

Réseaux neuronaux récurrents et les LSTMs

TALN Semaine 6

Merci à Dan Jurafsky pour l'inspiration des diapos de cette semaine !

Plan pour aujourd'hui

1. Réseaux neuronaux récurrents simples (RNN)
2. Architectures RNN
 1. Pour la modélisation du langage
 2. Pour la classification
3. Long Short Term Memory Units (LSTMs)
4. *Exercices de groupe*

Nous étudierons les modèles de langage, ajoutant à leur complexité à chaque niveau :

MODÈLES DE LANGUE

- **Modèles de langage N-gram** – Modèles probabilistes de langage contextualisé
- **Embeddings** – Sémantiques vectorielles
- **Modèles de langage neuronaux récurrents, LSTMs** – Modèles de langage neuronaux
- **Transformers** – Grand modèle de langage ou LLM
- **Modèles encodeurs** – Tâches de notation
- **Modèles encodeur-décodeurs** – tâches génératives
- **Peaufinage et apprentissage en-contexte** – le paradigme actuel de modèle fondateur

Modéliser le temps dans les réseaux neuronaux

Le langage est intrinsèquement temporel.

Pourtant, les classificateurs simples en TALN que nous avons vus (par exemple pour l'analyse des sentiments) ignorent le temps

- (Les LM FNN et les Transformer que nous verrons la semaine prochaine utilisent une approche de "fenêtre mobile" du temps.)

Ici, nous introduisons une architecture avec une façon différente de représenter le temps

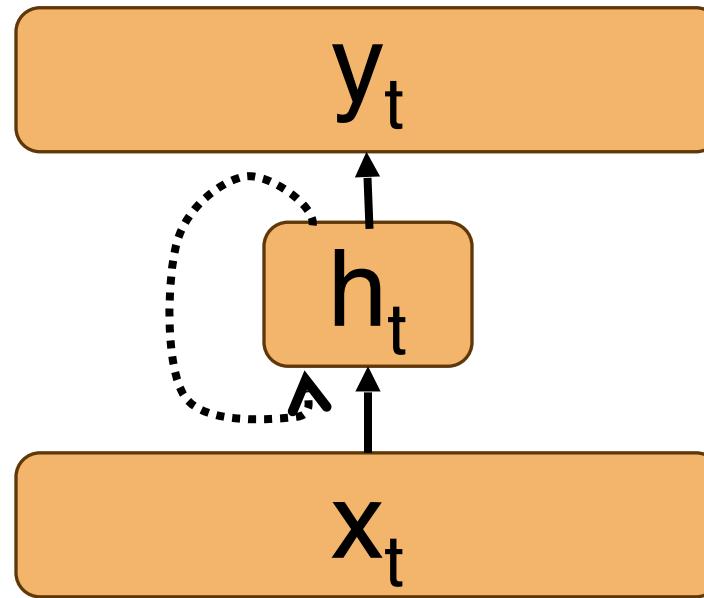
- **RNNs and leurs variants comme LSTMs**

Réseaux neuronaux récurrents (RNN)

Tout réseau qui contient un cycle dans ses connexions.

La valeur d'une unité neuronale dépend directement ou indirectement de ses propres sorties antérieures en tant qu'entrée.

Unités neuronales récurrentes simples (SRN ou Elman Net)



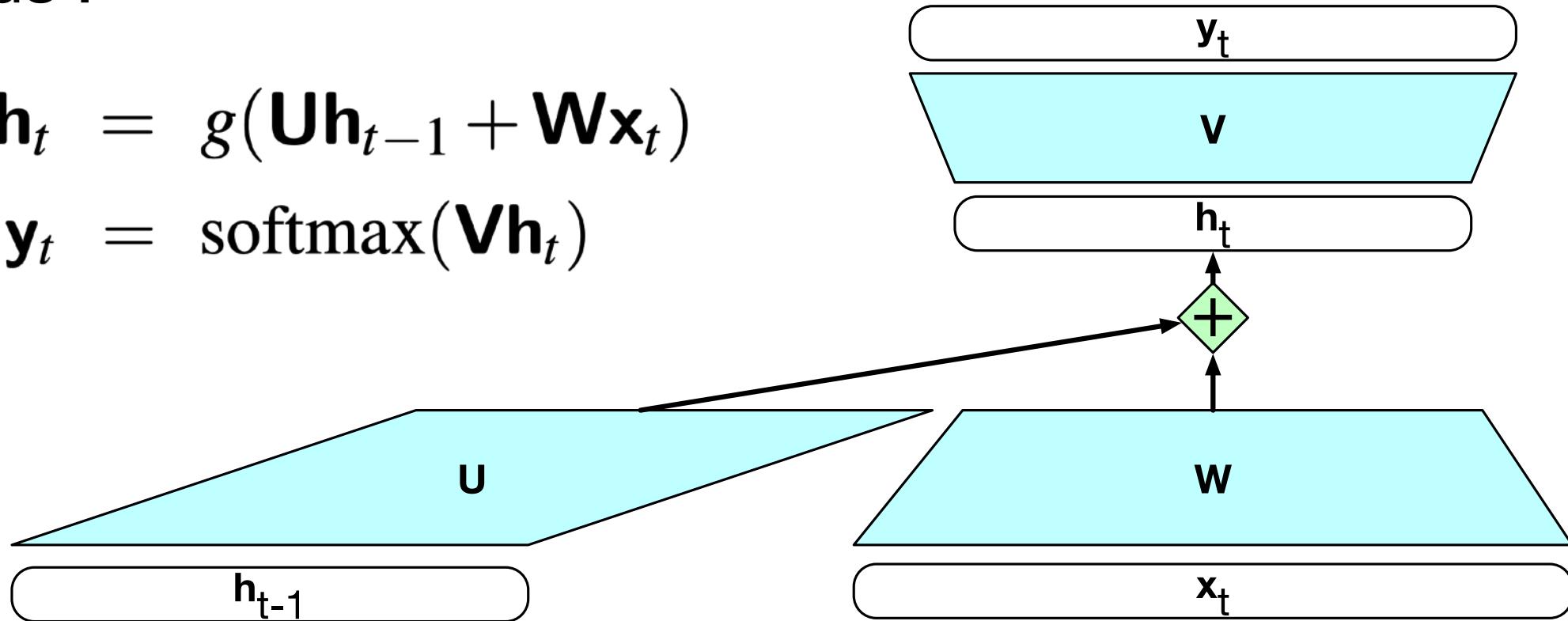
La couche cachée a une récurrence dans ses valeurs d'entrées
La valeur d'activation h_t depend de x_t et aussi de h_{t-1} !

Inférence vers l'avant dans des RNN simples

Très similaire aux réseaux feedforward que nous avons vus !

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$



L'inférence doit être incrémental

Calculer \mathbf{h} au temps t nécessite que nous calculons d'abord \mathbf{h} au temps précédent !

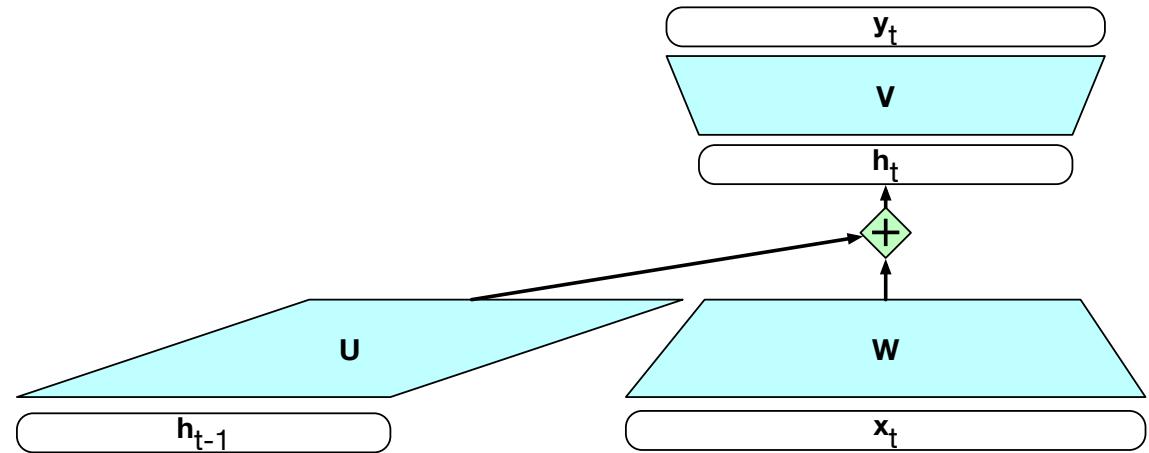
function FORWARDRNN($\mathbf{x}, network$) **returns** output sequence \mathbf{y}

```
 $\mathbf{h}_0 \leftarrow 0$ 
for  $i \leftarrow 1$  to LENGTH( $\mathbf{x}$ ) do
     $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$ 
     $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$ 
return  $y$ 
```

Entraîner un RNN simple

Tout comme la l'entraînement de FNN:

- Ensemble d'entraînement,
- Fonction de perte,
- backprop



Poids qui doivent être mis à jour :

- W , Les poids de la couche d'entrée à la couche cachée,
- U , Les poids de la couche cachée précédente à la couche cachée actuelle,
- V , Les poids de la couche cachée à la couche de sortie.

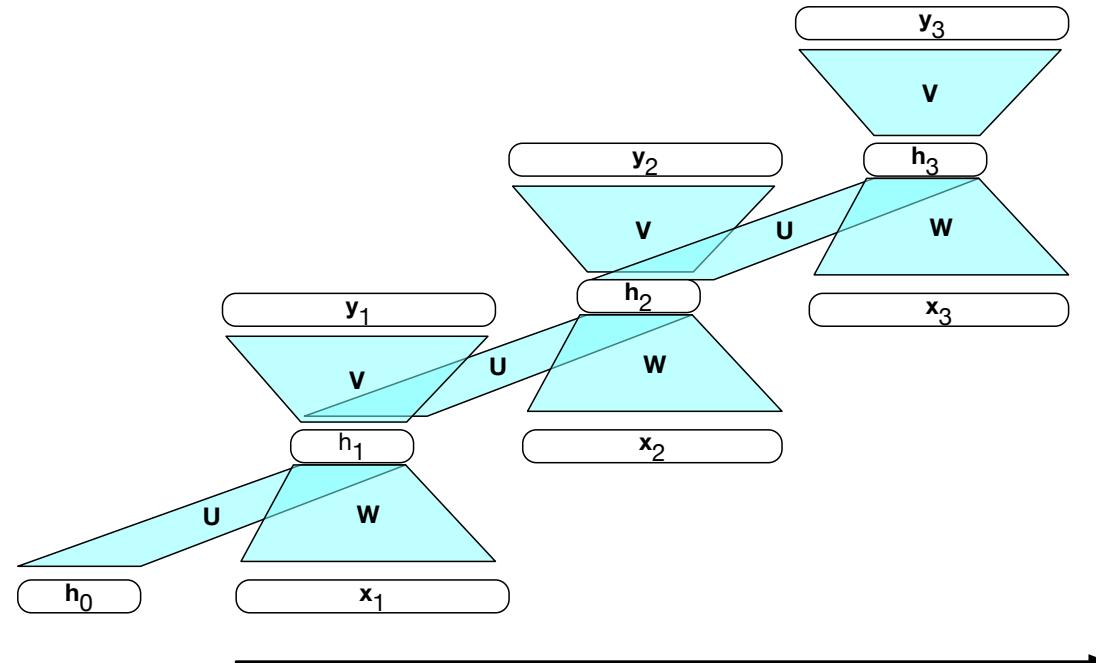
Entraîner un RNN simple : déroulement du temps

Contrairement aux FNNs:

1. Pour calculer la fonction de perte pour la sortie au moment t , nous avons besoin de la couche cachée du temps $t-1$.
2. La couche cachée au moment t influence la sortie au temps t et la couche cachée au temps $t+1$ (et donc la sortie et la perte à $t+1$).

