

Réseaux de neurones récurrents

- Savoir appliquer les réseaux de neurones aux séquences
- Comprendre les contraintes et avantages des réseaux récurrents

Traiter une séquence d'inputs :

- Séquence de mots (phrase)
- Séquence de caractères (mot)
- Séquence d'actions sur une page web
- Séquence de traces dans un log
- Séquence de mesures en géologie / océanographie / météorologie
- ...

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

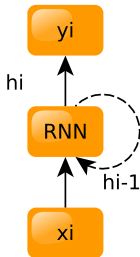
Idée

- 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 ?**

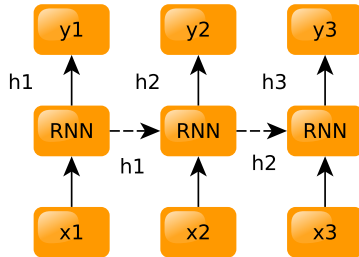
- Taille d'input variable
- utilisation d'un même réseau pour chaque élément de l'input, séquentiellement



Pour l'élément 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.
- Connexions horizontales (calcul pas à pas).

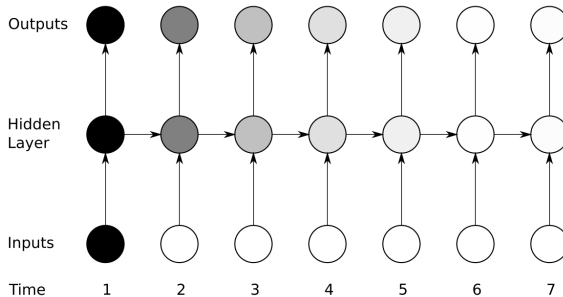
Adaptation de la rétropropagation des gradients

- 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 chaînage).

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

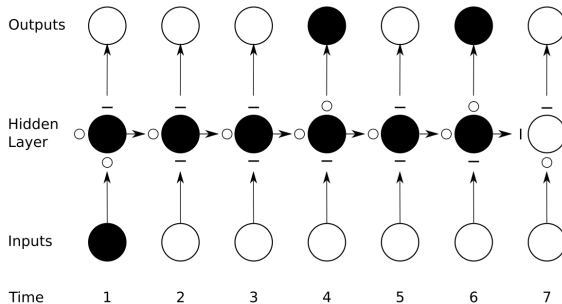
Problème principal

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



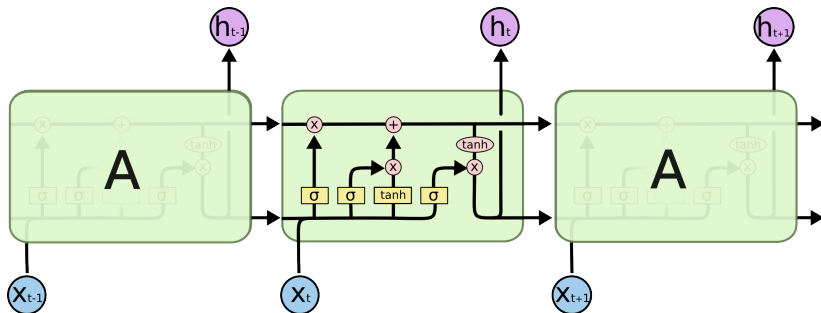
Solution : LSTM

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



Approche appelée Long Short-Term Memory Networks [?].

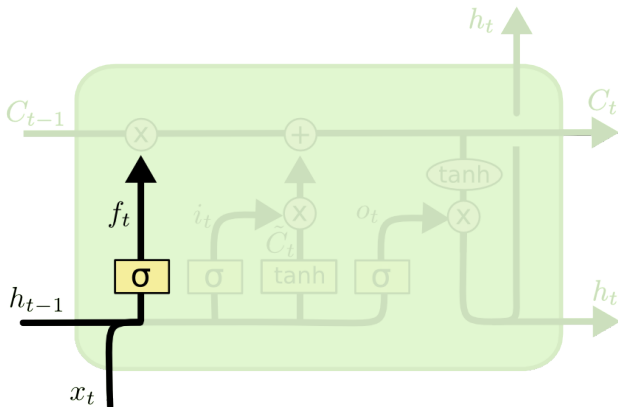
Cellule LSTM



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

LSTM : étape 1/4

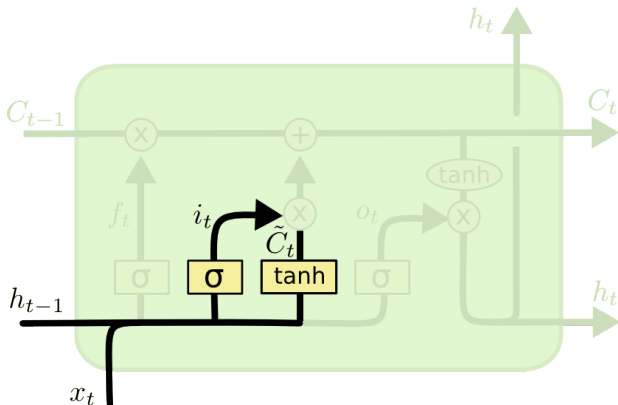
Conserver ou non les informations en mémoire [?] :



