Réseaux de neurones

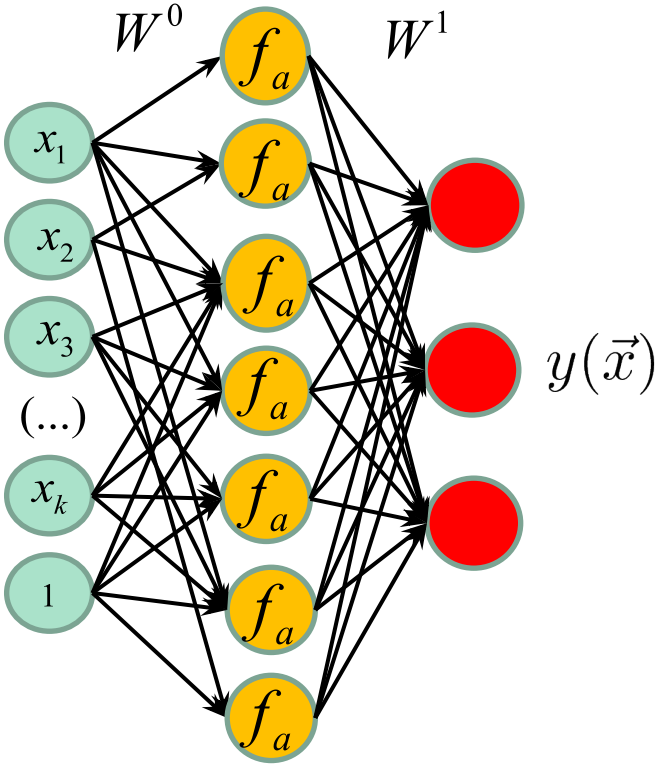
IFT 780

Réseaux récurrents

Par

Pierre-Marc Jodoin

Réseau de neurones de base (régression)



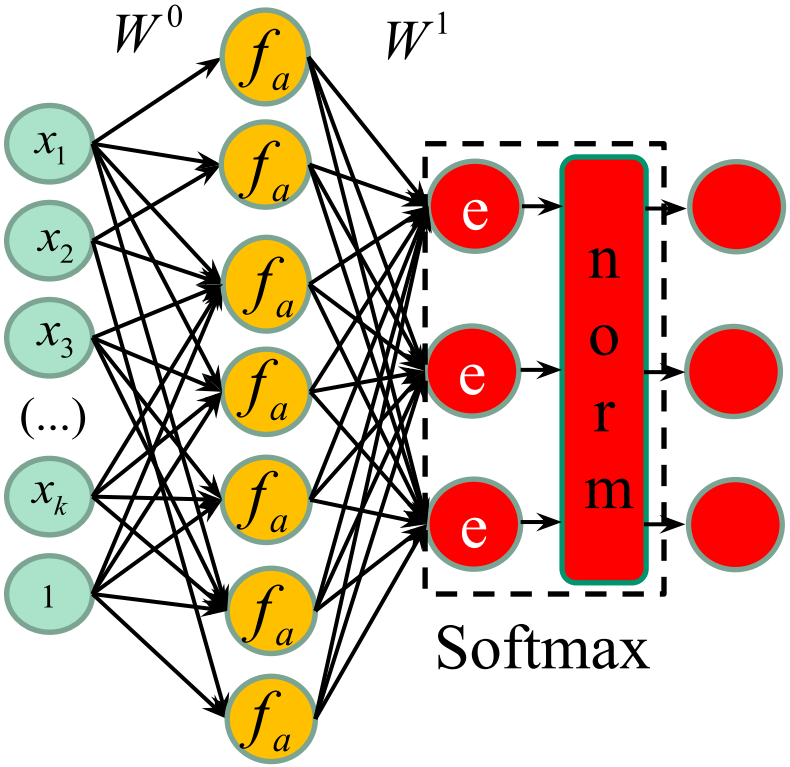
$$y(\vec{x}) = W^1 f_a(W^0 \vec{x})$$

$$\vec{h} = f_a(W^0 \vec{x})$$

$$y(\vec{x}) = W^1 \vec{h}$$

f_a : fonction d'activation

Réseau de neurones de base (classification)



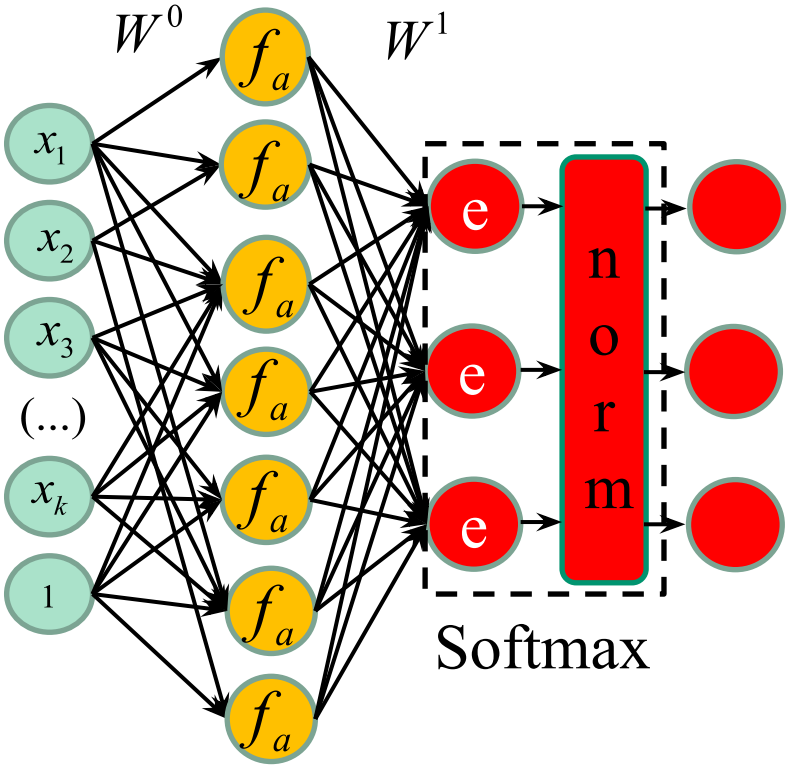
$$y(\vec{x}) = \text{softmax}(W^1 f_a(W^0 \vec{x}))$$

$$\vec{h} = f_a(W^0 \vec{x})$$

$$\hat{y} = W^1 \vec{h}$$

$$y(\vec{x}) = \text{softmax}(\hat{y})$$

Réseau de neurones de base (classification)



$y(\vec{x})$

$$y(\vec{x}) = \text{softmax}(W^1 f_a(W^0 \vec{x}))$$

$$\vec{h} = f_a(W^0 \vec{x})$$

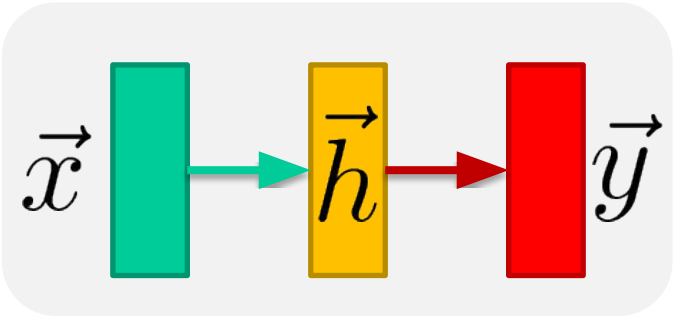
$$\hat{y} = W^1 \vec{h}$$

$$y(\vec{x}) = \text{softmax}(\hat{y})$$

Ne permettent que des tâches “1 pour 1”

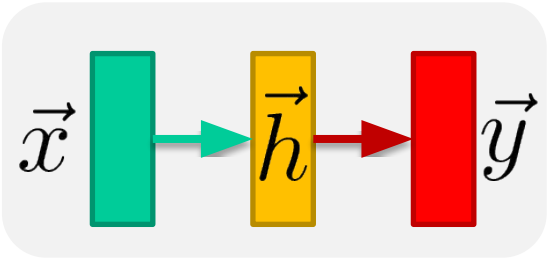
- Classification (1 image = 1 label)
- Régression (1 donnée = 1 chiffre)
- Localisation (1 boîte = 1 classification + 1 régression)

Illustration simplifiée



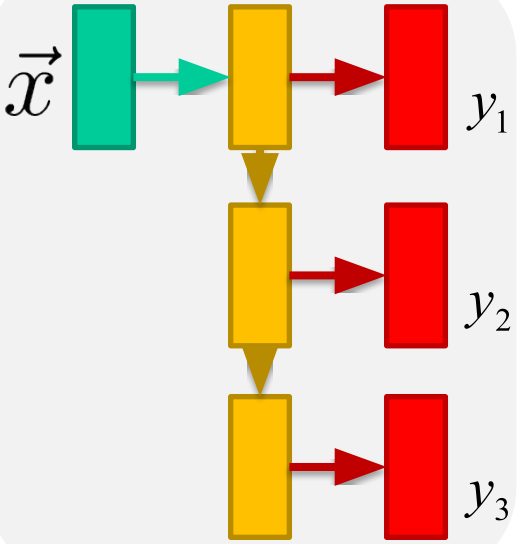
RN de base

Différentes configurations pour différentes applications



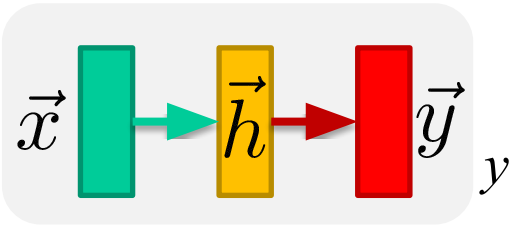
RN de base

1 entrée et N sorties



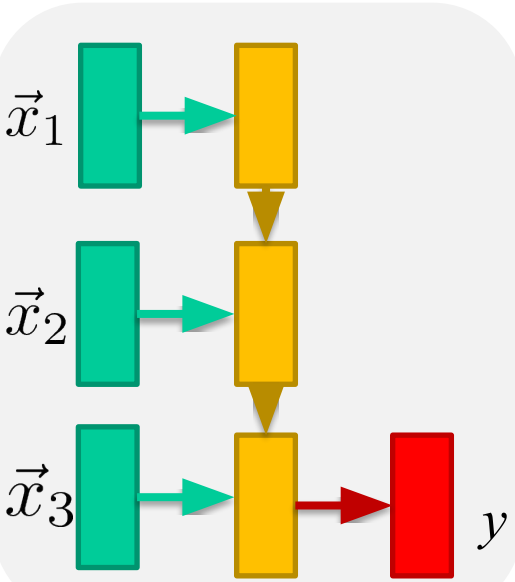
Ex.: description d'une image
1 image => N mots

Différentes configurations pour différentes applications



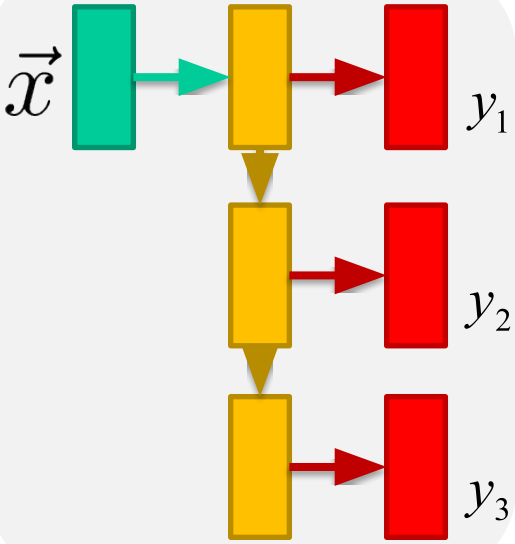
RN de base

1 entrée et 1 sorties



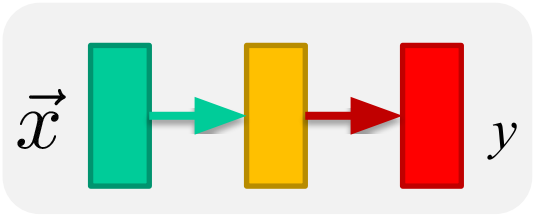
Ex.: Classification de texte
N mots => 1 classe

