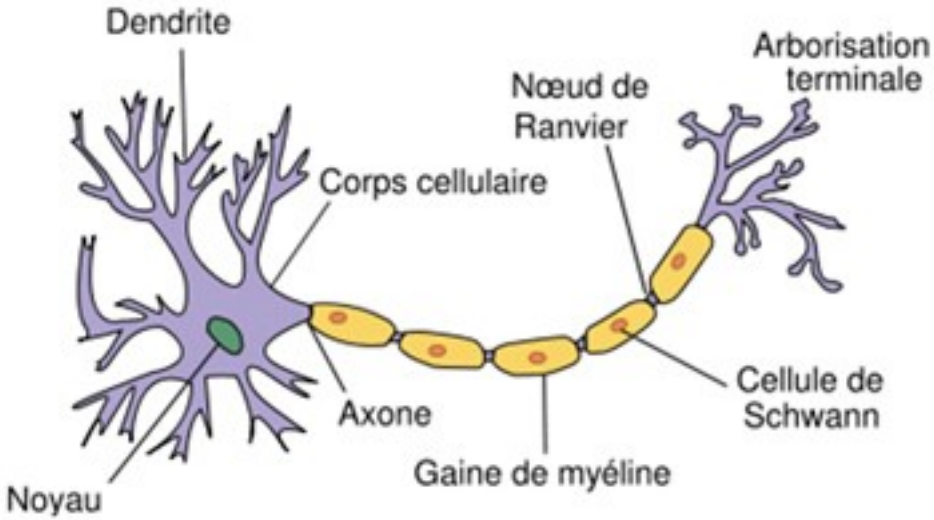
Réseaux de neurones récurrents artificiels

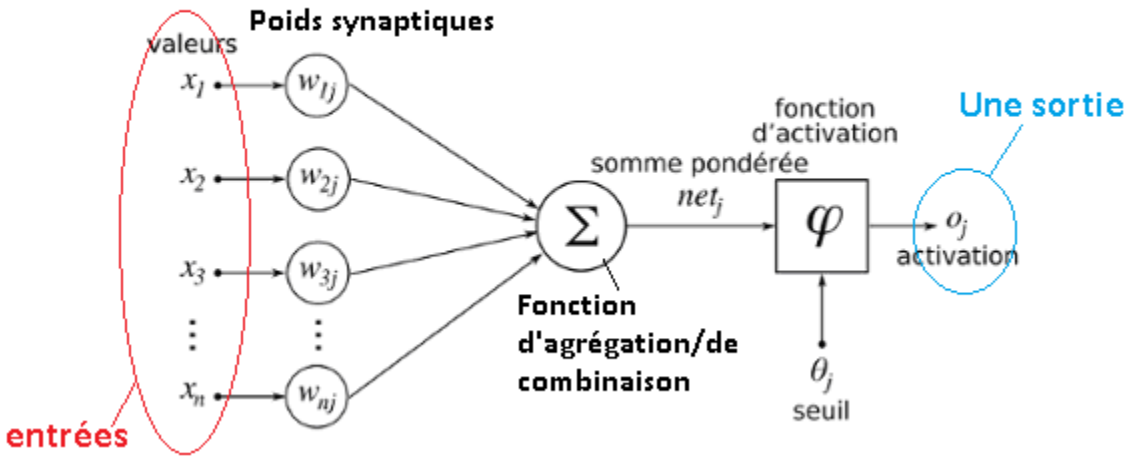
Paul Bernard (Paul.Bernard@inria.fr)

Equipe Mnemosyne - Inria/IMN/LaBRI

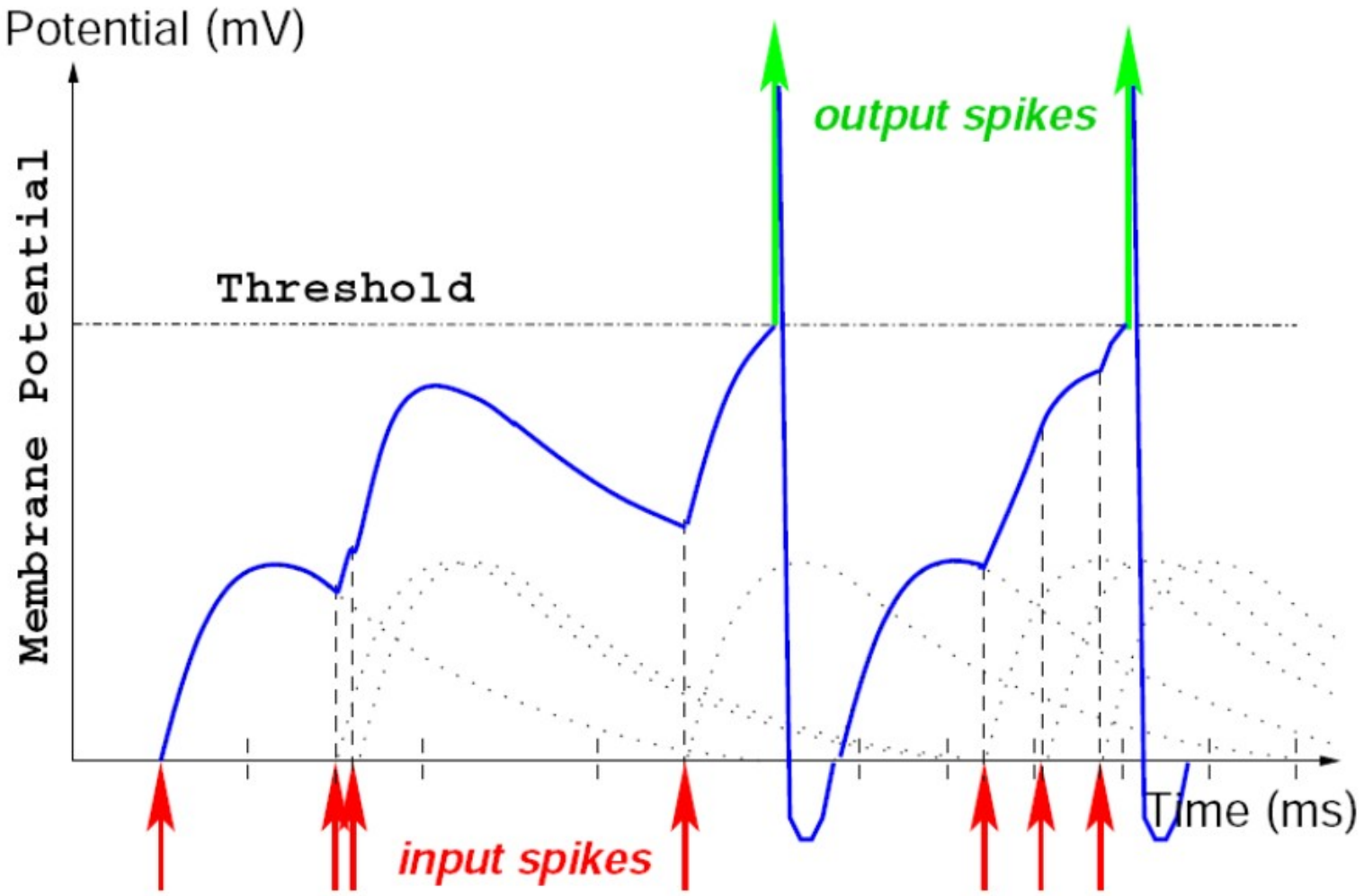
Un neurone artificiel : abstraction d'un neurone biologique

- Entrées:
  - activités provenant d'autres neurones
- Corps cell.:
  - combinaison des entrées
- Sortie:
  - activité du neurone
  - Taux de décharge moyen





Integrate and Fire model



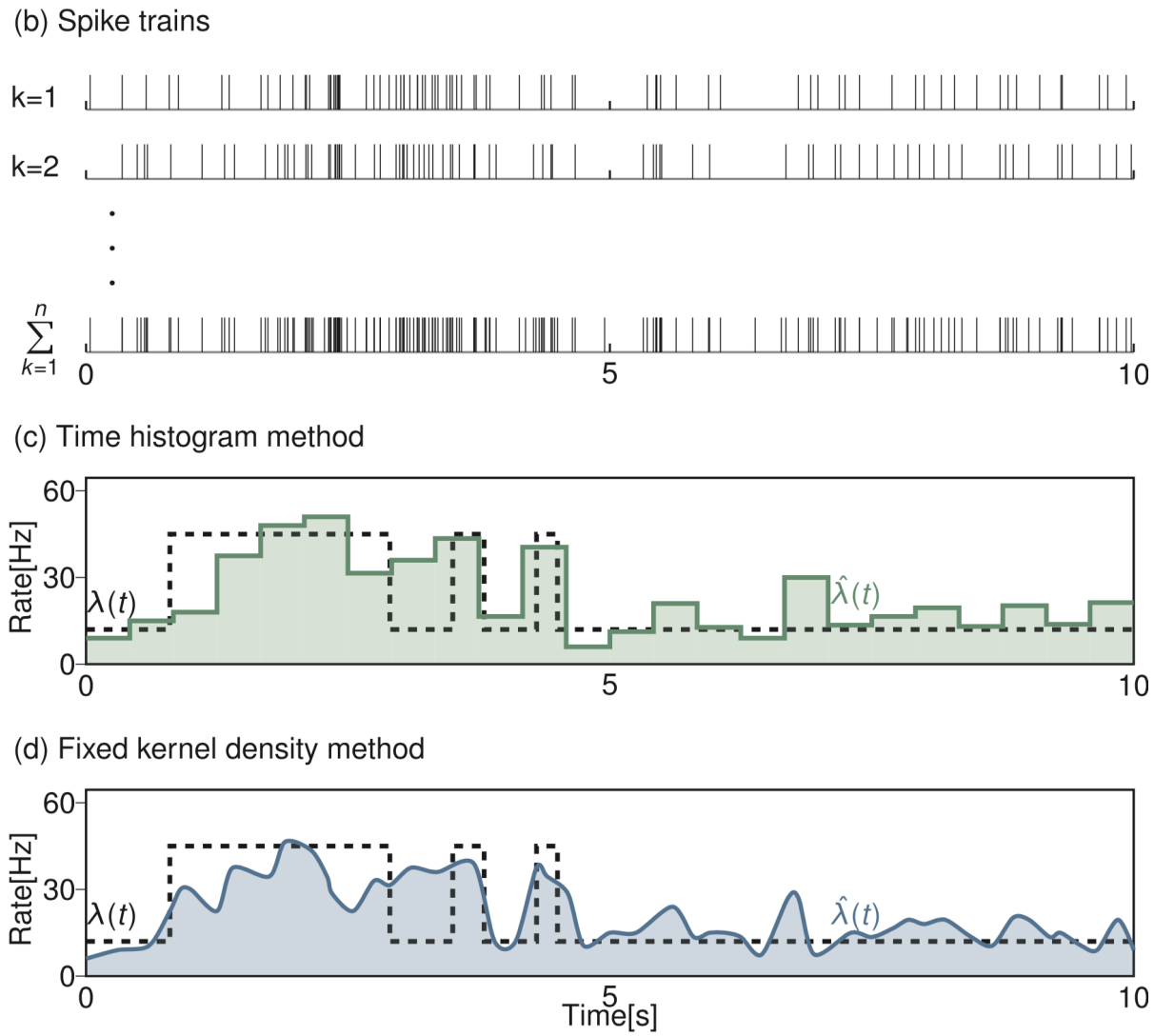
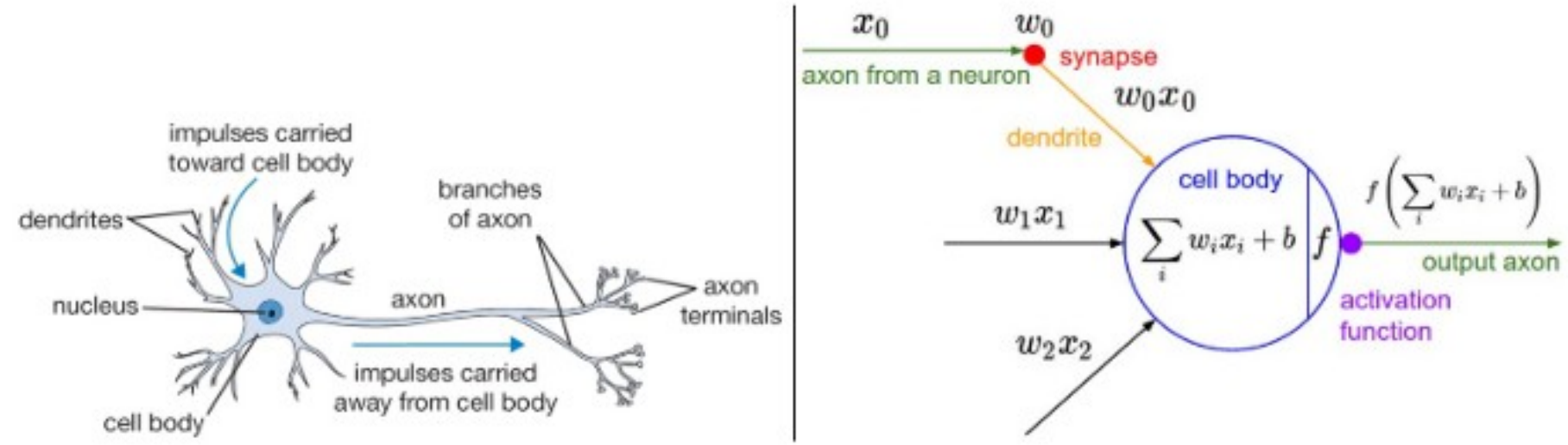


Fig. 1. Neuronal spiking activity and firing rate estimation. (a) Spikes occur at times S_i; the corresponding interspike intervals (ISIs) are denoted as X_i, and the associated counting process up to time instant t is $N(0, t)$. (b) Examples of spike trains simulated under identical statistical conditions. For most of the firing rate estimation methods, individual spike trains are superimposed but few methods, like Frequencygram (Section 4.1) and Bayesian Adaptive Kernel Smoothing (Section 4.8), work with individual recordings as well. (c) Estimation of underlying firing rate (green) from the spike train data, using the time histogram method, compared with the true underlying firing rate (dashed). (d) Firing rate estimation (blue) using the kernel smoothing method with Gaussian kernel, against the true underlying firing rate (dashed). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Tomar 2019
doi.org/10.1016/j.biosystems.2019.103980

Calcul fait dans un neurone articiel à taux de décharge moyen

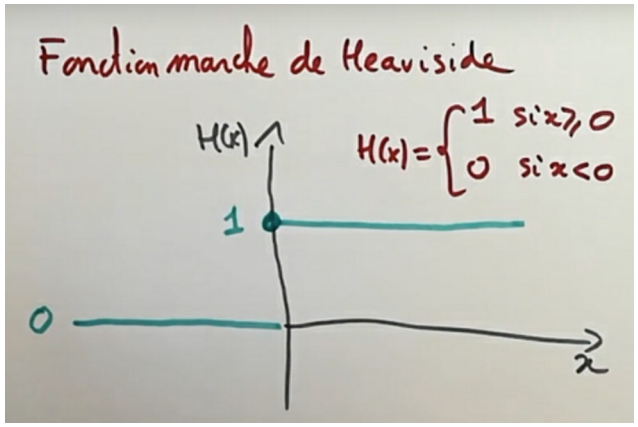


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

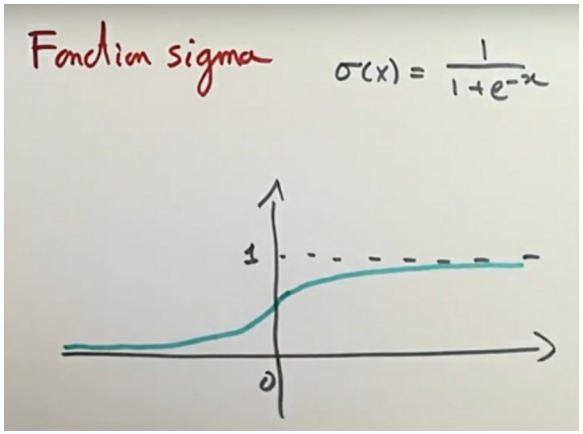
Composants du neurone:

- chaque **entrée** provient de la sortie d'un autre neurone
  - **pondérée** par un **poids w_i** (w comme *weight* en Anglais)
- le calcul interne (= **potentiel de membrane**) est la somme de ces entrées pondérées
  - + **un biais** (qui peut être positif, négatif ou nul)
  - si le biais est nul, cela correspond à l'absence de biais
- la **fonction d'activation *f*** qui détermine l'activité de la sortie du neurone
  - en fonction du potentiel de membrane : sortie = *f*(potentiel de membrane)

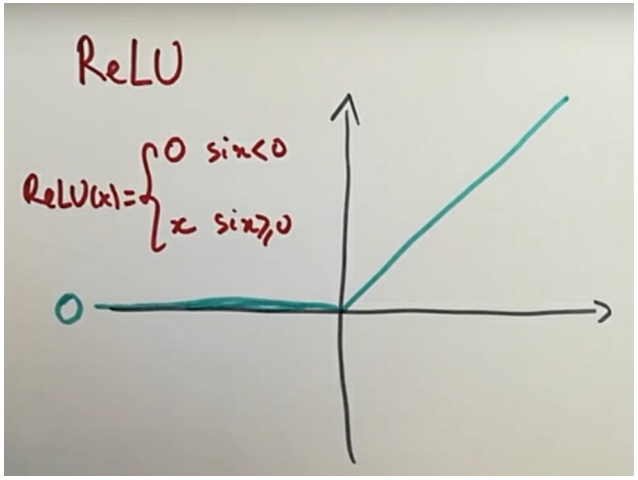
Fonctions d'activation (= fonctions de transfert)