Donc: Pour mesurer l'erreur dû à h_t ,

Nous devons connaître son influence à la fois sur la sortie actuelle ainsi que sur celles qui suivent.

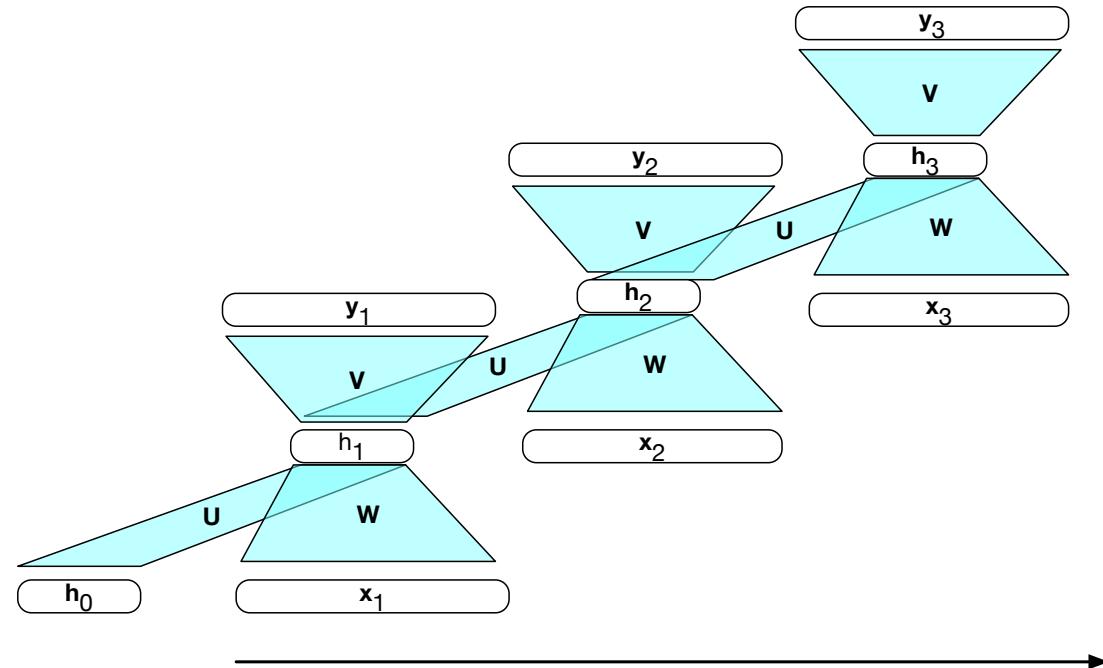


Entraîner un RNN simple : déroulement du temps

Nous déroulons le RNN en un graphe de calcul feedforward éliminant la récurrence !

Étant donné une séquence d'entrée :

1. Générer un réseau feedforward déroulé spécifique à l'entrée
2. Utilisez le graphique pour entraîner les poids directement via backprop ordinaire



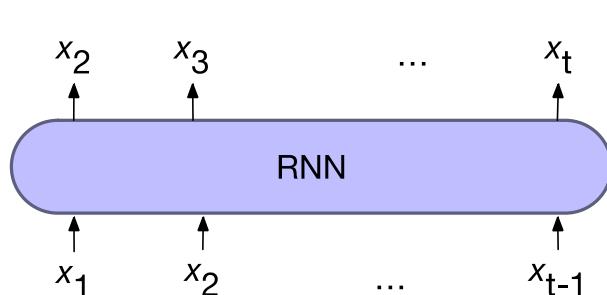
Architectures RNN

Nous verrons trois types d'architectures RNN qui peuvent être utilisées pour différentes tâches :

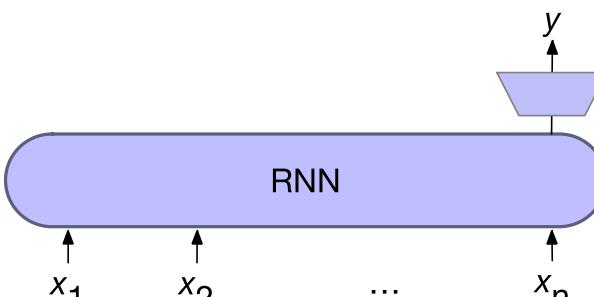
1. RNN pour la modélisation du langage

2. RNN pour la classification

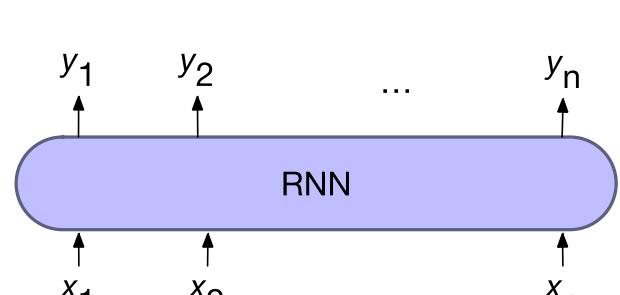
3. RNN pour l'étiquetage séquentiel



c) language modeling



b) sequence classification



a) sequence labeling

Rappel : Modélisation du langage

Modéliser la probabilité des mots en contexte.

Tâche: Prédire le mot suivant w_t

étant donné les mots antérieurs $w_{t-1}, w_{t-2}, w_{t-3}, \dots$

Problème avec les NN N-gram et feedforward : Traiter des séquences de longueur arbitraire.

Solution précédente: Fenêtres coulissantes (de longueur fixe)

$$P(w_t | w_{t-N+1}^{t-1})$$

Et les RNNs ? Aucune limite sur la taille du contexte ! Tous les mots précédents comptent. $P(w_t | w_1^{t-1})$

La taille du contexte de conditionnement pour différents LM

N-gram LM:

La taille du contexte est le $n - 1$ mots antérieurs.

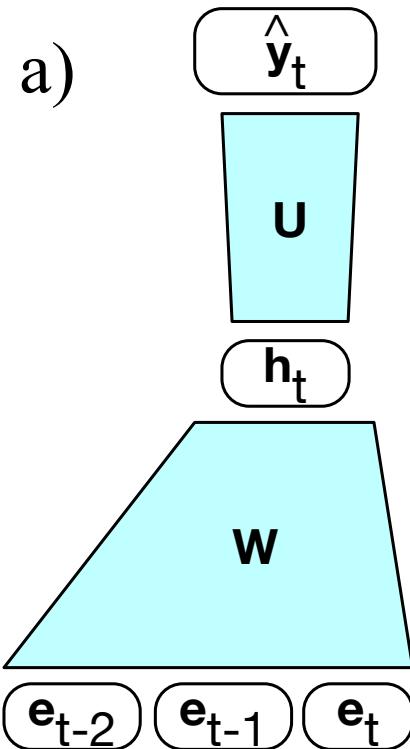
FNN LM:

Le contexte est la taille de la fenêtre.

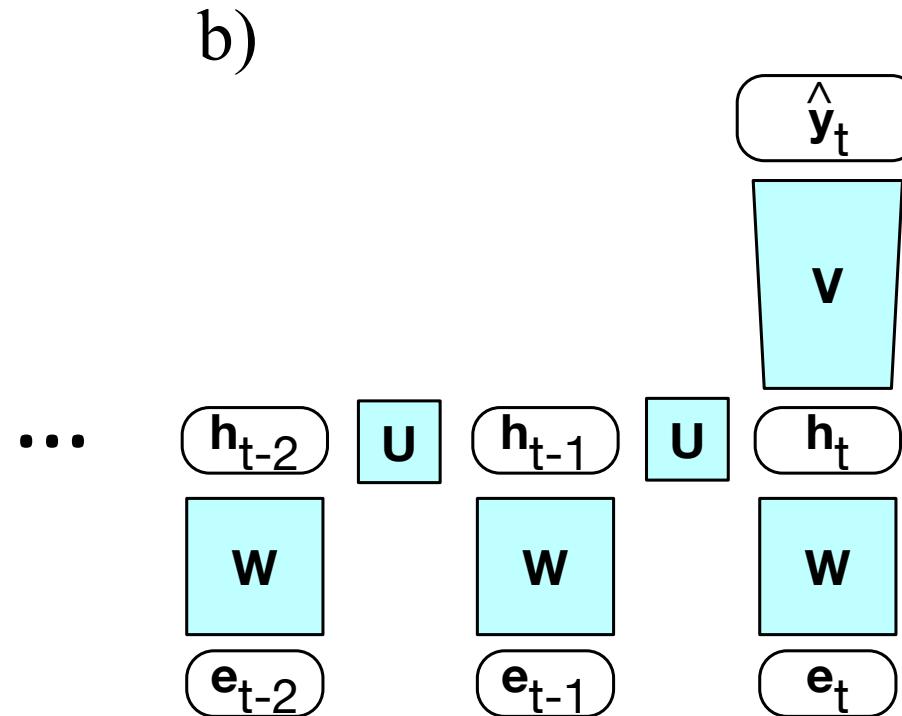
RNN LM:

Pas de taille de contexte fixe; h_{t-1} Représente toute l'historique.

