UNIVERSITÉ LAVAL

GLO-4030/7030
APPRENTISSAGE PAR RÉSEAUX DE NEURONES PROFONDS

Réseaux Récurrents avec *gate* (LSTM et GRU)

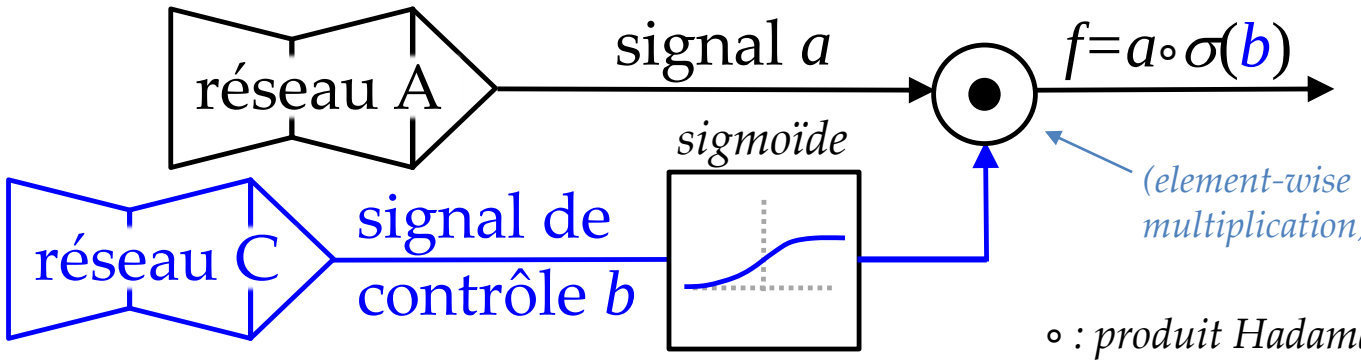
Hiver 2020

Philippe Giguère

LSTM (1997)

- Toujours d'actualité
- Résoudre les problèmes du RNN :
  - difficulté de la longue portée
  - *vanishing gradient*
- Idée maîtresse : **cellule(s) à état (cell state)** c_t
  - peut y ajouter/retirer/exposition de l'information via des *gates* (contrôle flux d'information)

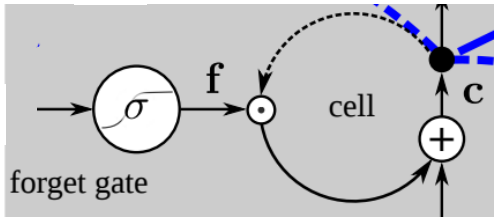
Rappel



  - similitude avec highway network/ResNet

LSTM : cellule + 3 gates

(beaucoup de diagrammes)



Cellule : (déroulée)

(vecteur) c_{t-1}

c_t

(opérations linéaires, peu modifiée, highway)