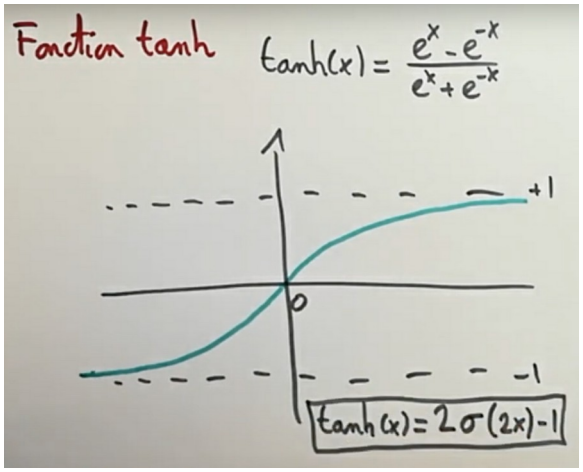
fonction **seuil**



fonction **sigmoïde**



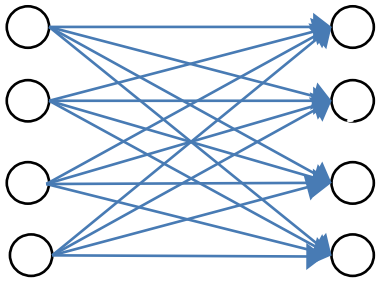
fonction **Re**ctified **L**inear **U**nit (**ReLU**)



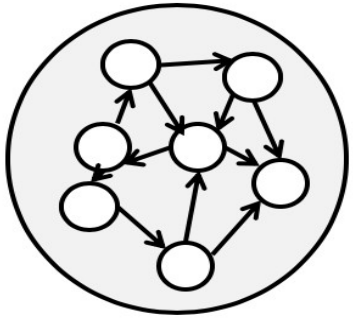
fonction tangente hyperbolique (**tanh**)

[Deepmath] 1.4. Fonctions d'activation https://www.youtube.com/watch?v=q0ZZ695vjkY (de 0:00 à 9:00)

2 types de réseaux



- Réseau de neurones en couches
  - Connexions entre neurones de couches différentes
  - Pas de mémoire

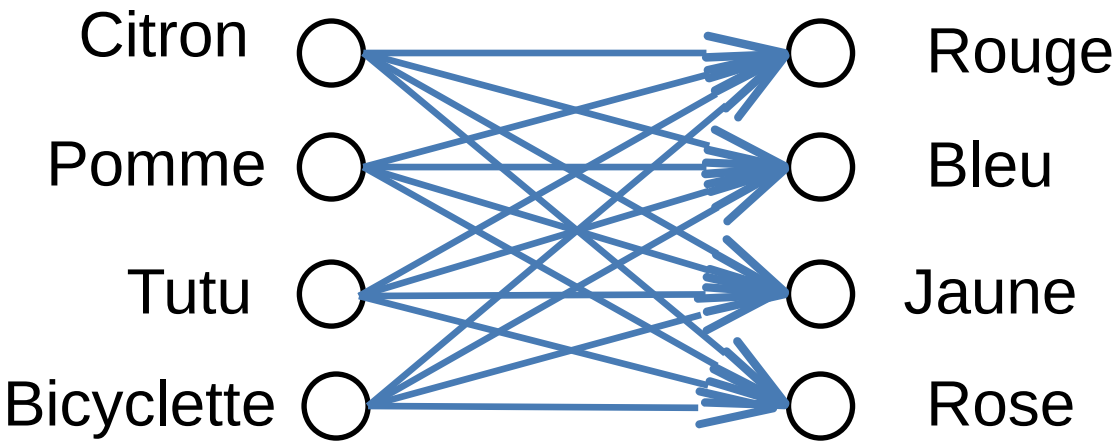


- Réseau de neurones récurrent
  - Mémoire dynamique
  - Représentation du contexte temporel

Neurone

Axone (connexion entre deux neurones)

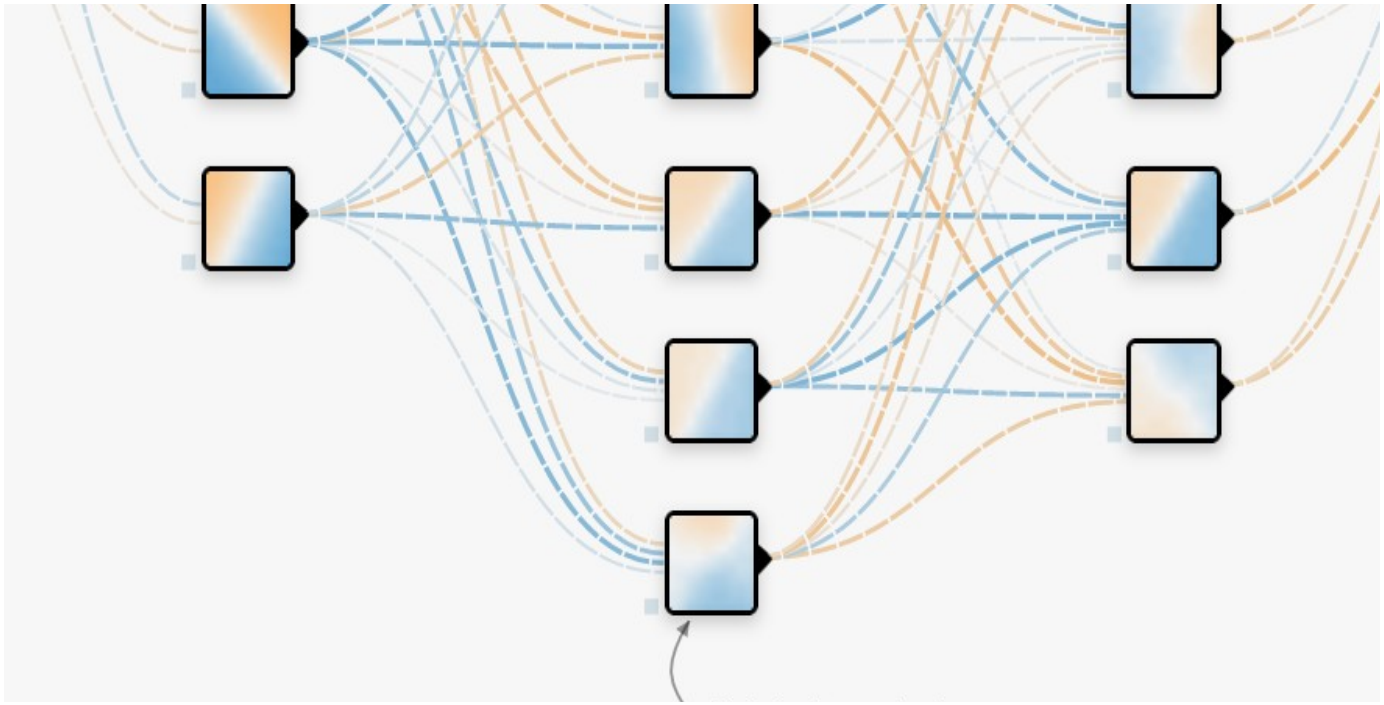
Un réseau de neurones en couches



- Loi de Hebb (1949)
  - lorsque 2 neurones sont actifs en même temps leurs connexions sont renforcés
    - « Neurons that fire together, wire together »

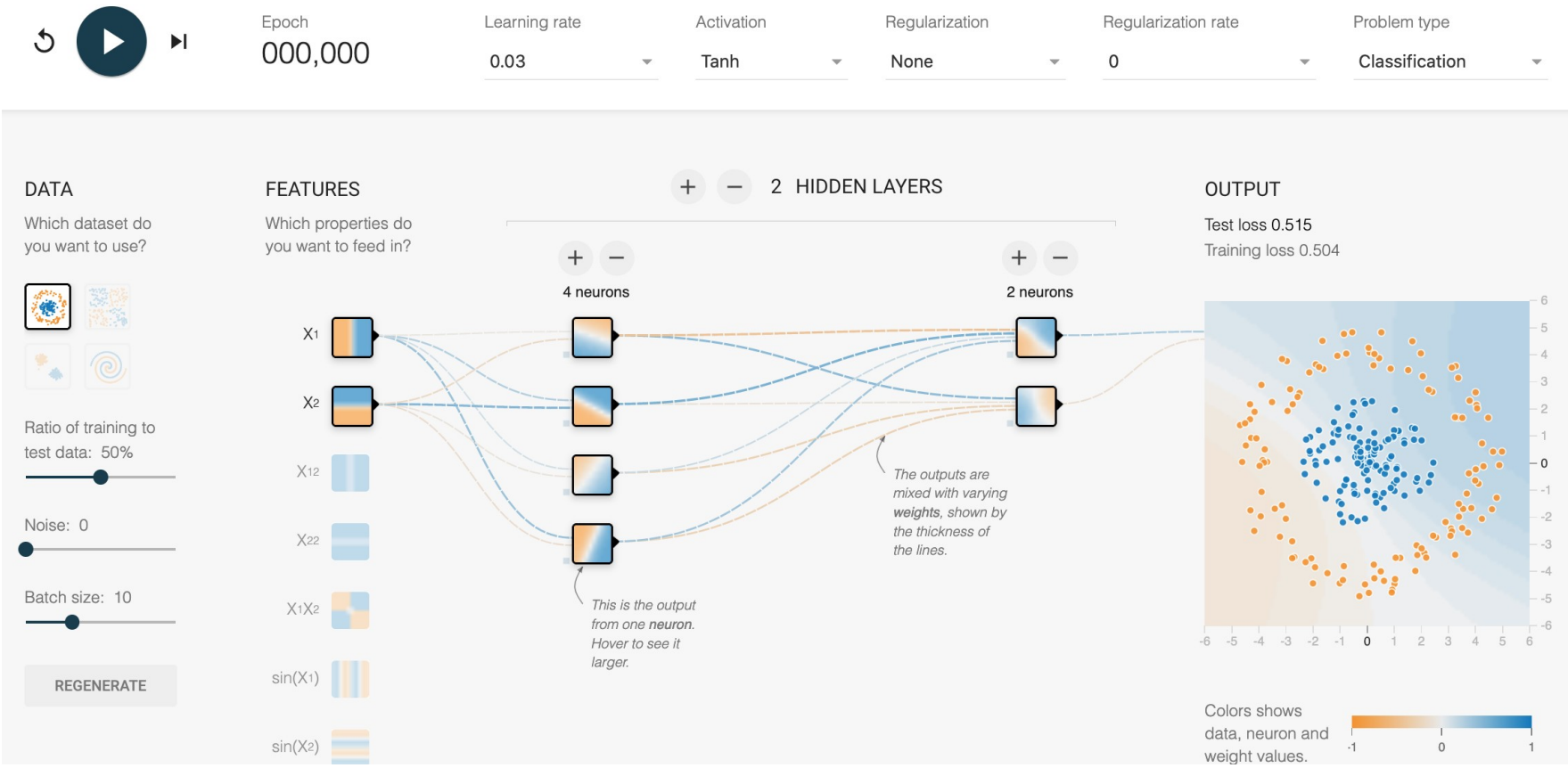
Un réseau de neurones en couches

- Avec l'algorithme de rétro-propagation du gradient, on peut apprendre avec un empilement de couches



playground.tensorflow.org

Jouer avec les neurones …



- http://playground.tensorflow.org
- http://binaire.blog.lemonde.fr/2017/10/20/jouez-avec-les-neurones-de-la-machine/

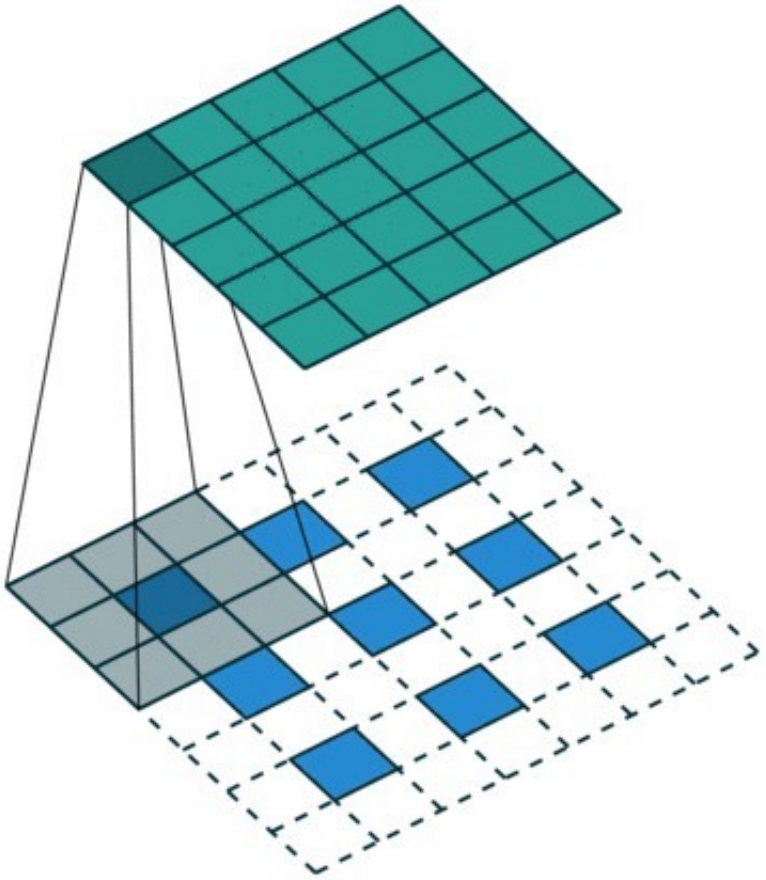
Algorithme le plus utilisé pour apprendre dans un réseau de neurones: la **Rétro-propagation du Gradient** (de l'erreur)

- En Anglais:
  - "Gradient Back-propagation"
  - ou "back-prop" pour les intimes
- Vidéos d'explications de @Science4All
  - descente du gradient stochastique (1'48 à 6'10)
    - https://www.youtube.com/watch?v=Q9-vDFvDdfg
  - rétropropagation du gradient (0'00 à 9'57)
    - https://www.youtube.com/watch?v=OgSA7liZMXI

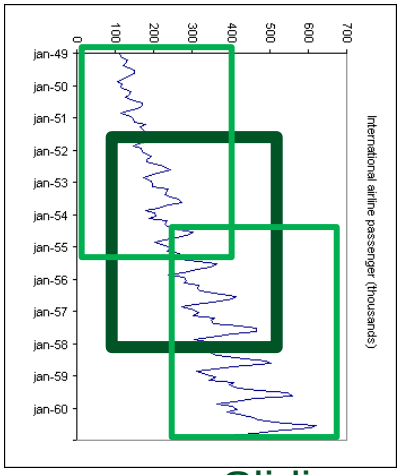
“Jouer avec la descente de gradient …”

- Comment marche la convergence améliorée
  - de l’algorithme d’apprentissage (“back-propagation”)
  - avec du “momentum” (inertie)
  - http://www.distill.pub/2017/momentum/
  - Goh, "Why Momentum Really Works", Distill, 2017. http://doi.org/10.23915/distill.00006

Rappels Convolution 2D



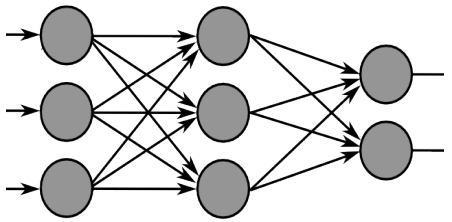
Fenêtre glissante traiter des series temporelles



Time

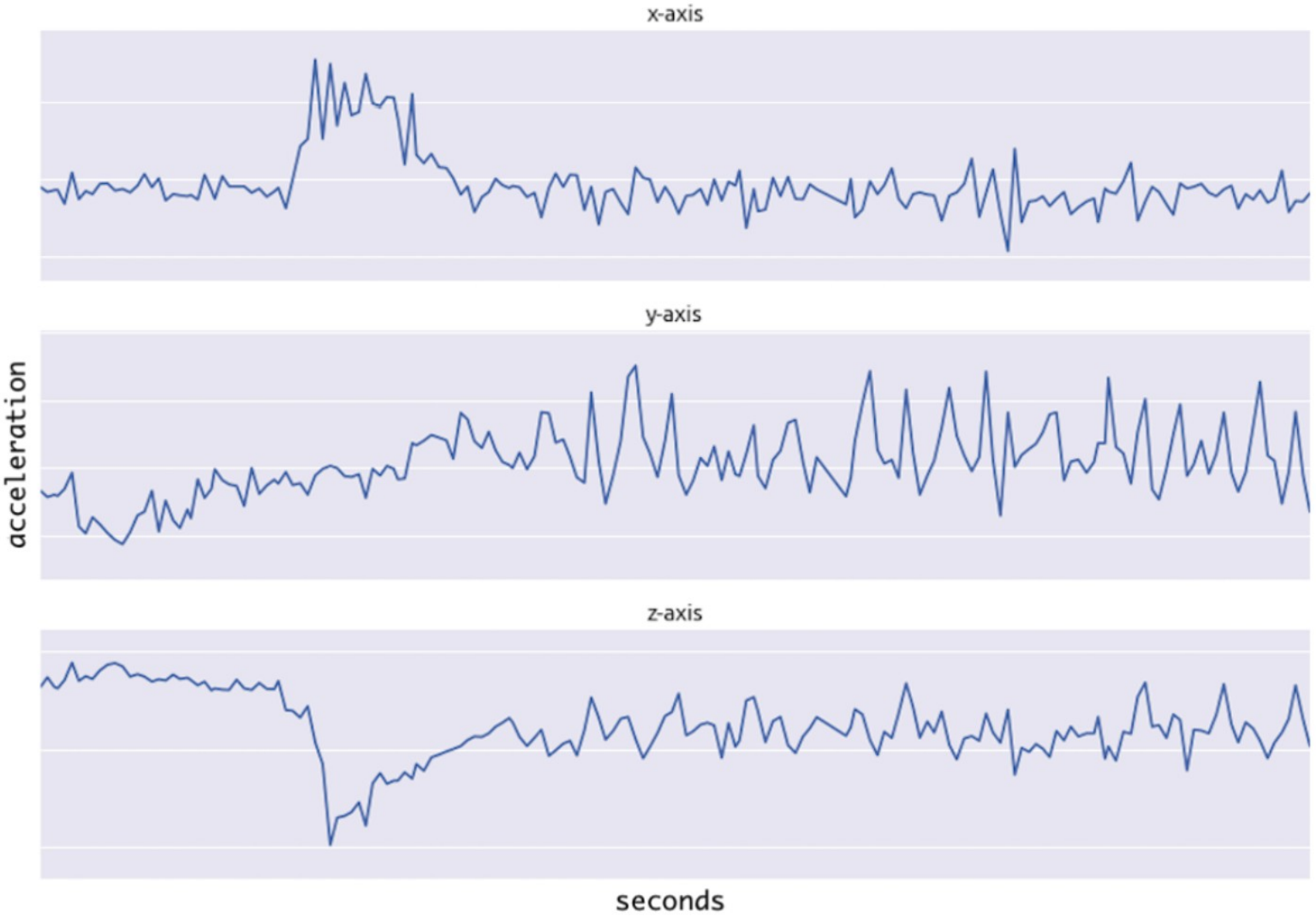
Sliding window

Processing



Feed Forward Neural Net.

