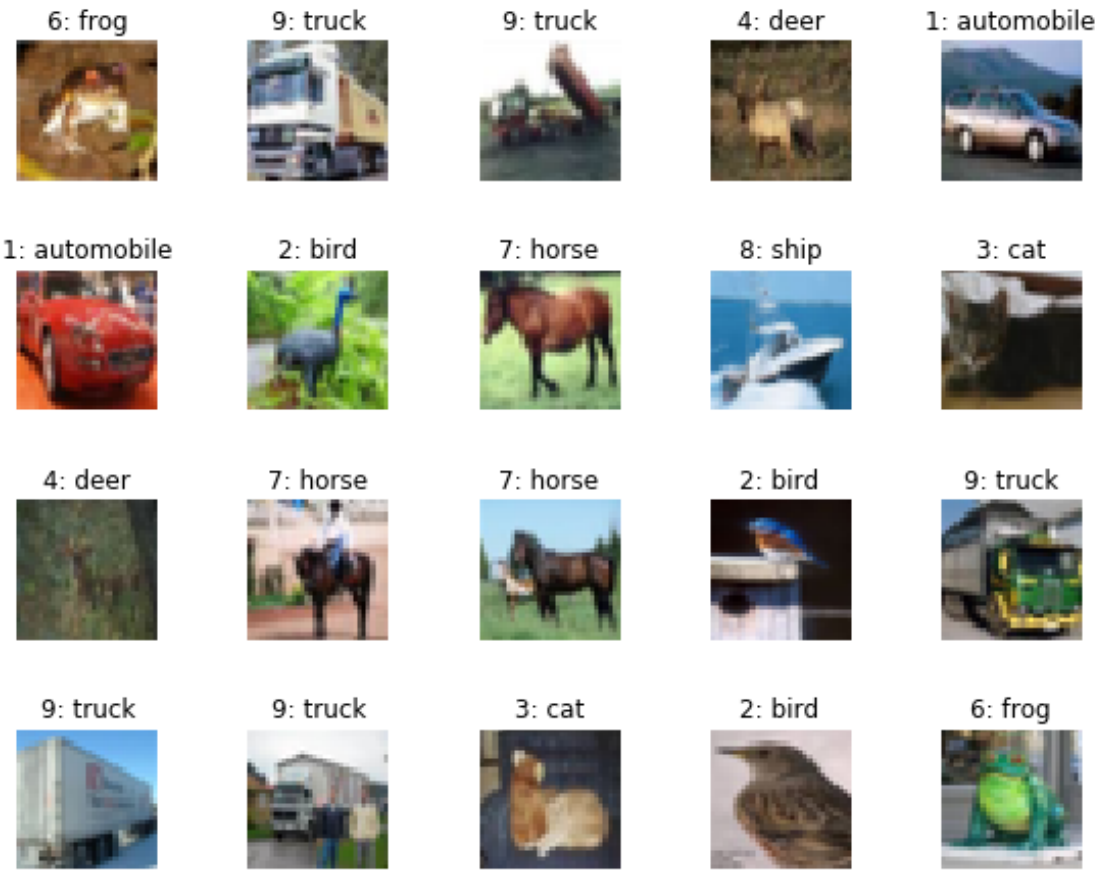
Réseaux de neurones récurrents

ENS'IA

Ensimag 2019-2020

11 mars 2020

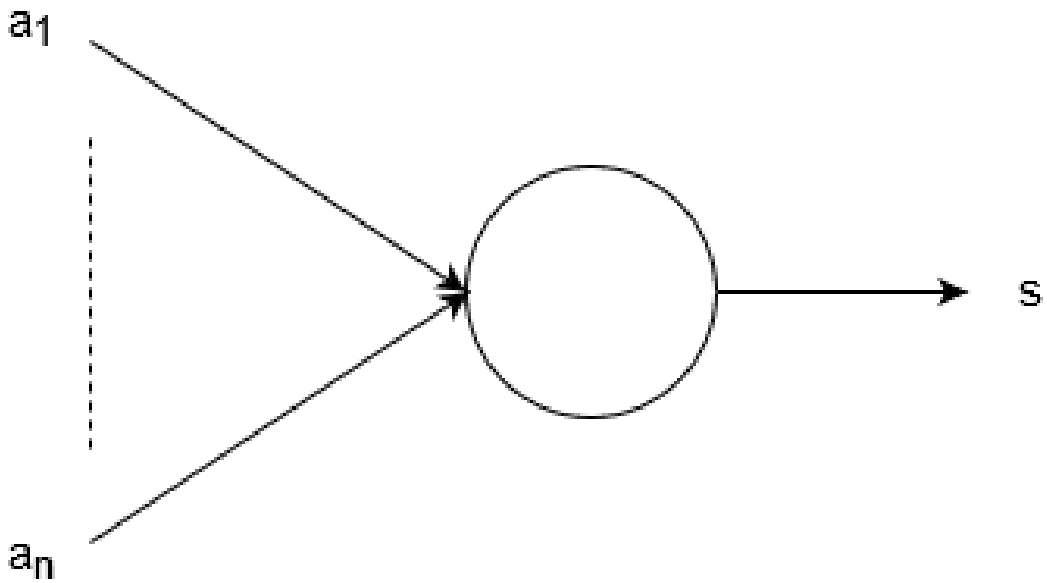
Rappel : objectif



Classification
$\rightarrow$ Besoin d'une fonction f tel que $f(x) = y$

Comment approximer cette fonction ? $\rightarrow$ Réseau de neurones

