Description d'une image