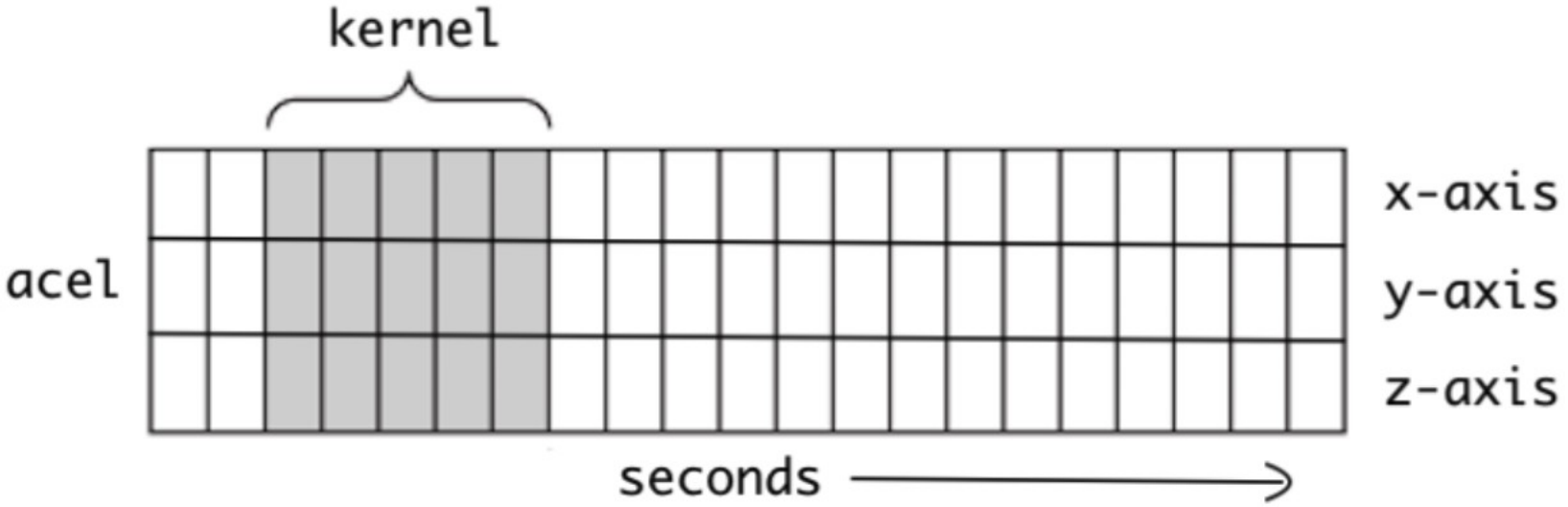
Convolution 1D pour les séries temporelles



Time series data from an accelerometer

Convolution 1D pour les séries temporelles

- On peut utiliser une convolution 1D de la même façon qu'une fenêtre glissante



Kernel sliding over accelerometer data

Convolution 1D pour les séries temporelles

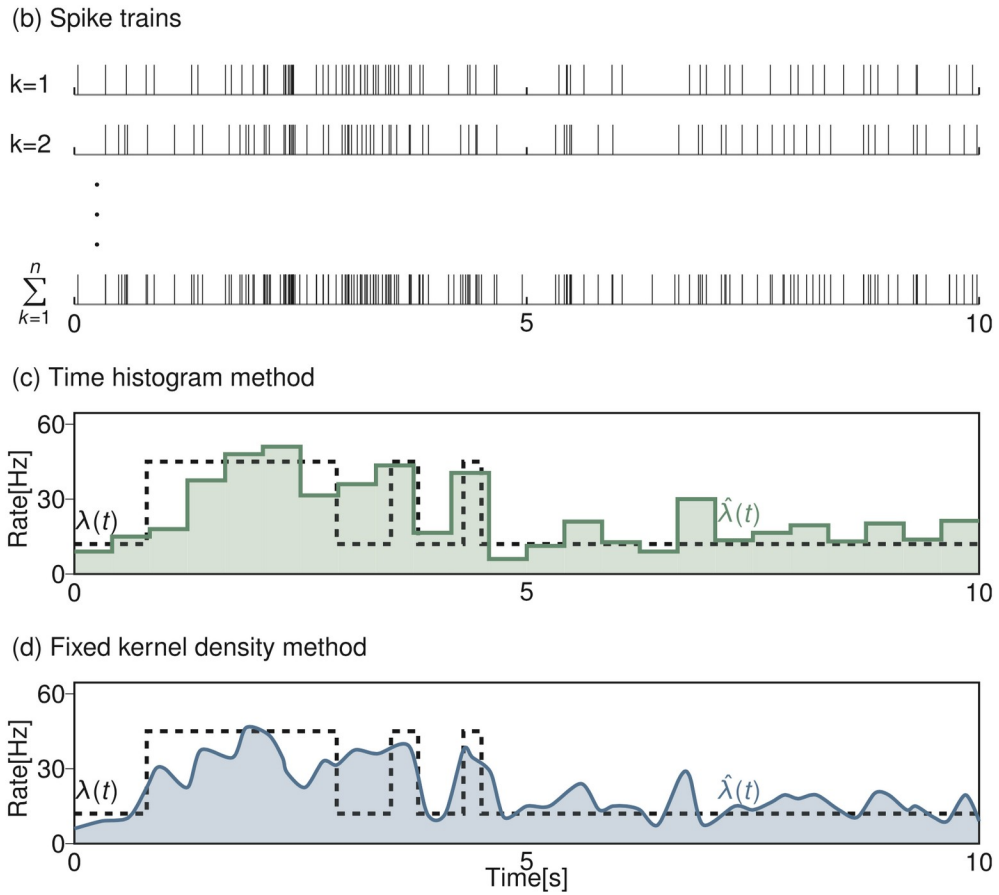
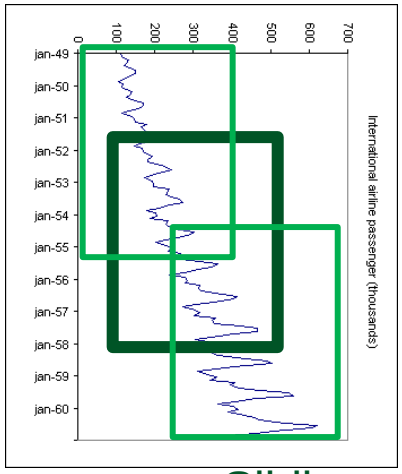


Fig. 1. Neuronal spiking activity and firing rate estimation. (a) Spikes occur at times S_i; the corresponding interspike intervals (ISIs) are denoted as X_i, and the associated counting process up to time instant t is $N(0, t)$. (b) Examples of spike trains simulated under identical statistical conditions. For most of the firing rate estimation methods, individual spike trains are superimposed but few methods, like Frequencygram (Section 4.1) and Bayesian Adaptive Kernel Smoothing (Section 4.8), work with individual recordings as well. (c) Estimation of underlying firing rate (green) from the spike train data, using the time histogram method, compared with the true underlying firing rate (dashed). (d) Firing rate estimation (blue) using the kernel smoothing method with Gaussian kernel, against the true underlying firing rate (dashed). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$

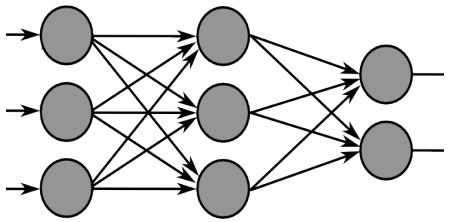
Quel type de reseau à la mode utilise également ce concept de fenêtre glissante ?



Time

Sliding window

Processing



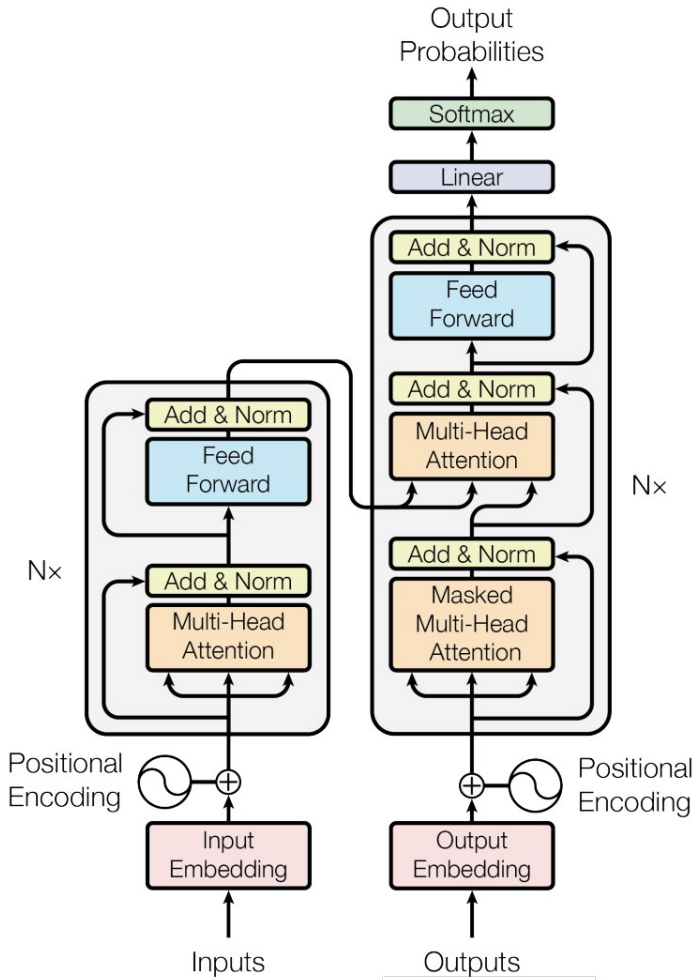
Feed Forward Neural Net.

Les transformers (aussi des FFNN) traitent les sequences similairement à des fenêtres glissantes

BERT

Encoder

Représentation du contexte “à gauche” et “à droite”
t – n … t … t+n



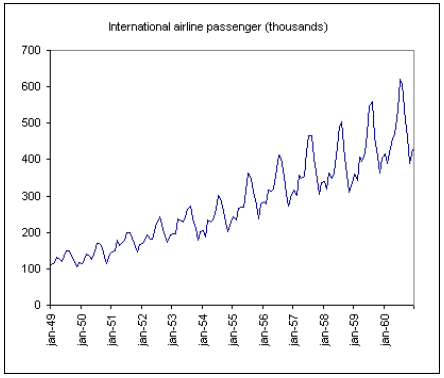
https://www.youtube.com/watch?v=4Bdc55j80l8

GPT

Decoder

Que dans le futur : Prédiction du prochain mot
t+1

Réseau de neurones recurrents traitent les séries temporelles et sequences sans fenêtre glissante

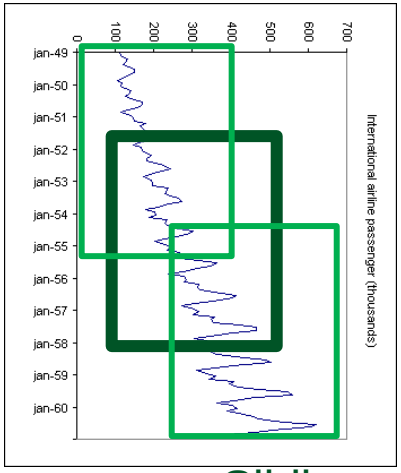


Time

Processing

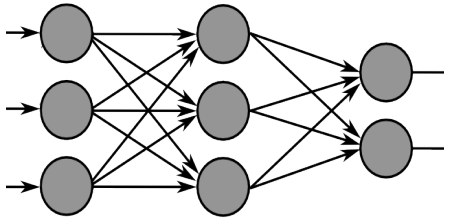
Recurrent Neural Net. (RNN)

Time



Sliding window

Processing



Feed Forward Neural Net.

Pourquoi utiliser des réseaux récurrents ?

« Je déteste les lundis »

« J'aime la prose »

« La bicyclette est bleue »

« Comme c'est bizarre ! »

?

◯ Mme Martin

◯ Jean

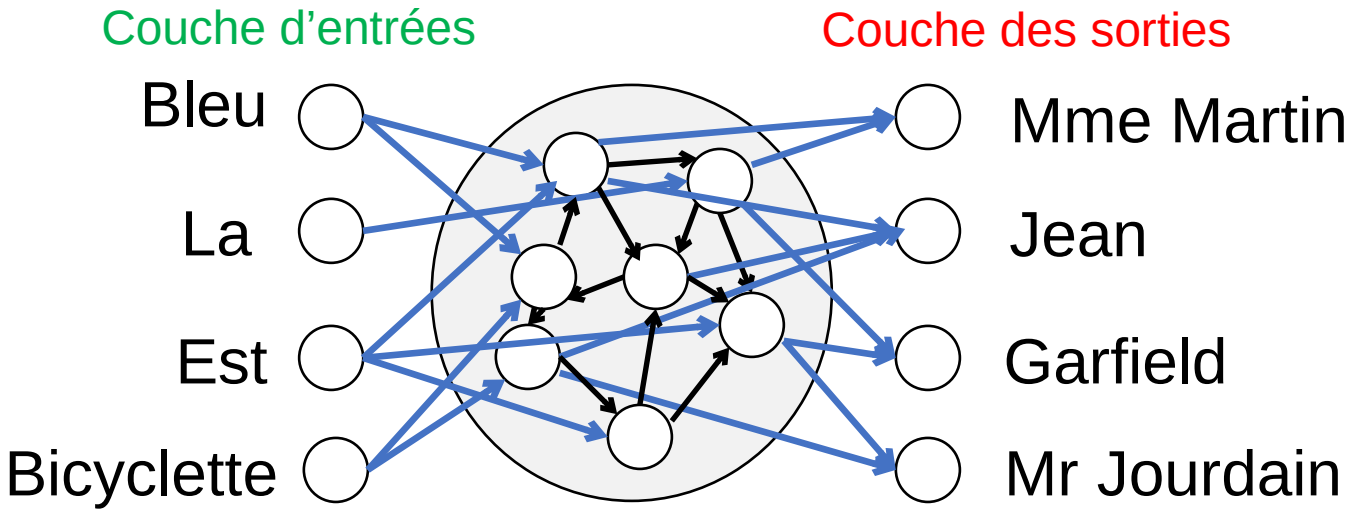
◯ Garfield

◯ Mr Jourdain

- Comment associer des phrases avec les personnes associées ?
  - Les mots sont dits les uns à la suite des autres
  - Il faut donc une mémoire de l'ordre des mots

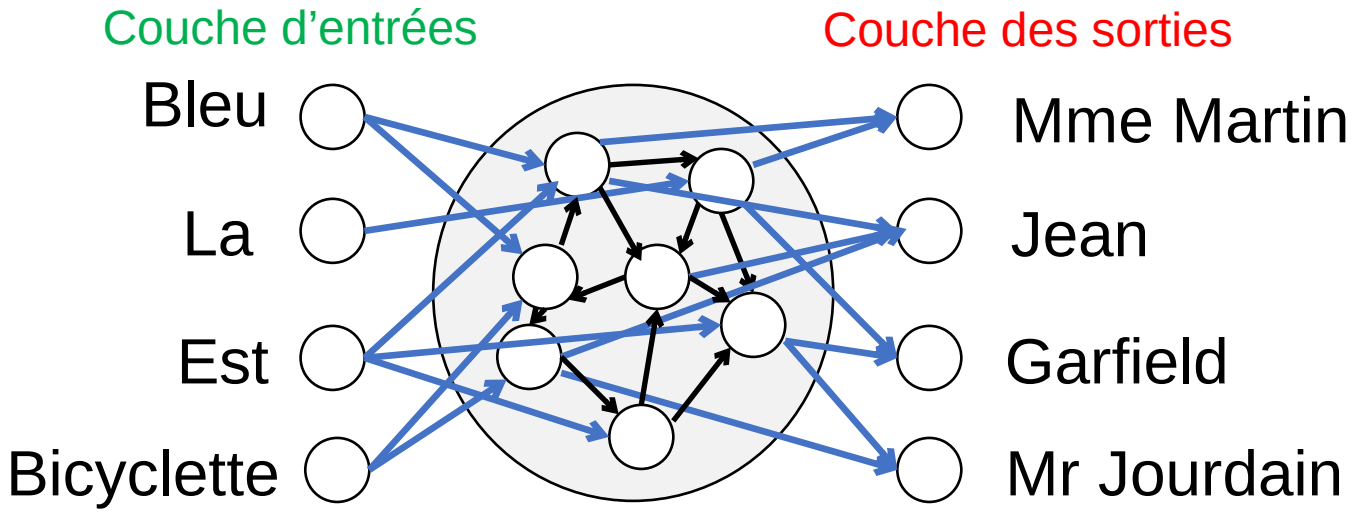
Réseau récurrent pour les séquences

« La bicyclette est bleue »