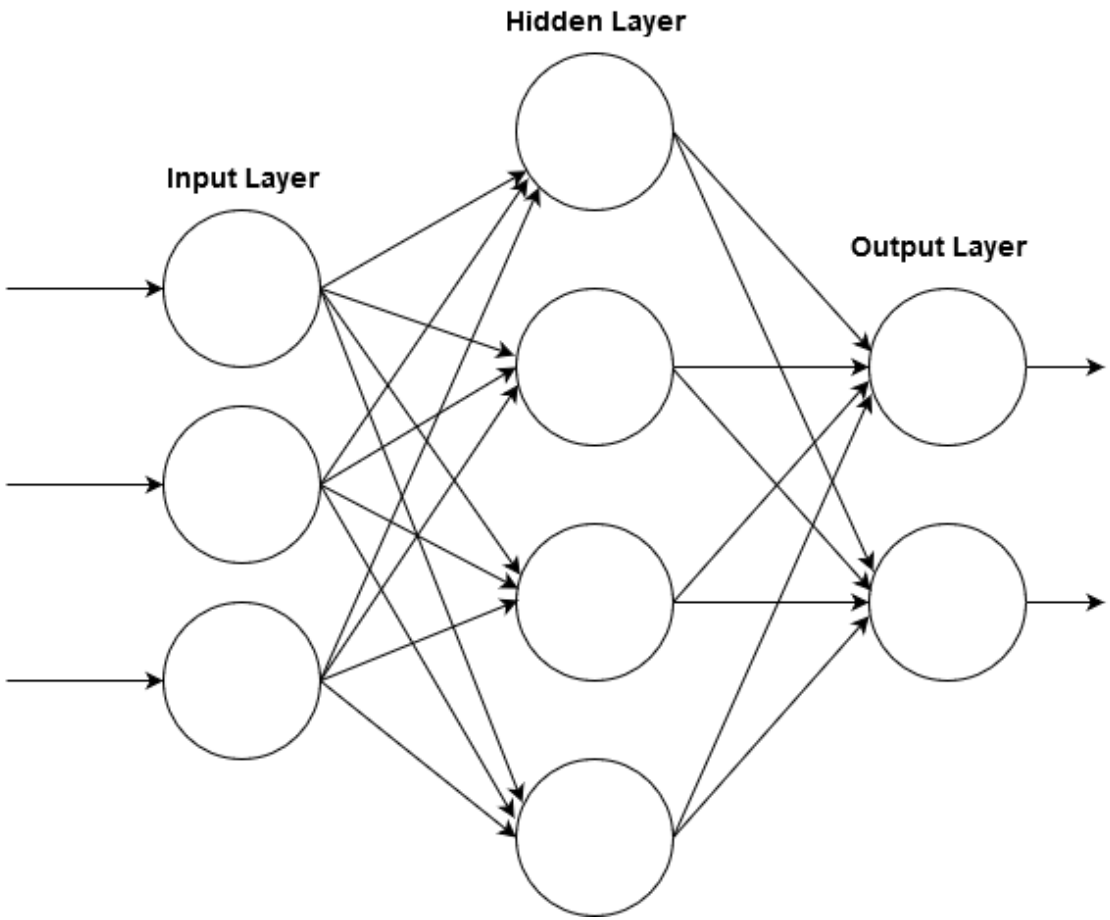
Rappel : Sigmoid neuron



$a_1, ..., a_n \in [0, 1]$

$s = \sigma(\sum_{i=0}^{n} a_i * w_i + b)$ où $\sigma(x) = \frac{1}{1+e^{-x}}$

Rappel : Réseau de neurones



→ Succession de couches de neurones

Rappel : Aspects techniques

Fonctions d'activation :