1 entrée et N sorties

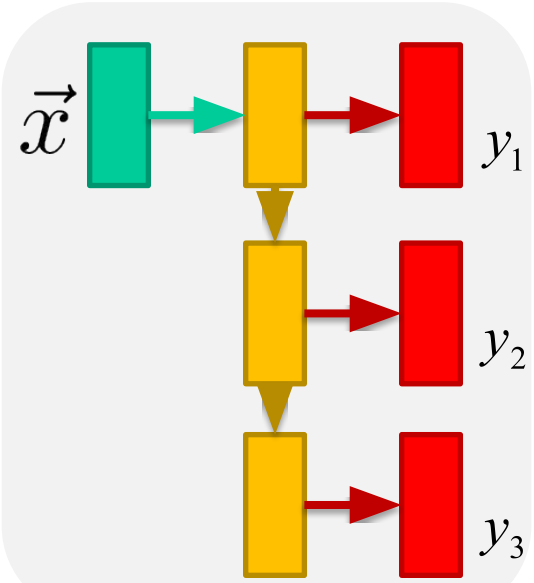


Différentes configurations pour différentes applications

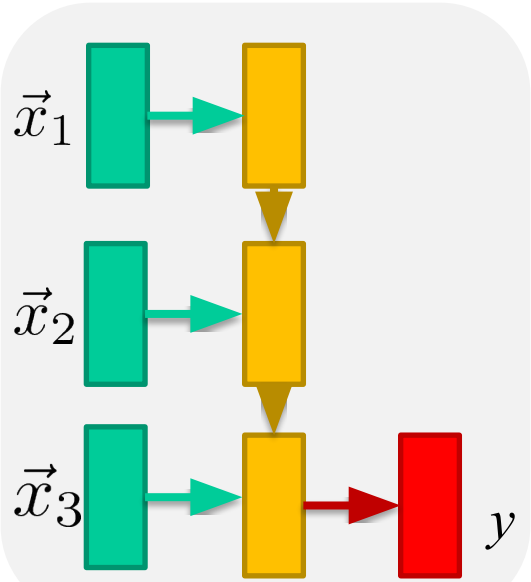
1 entrée et 1 sortie



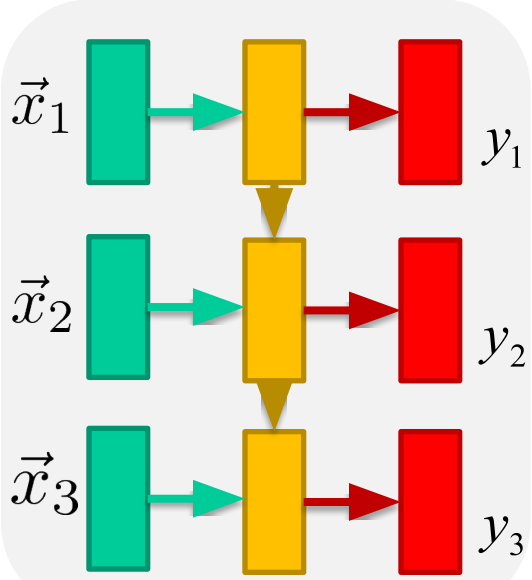
1 entrée et N sorties



N entrées et 1 sortie



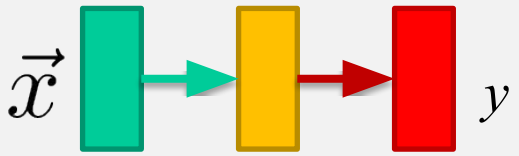
N entrées et N sorties



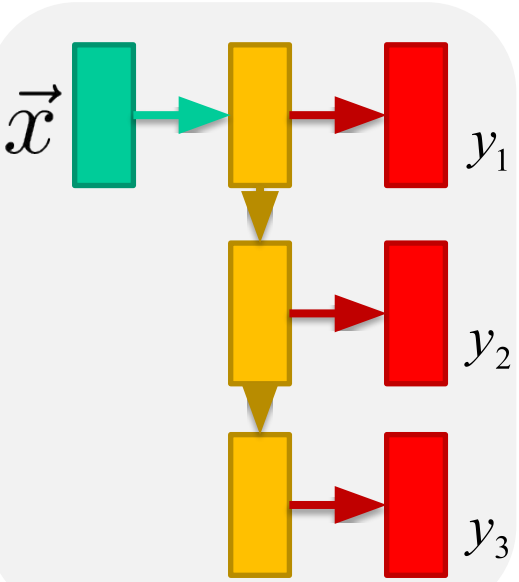
Ex.: Classification d'images vidéo
N images => N prédictions