Réseau récurrent pour les séquences

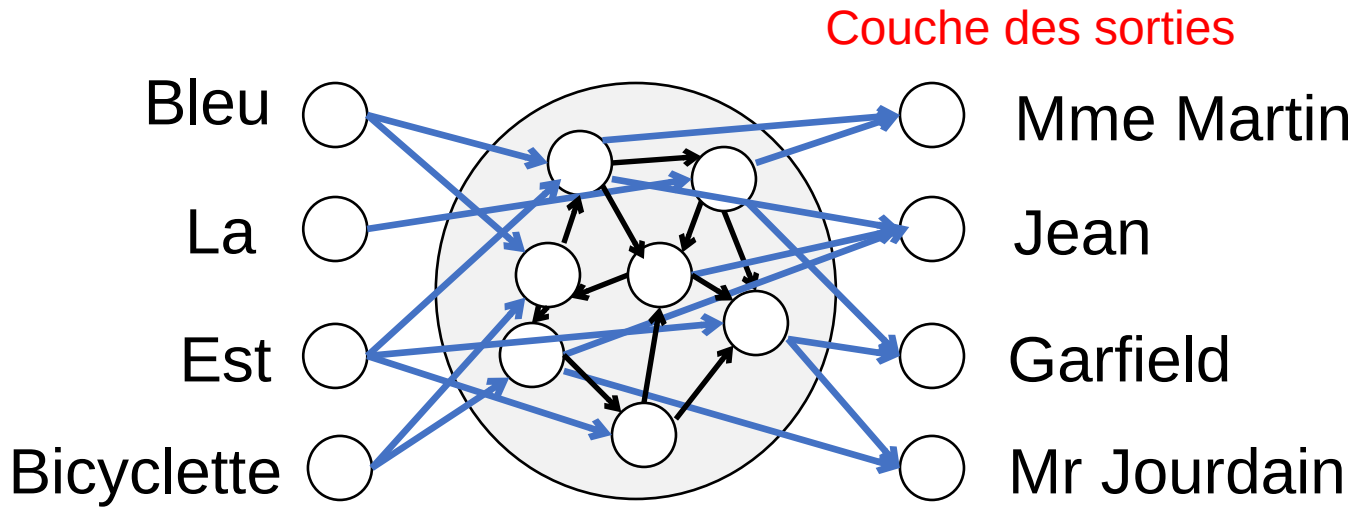
« La bicyclette est bleue »



- On va connecter la couche d'entrée (les mots possibles de la phrase) à la couche récurrente.

Réseau récurrent pour les séquences

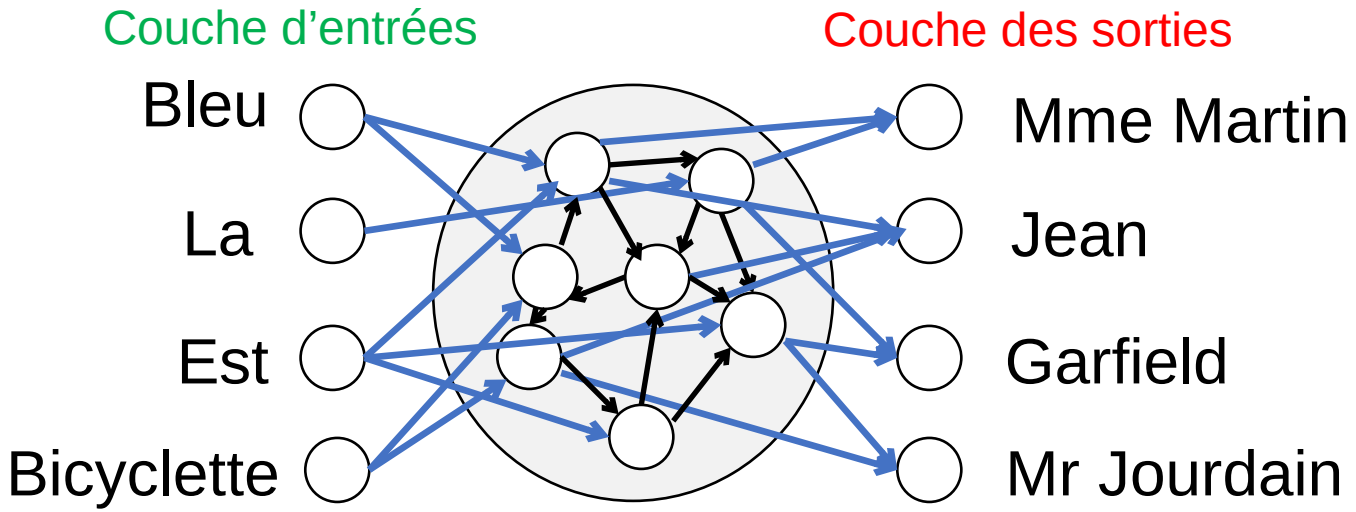
« La bicyclette est bleue »



- On va connecter la couche d'entrée (les mots possibles de la phrase) à la couche récurrente.
- Puis on va connecter la couche récurrente à la couche de sortie (les noms des auteurs possibles)

Réseau récurrent pour les séquences

« La bicyclette est bleue »



- On rentre les mots un par un.
- Les activations de neurones vont générer une mémoire dynamique des mots de la phrase.
- Représentation du contexte temporel.

Comment apprendre dans un réseau récurrent ?

- Si on veut appliquer l'algorithme de rétro-propagation du gradient à un réseau récurrent,
- cela implique de « virtualiser » le temps

Le réseau de Elman = Simple Recurrent Network (SRN)

Un des modèles de réseau de neurones récurrent qui a longtemps été le plus connus et utilisé est le Simple Recurrent Network de Elman (1990).

La présence de connexions récurrentes, grâce à la couche cachée, va permettre de représenter le contexte temporel des entrées.

Le réseau de Elman = Simple Recurrent Network (SRN)

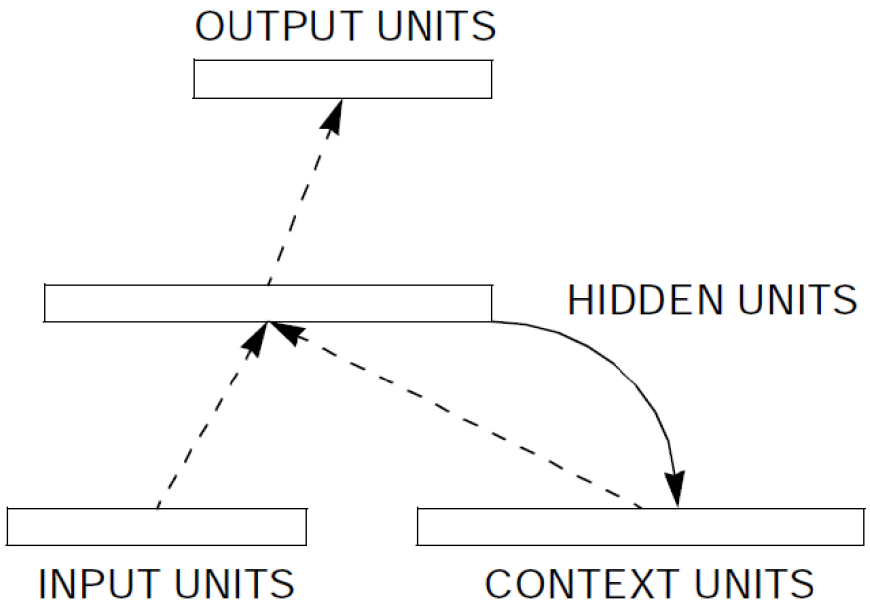


FIGURE 2.3 – Le Simple Recurrent Network de Elman. Les connexions en pointillés représentent les connexions modifiées par apprentissage. La particularité de ce modèle, par rapport à un réseau de neurones *feedforward*, vient de la présence de la couche de contexte qui reçoit des copies des activations de la couche cachée. Image provenant de [Elm04].

Le réseau de Elman = Simple Recurrent Network (SRN)

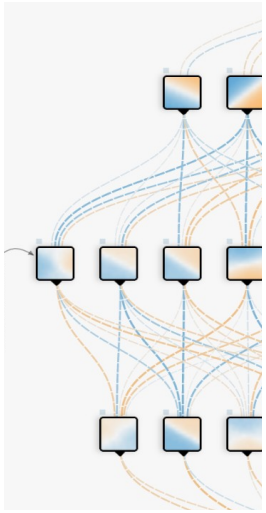
- Dans son article de 1990, Elman utilise l'algorithme de rétro-propagation (back-propagation) de l'erreur pour apprendre les connexions du réseau.
- Les connexions récurrentes de la couche cachée vers la couche de contexte sont fixées à 1 et ne sont pas modifiées par l'algorithme.
- L'algorithme est utilisé de la même façon que pour un réseau feedforward : chaque fois qu'une sortie a été produite, et donc à chaque pas de temps.

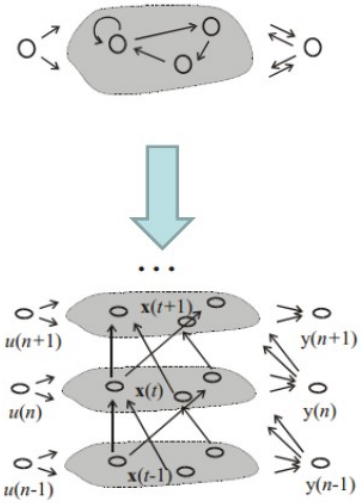
Le réseau de Elman = Simple Recurrent Network (SRN)

- En d'autres termes, le réseau de Elman ou SRN est un réseau sur lequel on applique la rétropropagation du gradient sur un pas de temps en arrière seulement.

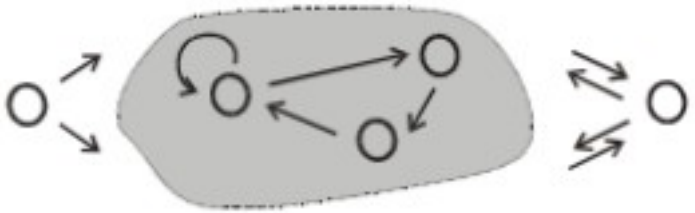
Comment apprendre dans un réseau récurrent ?

- Si on veut appliquer l'algorithme de rétro-propagation du gradient à un réseau récurrent,
- cela implique de « virtualiser » le temps
- et donc de dupliquer le réseau à chaque pas de temps
  - (cela implique de définir un horizon maximum, qui peut être de la taille de la série temporelle)
- afin appliquer l'algo. comme si c'était un réseau en couches



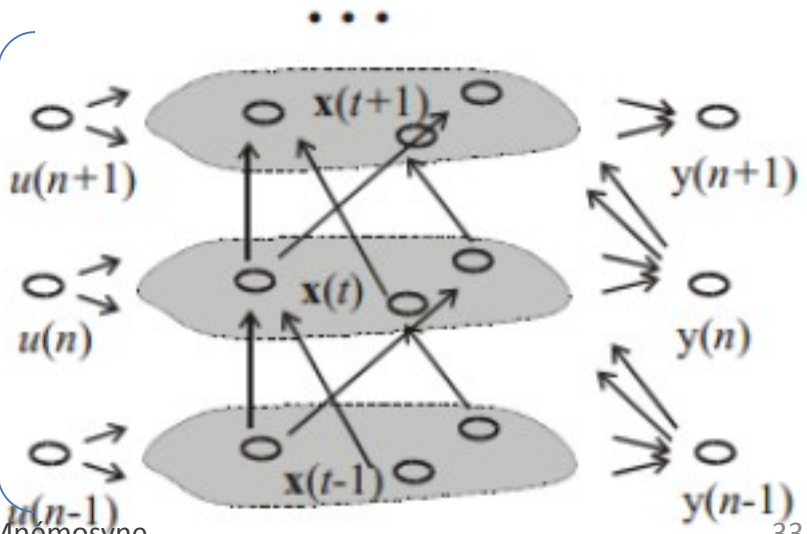


Back-Propagation Through Time (BPTT)



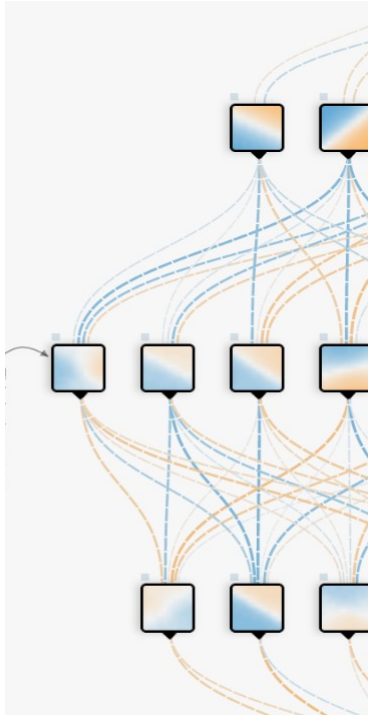
“depliage du temps”

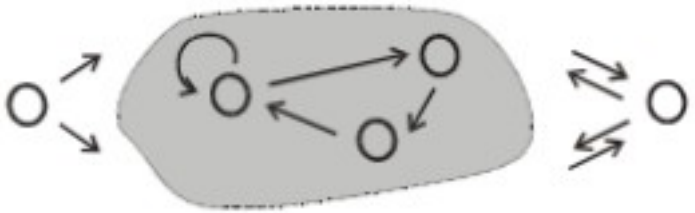
Dupliquer le reseau pour chaque pas de temps t



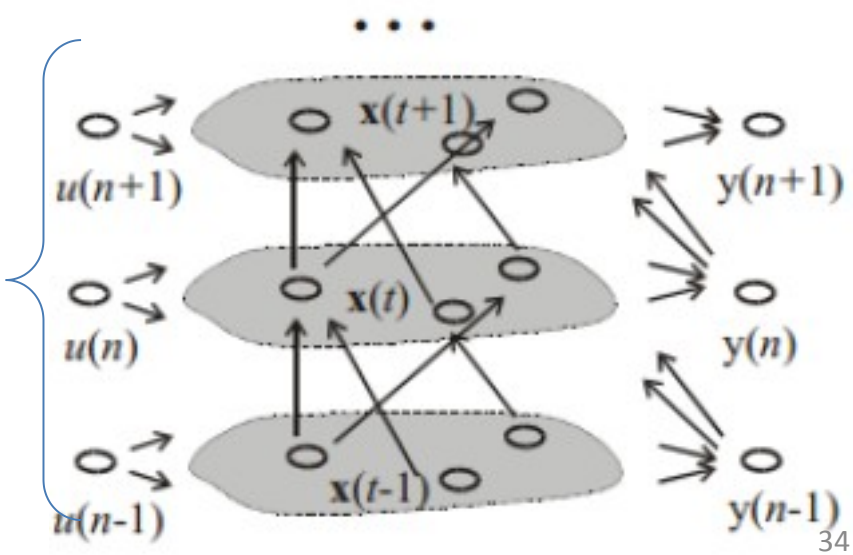
Back-Propagation Through Time (BPTT)

L'état interne à l'instant t du RNN va être équivalent à l'état de la couche t du reseau déplié en couches



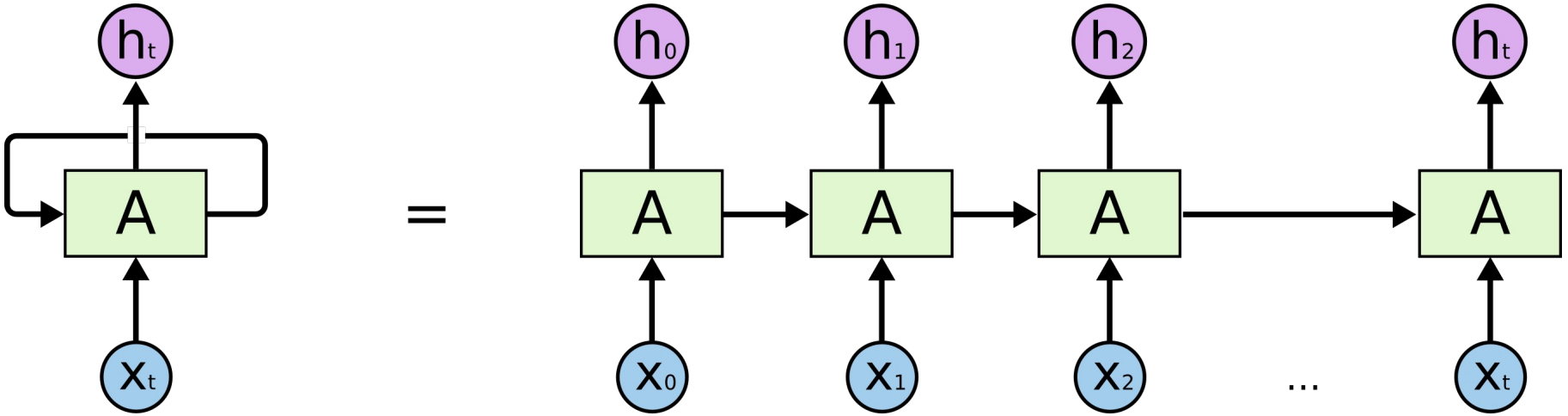


"depliage du temps"