- sigmoid $\rightarrow a(x) = \frac{1}{1+e^{-x}}$
- ReLU $\rightarrow a(x) = \max(0, x)$
- tanh $\rightarrow a(x) = \tanh x$

Algorithmes d'optimisation :

- Stochastic gradient descent (SGD)
- RMSProp
- Adam

Rappel : en pratique

→ Utilisation de `Keras`

```
model = Sequential()
model.add(Dense(64, activation = "relu", input_dim=128))
model.add(Dense(64, activation = "relu"))
model.add(Dense(10, activation = "softmax"))
model.summary()
model.compile(loss = loss, optimizer = optimizer, metrics =
[metrics])
```

Rappel : Réseau de neurones

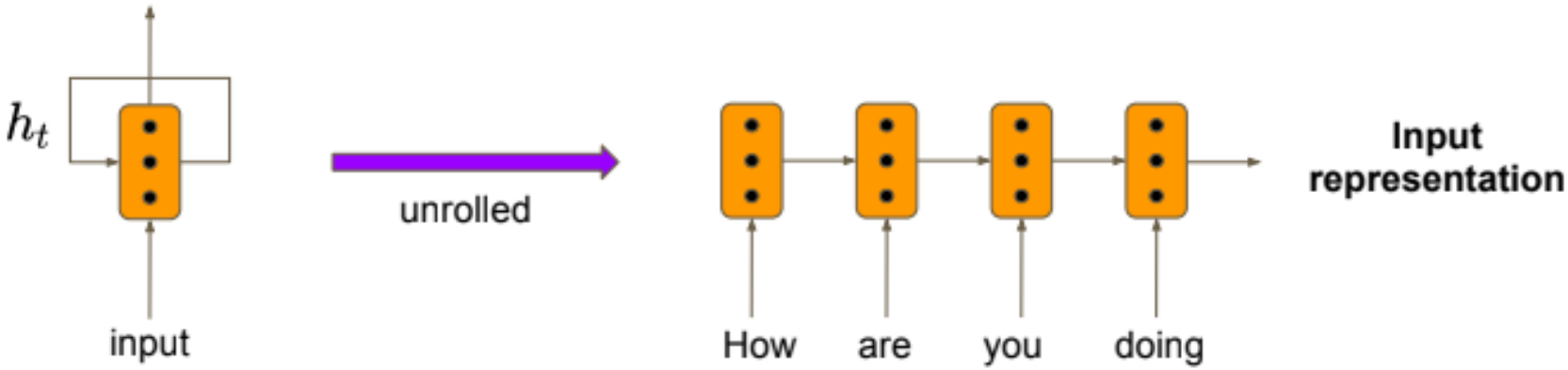
→ Problème : données liées dans le temps ou *séquences*