Différentes configurations pour différentes applications

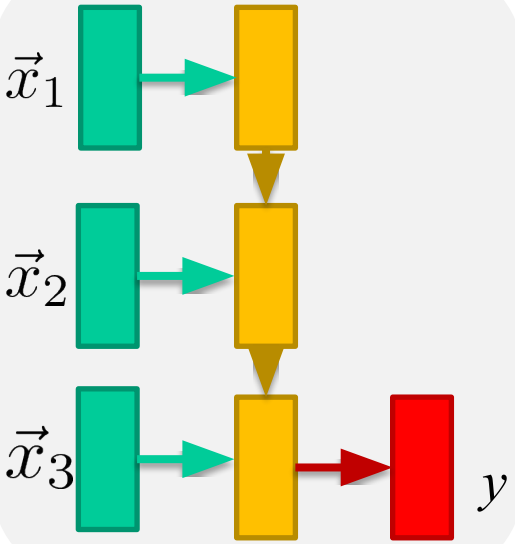
1 entrée et 1 sortie



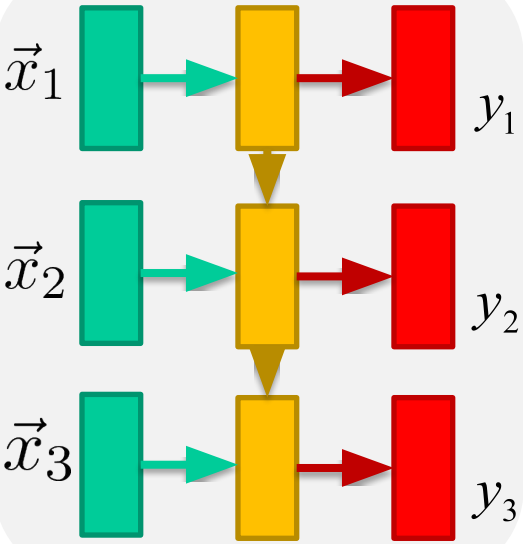
1 entrée et N sorties



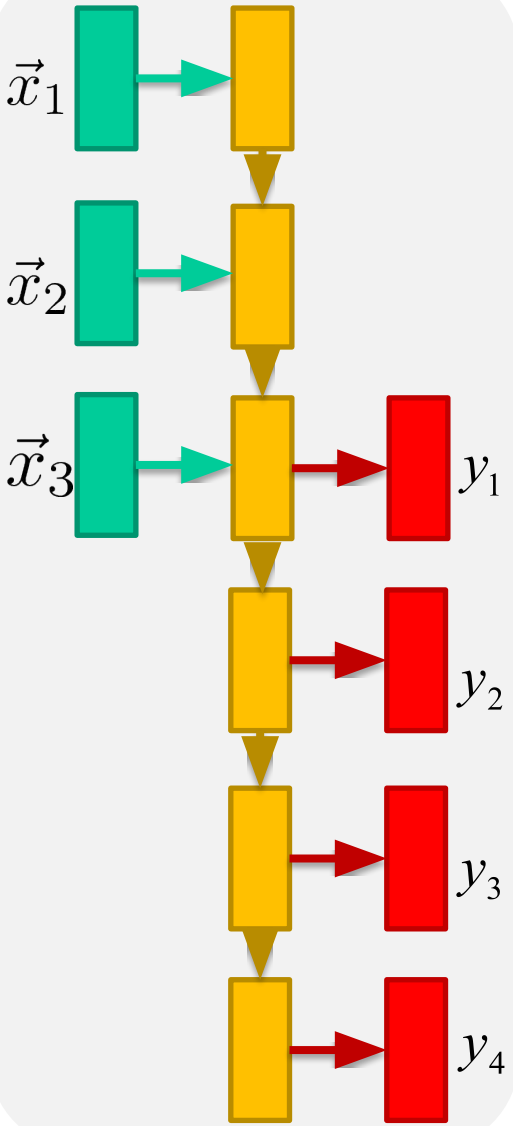
1 entrée et N sorties



N entrées et N sorties

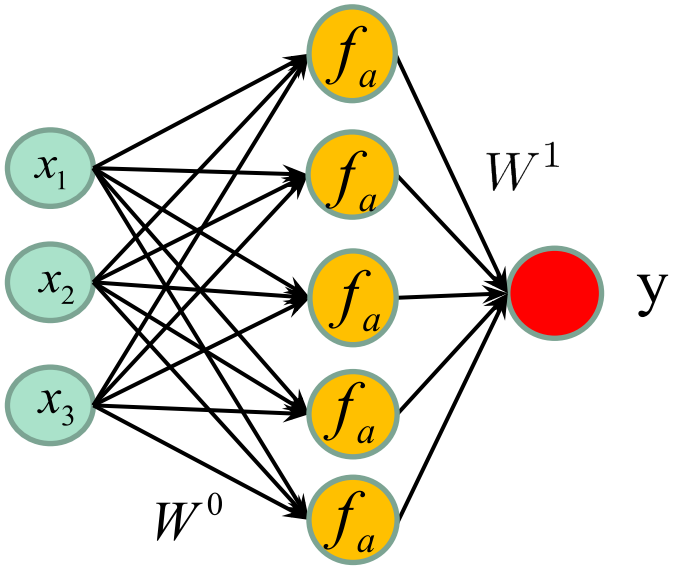


M entrées et N sorties

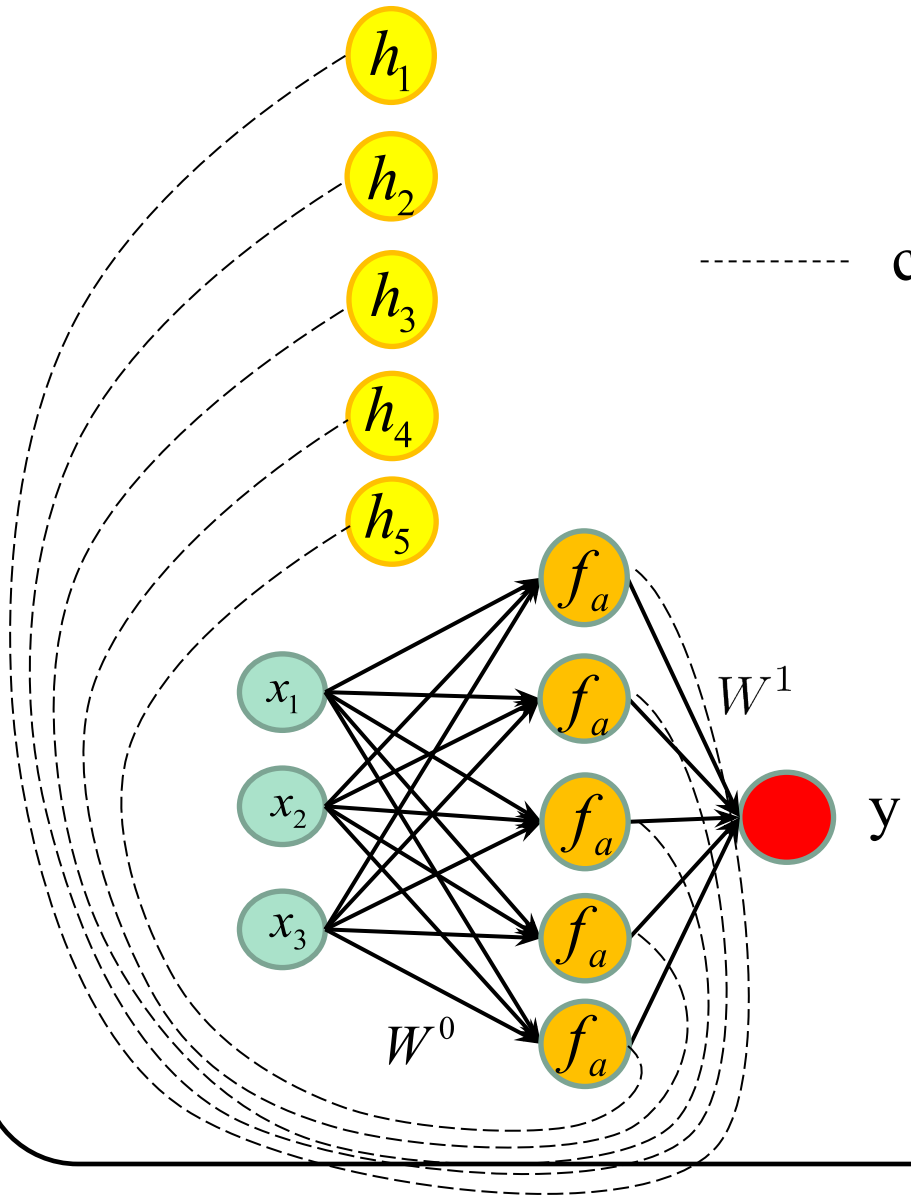


Ex.: Traduction Français-Anglais
N mots => M mots

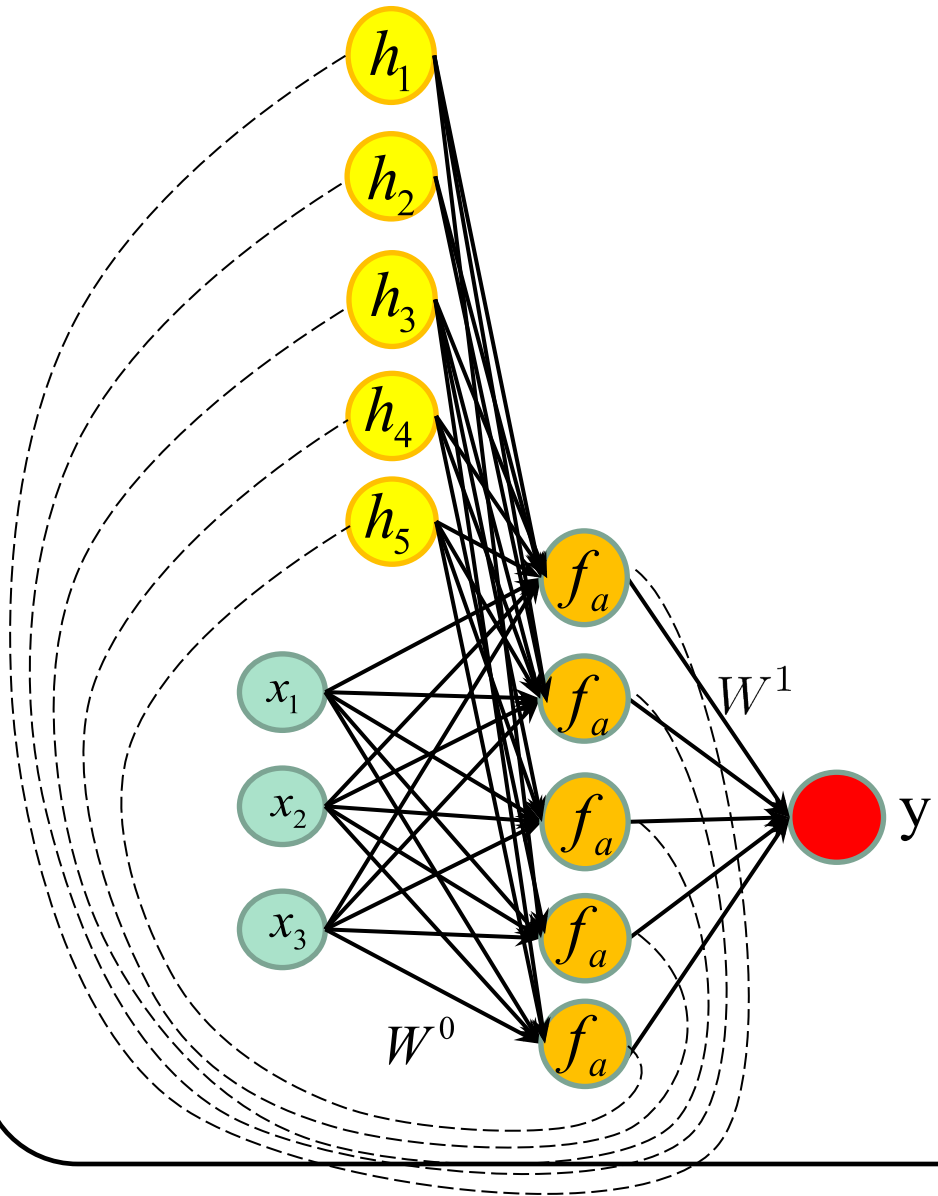
Réseau récurrent : la sortie des neurones est réinjectée dans leur entrée



Réseau récurrent : la sortie des neurones est réinjectée dans leur entrée



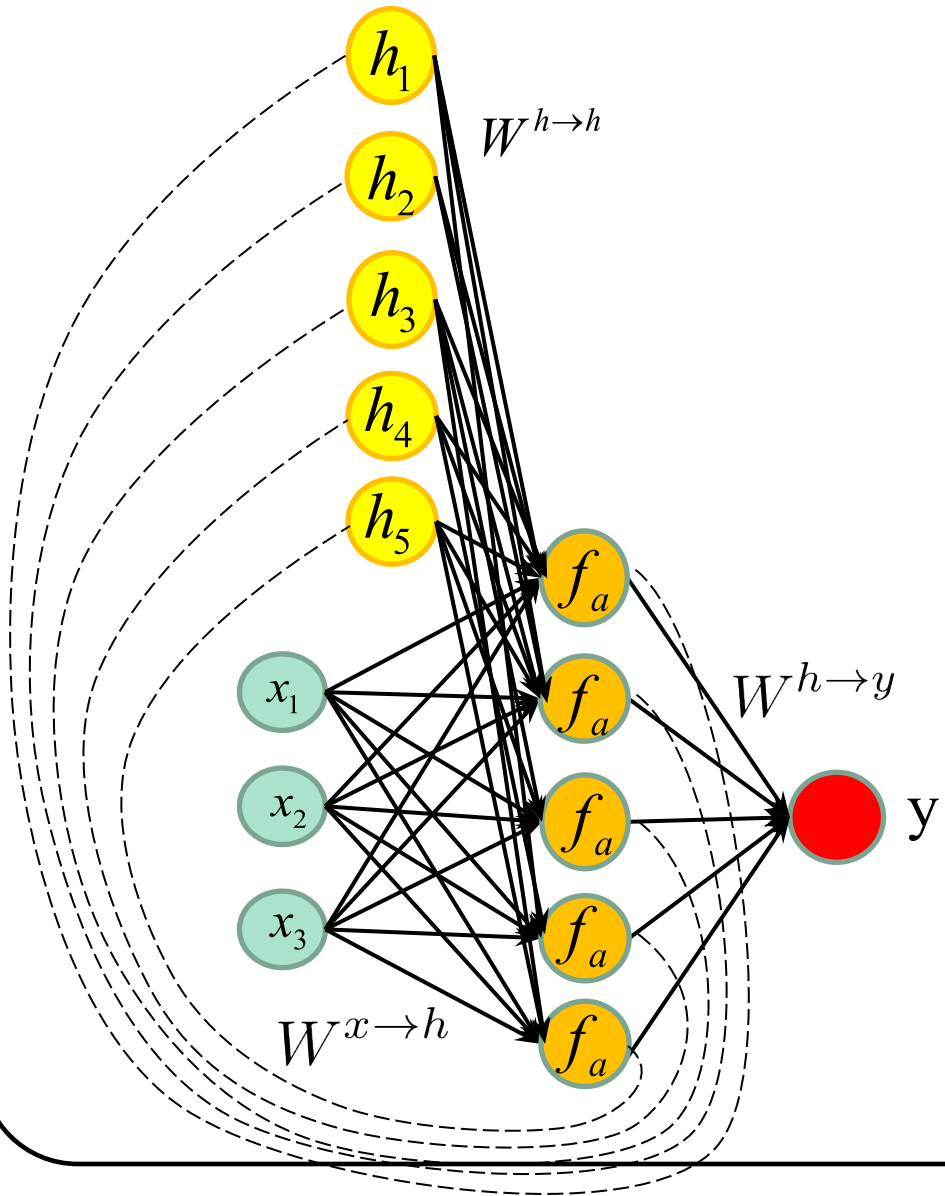
Réseau récurrent : la sortie des neurones est réinjectée dans leur entrée



copie

Ici, au lieu d'avoir 3 entrées, chaque neurone a **3+5=8 entrées.**

Réseau récurrent : la sortie des neurones est réinjectée dans leur entrée

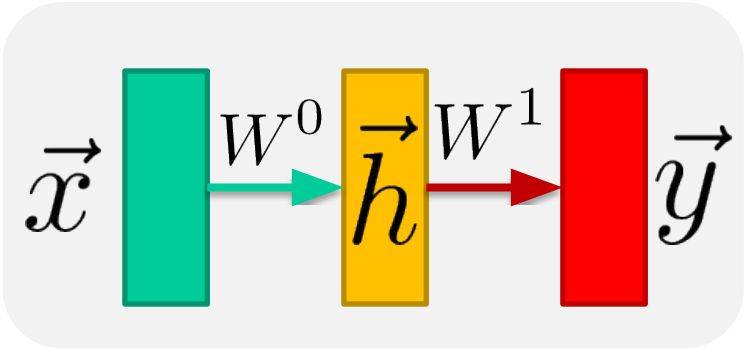


$$y(\vec{x}) = W^{h\to y} f_a(W^{x\to h}\vec{x} + W^{h\to h}\vec{h})$$

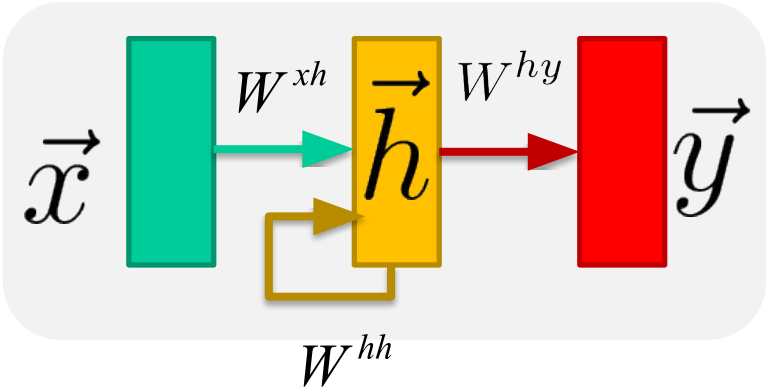
$$\vec{h} = f_a(W^{x\to h}\vec{x} + W^{h\to h}\vec{h})$$

$$y(\vec{x}) = W^{h\to y}\vec{h}$$

Illustration simplifiée

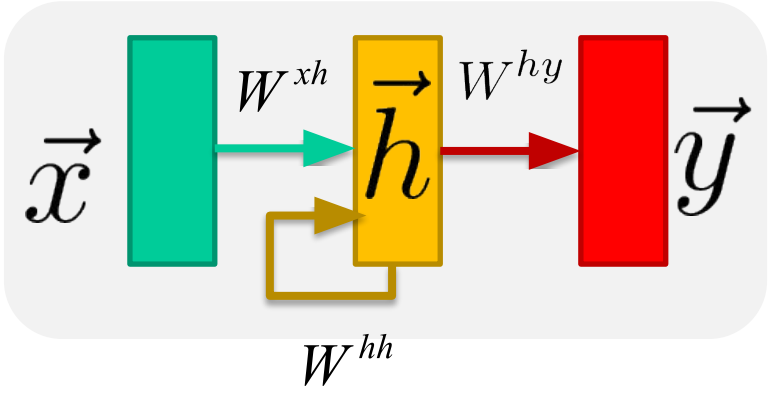


RN de base



RN récurrent

Dans le cas général avec K sorties (régression)