Feedforward LMs vs RNN LMs



FNN



RNN

Inférence vers l'avant avec un RNN LM

Étant donné la séquence \mathbf{X} de N tokens

$$\mathbf{X} = [\mathbf{x}_1; \dots; \mathbf{x}_t; \dots; \mathbf{x}_N]$$

Utilisez la matrice d'embeddings pour obtenir l'embedding du token actuel \mathbf{x}_t

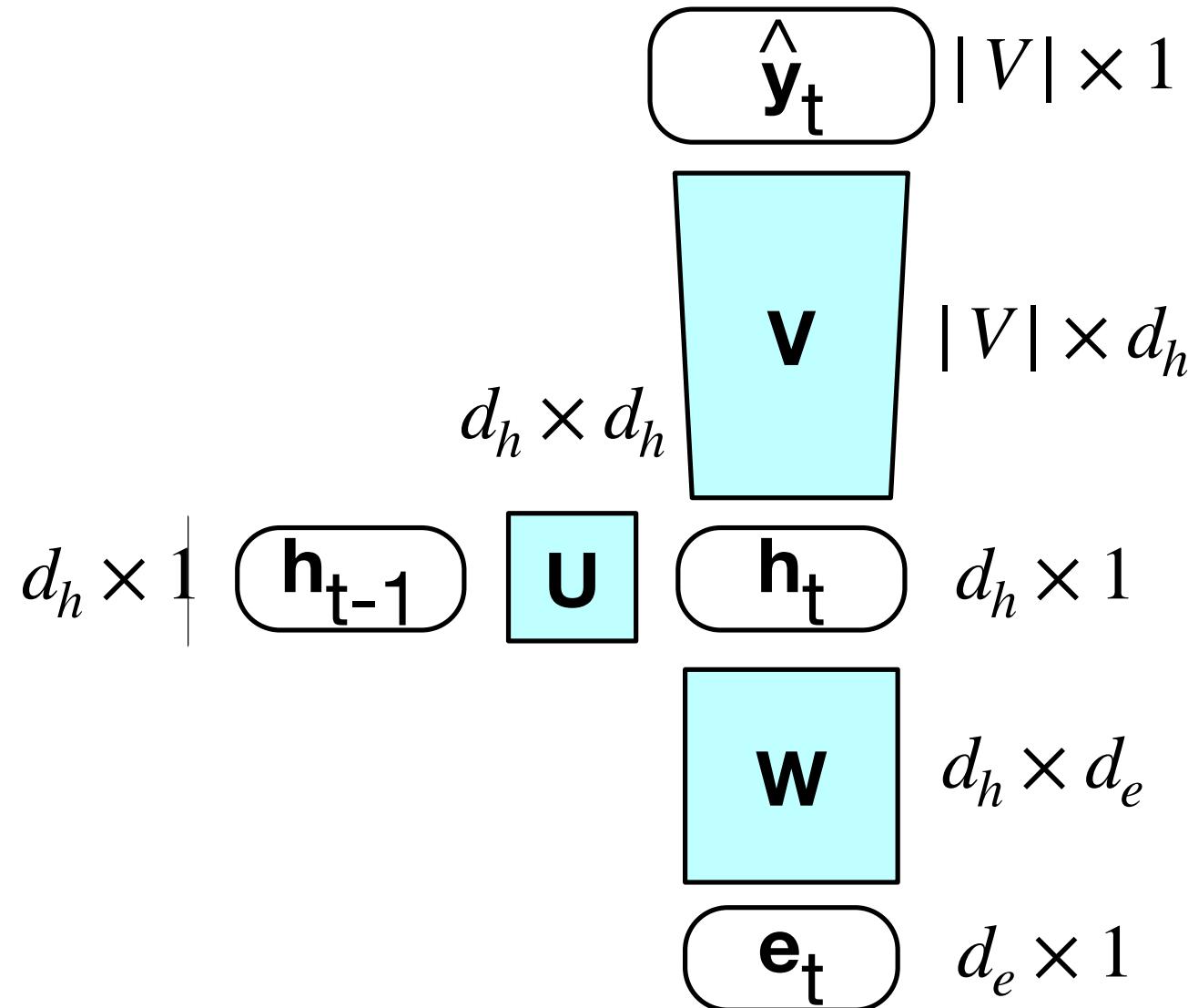
$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$

Puis combinez ...

$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

Les tailles des matrices et vecteurs



Calculer la probabilité que le mot suivant soit le mot k

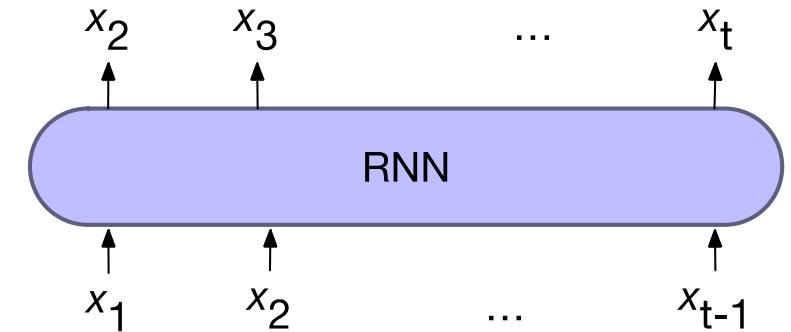
$$P(w_{t+1} = k | w_1, \dots, w_t) = \hat{\mathbf{y}}_t[k]$$

Calcul de la probabilité, ou évaluer, une phrase

$$\begin{aligned} P(w_{1:n}) &= \prod_{i=1}^n P(w_i | w_{1:i-1}) \\ &= \prod_{i=1}^n \hat{\mathbf{y}}_i[w_i] \end{aligned}$$

Entraînement de RNN LM

- **Auto-supervision (comme avec les FNN)**
 - Prendre un ensemble de texte comme matériel de formation
 - Demandez au modèle de prédire le mot suivant
 - (Contrairement au FNN) à chaque temps t !
- **Pourquoi ça s'appelle auto-supervision ?** Nous n'avons pas besoin d'étiquettes humaines ; le texte est son propre signal de supervision
- **Nous formons le modèle à :**
 - Minimiser l'erreur dans la prédiction du vrai mot suivant dans la séquence d'entraînement, en utilisant l'entropie croisée comme fonction de perte.



c) language modeling

Erreur d'entropie croisée (CE loss)

Minimise la différence entre (1) une distribution de probabilité prédite et (2) la vraie distribution.

$$L_{CE} = - \sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

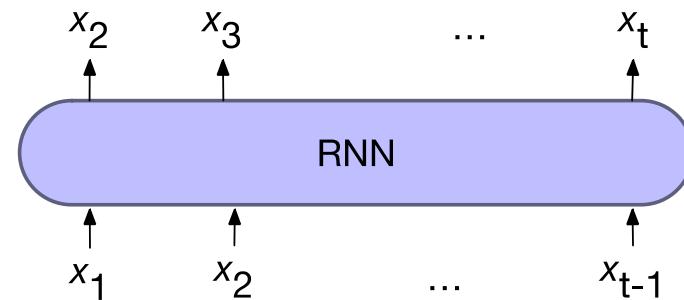
La fonction de perte CE pour les LM peut être simplifiée puisque :

- La vraie distribution \mathbf{y}_t est un vecteur one-hot sur le vocabulaire (i.e. Où la probabilité du mot suivant réel est 1, et toutes les autres entrées sont 0.)
- CE loss pour LMs n'est déterminé que par la probabilité du mot suivant.
- Donc au temps t, CE loss est: $L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$

Apprentissage dirigé (Teacher forcing)

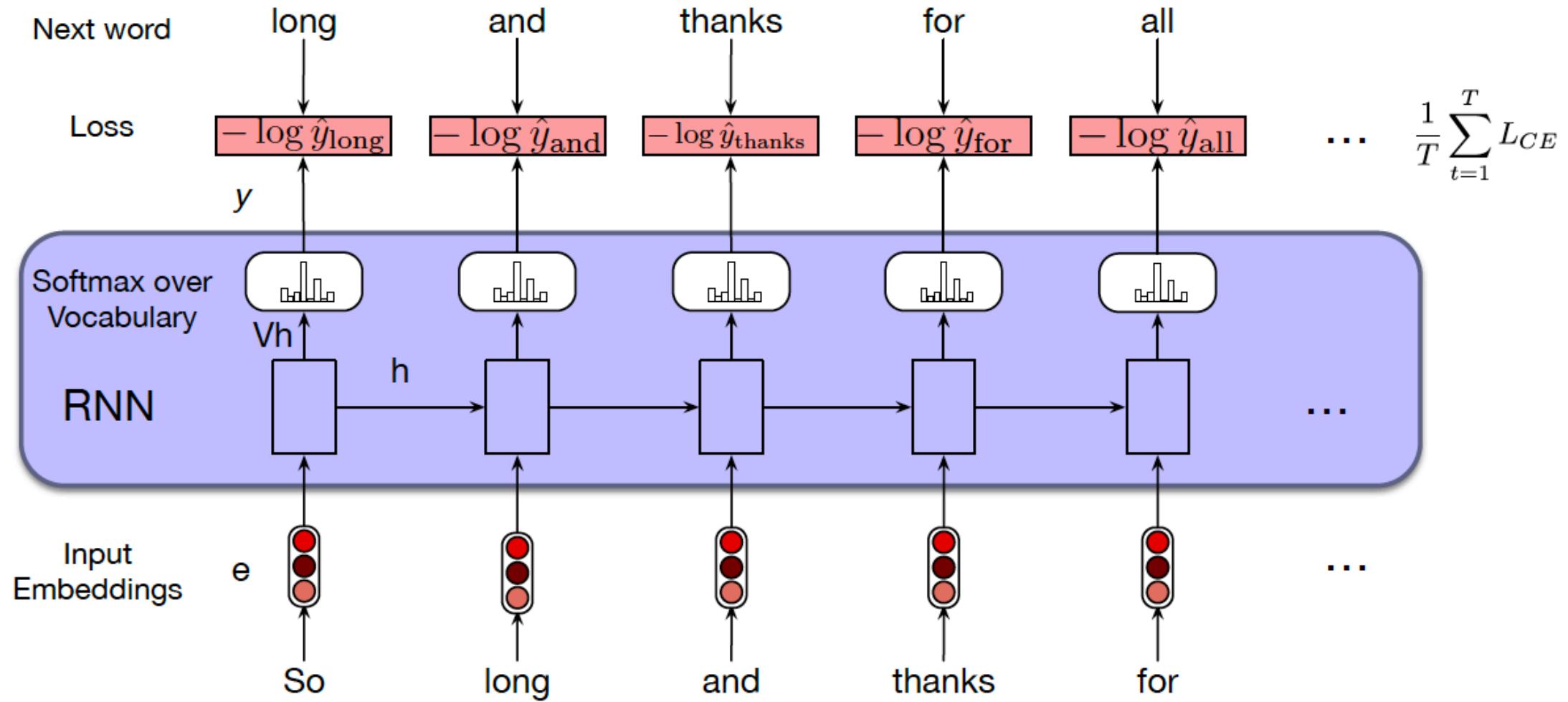
Nous donnons toujours au modèle le vrai historique pour prédire le mot suivant (plutôt que d'alimenter le modèle avec la prédiction du mot suivant au temps précédent - ce qui pourrait être faux).

C'est ce qu'on appelle l'apprentissage dirigé (durant l'entraînement, nous forçons le contexte à être correct en fonction des données de référence).



c) language modeling

Résumé : Entraînement de RNN pour la modélisation du langage



Attacher les poids (Weight tying)

Une modification architecturale facultative lorsque $d_e = d_h$.