Forget gate

(vecteur) h_{t-1}

h_t

Input gate

Output gate

X_t

Neural Network Layer

Pointwise Operation

Vector Transfer

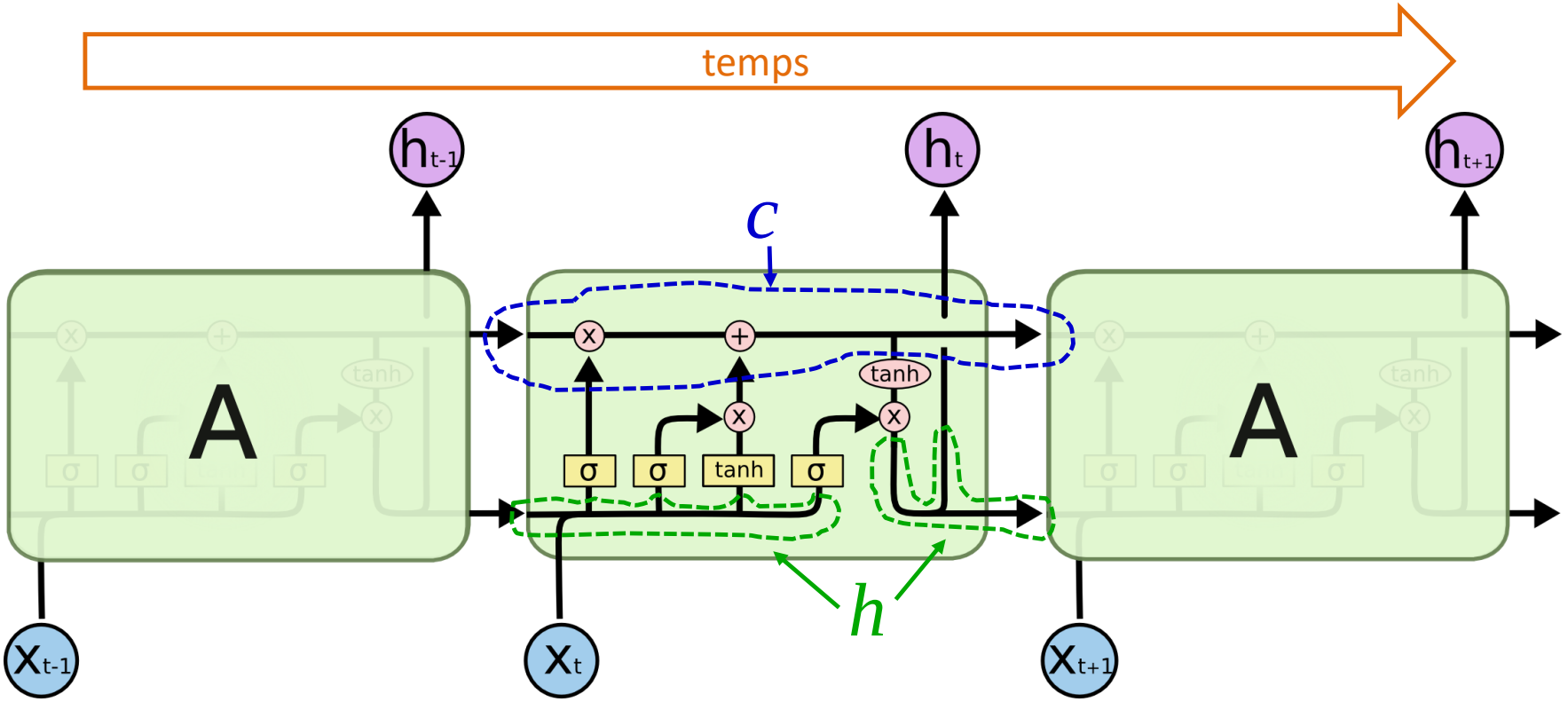
Concatenate

Copy

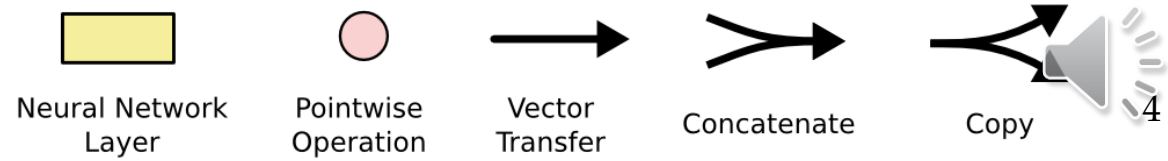
Illustrations : Christopher Olah, avec permission

LSTM : récursivité déroulée

- « État caché » est (h_t, c_t)