Vanilla RNN

$\rightarrow$ Idée : conserver de l'information d'une donnée à une autre

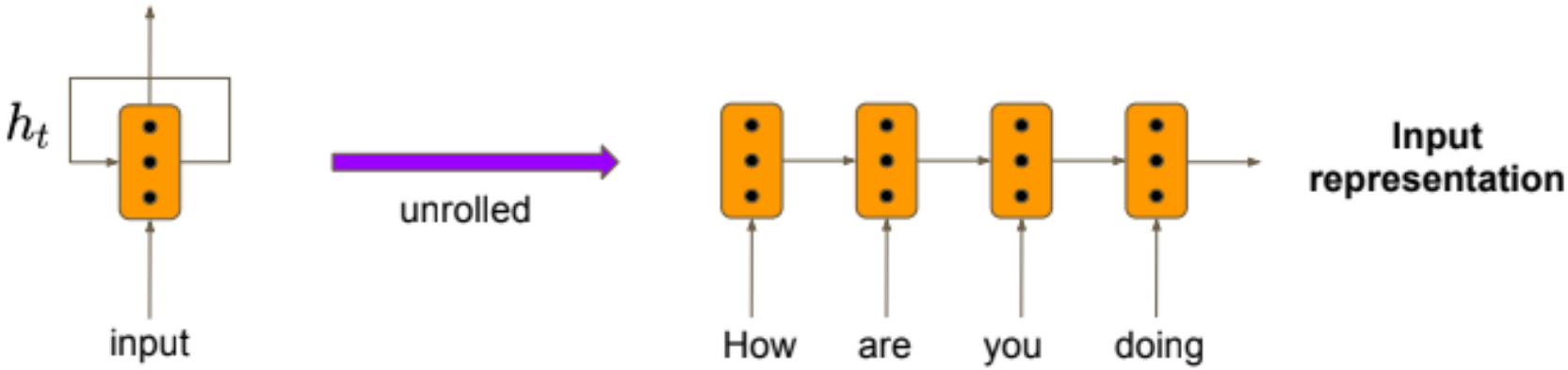
Vanilla RNN

→ Idée : conserver de l'information d'une donnée à une autre



Vanilla RNN

→ Idée : conserver de l'information d'une donnée à une autre



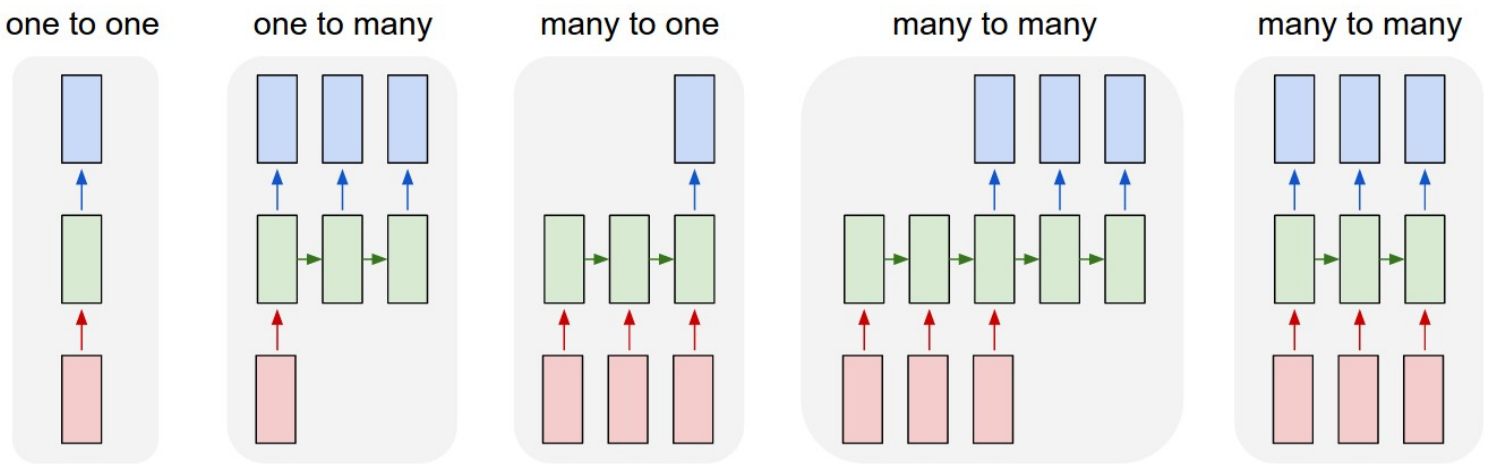
→ En pratique : mis à jour et transfert d'un état interne