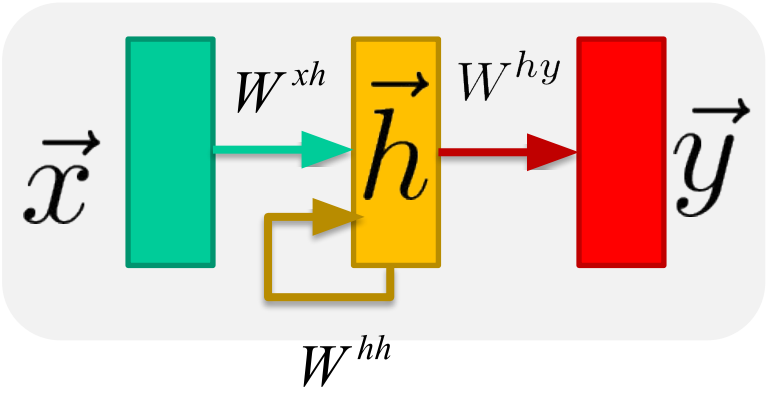


$$y(\vec{x}) = W^{hy} f_a(W^{xh}\vec{x} + W^{hh}\vec{h})$$

$$\vec{h} = f_a(W^{xh}\vec{x} + W^{hh}\vec{h})$$

$$\vec{y}(\vec{x}) = W^{hy}\vec{h}$$

Dans le cas général avec K sorties (classification)



$$y(\vec{x}) = W^{hy} f_a(W^{xh}\vec{x} + W^{hh}\vec{h})$$

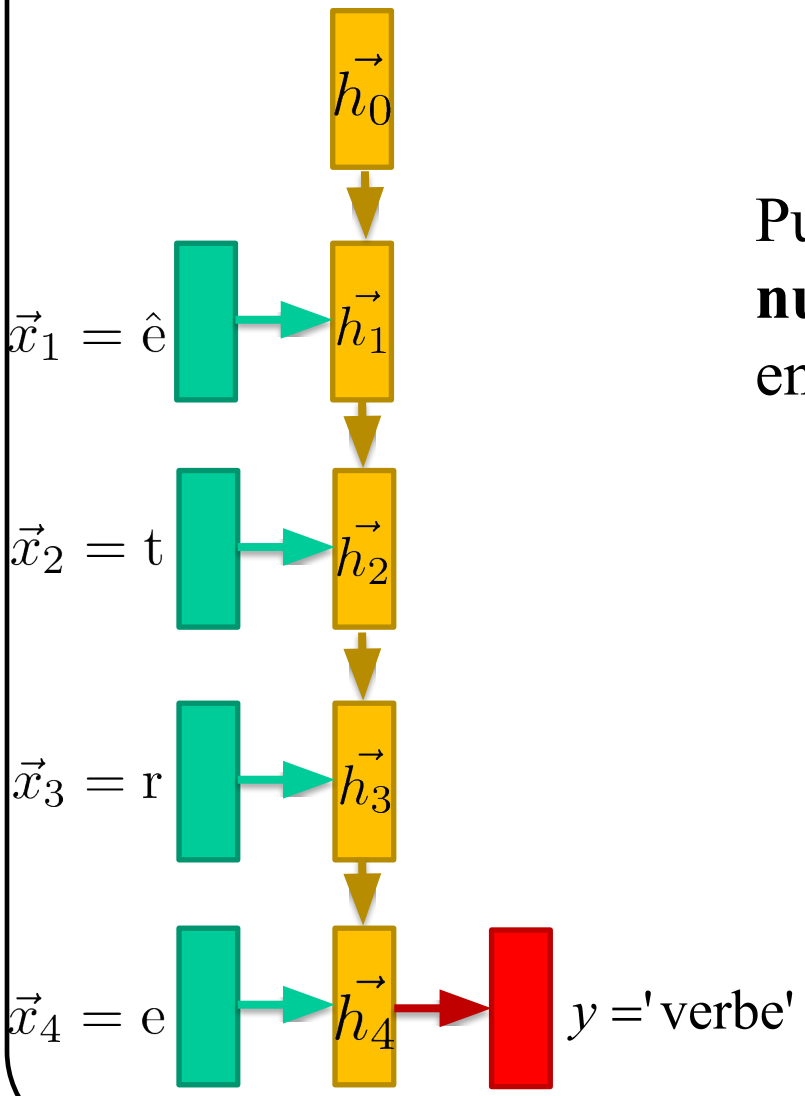
$$\vec{h} = f_a(W^{xh}\vec{x} + W^{hh}\vec{h})$$

$$\hat{y} = W^{hy}\vec{h}$$

$$\vec{y}(\vec{x}) = \text{softmax}(\hat{y})$$

Exemple pour N entrées et 1 sortie:

Analyse grammaticale (classification) : (ê.t.r.e)=>'verbe'



Puisque $\vec{x}, \vec{h}$ et y doivent être des **variables numériques** on utilise généralement un encodage de type « *one hot* ».

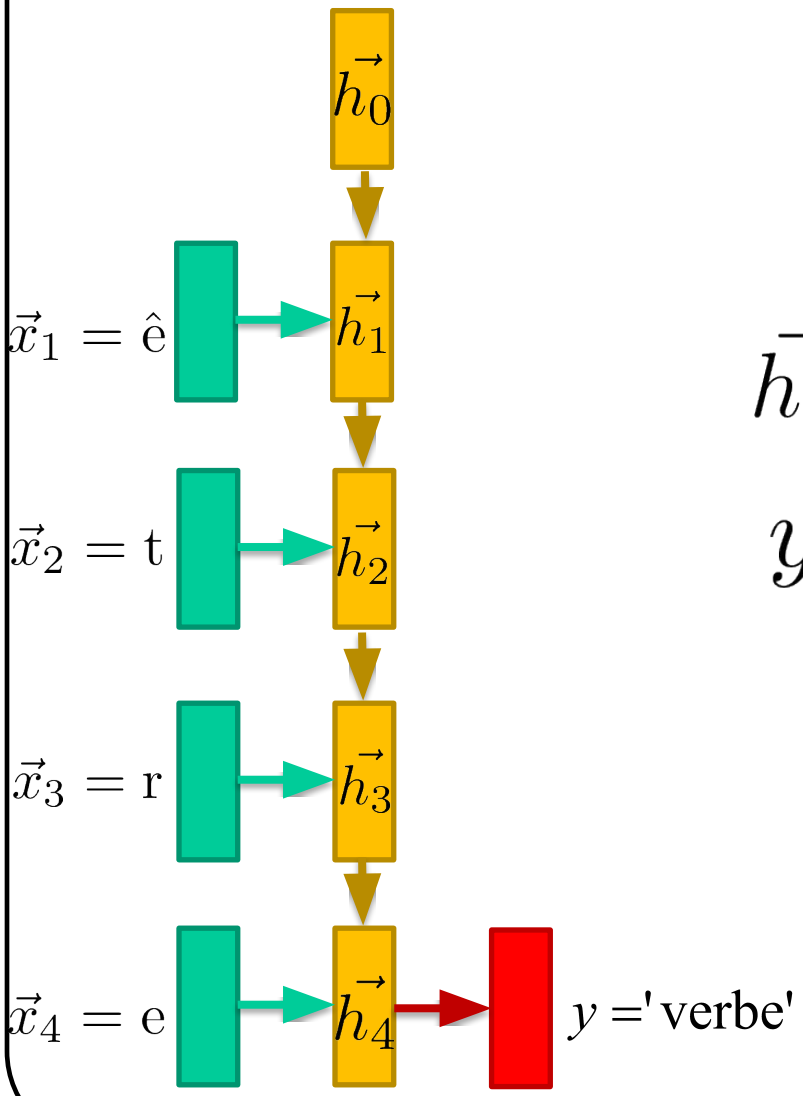
$$\left.\begin{array}{l} 'a' = [1,0,0,...,0] \\ 'b' = [0,1,0,...,0] \\ 'c' = [0,0,1,...,0] \end{array}\right\} \in R^{256}$$

....

$$\left.\begin{array}{l} 'verbe' = [1,0,0...0] \\ 'nom' = [0,1,0,...0] \\ 'adjectif' = [0,0,1,...0] \end{array}\right\} \in R^{M}$$

Exemple pour N entrées et 1 sortie:

Analyse grammaticale (classification) : (ê.t.r.e)=>'verbe'