Note : c et h ont la même taille



Illustrations : Christopher Olah, avec permission

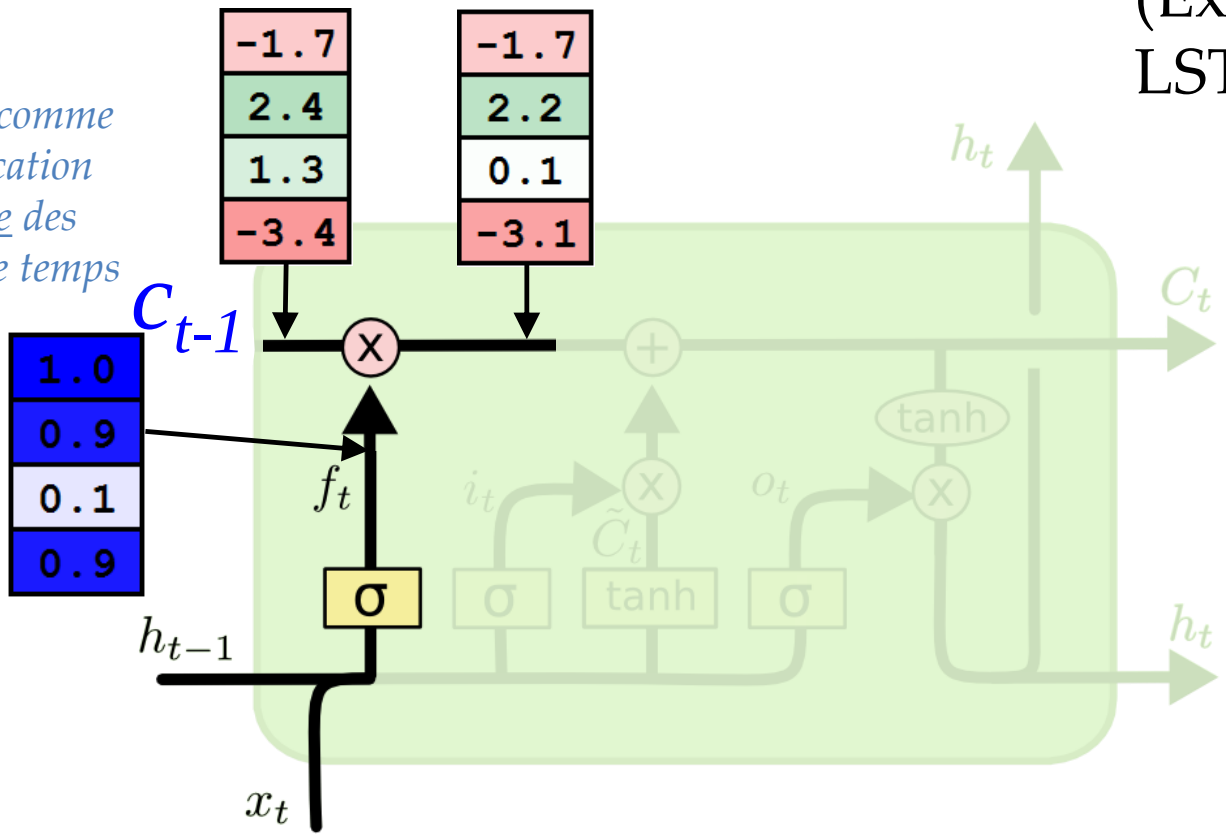
LSTM : Étape 1

- Quelle information **retirer** de la cellule ?
- forget gate (pensez plus : *remember gate*)

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

(Exemple pour LSTM à 4 unités)

Peut le voir comme une modification dynamique des constantes de temps



Illustrations : Christopher Olah, avec permission

LSTM : Étape 2

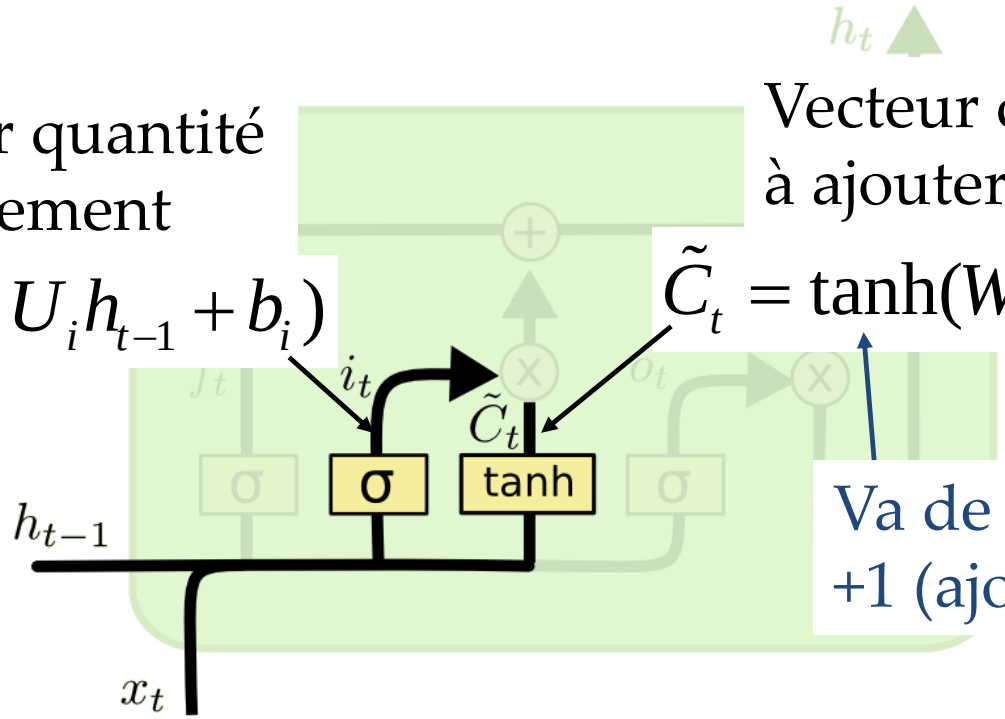
- Choisir l'information à ajouter à la cellule