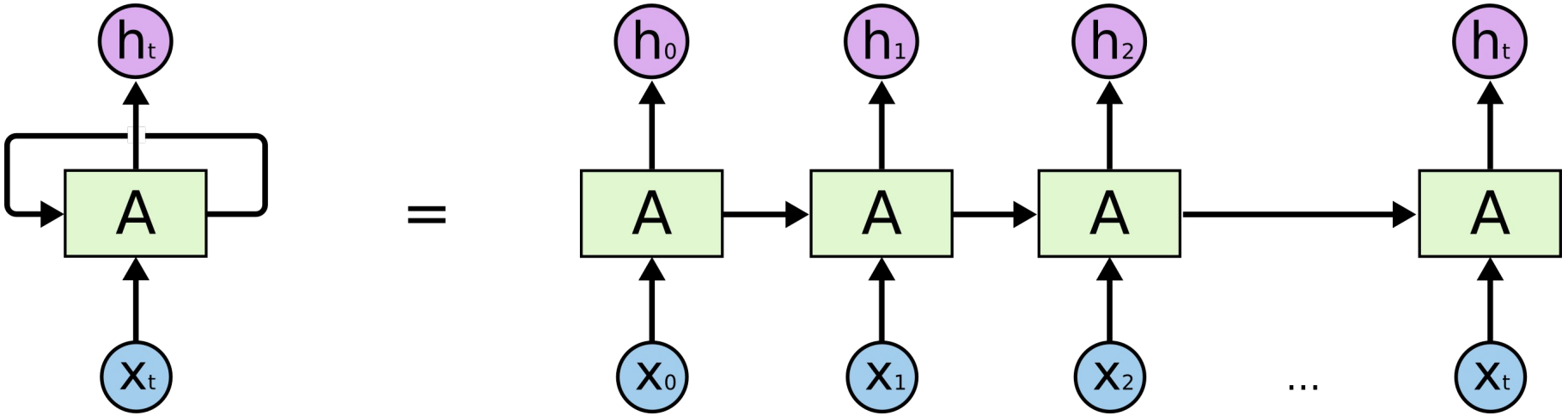
Réseau de neurone recurrent “déplié”

§ Un réseau de neurone “déplié”/“déroulé” (*unrolled*) dans le temps



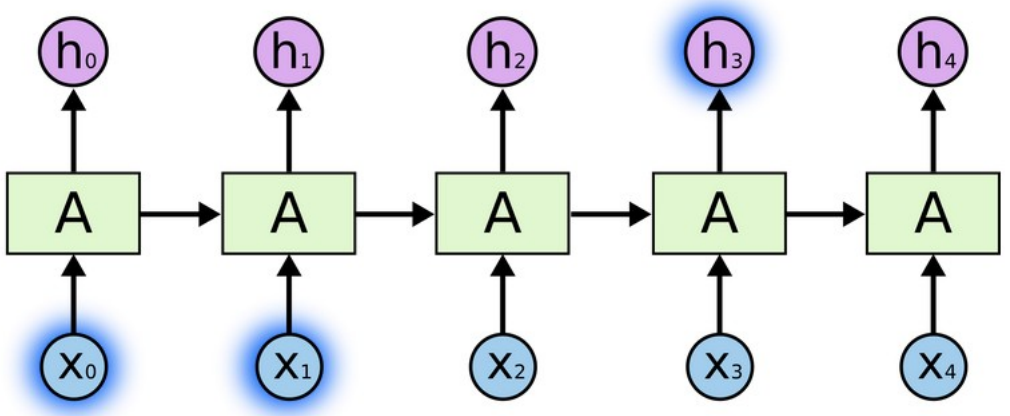
Réseau de neurone recurrent “déplié”

- Un réseau de neurone “déplié”/“déroulé” (*unrolled*) dans le temps
- Cela revient à avoir un réseau feedforward avec des poids de connexion identiques à plusieurs endroits du réseau.
- En appliquant la BPTT (backprop through time) on va tenir compte de cette contrainte (d’avoir les mêmes poids de connexions à différents endroits)

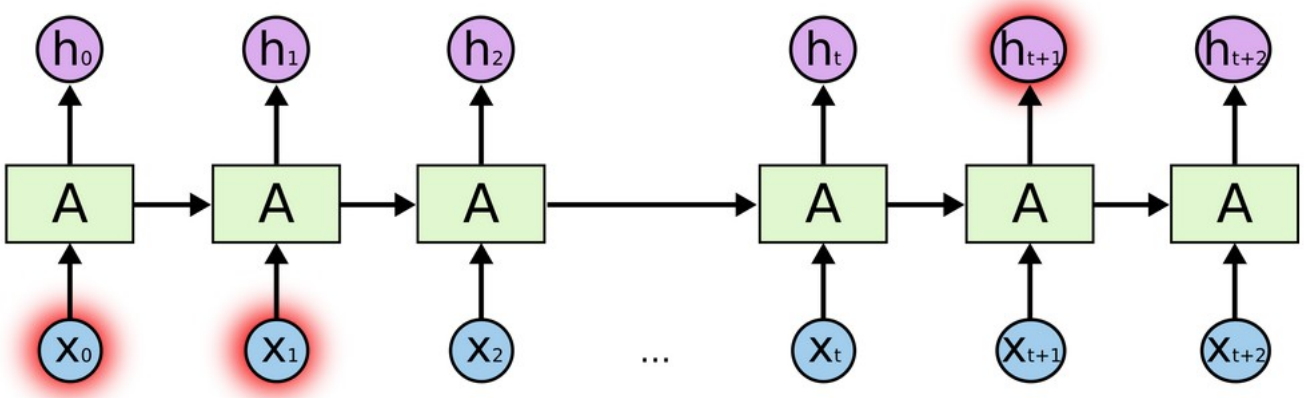


Comment gérer les dépendances à long terme ? (= Long-Time Dependencies)

Short

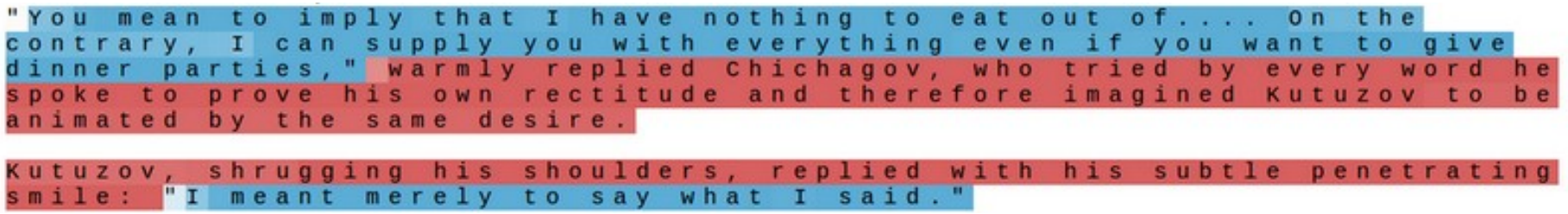


Long



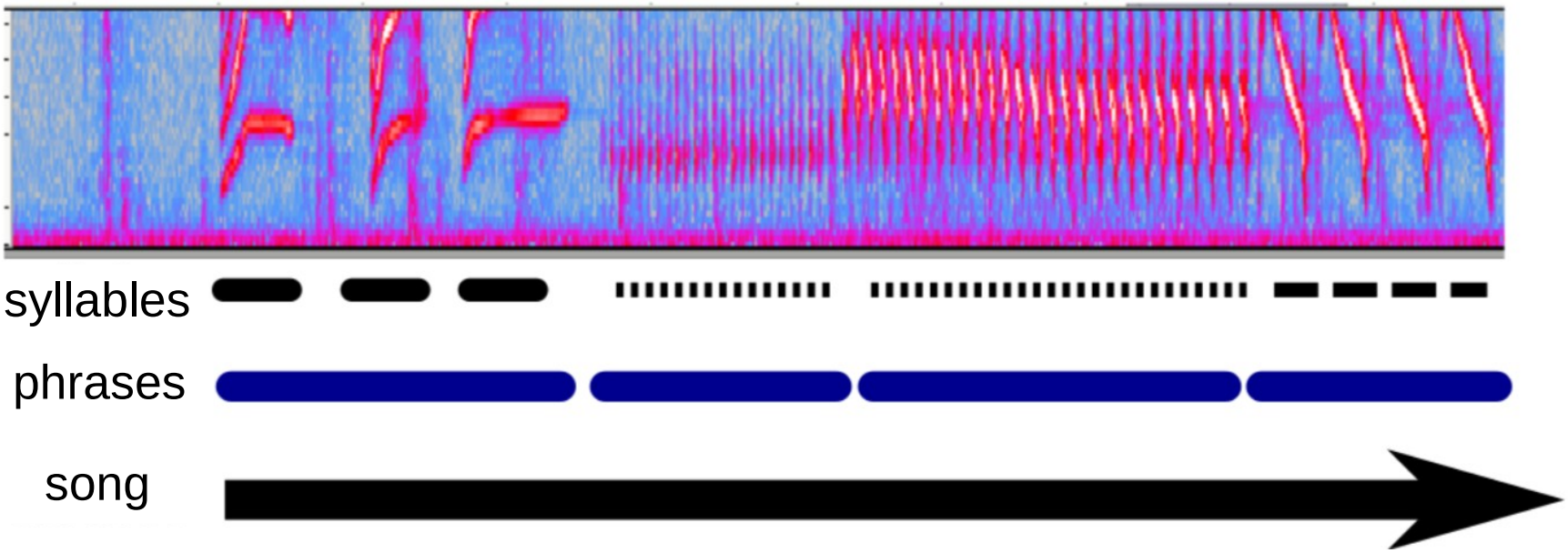
Comment gérer les dépendances à long terme ? (= Long-Time Dependencies)

Retenir ce qui vient avant, pendant ou après une citation implique d'avoir une mémoire de travail assez longue, car cela implique de gérer les dépendances à long terme sous-jacentes.

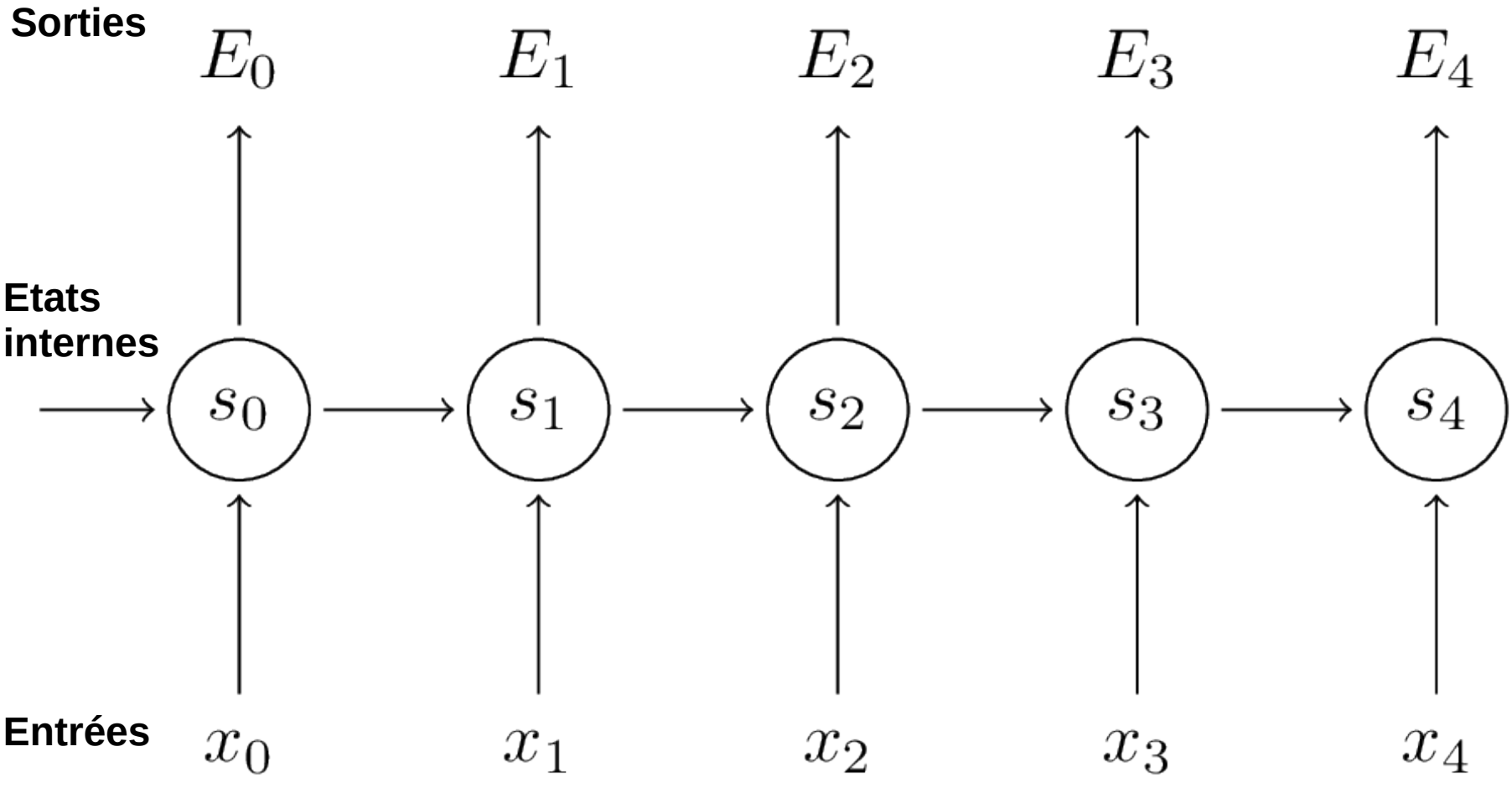


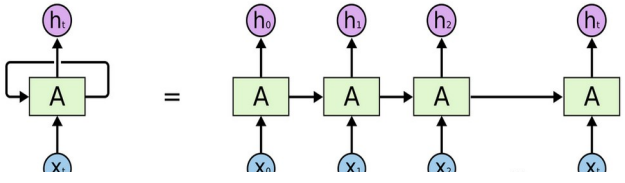
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Canary Songs