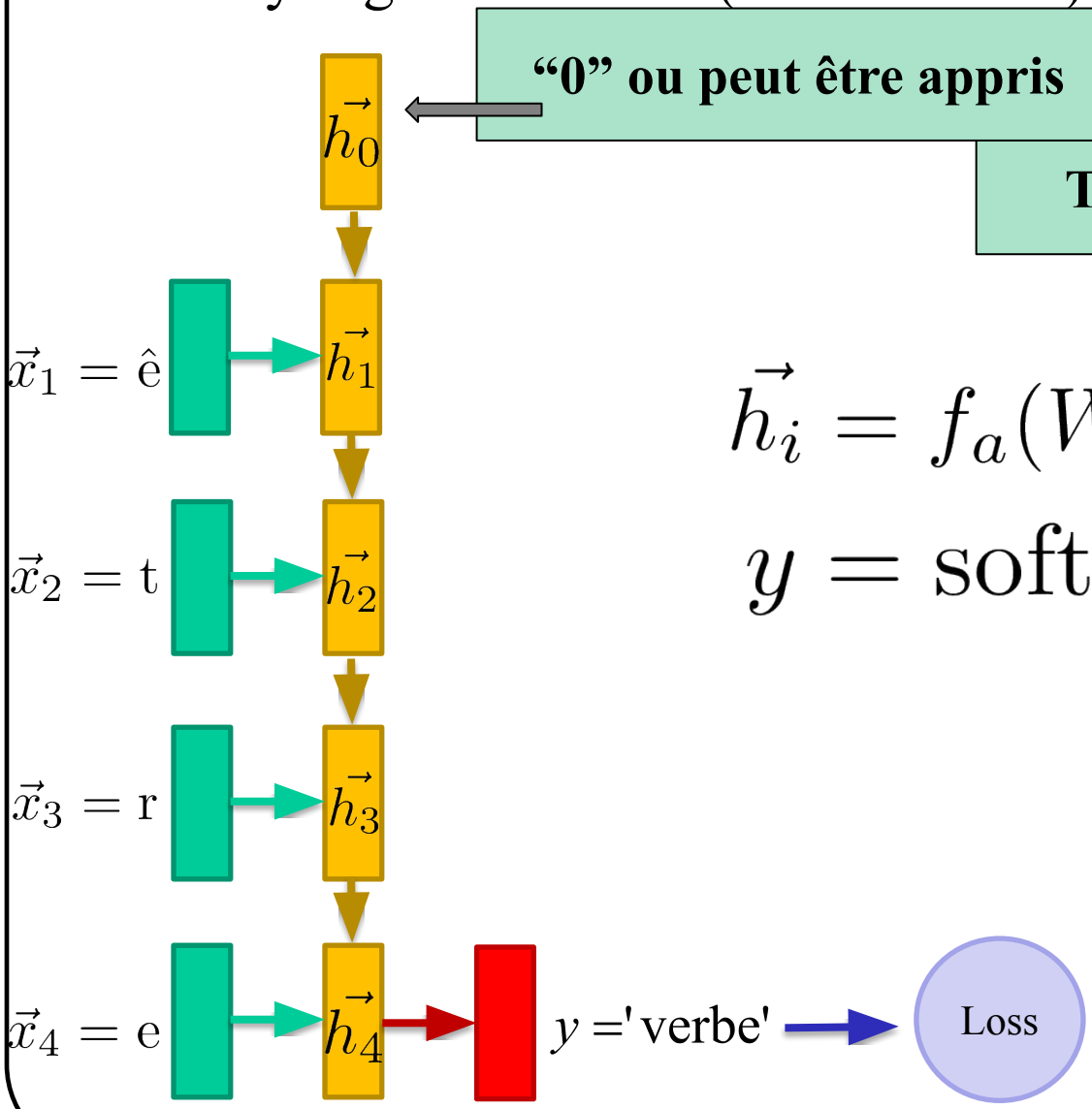


$$\vec{h_i} = f_a(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

$$y = \text{softmax}(W^{hy}\vec{h}_{i-1})$$

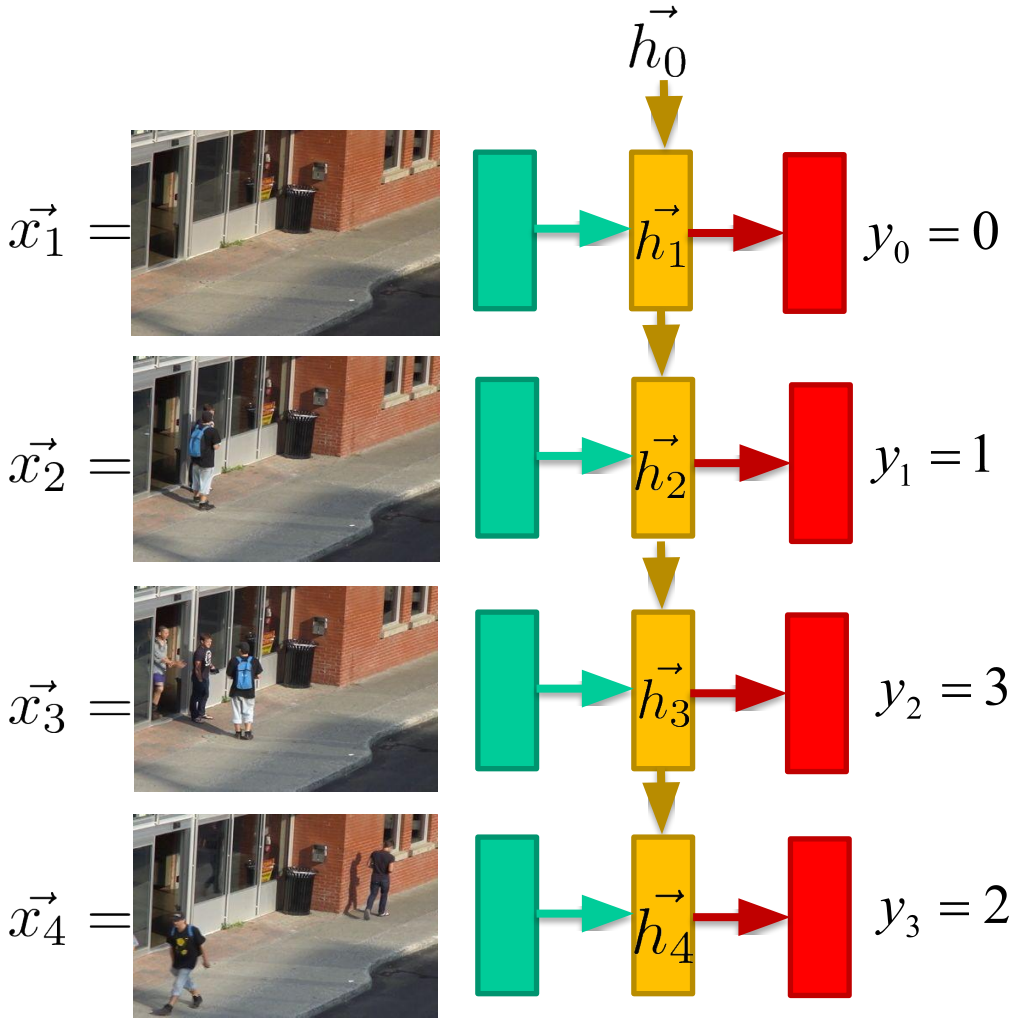
Exemple pour N entrées et 1 sortie:

Analyse grammaticale (classification) : (ê.t.r.e)=>'verbe'



Même idée pour N entrées et N sorties:

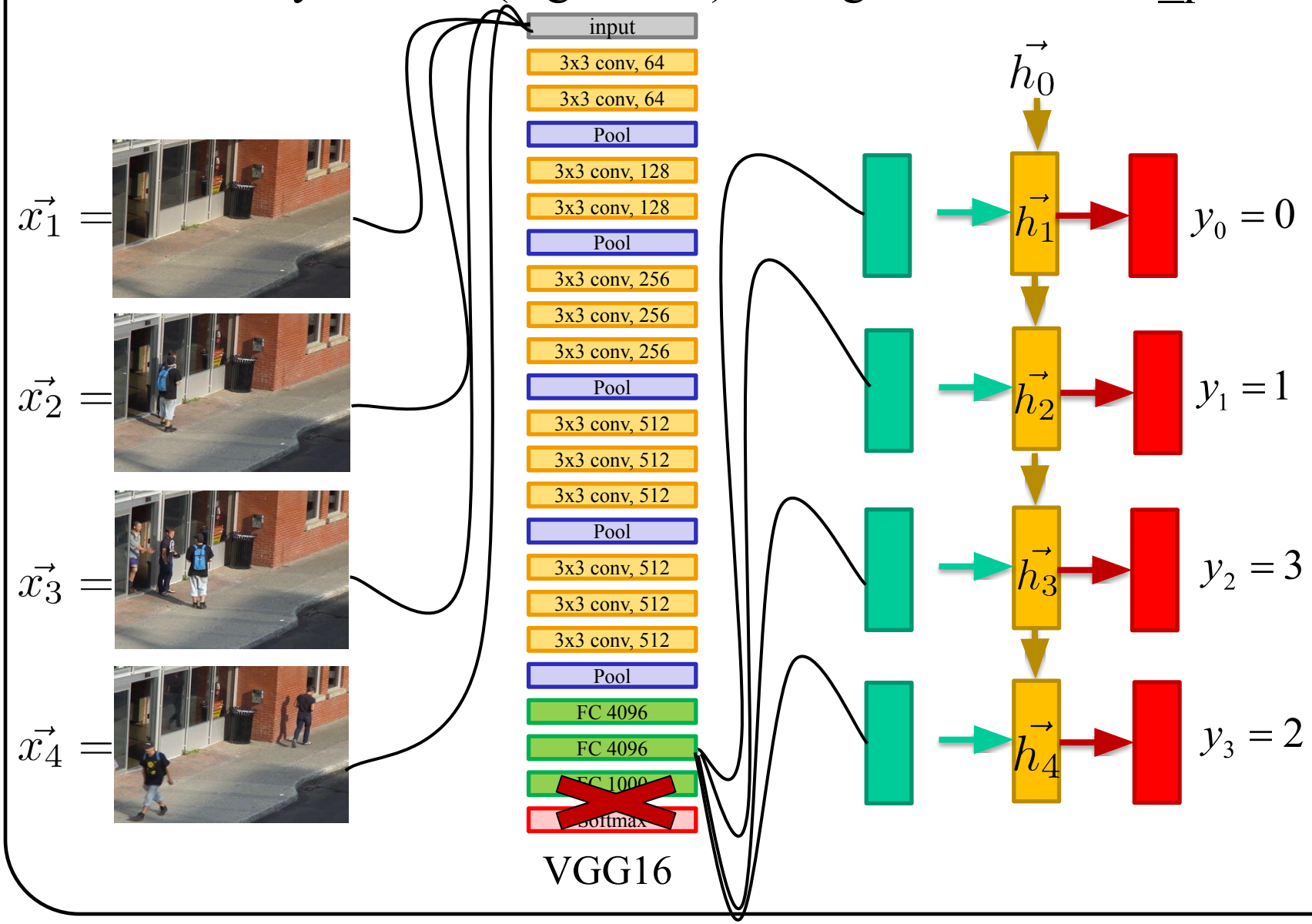
ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons



$$\vec{h_i} = f_a(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$y_i = W^{hy}\vec{h_i}$$

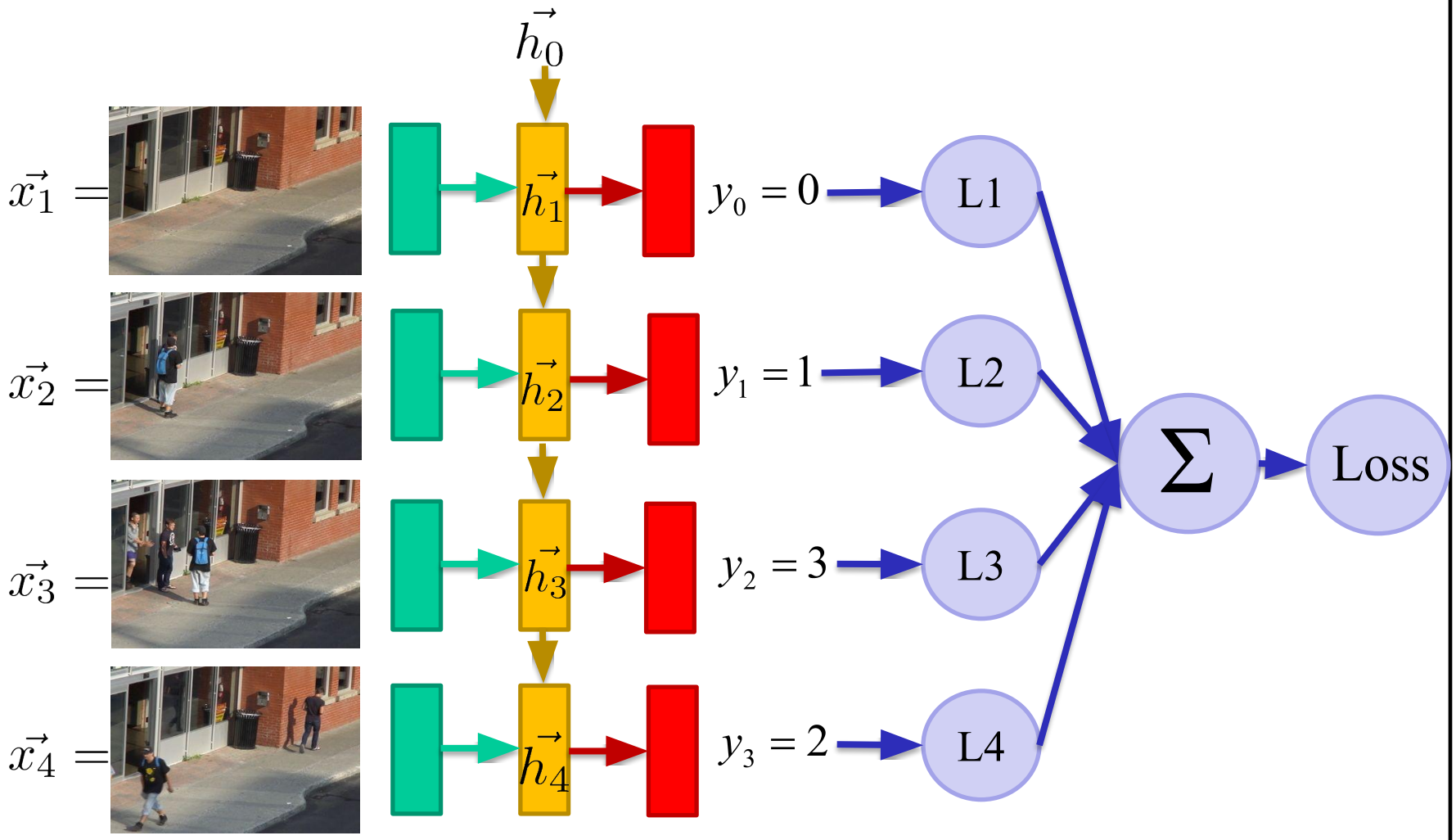
Même idée pour N entrées et N sorties:

ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons



Même idée pour N entrées et N sorties:

ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons



Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet jouet :[**a,e,m,s**]

Représentation « one hot » jouet:

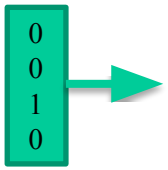
‘a’ = [1, 0, 0, 0]
‘e’ = [0, 1, 0, 0]
‘m’ = [0, 0, 1, 0]
‘s’ = [0, 0, 0, 1]

But : Entraîner un modèle à prédire les lettres du mot « **masse** ».

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».

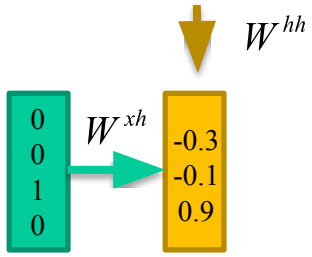
$\vec{x_1} = \mathrm{m}$ 

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».

$\vec{x_1} = \mathrm{m}$



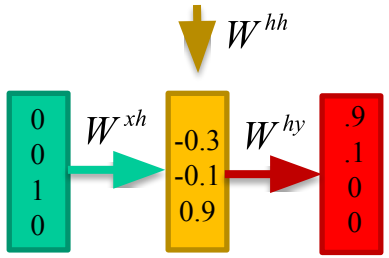
$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».

$\vec{x_1} = \text{m}$



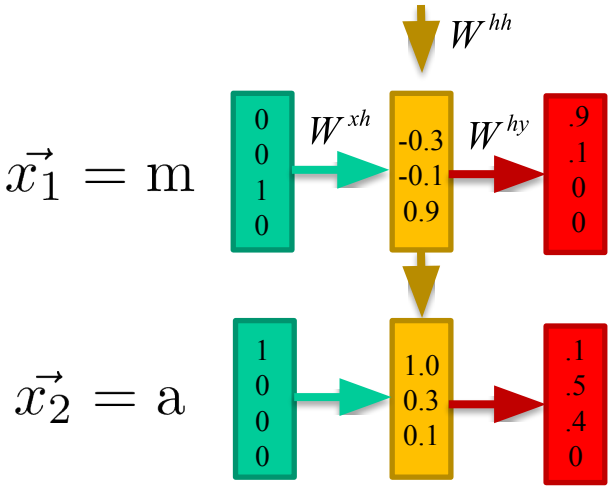
$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

$$\vec{y}(\vec{x_i}) = \text{softmax}(W^{hy}\vec{h_i})$$

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».



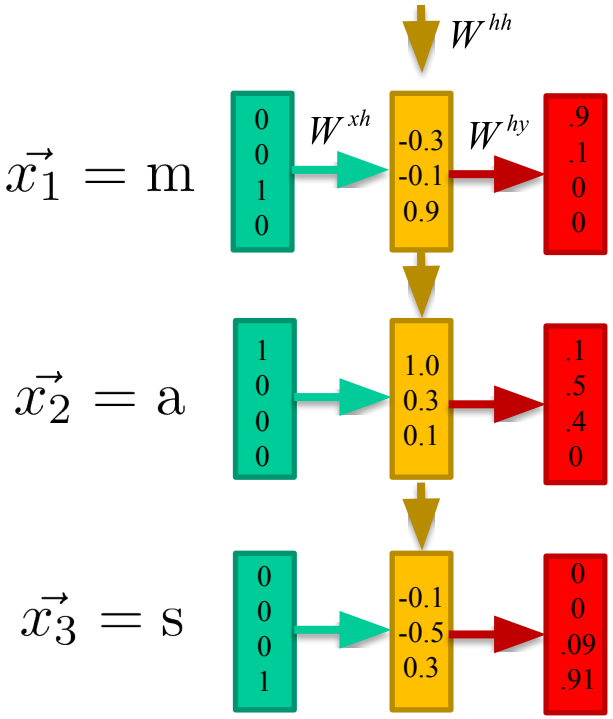
$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

$$\vec{y}(\vec{x_i}) = \mathrm{softmax}(W^{hy}\vec{h_i})$$

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».



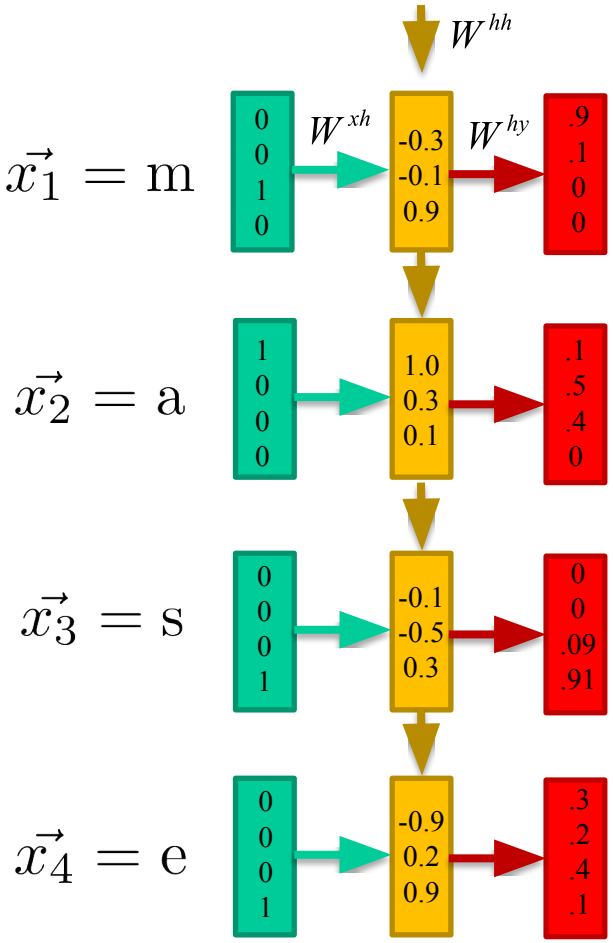
$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

$$\vec{y}(\vec{x_i}) = \mathrm{softmax}(W^{hy}\vec{h_i})$$

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».



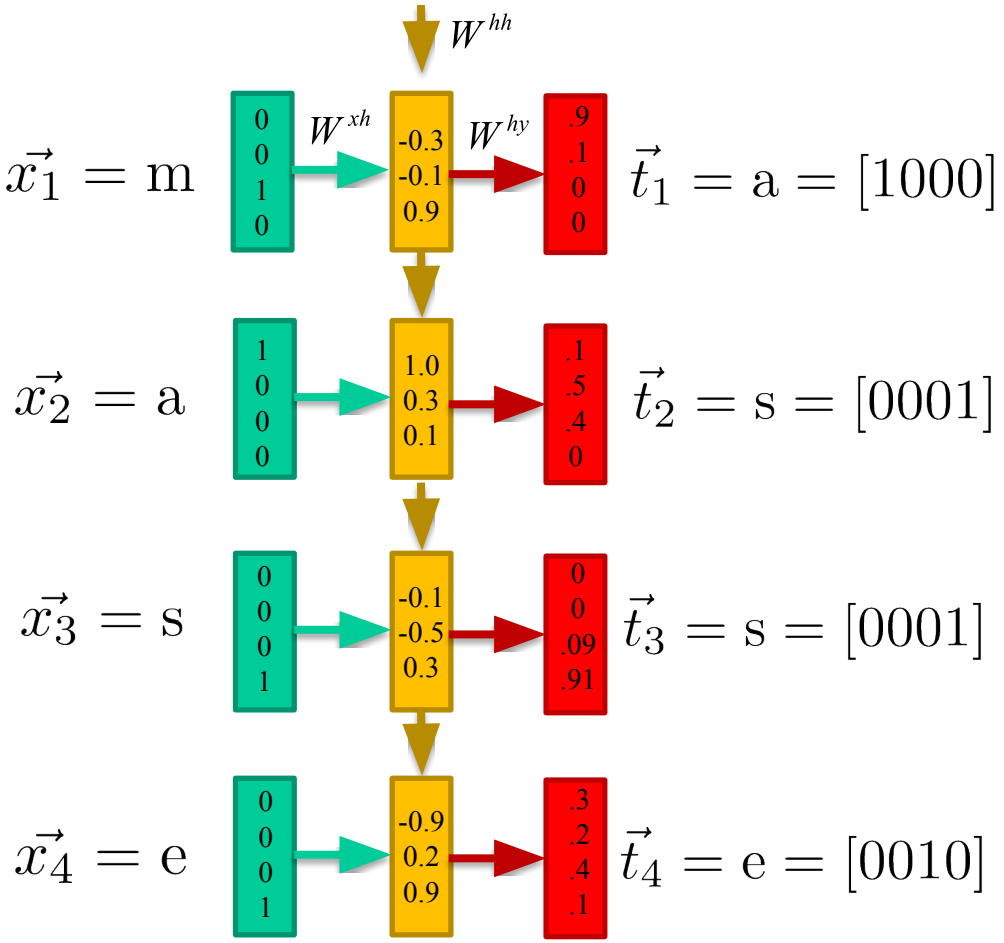
$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

$$\vec{y}(\vec{x_i}) = \mathrm{softmax}(W^{hy}\vec{h_i})$$

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».