Lorsque la dimension d'embedding et la dimension cachée sont identiques, alors la matrice d'embedding **E** and et la matrice de poids de la couche finale **V** ont des formes similaires: **E** est $d \times |V|$ et **V** est $|V| \times d$

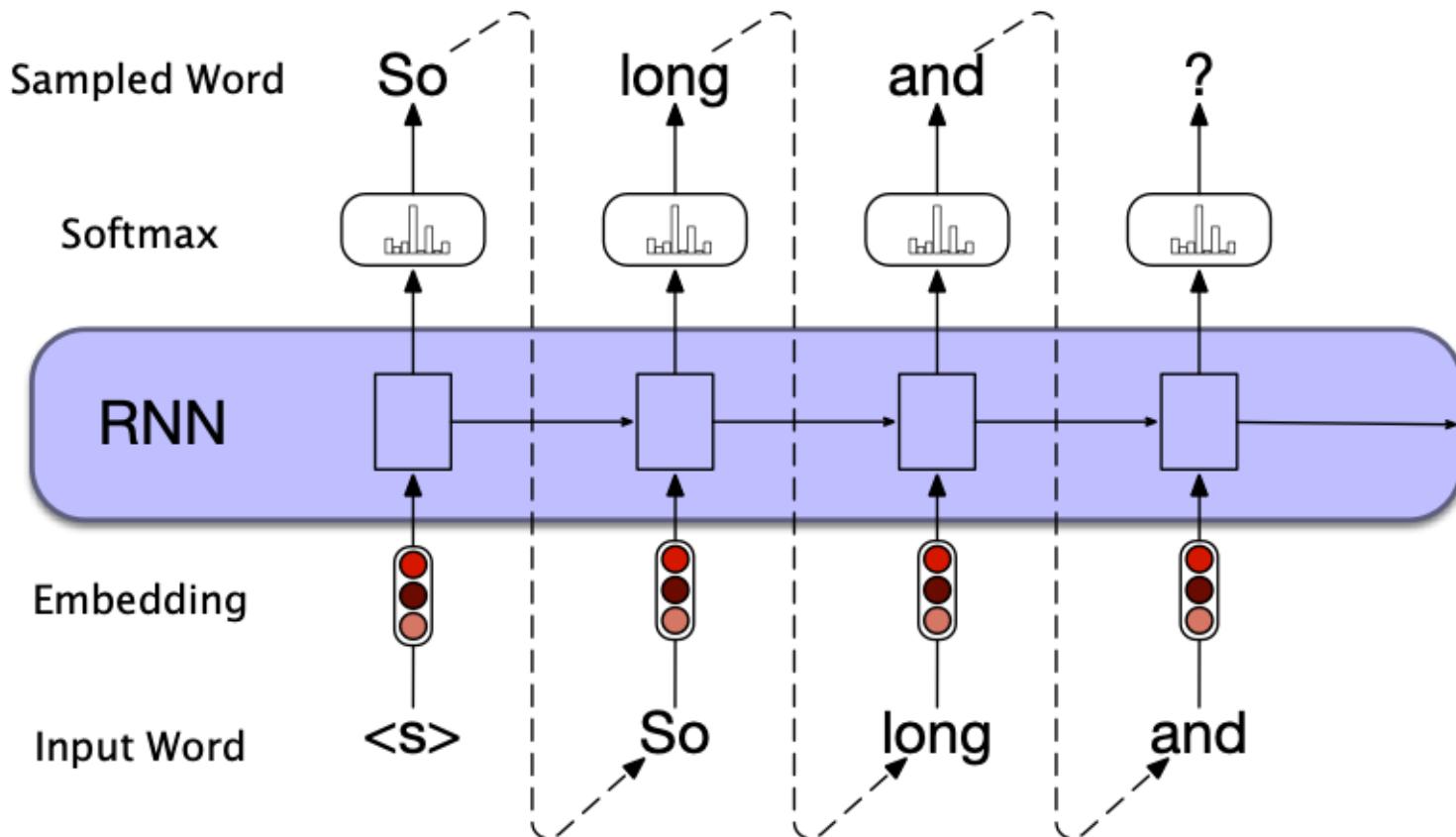
Au lieu d'avoir des matrices séparées, nous les attachons simplement ensemble, en transposant **E^T** pour remplacer **V**, donc les embeddings y sont deux fois:

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$

$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{E}^T \mathbf{h}_t)$$

Génération autorégressive à partir d'un RNN LM



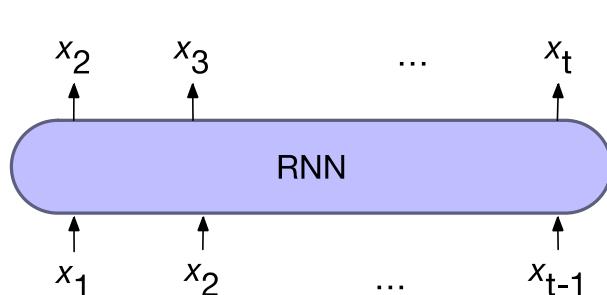
Architectures RNN

Nous verrons trois types d'architectures RNN qui peuvent être utilisées pour différentes tâches :

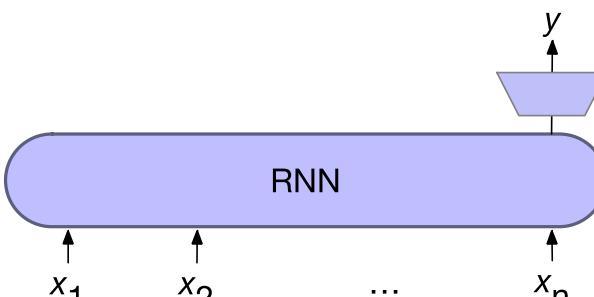
1. RNN pour la modélisation du langage

2. RNN pour la classification

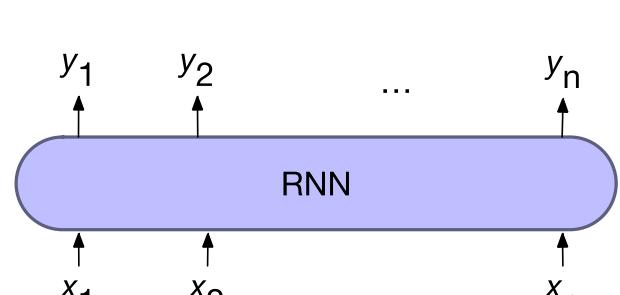
3. RNN pour l'étiquetage séquentiel



c) language modeling



b) sequence classification

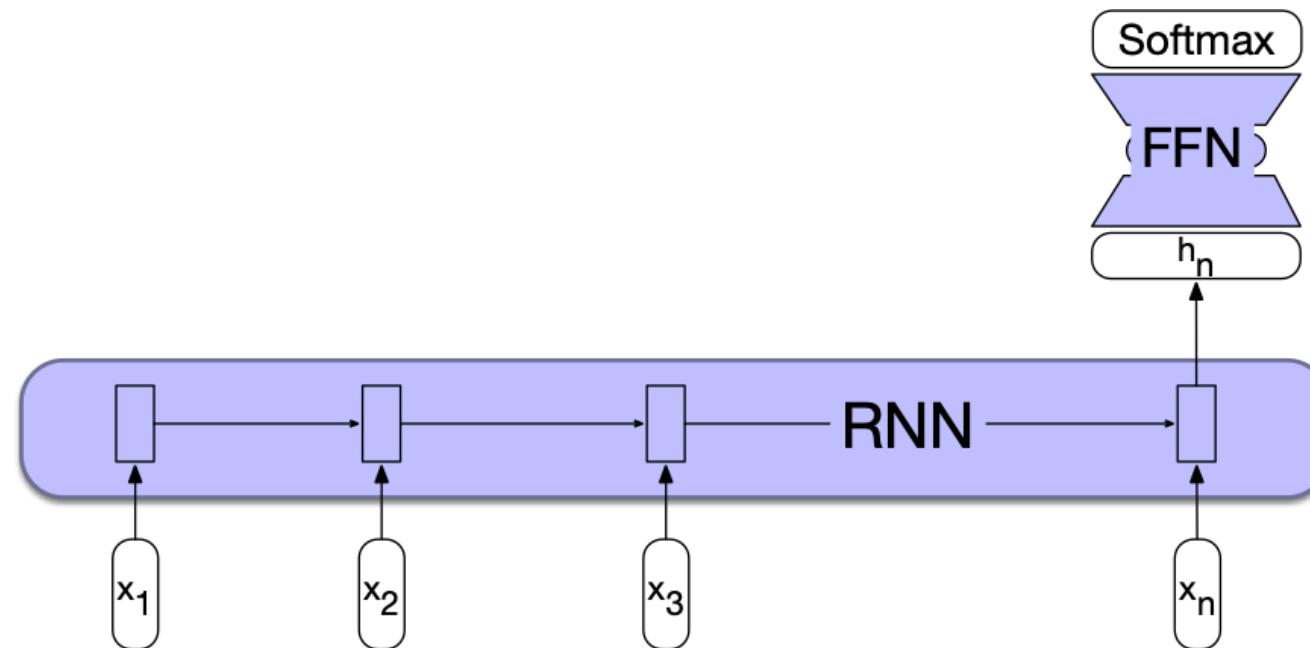


a) sequence labeling

RNN pour la classification de textes

Il n'y a qu'une seule prédiction de sortie finale, plutôt qu'une sortie à chaque état du temps.

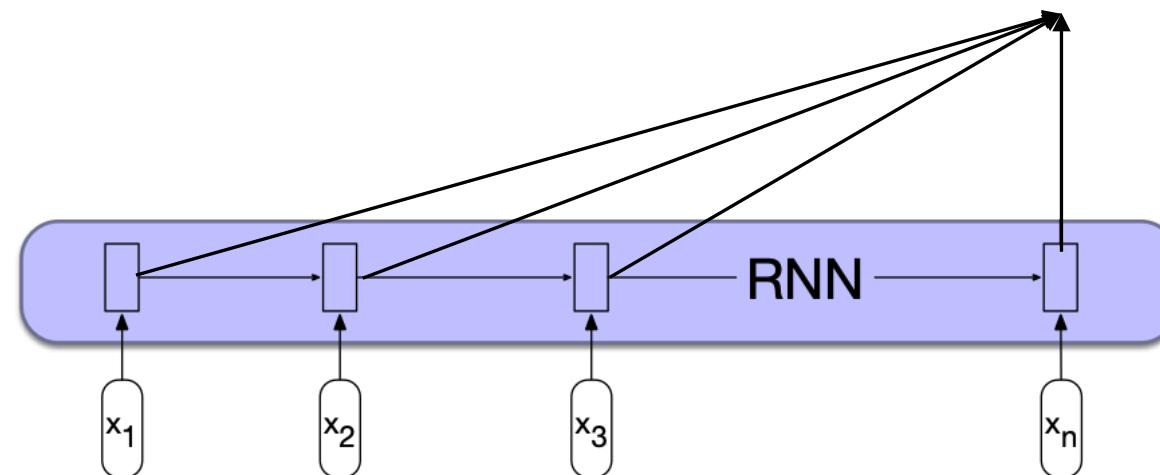
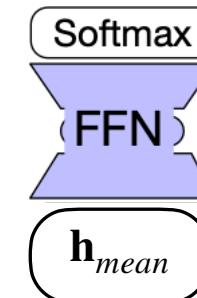
Nous pouvons utiliser juste l'état caché final comme codage de séquence qui est transmis à un NN feedforward régulier pour la classification.