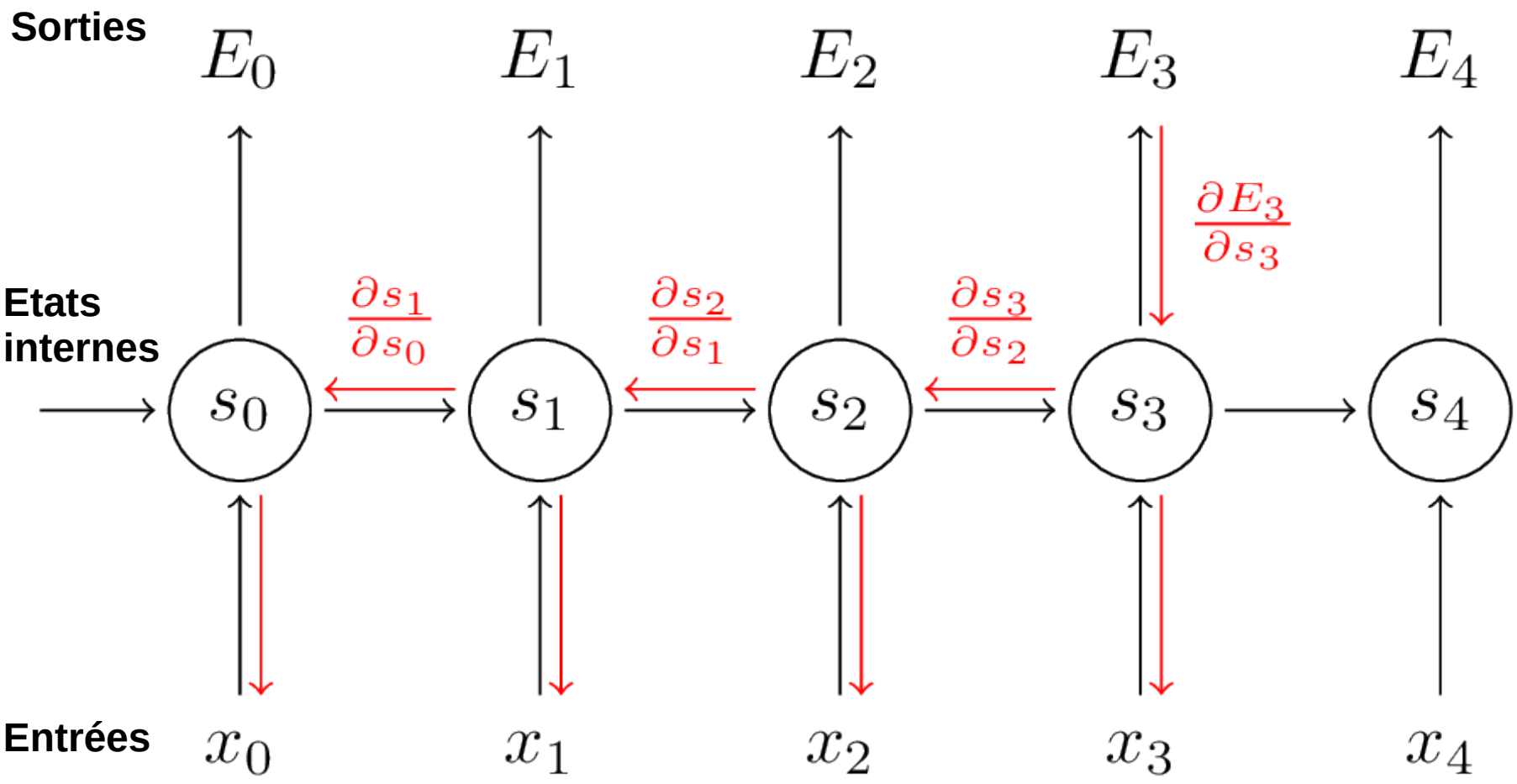


Back-Propagation Through Time

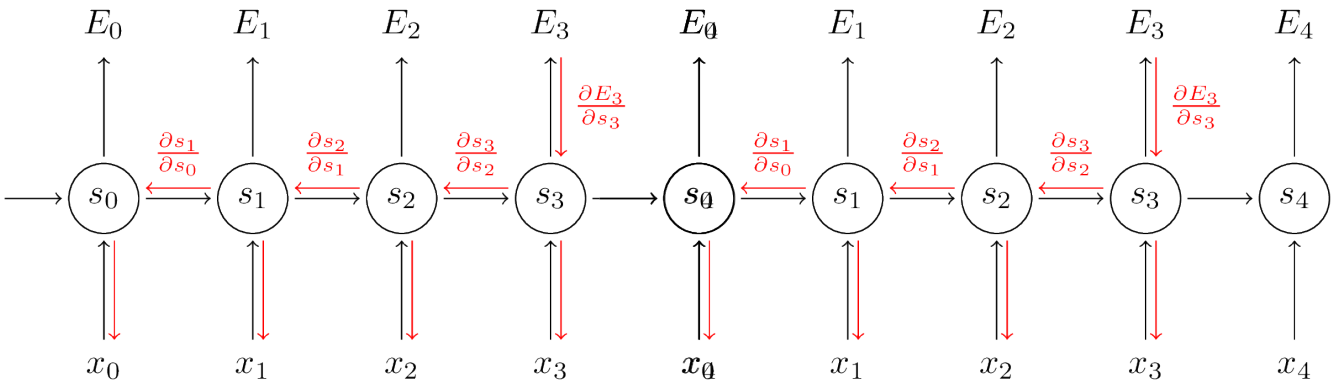




Back-Propagation Through Time

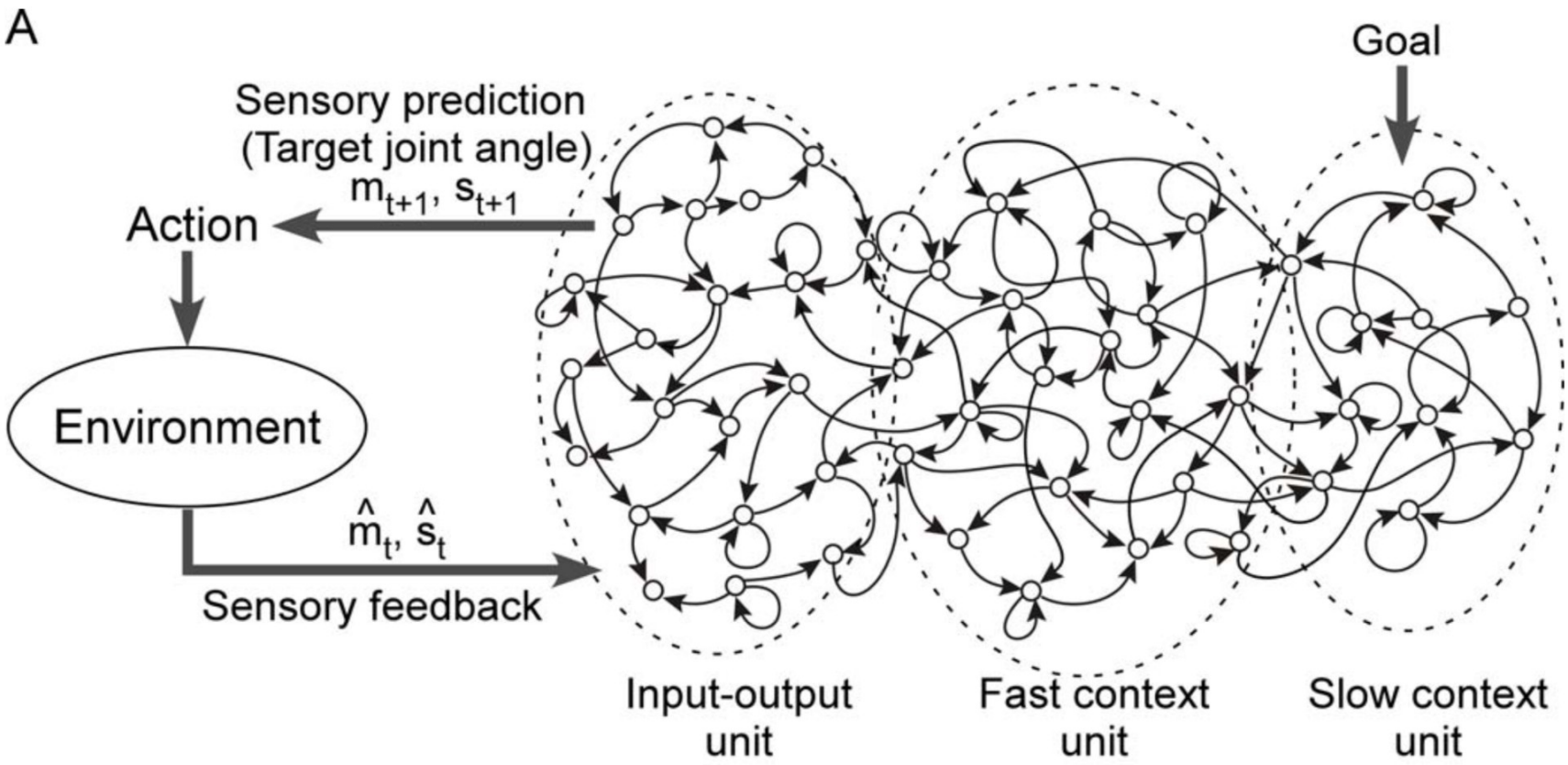


Back-Propagation Through Time



Problem of vanishing or exploding gradient the more where go through time …

Use RNN with Multiple Time-Scales



https://www.youtube.com/watch?v=n9NYcG8xlYs

Yamashita & Tani 2008

Use RNN with Multiple Time-Scales

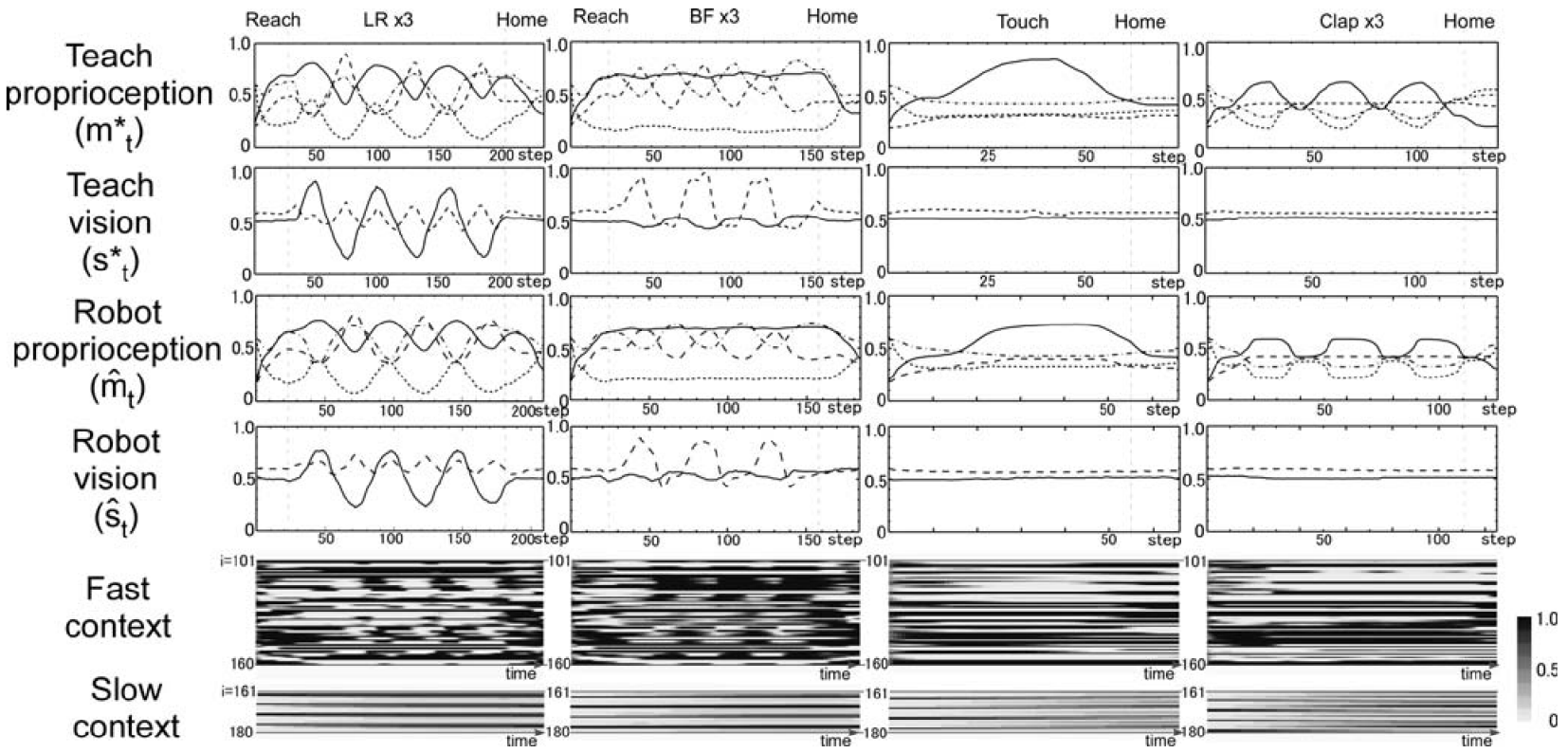


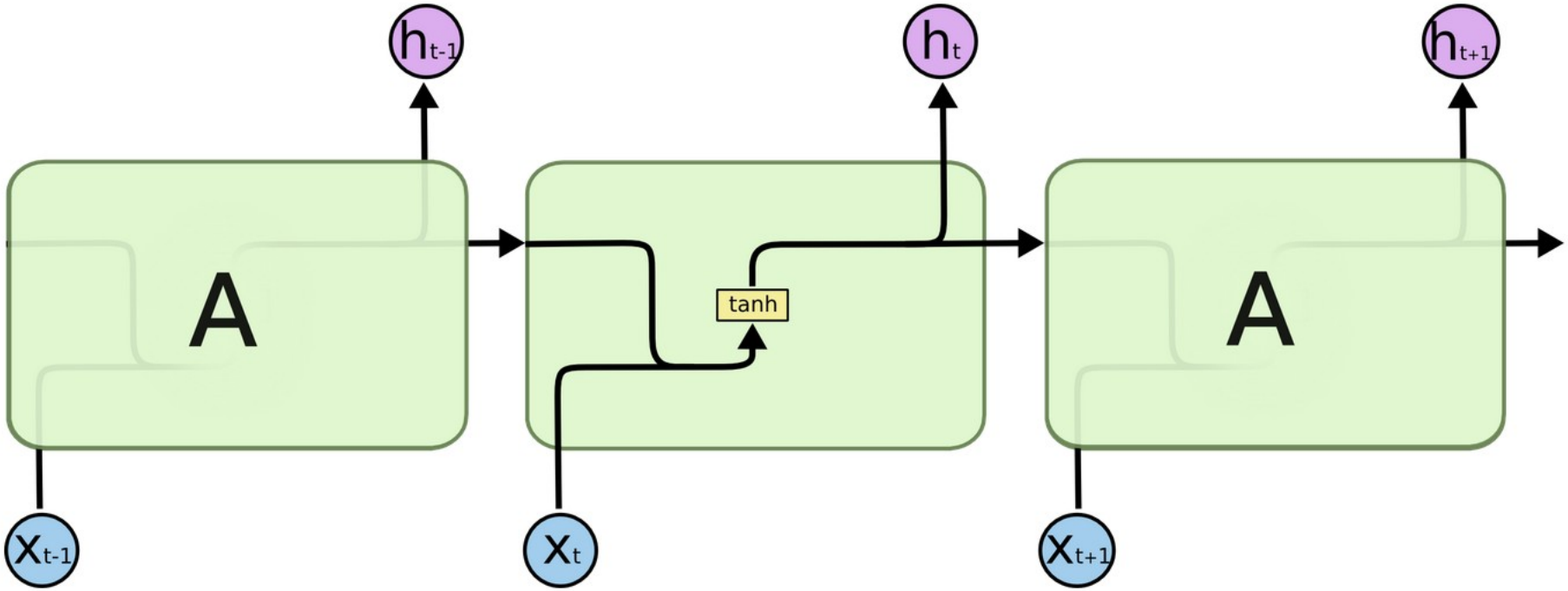
Figure 5. Example of behavior sequences for other basic behavior. Proprioception, vision, fast and slow context activation of teaching signal and actual values in physical environment during left-right (LR: first column), backward-forward (BF: second column) touch with single hand (Touch: third column) and clapping hands (Clap: fourth column) behavior at position 3 are shown. Correspondences for line types in each graph are the same as in Figure 4. Reach: reach to the object, Home: return to the home position.
doi:10.1371/journal.pcbi.1000220.g005

Yamashita & Tani 2008

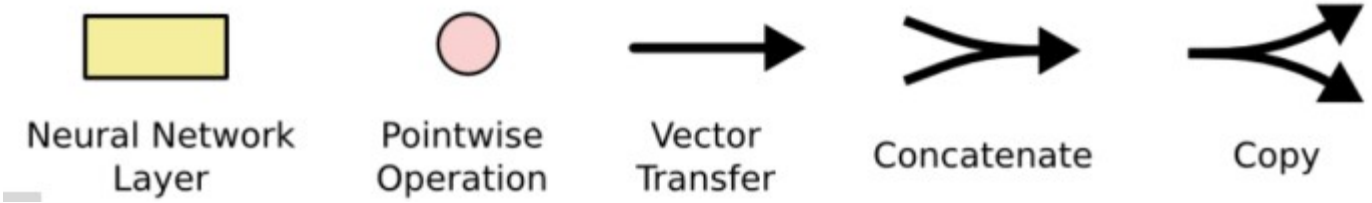
Plus loin que les RNNs classiques

- Pour pallier au problème de mémoire à long terme dans les RNNs
  - Tani et al. on utilisé les MTRNN (multi-time scale RNN). Plos Comp Bio 2008
    - « L'état interne » de chaque couche du RNN est représente un contexte différent
    - Contexte dans une action (état interne de la couche rapide)
    - Contexte dans une séquence d'actions (é.i. de la couche lente)
- Pour pallier au problème d'« explosion ou évanescence » du gradient (de la *back-propagation*)
  - LSTM: Long-Short Term Memory network (1997)

Classical RNNs

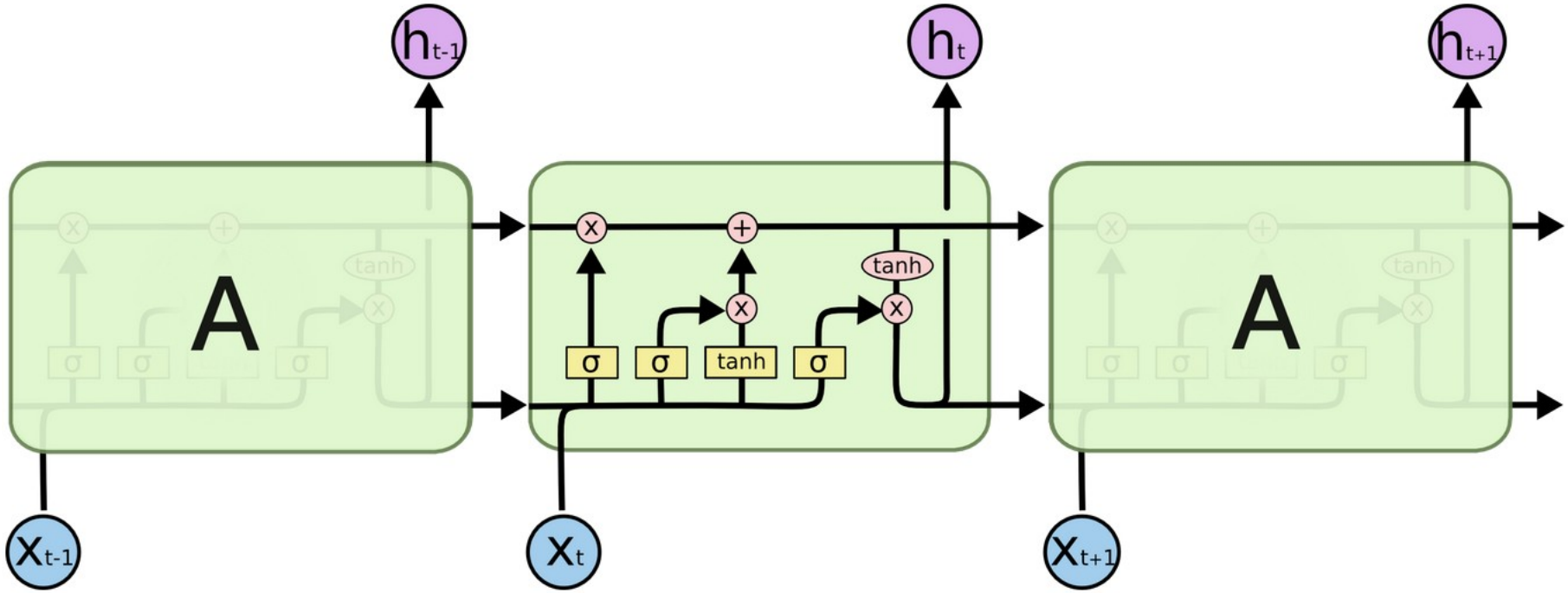


The repeating module in a standard RNN contains a single layer.