Vecteur pour quantité de changement

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

Vecteur d'information à ajouter

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$



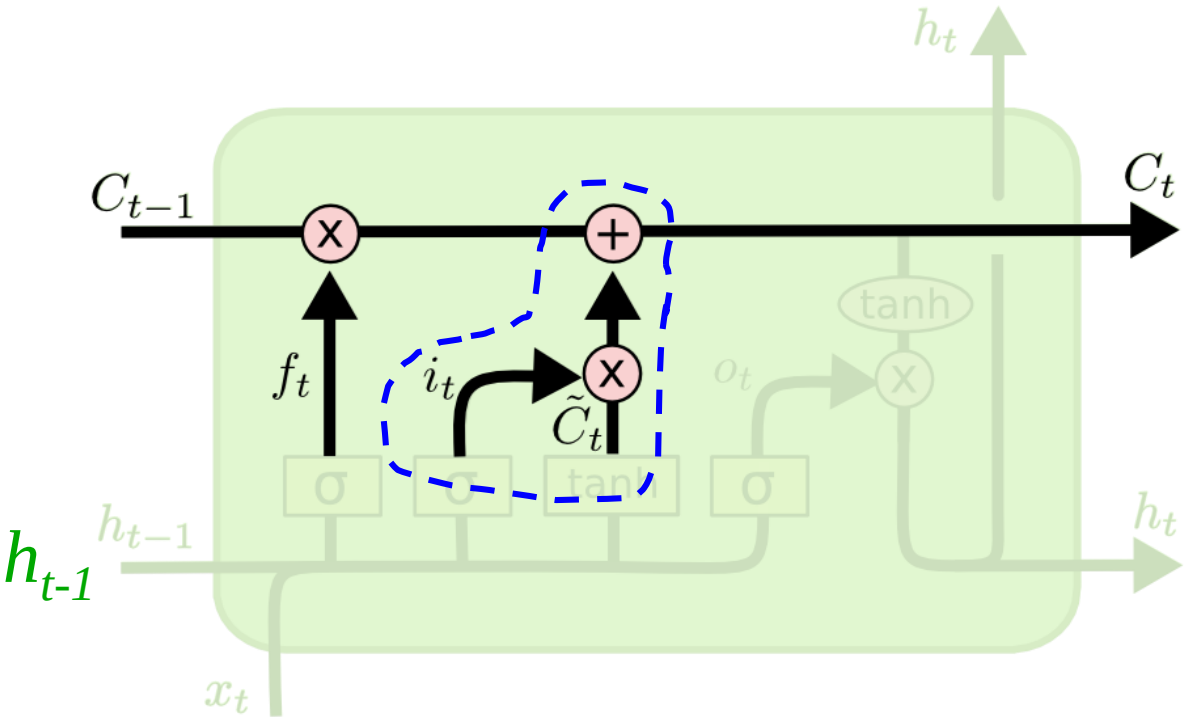
Va de -1 (enlever) à +1 (ajouter)

Illustrations : Christopher Olah, avec permission

LSTM : Étape 3

- La cellule est mise-à-jour indirectement par l'état caché h_{t-1}
  - voir comme des ajustements incrémentaux (résiduel)