RNN pour la classification de textes

Alternativement, au lieu de prendre le dernier état caché seulement, nous pouvons utiliser une fonction qui prend en compte tous les états cachés, comme le sous-échantillonnage de la valeur moyenne (**mean pooling**) .

$$\mathbf{h}_{mean} = \frac{1}{n} \sum_{i=1}^n \mathbf{h}_i$$



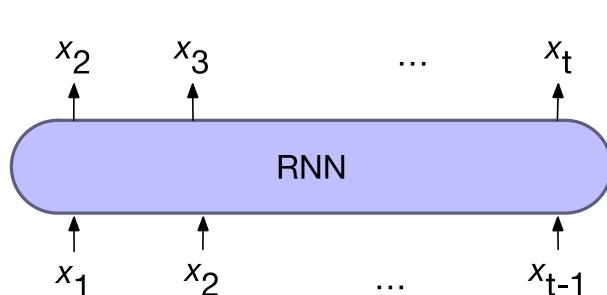
Architectures RNN

Nous verrons trois types d'architectures RNN qui peuvent être utilisées pour différentes tâches :

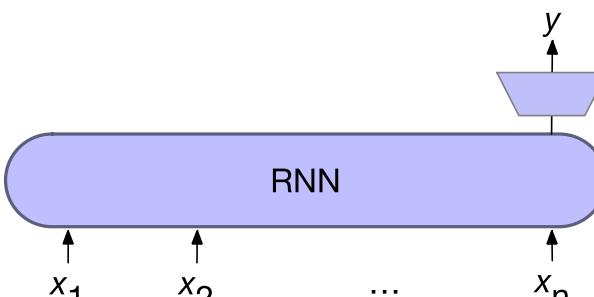
1. RNN pour la modélisation du langage

2. RNN pour la classification

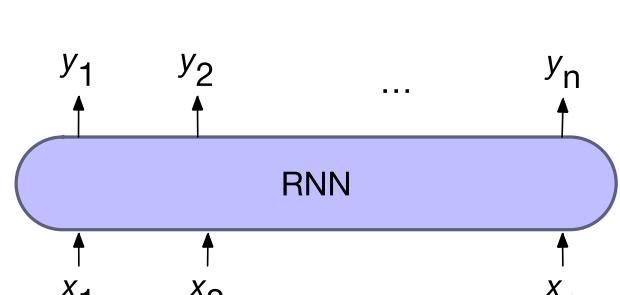
3. RNN pour l'étiquetage séquentiel



c) language modeling



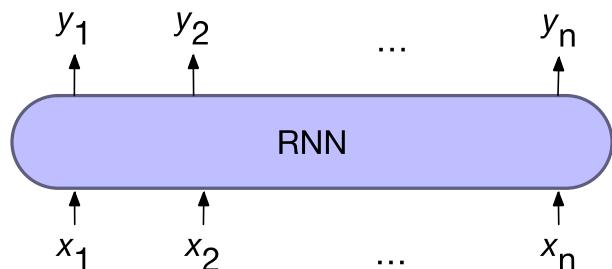
b) sequence classification



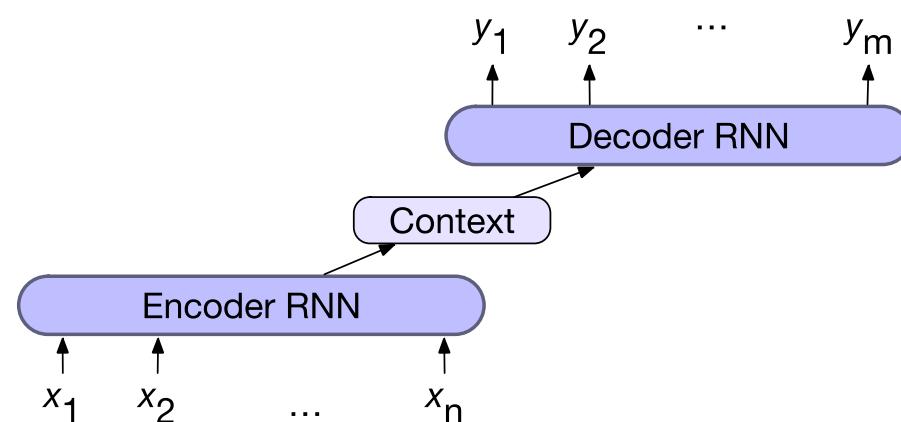
a) sequence labeling

Deux types de tâches de prédiction de séquence

- 1. Étiquetage séquentiel** : Lorsque la longueur de la séquence de sortie correspond à celle de l'entrée
- 2. modèle seq-à-séq ou encodeur-decodeur** : Lorsque les séquences de sortie et d'entrée sont de longueur différente
(Semaine 10 du cours)



a) sequence labeling

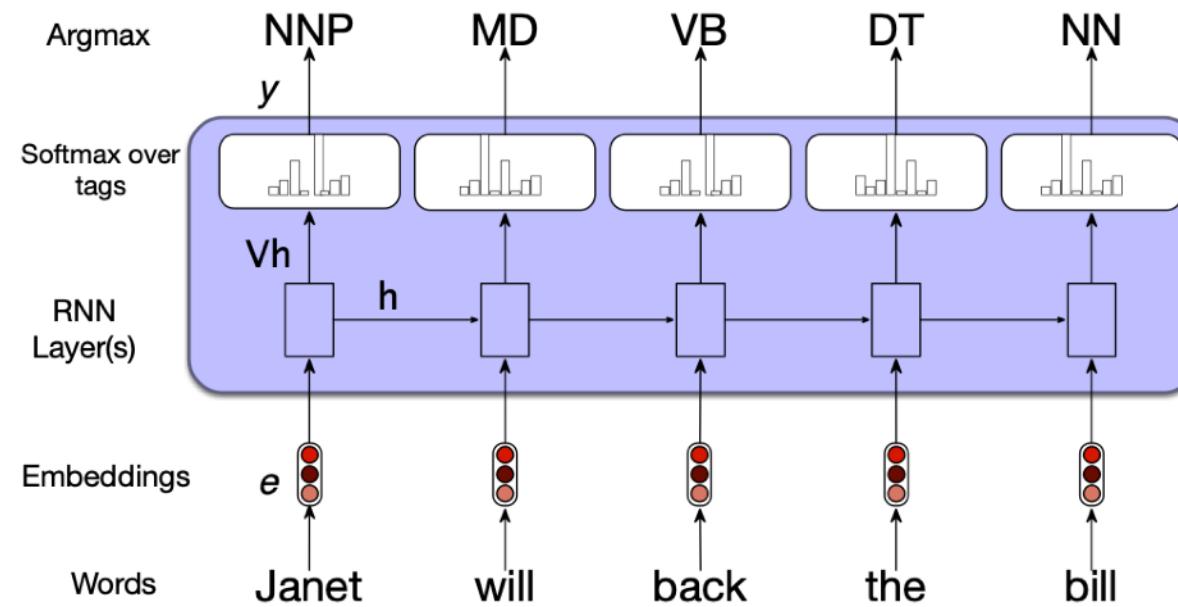


d) encoder-decoder

RNN pour l'étiquetage séquentiel

Attribuer une étiquette à chaque élément d'une séquence (ex. Étiquetage morphosyntaxique (POS tagging)):

- La principale différence avec l'architecture LM et l'architecture de classification est que la dimension de sortie est sur le nombre d'étiquettes (pas le vocabulaire), et qu'il y a une prédiction de sortie à chaque état de temps (pas seulement le dernier).



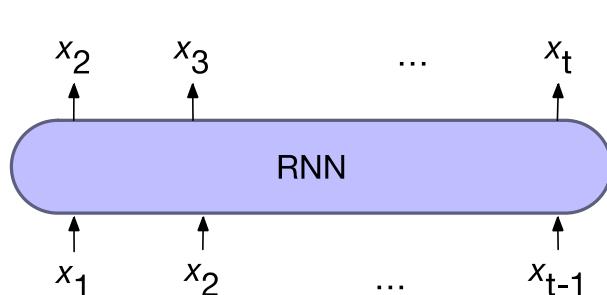
Architectures RNN

Nous verrons trois types d'architectures RNN qui peuvent être utilisées pour différentes tâches :