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

LSTM : étape 2/4

Prendre en compte ou non l'input :

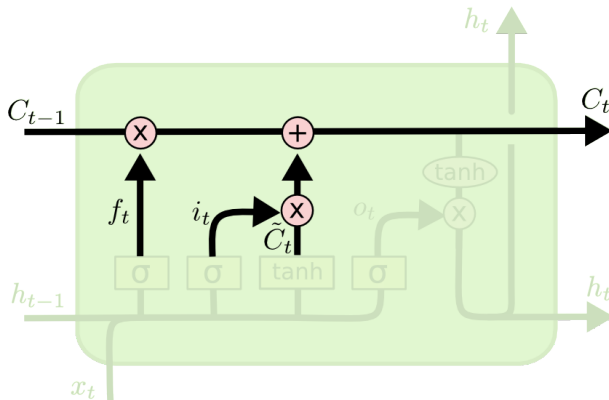


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

LSTM : étape 3/4

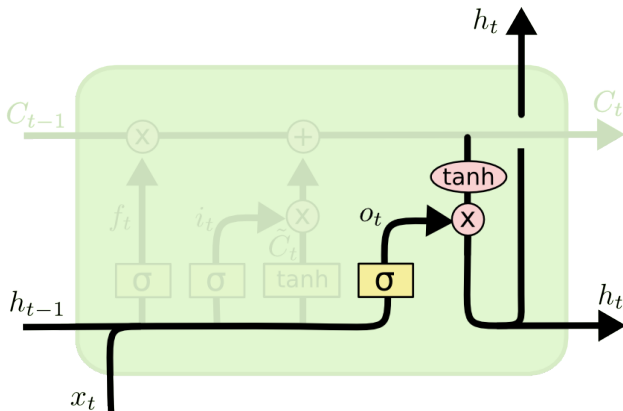
Mettre à jour l'état caché :



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

LSTM : étape 4/4

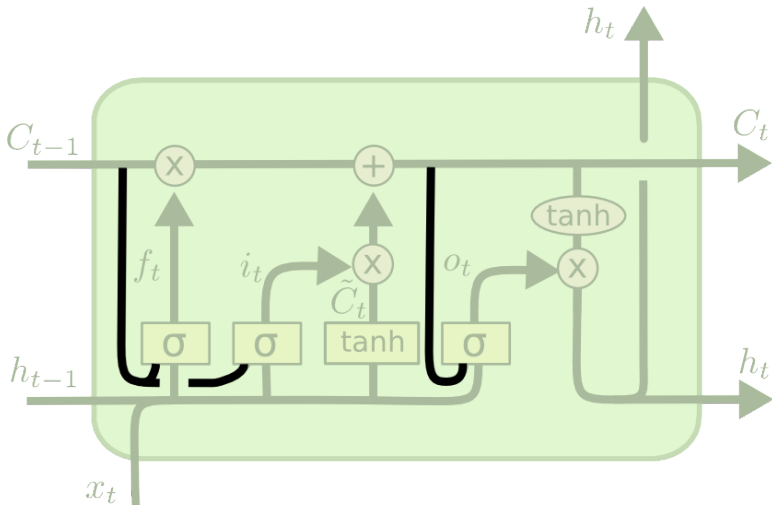
Contrôler si l'on produit ou non une valeur de sortie :



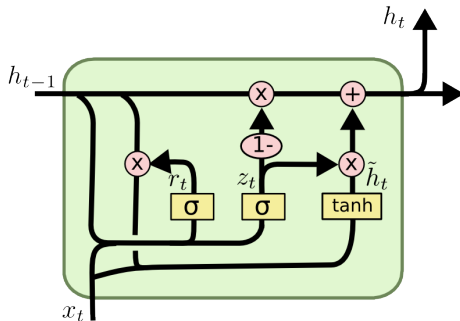
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

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

Extension : judas



Variante : GRU



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

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

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

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 deep learning