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



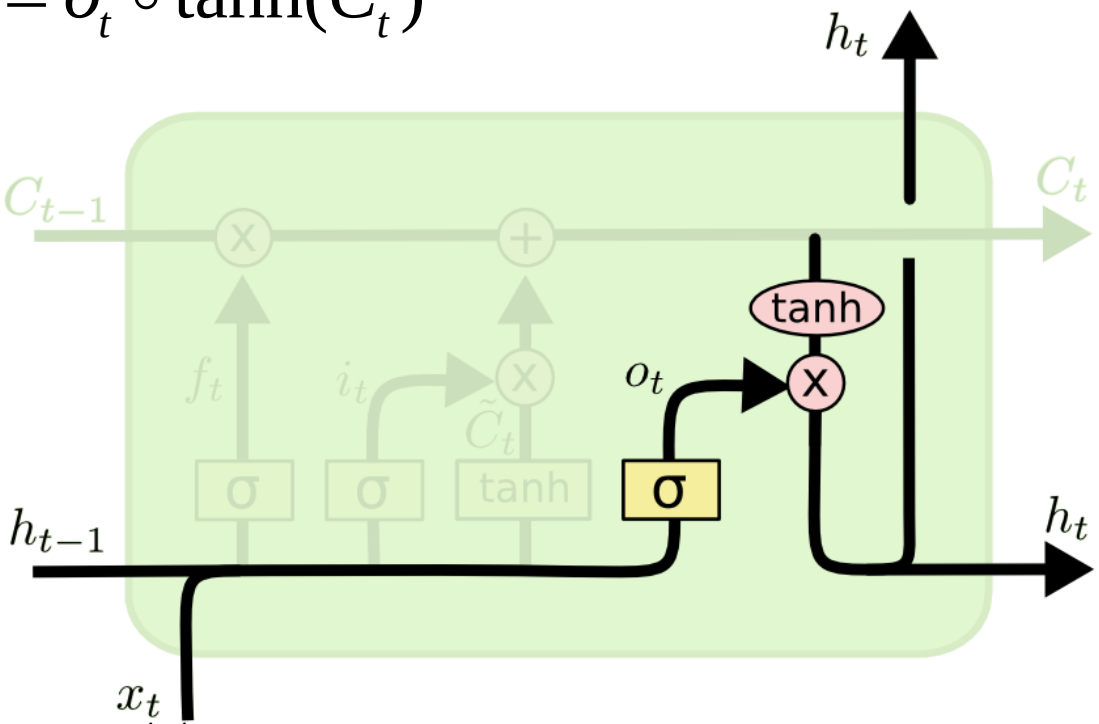
Illustrations : Christopher Olah, avec permission