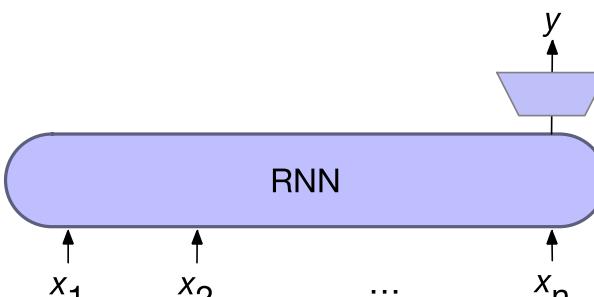
1. RNN pour la modélisation du langage

2. RNN pour la classification

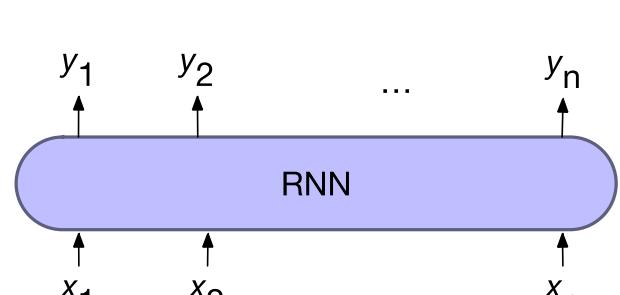
3. RNN pour l'étiquetage séquentiel



c) language modeling



b) sequence classification



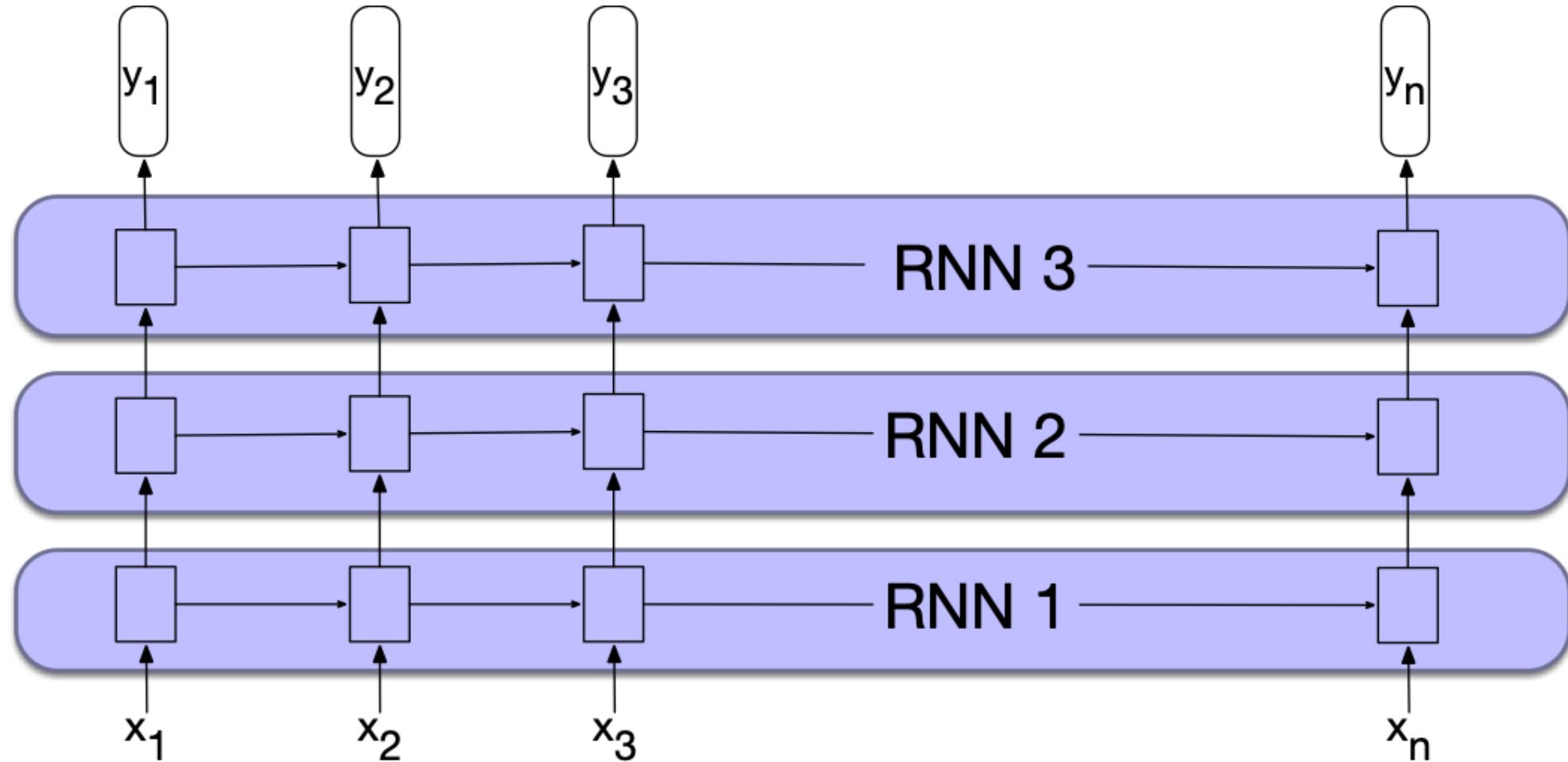
a) sequence labeling

Autres modifications architecturales

Nous avons vu différentes façons d'utiliser le RNN à couche simple de manière unidirectionnelle. Dans tous les cas d'utilisation, nous pouvons également modifier nos architectures pour inclure :

- 1. Plusieurs couches ou RNN empilés**
- 2. Encodage bidirectionnel**

RNNs empilés

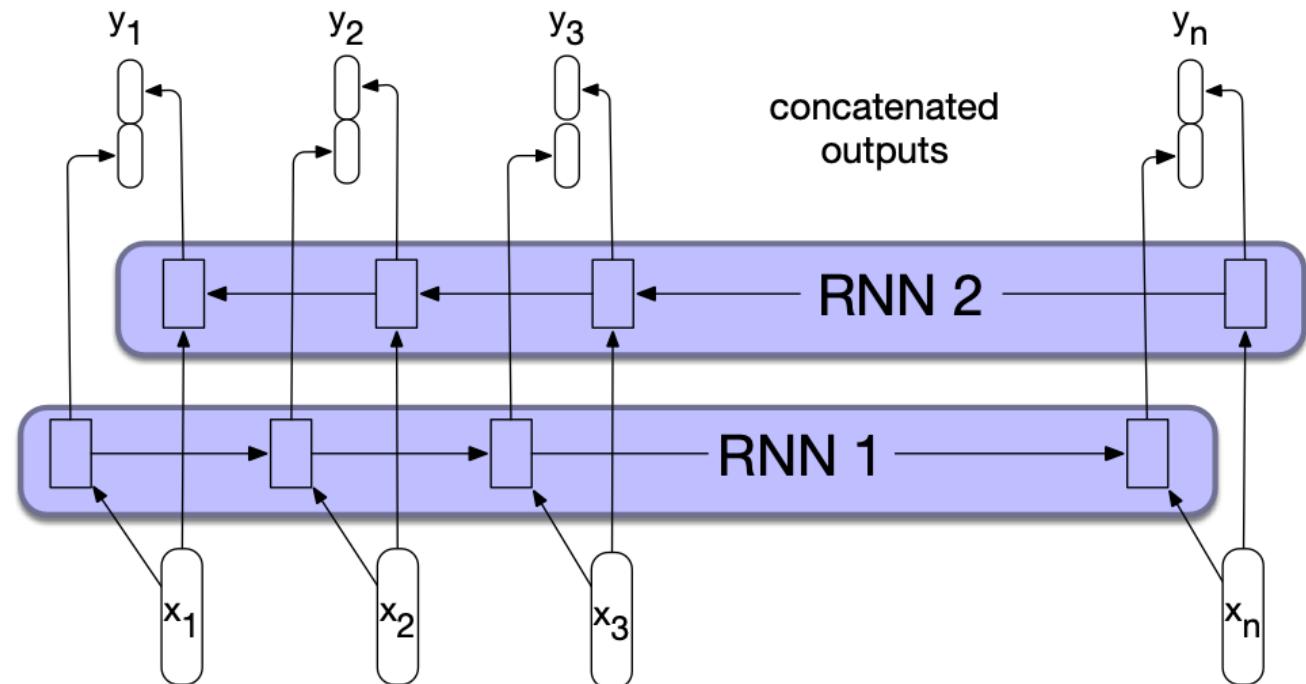


RNNs bidirectionnels

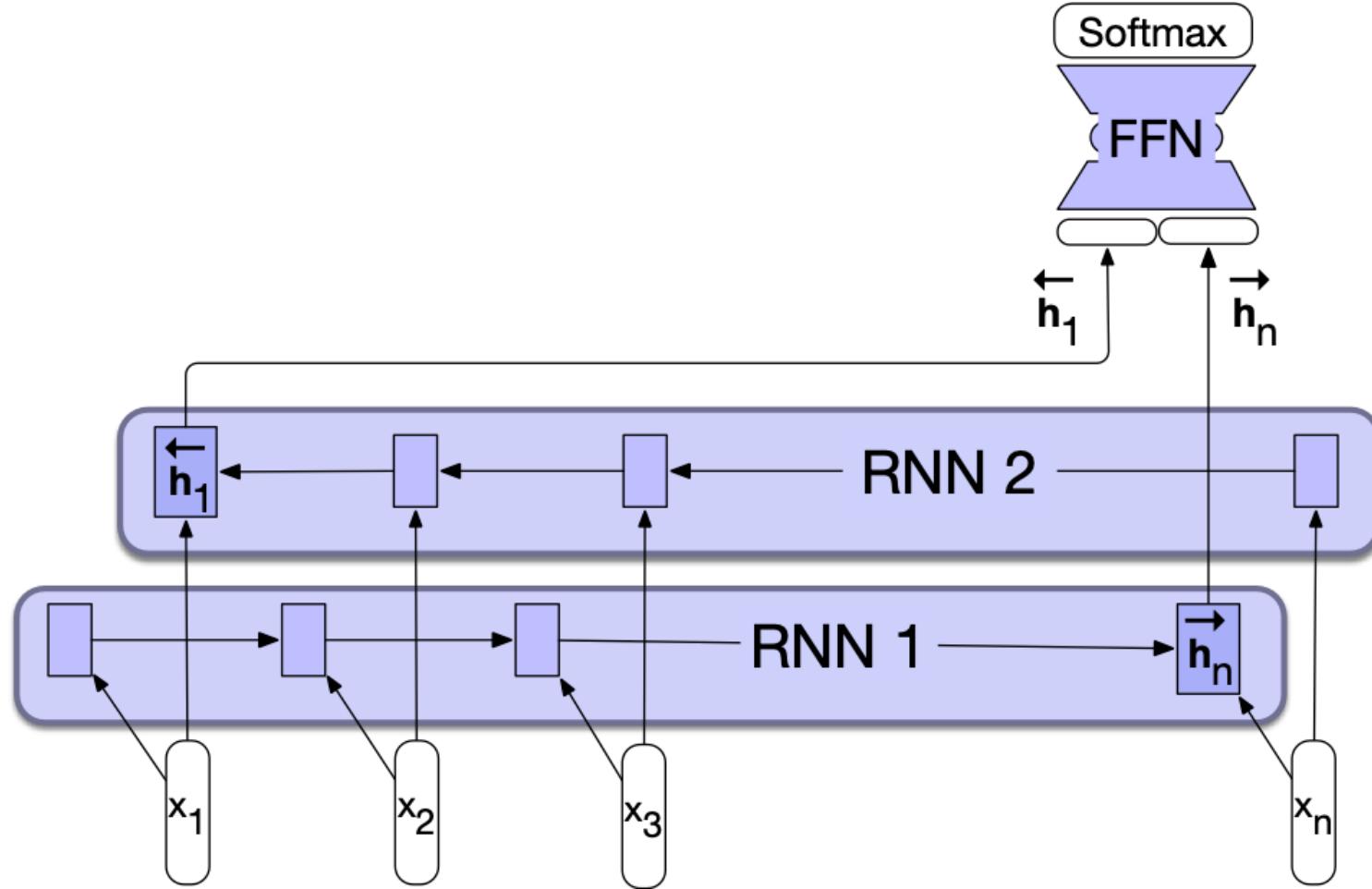
$$\mathbf{h}_t^f = \text{RNN}_{\text{forward}}(\mathbf{x}_1, \dots, \mathbf{x}_t)$$

$$\mathbf{h}_t^b = \text{RNN}_{\text{backward}}(\mathbf{x}_t, \dots, \mathbf{x}_n)$$

$$\begin{aligned}\mathbf{h}_t &= [\mathbf{h}_t^f ; \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b\end{aligned}$$



RNNs bidirectionnels pour classification



Autres types d'unités récurrentes

Rappel : RNN est tout réseau qui contient un cycle dans ses connexions réseau, c'est-à-dire que la valeur d'une unité dépend directement, ou indirectement, de ses propres sorties antérieures en tant qu'entrée.

Toutes les architectures que nous venons de voir peuvent être mises en œuvre avec N'IMPORTE QUEL type de RNN.