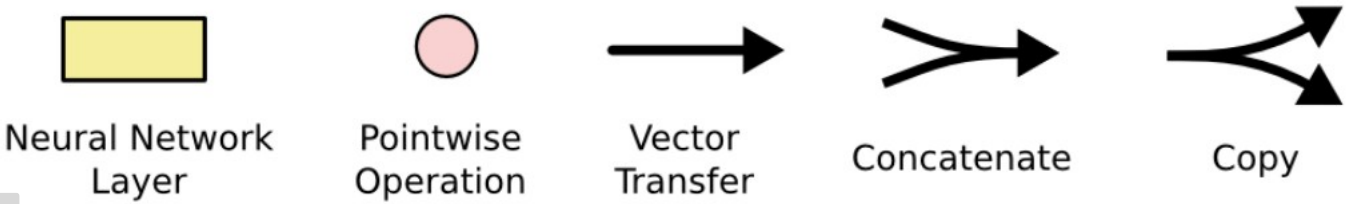


LSTM: Long Short-Term Memory Networks

Hochreiter & Schmidhuber (1997)



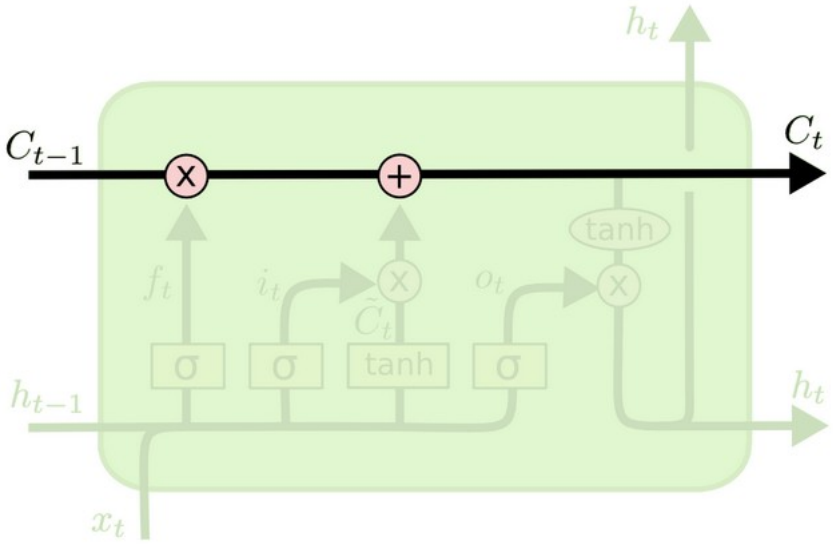
The repeating module in an LSTM contains four interacting layers.



colah.github.io

LSTM: the key component

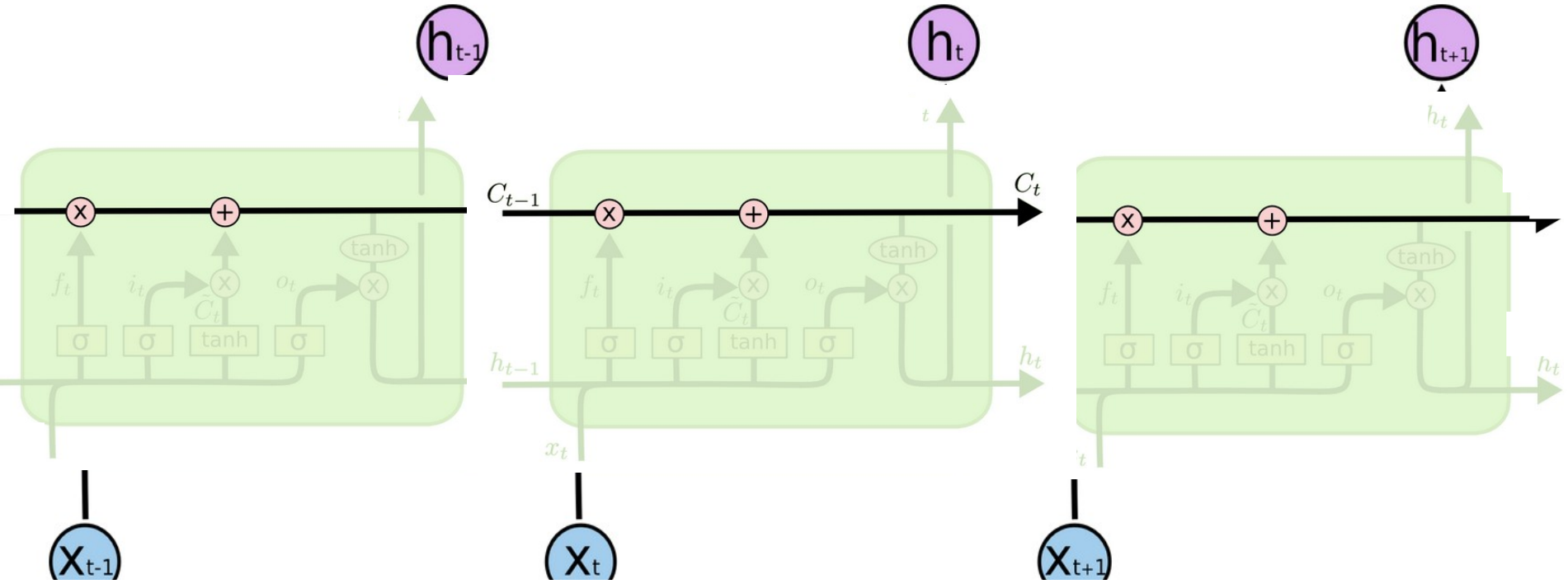
Composant clé:
« cell state »



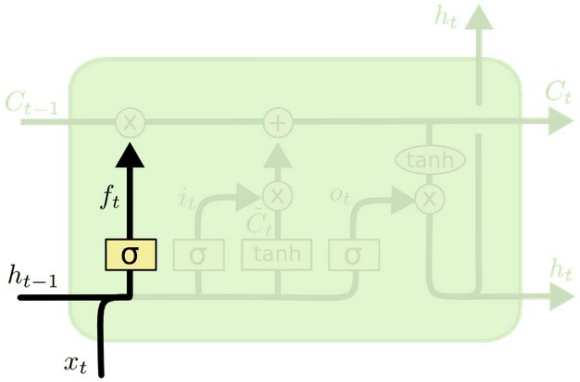
La « Cell state » permet de garder le gradient constant au cours du temps (lorsque c'est nécessaire).

Cela permet de résoudre le problème « d'explosion ou d'évaporation » du gradient.

LSTM: the key component



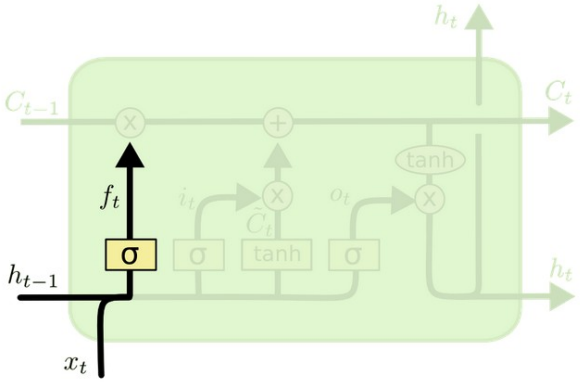
LSTM: 4 components



forget gate

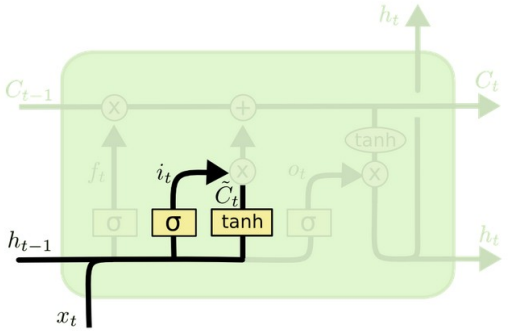
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM: 4 components



forget gate

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

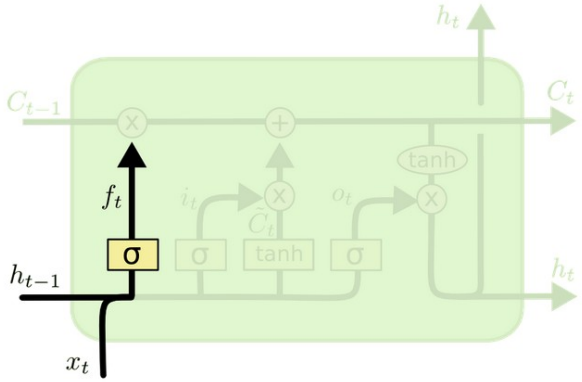


input gate

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

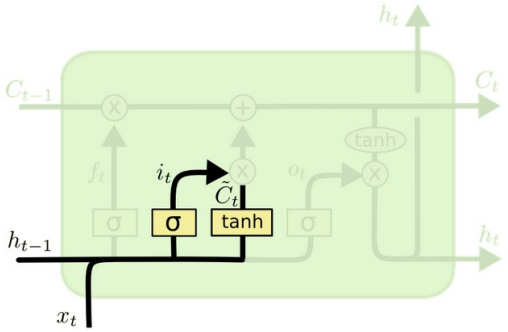
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM: 4 components



forget gate

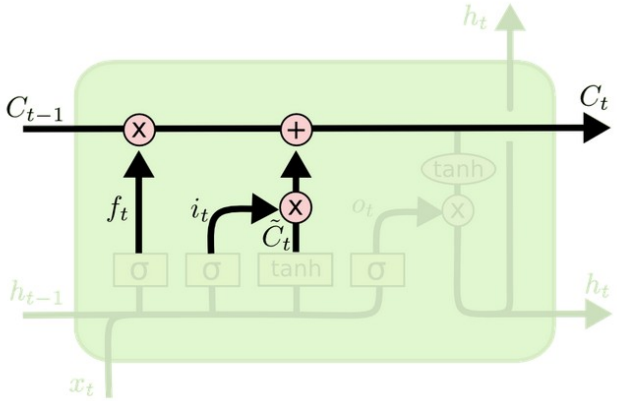
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



input gate

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

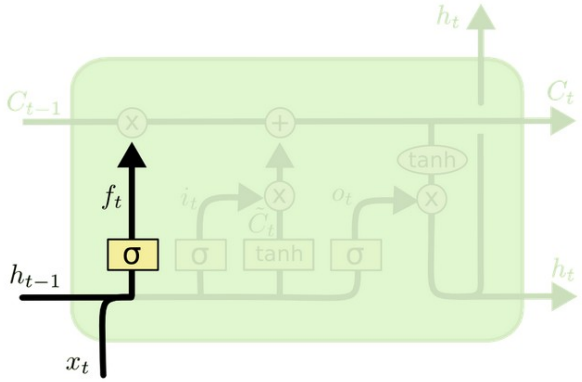
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



cell state

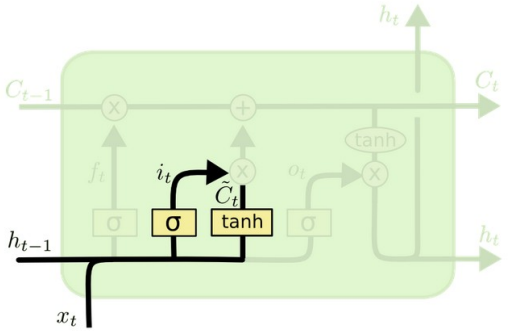
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM: 4 components



forget gate

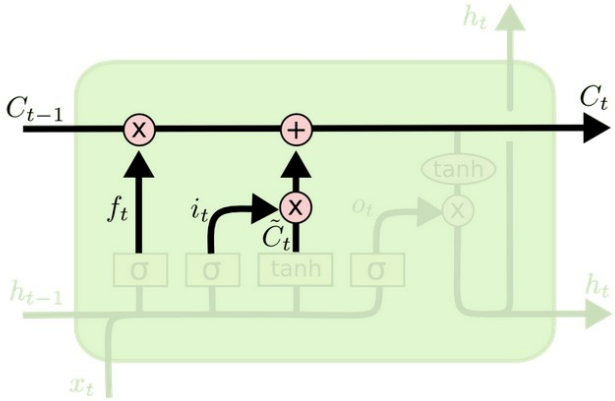
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



input gate

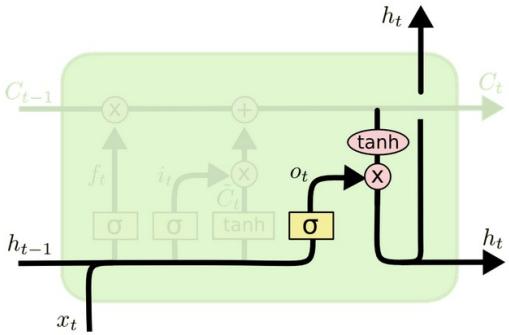
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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



cell state

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



output gate

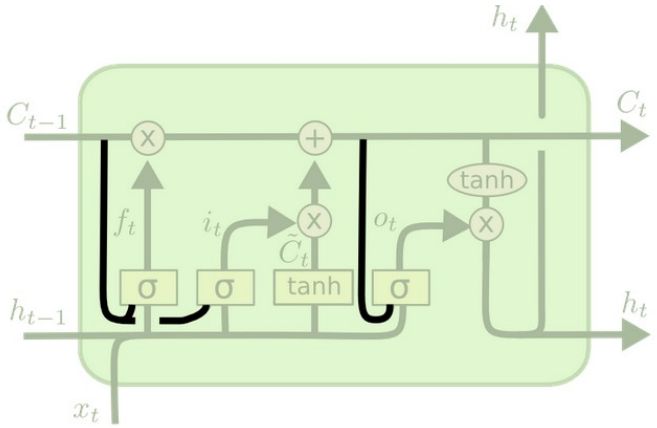
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

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

LSTM: Long Short-Term Memory Networks

- Il y a différentes versions de LSTM !
  - Version originale: Hochreiter & Schmidhuber (1997)
  - Version “2000”: Gers & Schmidhuber (2000)
  - GRU (Gated Recurrent Unit): Cho, et al. (2014)
  - …

LSTM Variants

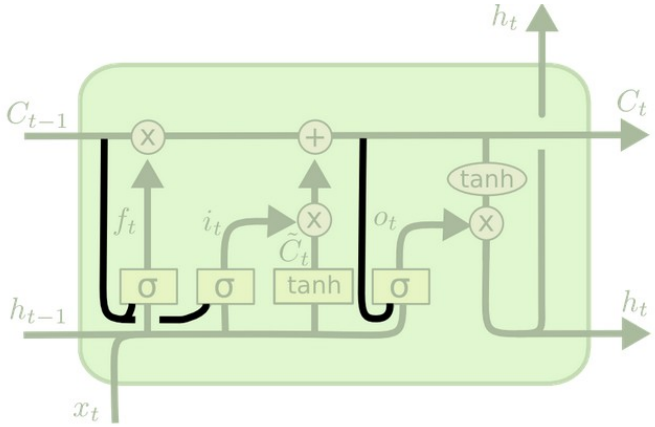


Gers & Schmidhuber (2000)

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

adding "peephole connections"

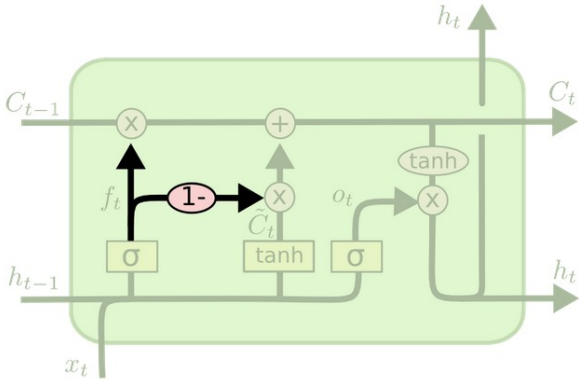
LSTM Variants



Gers & Schmidhuber (2000)

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

adding "peephole connections"

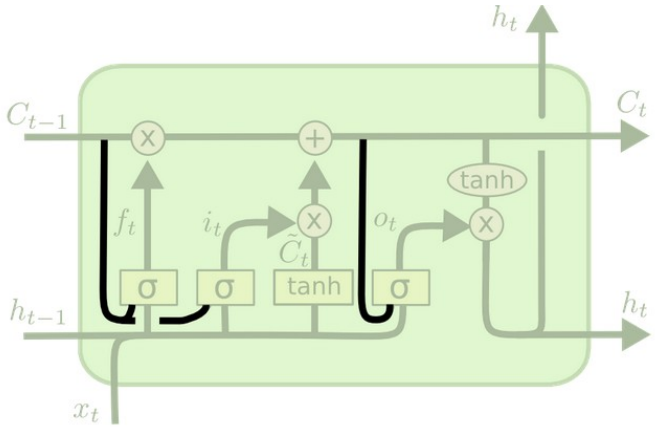


coupled forget and input gates

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

On décide simultanément ce que vont faire l'*input* et la *forget gate*:
On oublie seulement lorsque l'on va remplacer l'état par l'entrée.