LSTM : Étape 4

- h_t est une sortie de $tanh(c_t)$
- Modulée par l'output gate o_t

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

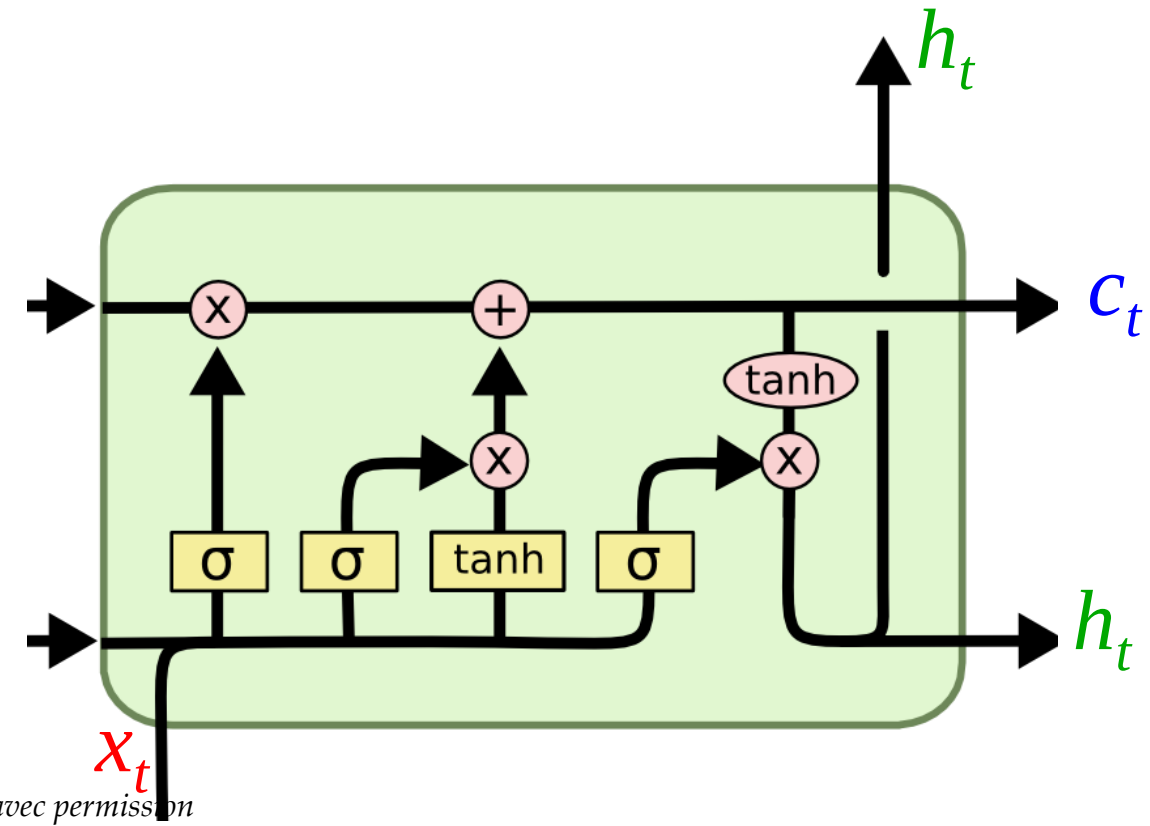
$$h_t = o_t \circ \tanh(C_t)$$



Illustrations : Christopher Olah, avec permission

LSTM

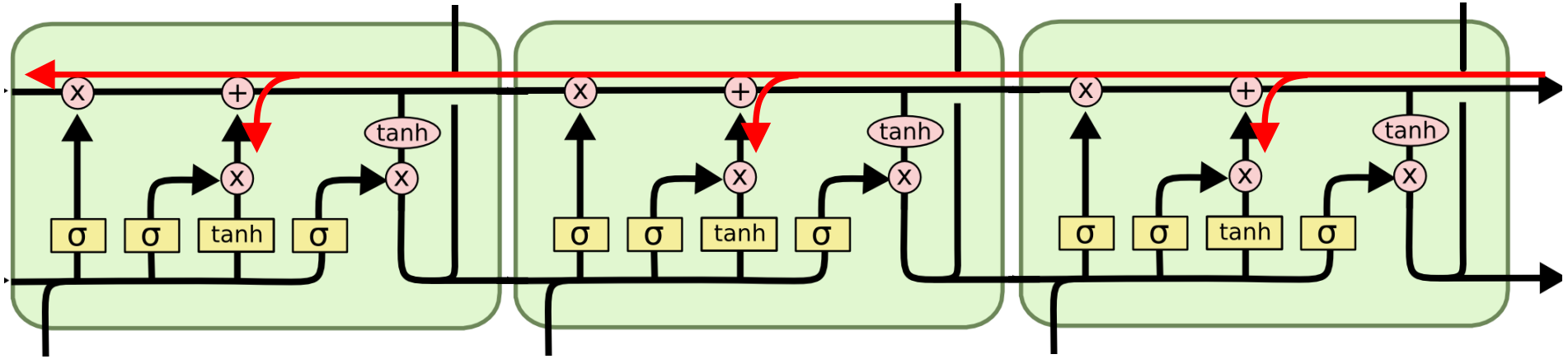
- La cellule c_t est affectée lentement (**slow state**)
- L'état h_t est affecté plus rapidement (**fast state**)



Illustrations : Christopher Olah, avec permission

Flot du gradient

- Gradient se propage mieux que RNN
  - via la cellule
  - pensez ResNet
- RNN : le gradient doit traverser un *tanh*