Nous avons vu les simple recurrent neural units (SRN). Deux autres types courants sont les gated recurrent units (GRU) and les **long-short-term memory units (LSTM)**

Motiver le LSTM : faire face au longue distance

Les couches cachées de RNN sont obligées de faire deux choses :

- a) Fournir des informations utiles pour la décision actuelle,
- b) Mettre à jour et reporter les informations requises pour les décisions futures.

Conduit à deux problèmes :

1. Il est difficile d'attribuer des probabilités avec précision lorsque le contexte est très éloigné : ‘The flights the airline was canceling were full’

Cela nécessite de connaître et de hiérarchiser les infos provenant d'états antérieurs lointains par rapport à des infos plus proches.

2. Pendant le backprop, nous devons multiplier à plusieurs reprises les gradients à travers le temps et de nombreux états cachés qui peuvent conduire au **problème de l'évanescence du gradient (vanishing gradient)**.

LSTM: Long short-term memory network

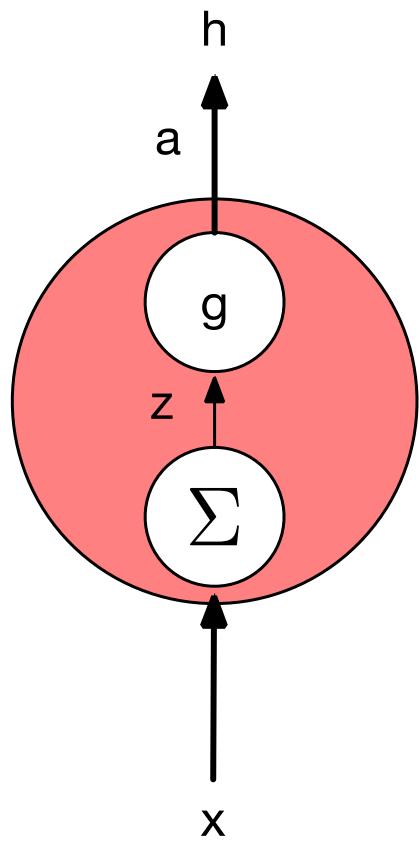
Le LSTM divise le problème de gestion du contexte en deux sous-problèmes :

1. Supprimer les informations qui ne sont plus nécessaires du de l'historique,
2. Ajouter des informations susceptibles d'être nécessaires pour une prise de décision ultérieure

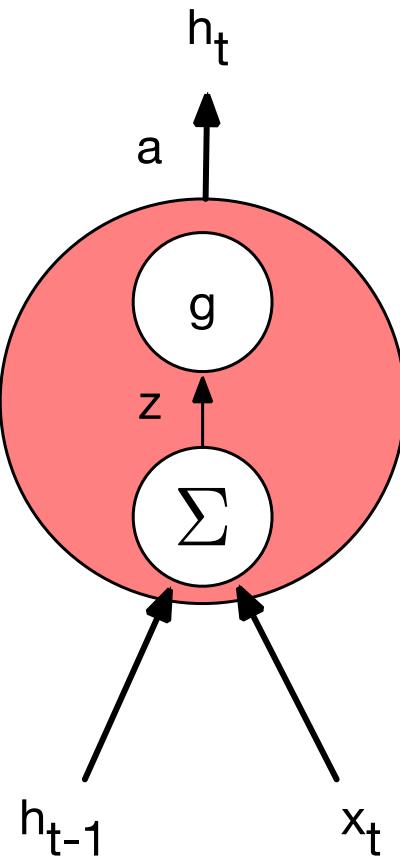
Le LSTM rajoute:

- une couche de contexte explicite
- Circuits neuronaux avec portes pour contrôler le flux d'informations