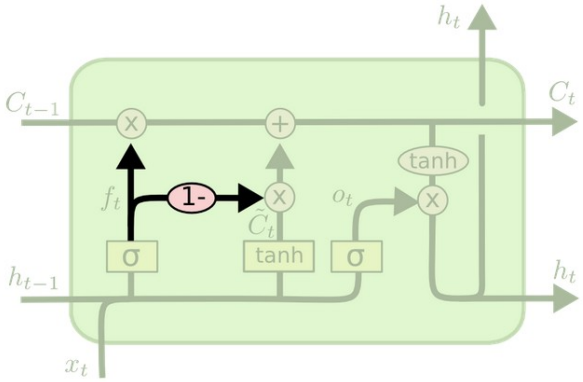
LSTM Variants



Gers & Schmidhuber (2000)

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

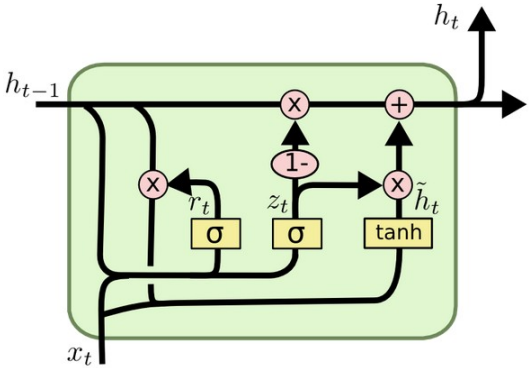
adding "peephole connections"



coupled forget and input gates

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

On décide simultanément ce que vont faire l'*input* et la *forget gate*:
On oublie seulement lorsque l'on va remplacer l'état par l'entrée.



GRU: Gated Recurrent Unit

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

colah.github.io

LSTM
Long Short-Term Memory Networks

§ A good blog to understand how it works:

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

§ Another explanation with a "football" analogy:

https://www.analyticsvidhya.com/blog/2019/01/fundamentals-deep-learning-recurrent-neural-networks-scratch-python/

'Soccer book concept' | ID 27329051 © Ponsuwan | Dreamstime.com

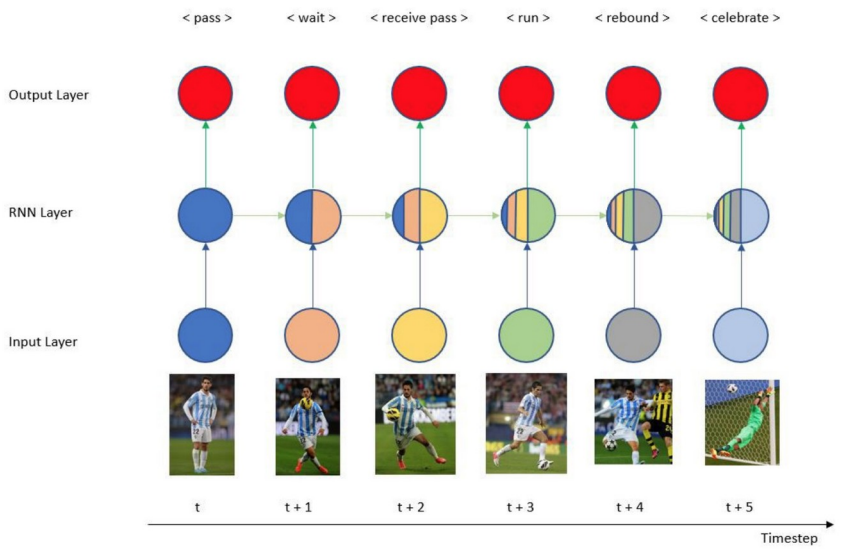
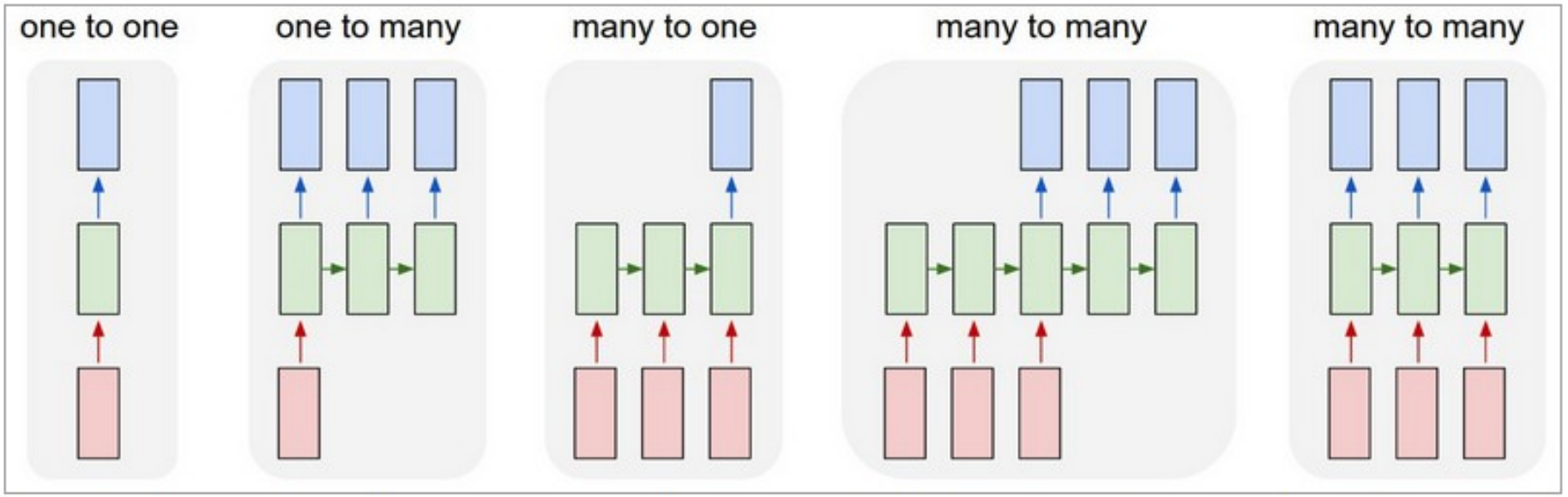


Image by author

Rappels sur les Réseaux de Neurones Récurrents (RNN) « classiques » (fin)

- Vidéo d'explications en Anglais
  - fonctionnement des LSTMs:
    - https://www.youtube.com/watch?v=8HyCNIVRbSU (2:07 à la fin)

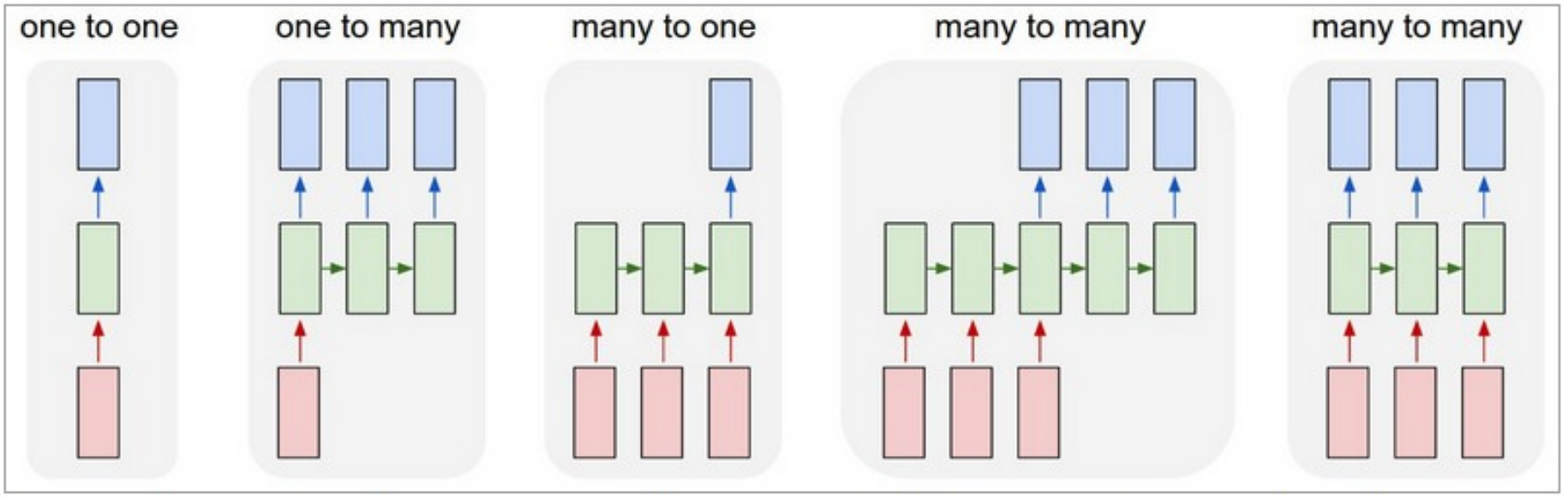
Summary on RNNs



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

source : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

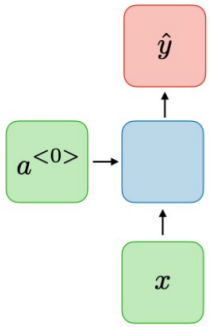
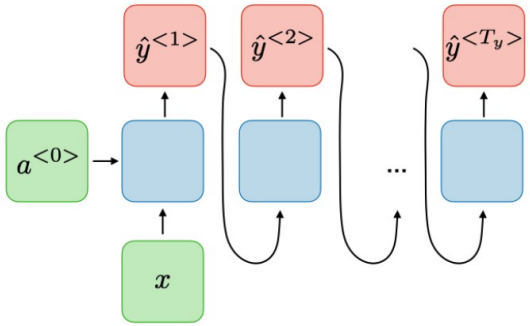
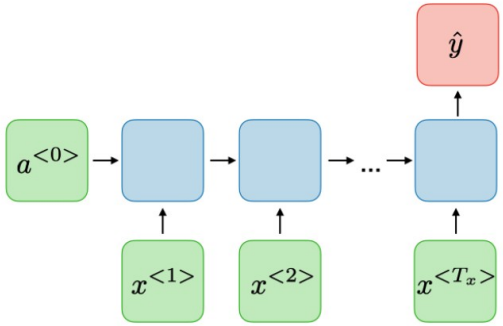
Summary on RNNs



- Translation ?
- Music generation ?
- Named entity recognition ?
- Sentiment classification ?

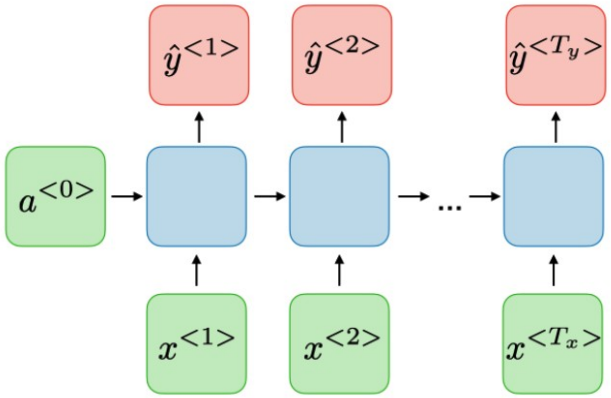
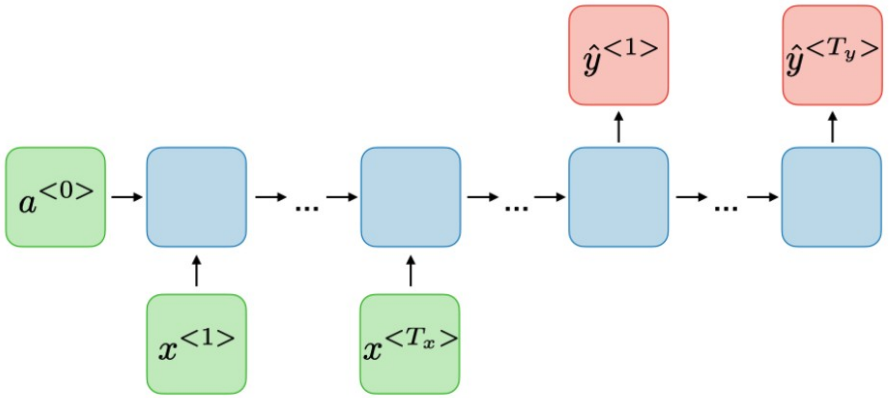
source : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Summary on RNNs

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

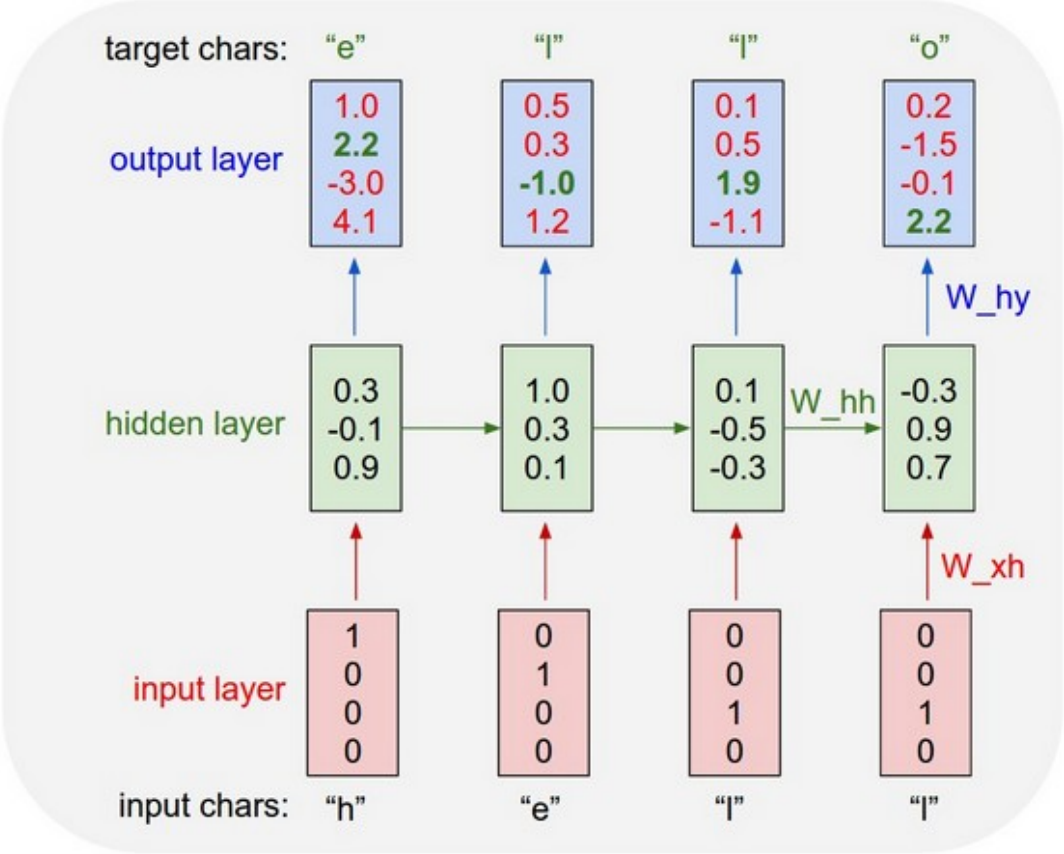
source : https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#

Summary on RNNs

Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

source : https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#

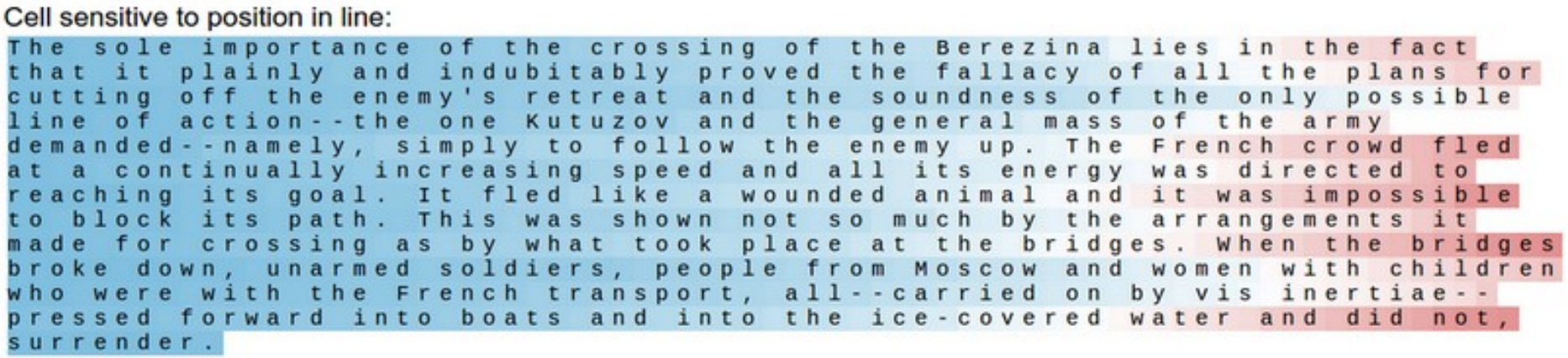
Next character prediction



An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low.

Next character prediction

- « Crazy things » that can do a RNN
  - A very nice blog about LSTMs:
    - http://karpathy.github.io/2015/05/21/rnn-effectiveness



source : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Next character prediction

- « Crazy things » that can do a RNN
  - A very nice blog about LSTMs:
    - http://karpathy.github.io/2015/05/21/rnn-effectiveness

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

source : http://karpathy.github.io/2015/05/21/rnn-effectiveness/