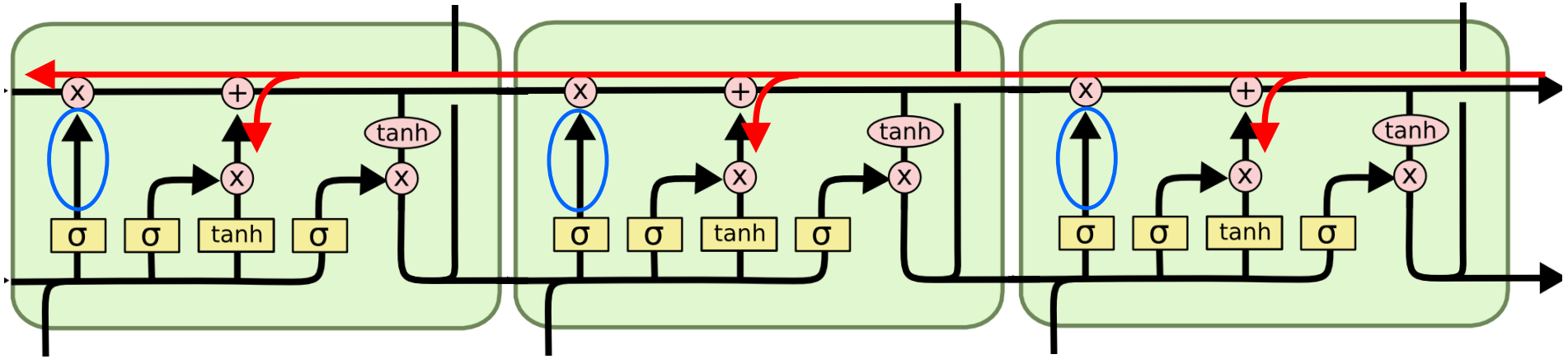


(Aussi à condition que *forget gate* ait des entrées proches de 1)

Illustrations adaptée de Christopher Olah, avec permission

Flot du gradient

- Gradient est multiplié par la sortie du forget gate
  - qui prend des valeurs différentes à chaque itération
  - évite le problème du gradient explosif ou évanescent du RNN : W^n



Illustrations adaptée de Christopher Olah, avec permission

Exemple : entraîné sur code C

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

- 474 Mo de code C sur Linux repo
- 10 millions de paramètres dans LSTM 3 couches
- Des cellules vont se spécialiser à :
  - Compter les incréments
  - Ouvertures/fermetures de parenthèses, accolade, etc.

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

Exemple : entraîné sur code C

- Capable de réciter la licence GNU!

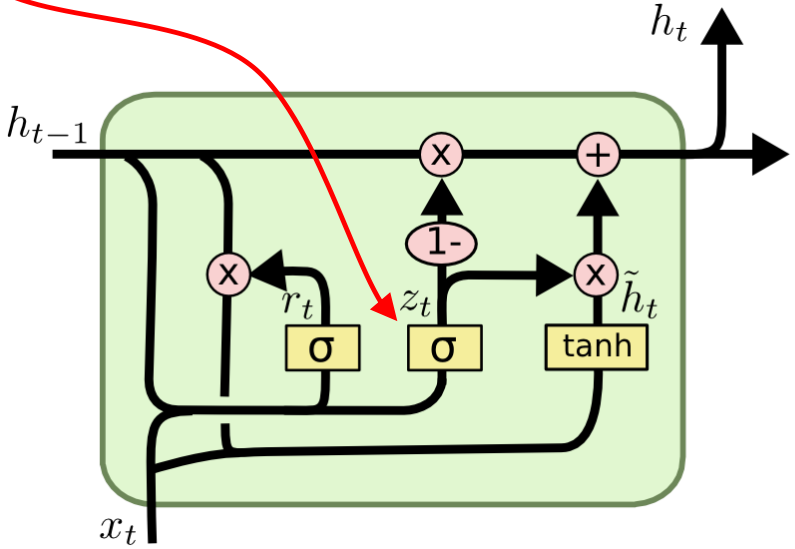
```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)
```

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

Variantes de LSTM

GRU : Gated Recurrent Unit

- Combine forget et input gate ensemble
  - update gate
  - « on oublie seulement si on modifie »
- Plus de séparation hidden *h* /cell *c*