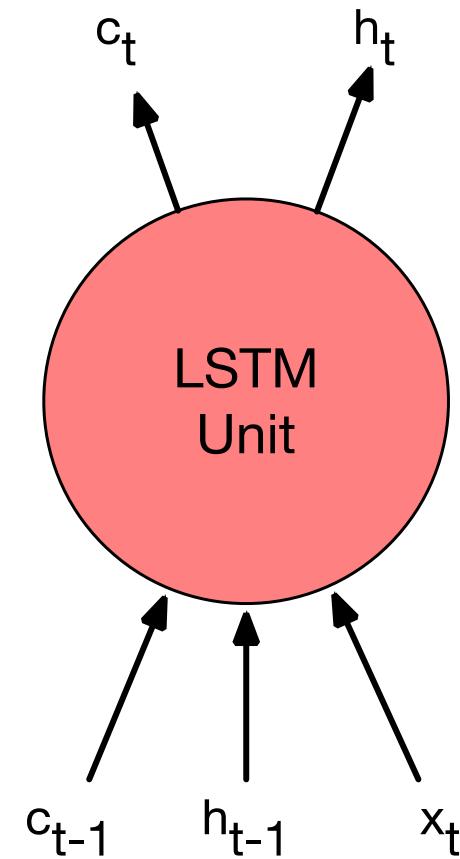
Différence entre les unités neuronales



(a)
Feedforward

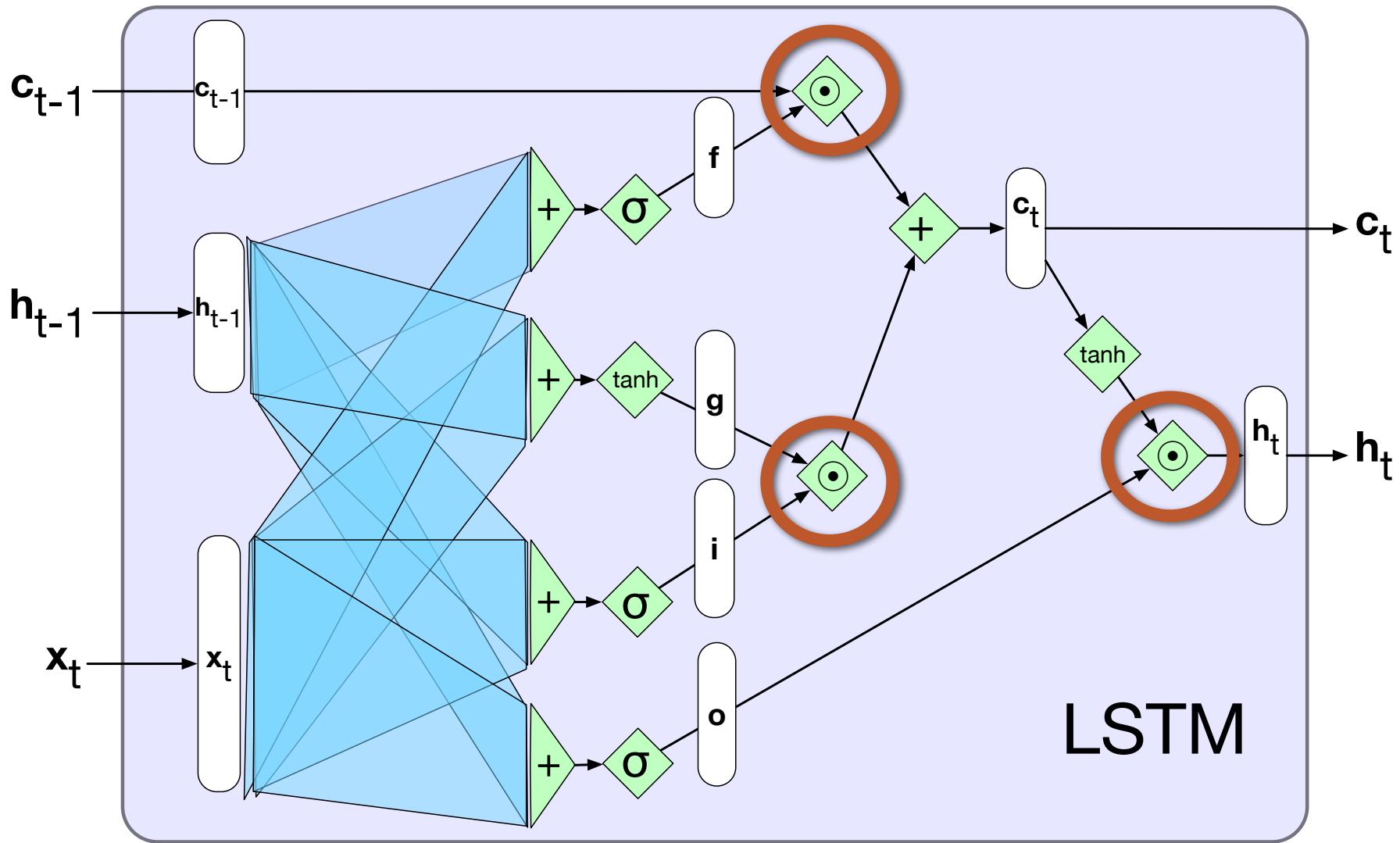


(b)
SRN

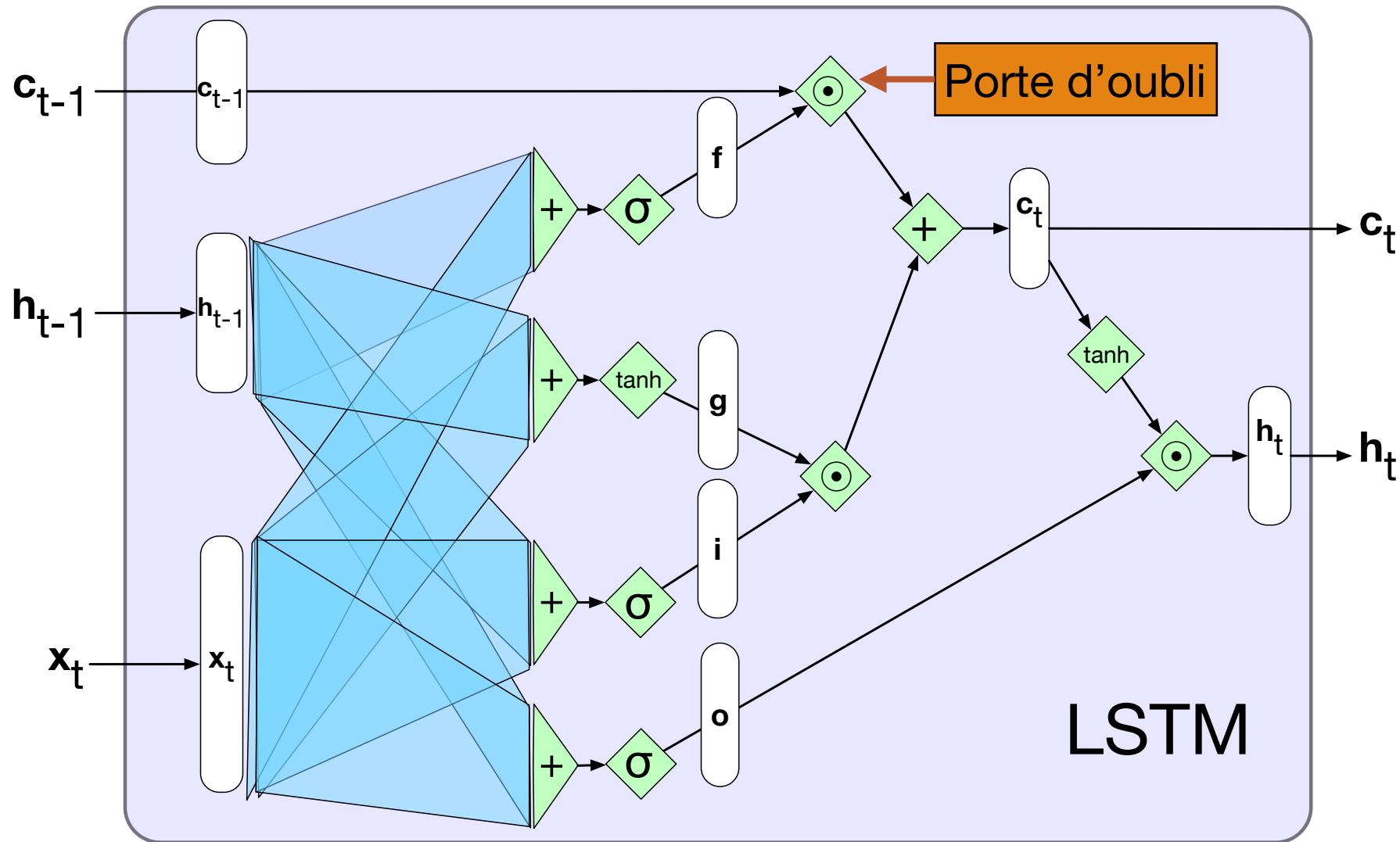


(c)
LSTM

Le LSTM



Le LSTM



Porte d'oubli

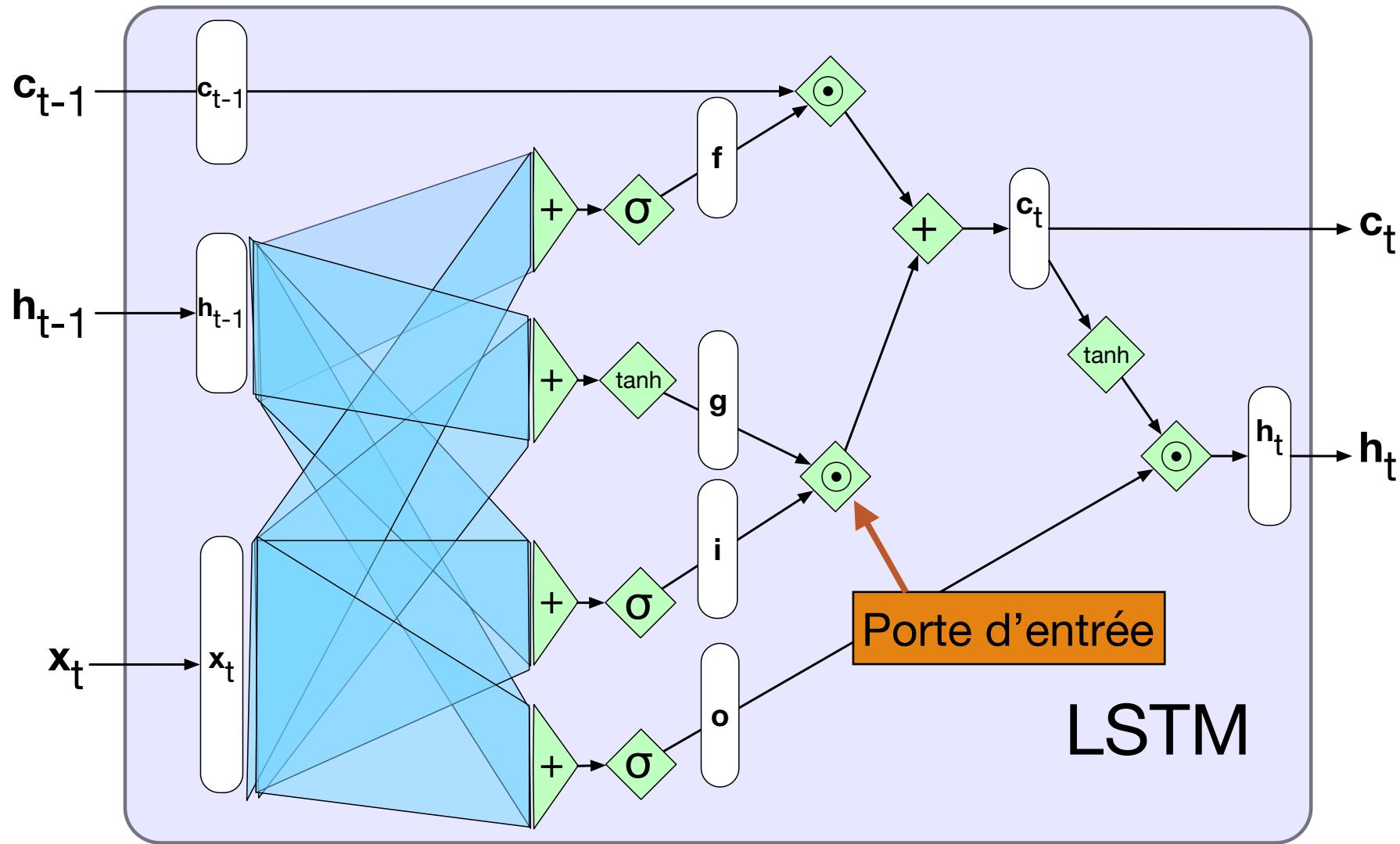
Son rôle : Supprimer les informations du contexte qui ne sont plus nécessaires.

Calcule la somme pondérée des états cachés précédents et de l'entrée actuelle et passe par l'activation sigmoïde.

Ce masque est ensuite multiplié par élément avec un vecteur de contexte pour supprimer les informations du contexte qui ne sont plus nécessaires.

$$\begin{aligned}\mathbf{f}_t &= \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t) \\ \mathbf{k}_t &= \mathbf{c}_{t-1} \odot \mathbf{f}_t\end{aligned}$$

Le LSTM



Porte d'entrée

Son rôle: Sélectionner des informations à ajouter au contexte actuel.

Calculez d'abord le passage régulier des informations (comme avec SRN) :

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$

Ensuite, sélectionnez de nouvelles informations à ajouter au contexte en multipliant par élément la sortie du sigmoïde avec la sortie de l'état actuel.

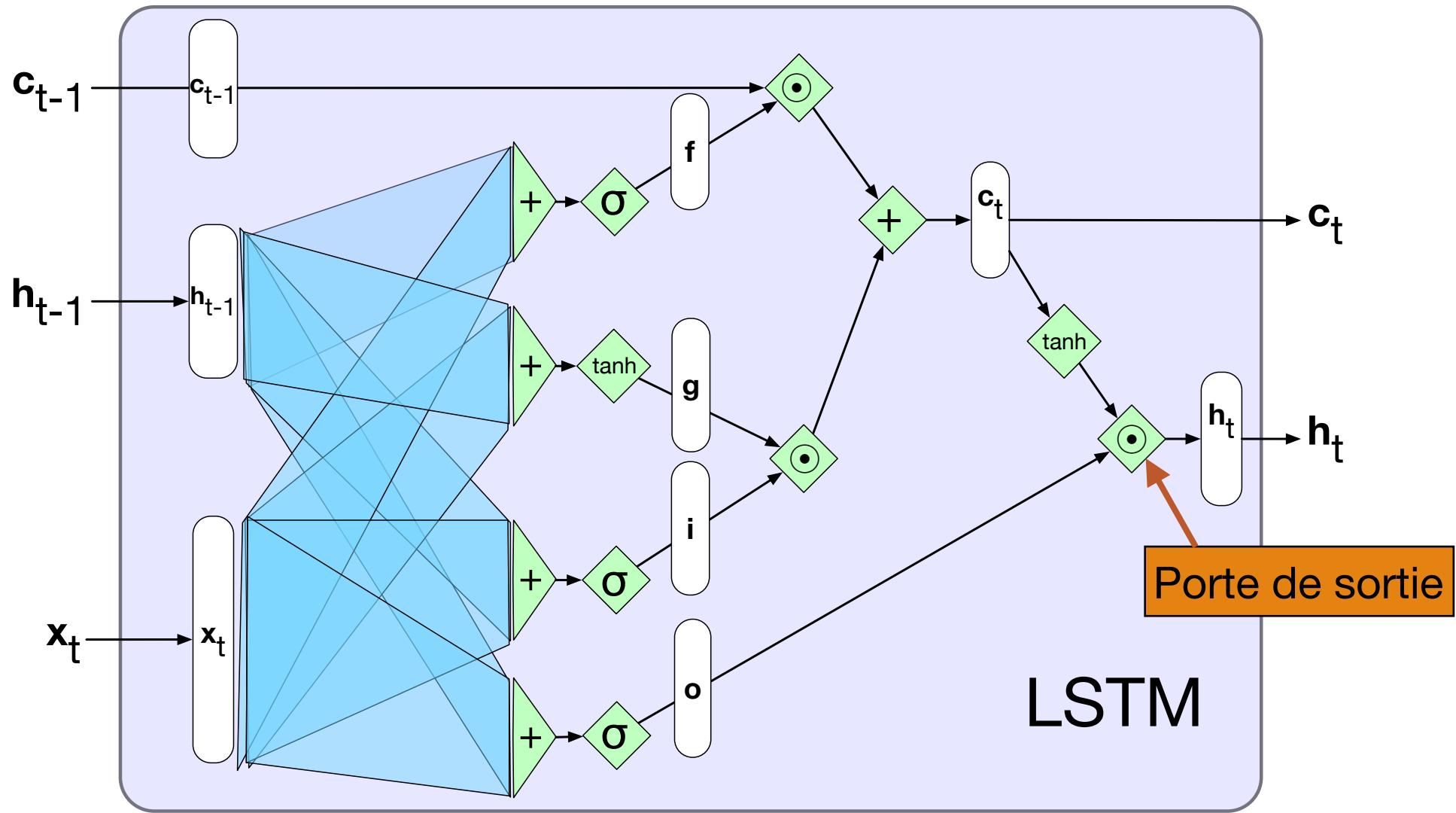
$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

Ajoutez ceci au vecteur de contexte modifié par la porte d'oubli pour obtenir notre nouveau vecteur de contexte.

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

Le LSTM



Porte de sortie

Son rôle: Décidez quelles informations sont nécessaires pour l'état caché actuel (par opposition aux informations qui doivent être conservées pour les décisions futures).

Multiplie par élément la sortie sigmoïde de l'état actuel par le nouveau contexte.

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM pour les données du langage naturel

Parce que le langage naturel a de nombreuses dépendances à longue distance, les LSTM sont le réseau neuronal récurrent de choix pour la modélisation du langage.



[Pause - 15 minutes]

Travailler avec des RNNs

Mettez-vous en équipe!

Ouvrez les exercices/week 6 dans votre dossier de cours et commencez à écrire/exécuter du code !