$$h_t = \tanh\left(W_{hh} h_{t-1} + W_{hx} x_t\right)$$
$$y_t = W_{hy} h_t$$

3 matrices à apprendre : W_{hh}, W_{hx} et W_{hy}

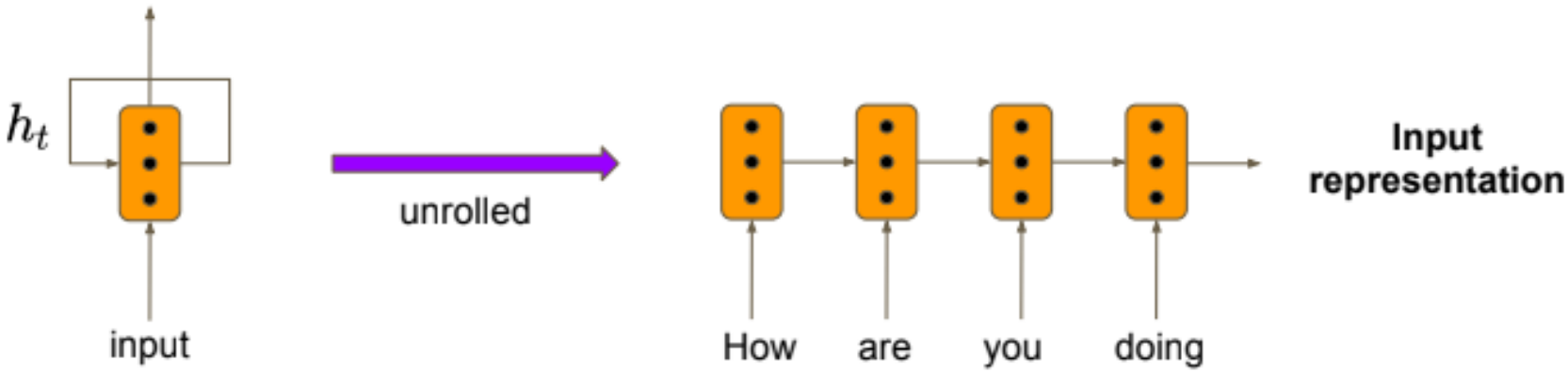
Vanilla RNN

Différentes architectures pour différents modèles



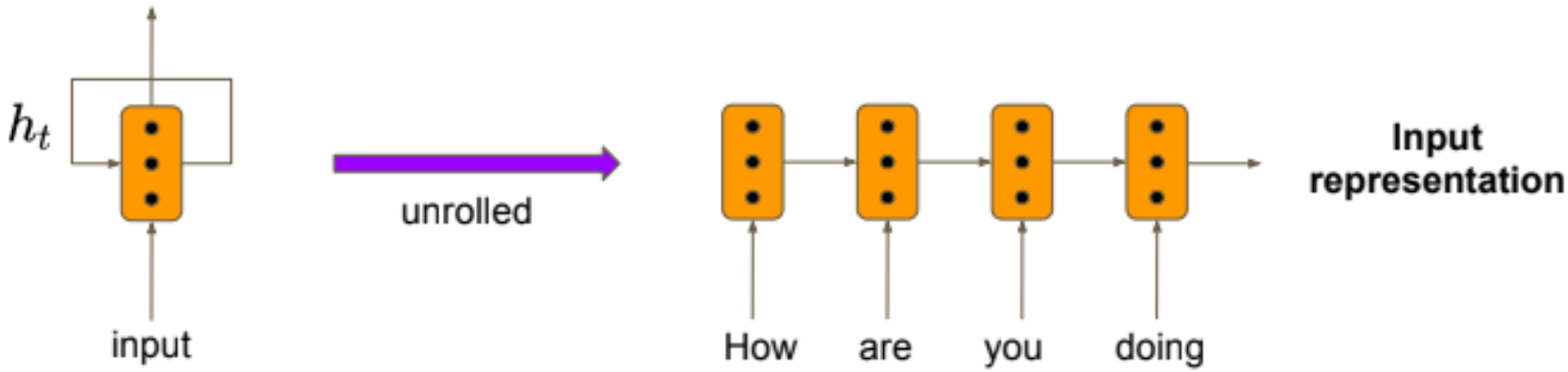
Vanilla RNN

Mais malheuresement quelques soucis



Vanilla RNN

Mais malheuresement quelques soucis

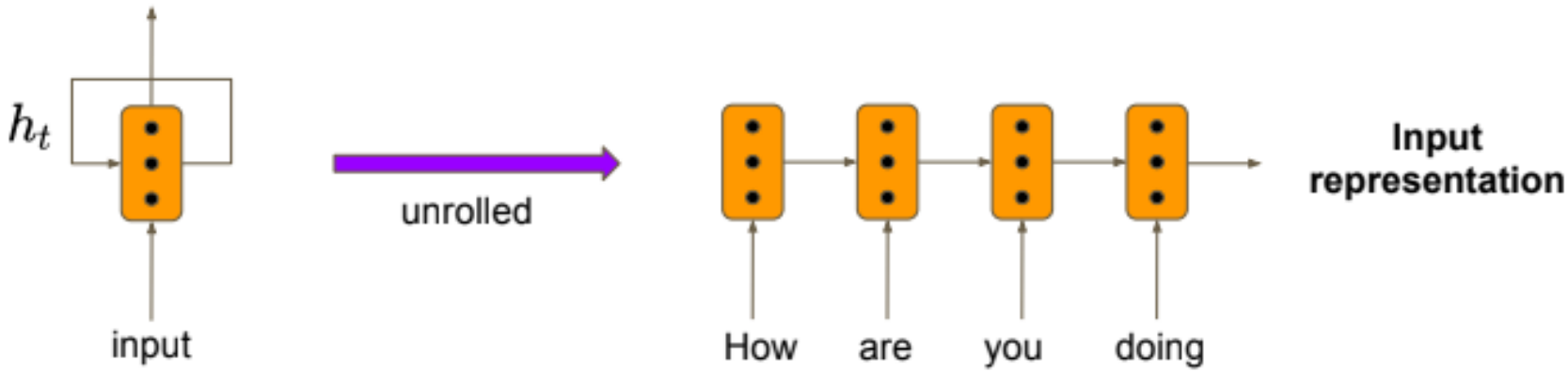


Problèmes :