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Moins de paramètres

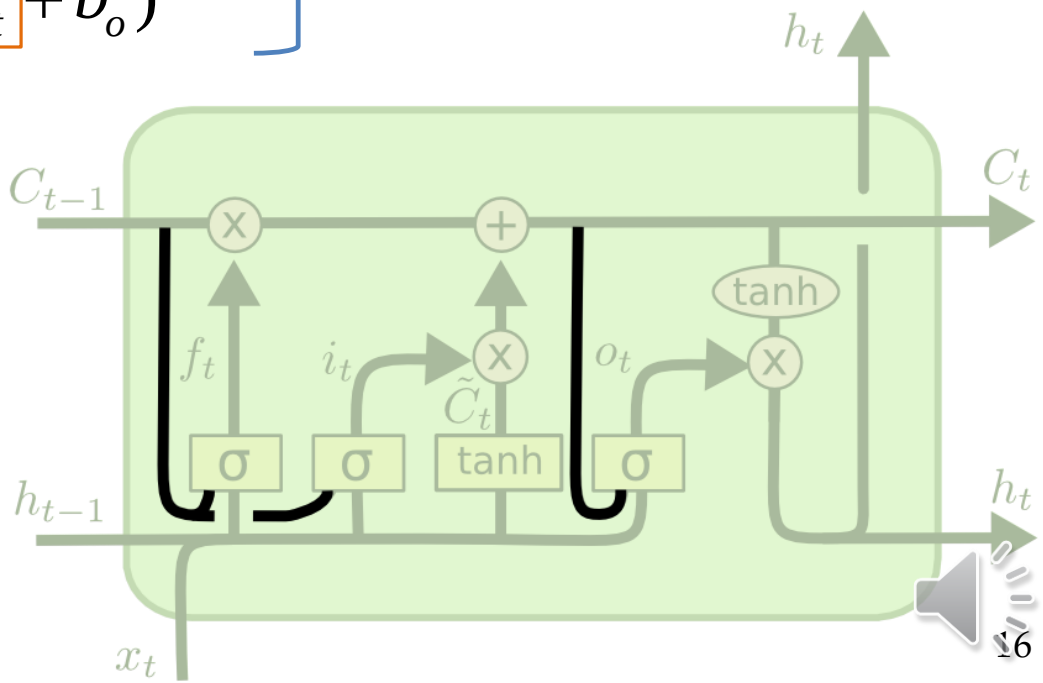
Illustrations : Christopher Olah, avec permission

Peephole connection

- Pour permettre à l'état de la cellule de contrôler les *gates*

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + P_f c_{t-1} + b_f)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + P_i c_{t-1} + b_i)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + P_o c_t + b_o)$$

Note : certaines variantes, U = 0



Illustrations : Christopher Olah, avec permission

LSTM: A Search Space Odyssey

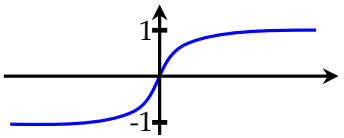
- 5400 tests de 8 variantes d'architecture sur 3 tâches :
  - modélisation acoustique
  - reconnaissance d'écriture manuscrite
  - modélisation musique polyphonique
- Aucune variante ne domine réellement
  - variante GRU a l'avantage d'avoir moins de paramètres
- Parties les plus importantes :
  - forget gate
  - output activation

An Empirical Exploration of Recurrent Network Architectures

- Essais de 10,000 architectures trouvées par processus d'évolution génétique (mutation)
- Ont identifié une architecture qui parfois dépasse le LSTM et le GRU, mais sur certaines tâches seulement
- Bref, LSTM/GRU encore compétitif!
  - **réduit l'écart GRU-LSTM en ajoutant un biais de b_f=1 pour le *forget gate* du LSTM**

Conclusion

- RNN et les LSTM (variantes) peuvent traiter des séquences de données de longueur indéterminée
- RNN vanille est facile à implémenter, mais ne marche pas toujours bien en pratique
  - *tanh* qui n'a pas un bon profil de gradient
  - Vanishing ou exploding gradient dû à la multiplication répétée de W



- LSTM : les gradients percolent beaucoup mieux, via la cellule c
  - Soumise à des simples opérations linéaires + ou ×

Évolution

RNN → LSTM → Attention