- Disparition de l'information au fur et à mesure par "compression"

Vanilla RNN

Mais malheuresement quelques soucis

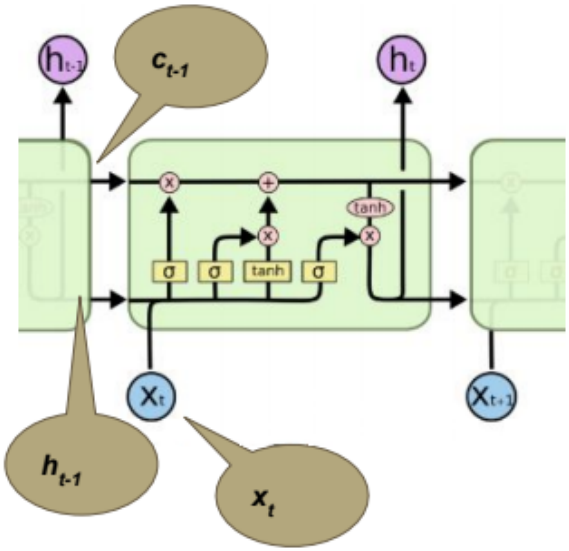


Problèmes :

- Disparition de l'information au fur et à mesure par "compression"
- *Vanishing Gradient* lors de l'apprentissage : apprentissage peu efficace...

LSTM

→ Solution : *Long Short Term Memory* (1997)

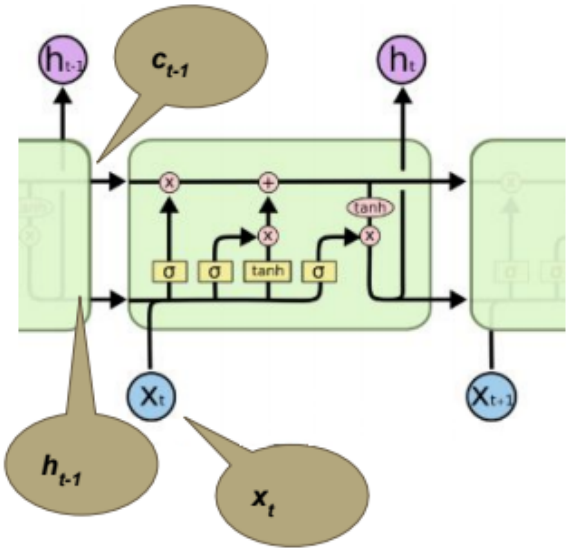


update
forget
output

$$
\begin{aligned}
\tilde{c}_t &= tanh(W_c * [h_{t-1}, x_t]) \\
u_t &= \sigma(W_t * [h_{t-1}, x_t]) \\
f_t &= \sigma(W_f * [h_{t-1}, x_t]) \\
o_t &= \sigma(W_o * [h_{t-1}, x_t]) \\
c_t &= u_t * \tilde{c}_t + f_t * c_{t-1} \\
h_t &= o_t * tanh(c_t)
\end{aligned}
$$

LSTM

→ Solution : *Long Short Term Memory* (1997)



update
forget
output

$$
\begin{aligned}
\tilde{c}_t &= tanh(W_c * [h_{t-1}, x_t]) \\
u_t &= \sigma(W_t * [h_{t-1}, x_t]) \\
f_t &= \sigma(W_f * [h_{t-1}, x_t]) \\
o_t &= \sigma(W_o * [h_{t-1}, x_t]) \\
c_t &= u_t * \tilde{c}_t + f_t * c_{t-1} \\
h_t &= o_t * tanh(c_t)
\end{aligned}
$$

4 parties :

- Forget gate : ce que l'on oublie
- "Input" gate : ce que l'on souhaite avoir
- Update gate : ce que l'on rajoute
- Output gate : ce que l'on sort

RNN - Résumé

Structure :

- Transformation des données : séquences
- Eventuellement word embedding : conversion mot en vecteur
- Recurrent layers
- Réseau classique / Dense layer

Sources

- cs231 Stanford
- http ://colah.github.io/
- http ://karpathy.github.io/