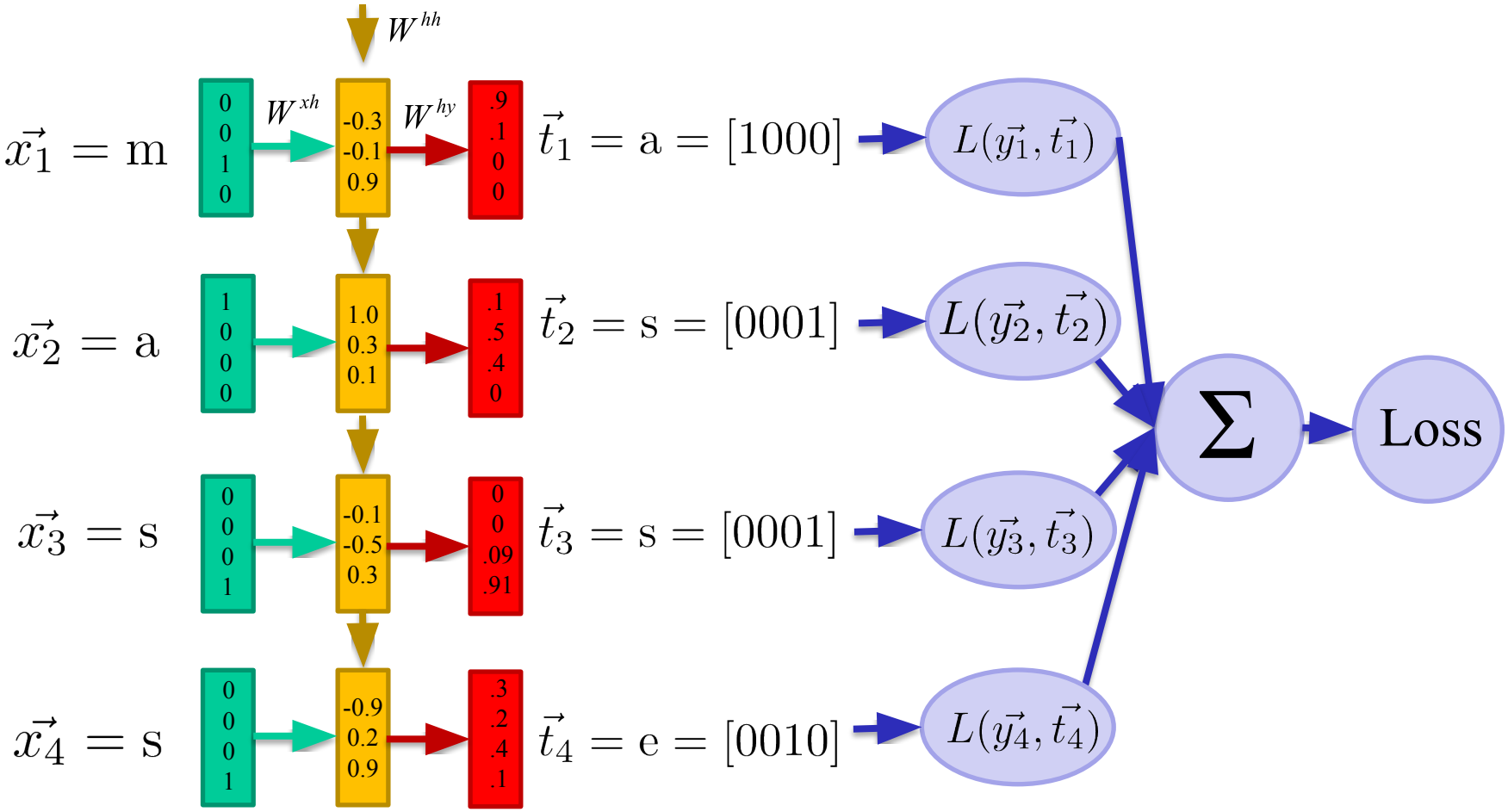


Cibles

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

Entraîner un modèle à prédire les lettres du mot « **masse** ».

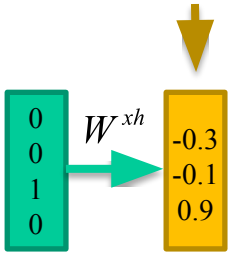


Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres

$\vec{x_1} = \text{m}$



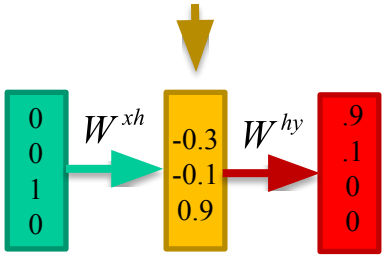
Étape 1 : Calcul de la couche cachée

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres

$\vec{x_1} = \text{m}$

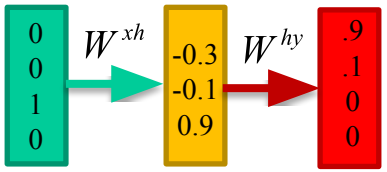


Étape 2 : Calcul de la sortie (softmax)

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres

$\vec{x_1} = \text{m}$ 

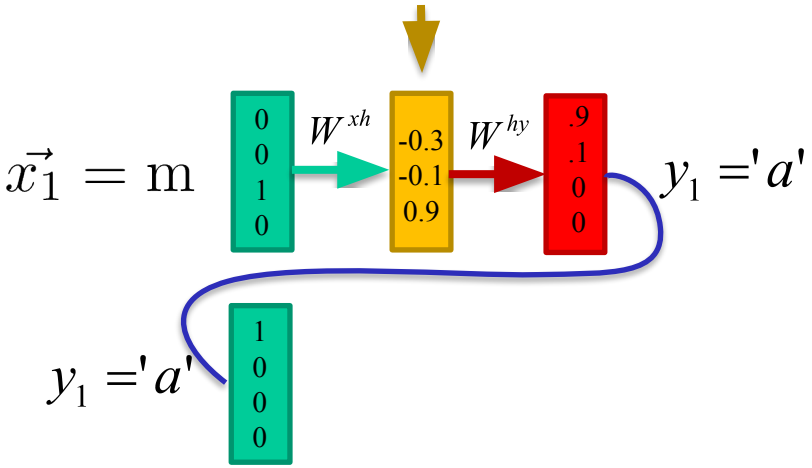
$y_1 = 'a'$

Étape 3 : Sélectionner le caractère le plus probable

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres

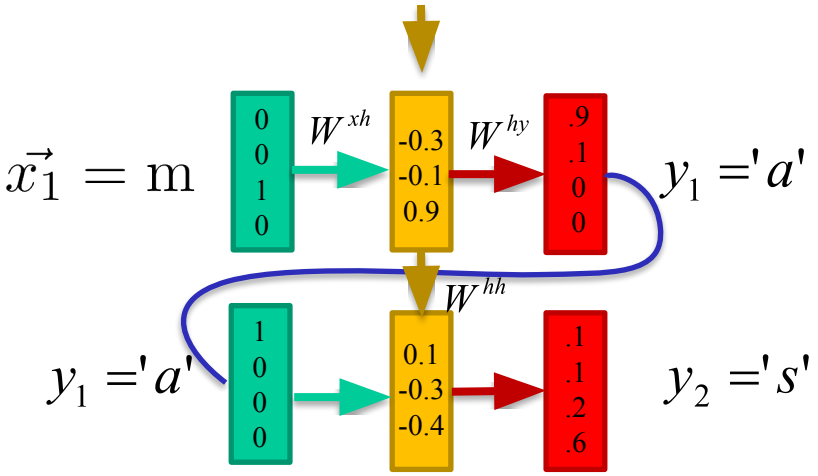


Étape 4 : Injecter le caractère prédit au début du réseau

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres

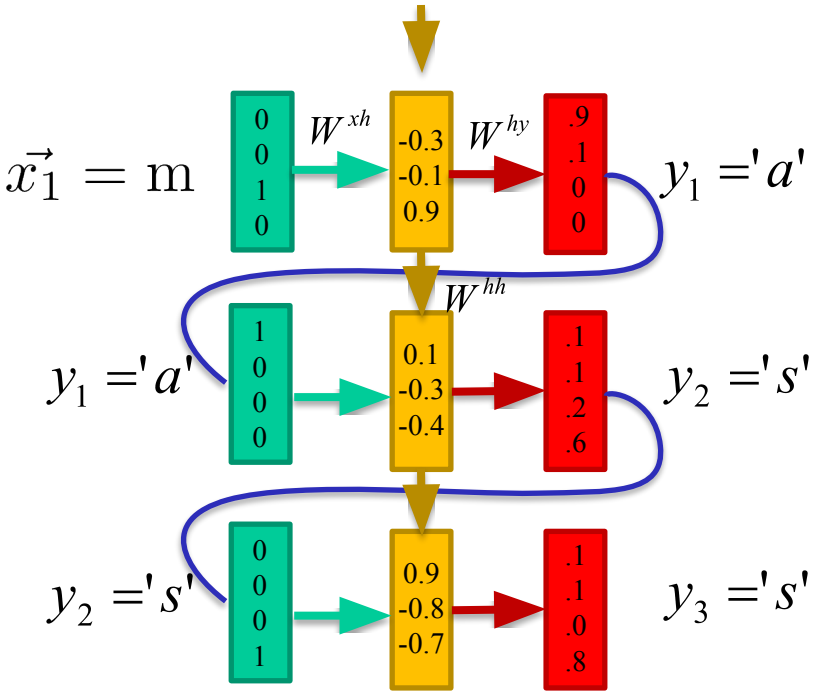


Et on recommence!

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

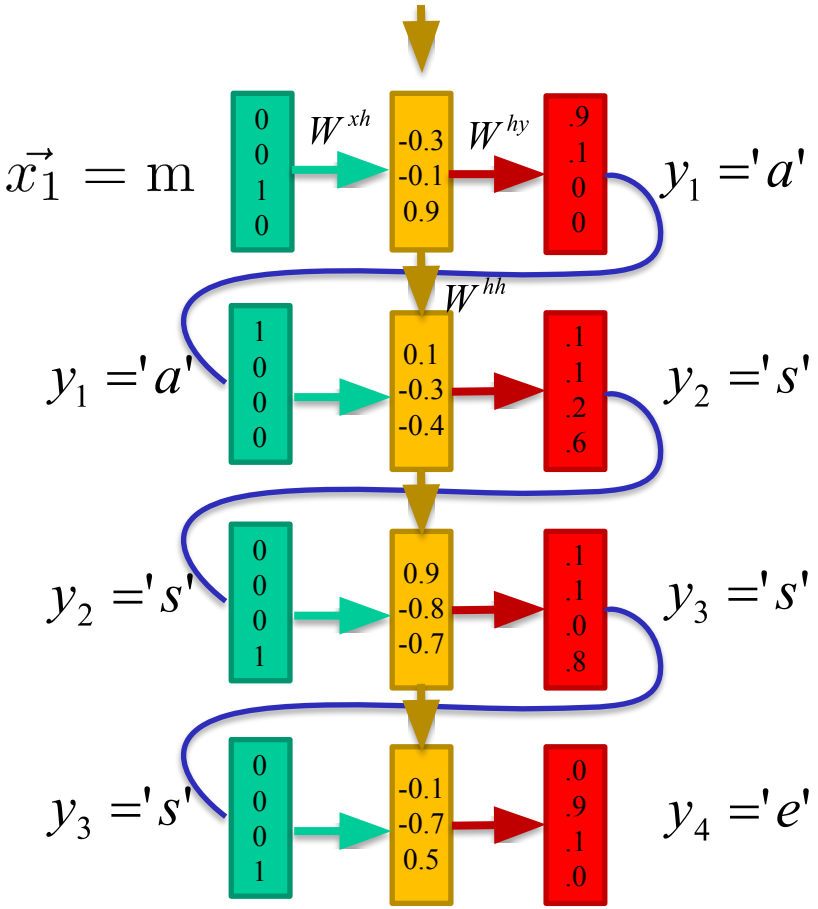
En test : prédire les lettres les unes après les autres



Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres



Autre exemple: **prédiction de caractères** (modèle de langue)

Code python: **“mini-char-RNN” de A. Karpathy**

https://gist.github.com/karpathy/d4dee566867f8291f086

Un RNN en 112 lignes !

Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy

min-char-rnn.py — Raw

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
```

$$\left.\begin{aligned} 'a' &= [1,0,0,\ldots,0] \\ 'b' &= [0,1,0,\ldots,0] \\ 'c' &= [0,0,1,\ldots,0] \end{aligned}\right\} \in R^{256}$$

…

Autre exemple: **prédiction de caractères** (modèle de langue)

Code python: **“mini-char-RNN” de A. Karpathy**
https://gist.github.com/karpathy/d4dee566867f8291f086

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.

$$\left.\begin{aligned} 'a' &= [1,0,0,\ldots,0] \\ 'b' &= [0,1,0,\ldots,0] \\ 'c' &= [0,0,1,\ldots,0] \end{aligned}\right\} \in R^{256}$$

$\ldots$

Autre exemple: **prédiction de caractères** (modèle de langue)

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Crédit: A. Karpathy, CS231

Autre exemple: **prédiction de caractères** (modèle de langue)

```
----
iter 1816300, loss: 39.494453
----
 nnes troi la nulits avovez son des préchâté, ne viche loi. Et serrité, ou présermoisse sent soclaine,
 cocraine, de la mas était pas. Devant lui quelque rumisent du cria entrois cepthib. Le _pouv
----
iter 1816400, loss: 39.437285
----
 s. Qu'est-c'était bâciante Clemigant du passée avez-logue une salée:

--Ains, honteatiées d'arrapa, à, emporle, Cosettes doncait. Marius cet hombre. Il a hume
grêts on éline, ce sans tous l
----
iter 1816500, loss: 39.626851
----
 côtée, et même des sordia:

--Il sunques nous exation de toleva entré, comme-hes
rivorme de rassieu. On a-Autant de bêtes, avernorbre que pierre dejoil, l'agraître. Disant dose de
l'édét
----
```

Crédit: A. Karpathy, CS231

Entraînement sur les cinq tomes de “Les misérables” provenant du *Gutenberg Project* (https://www.gutenberg.org/)

Autre exemple: **prédiction de caractères** (modèle de langue)

```
 l'eccure, à la dêtreste
probée.


«Elle qu'il senconches, plutige. Envreur de minetiesait soulèretominante? l'ontatilletier si moin, ell
e hautôt sa trieur,
ou serace, tanne, les inquiqueur
----
iter 994600, loss: 41.919709
----
 avait à retri dideux la sailtirssique
plus autré pars une soir finions M..- Jes marturiteme. Le chemtite un camme tembres! quand avait Litat
ions au moursues là vien de diclit-ilur et--- inguses, c
----
iter 994700, loss: 41.791706
----
 te, cielle, s'un dune autresses. Il surclait. Et par sur le saute, son êtres
come; faiement le bacevielle un
vandait la
pranconsenteu
de la croboti:
```

Entraînement sur Madame Bovary provenant du *Gutenberg Project* (https://www.gutenberg.org/)

Autre exemple: **prédiction de caractères** (modèle de langue)

Texte généré une fois le modèle entraîné

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Crédit: A. Karpathy, CS231

Autre exemple: **prédiction de caractères** (modèle de langue)

Entraînement sur le code source de Linux en C++

Texte généré une fois le modèle entraîné

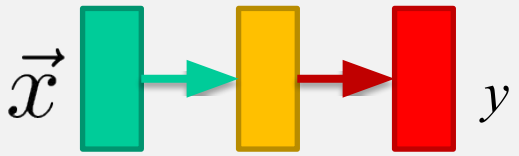
```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

```
#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)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

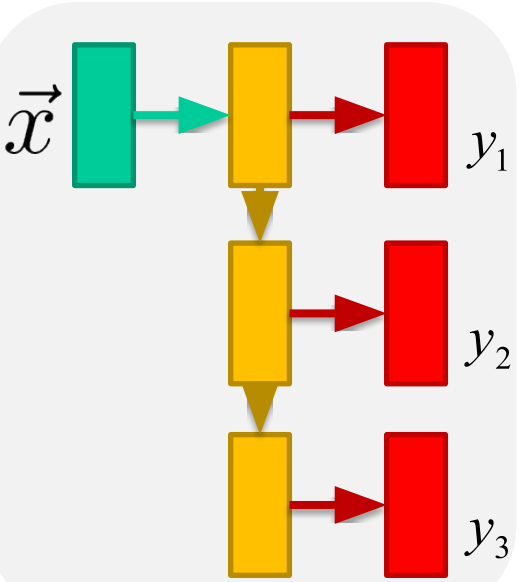
Crédit: A. Karpathy, CS231

Différentes configurations pour différentes applications

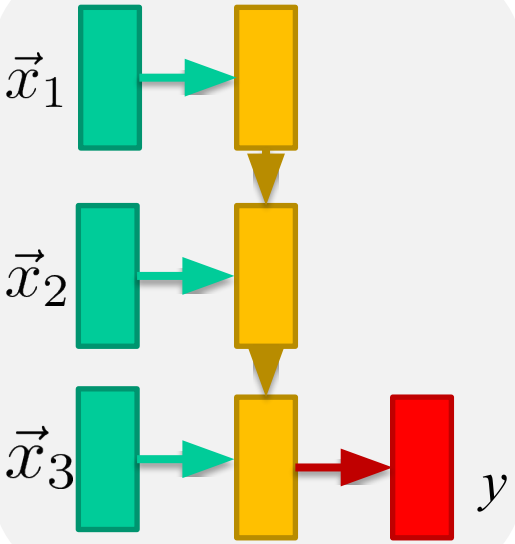
1 entrée et 1 sortie



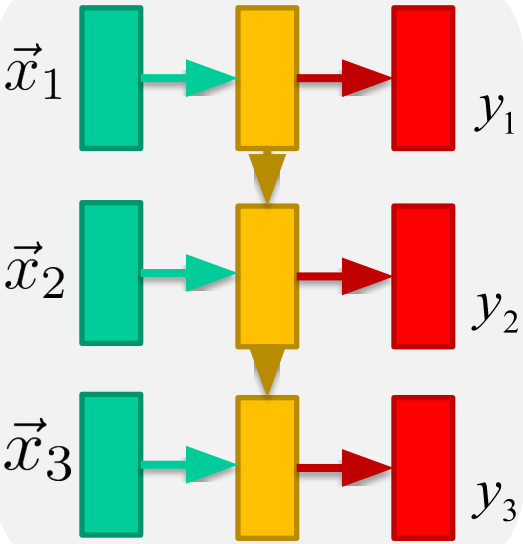
1 entrée et N sorties



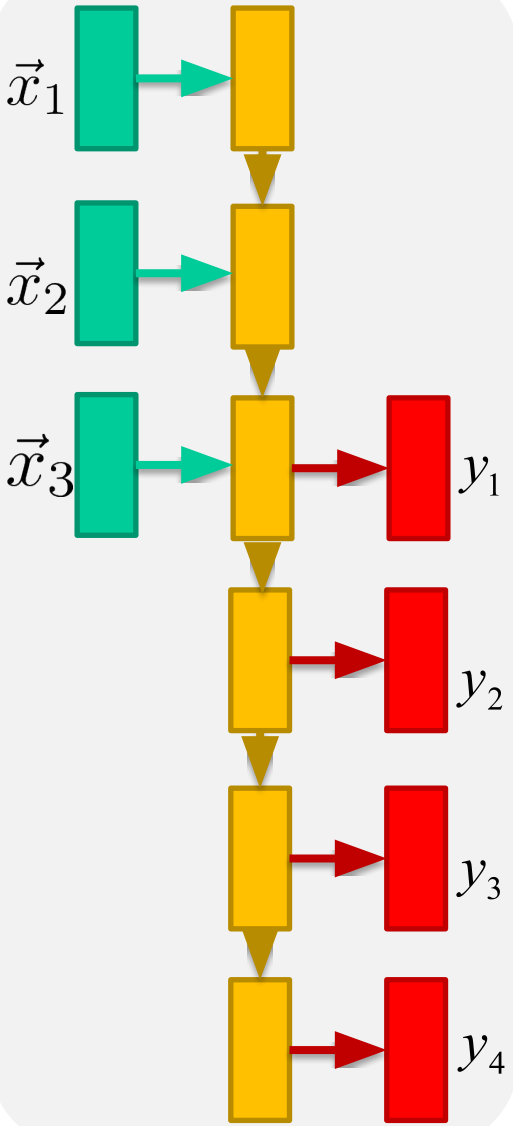
1 entrée et N sorties



N entrées et N sorties



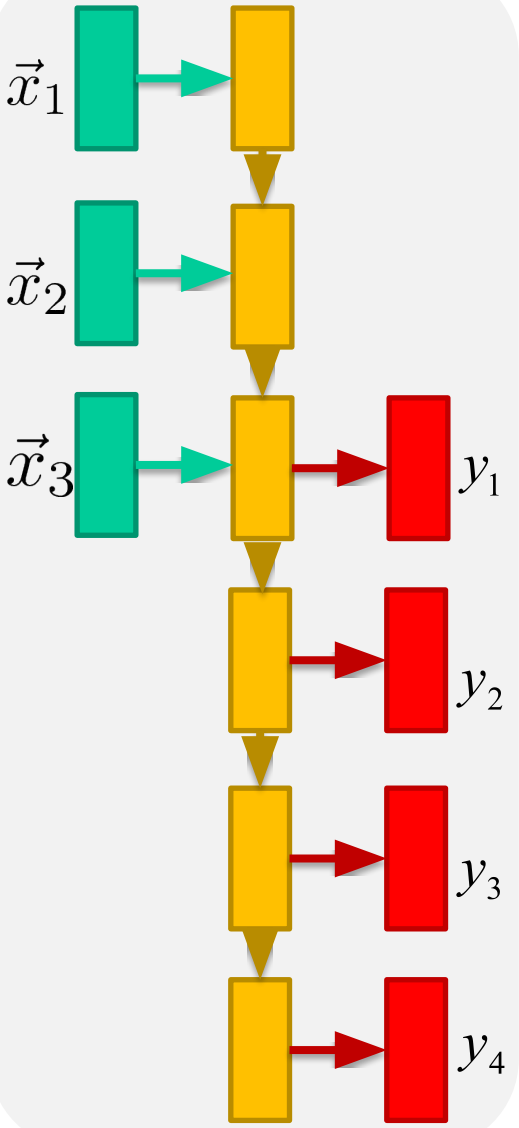
M entrées et N sorties



Ex.: Traduction Français-Anglais
N mots => M mots

Différentes configurations pour différentes applications

M entrées et N sorties



Ex.: Traduction Français-Anglais
N mots => M mots

Autre exemple: **traduction**

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

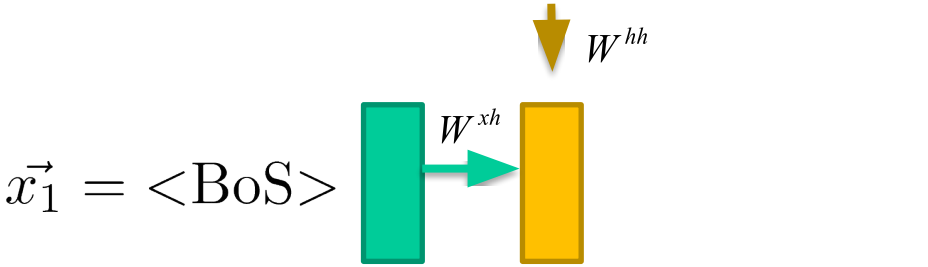
Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**

Pas le même nombre d’entrées et de sorties !

Autre exemple: **traduction**

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

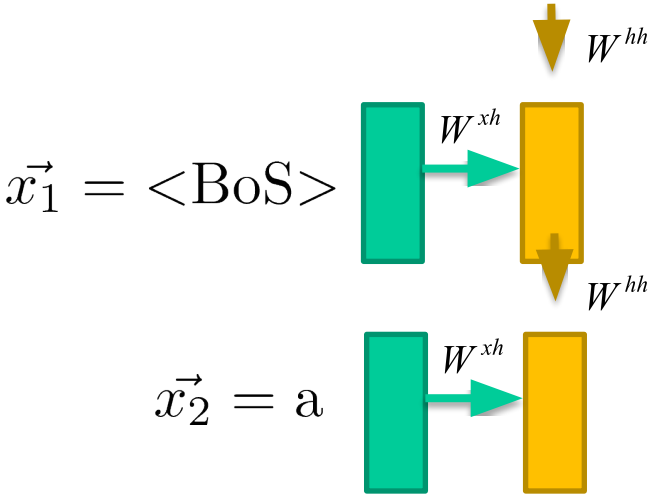
Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**



Autre exemple: **traduction**

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

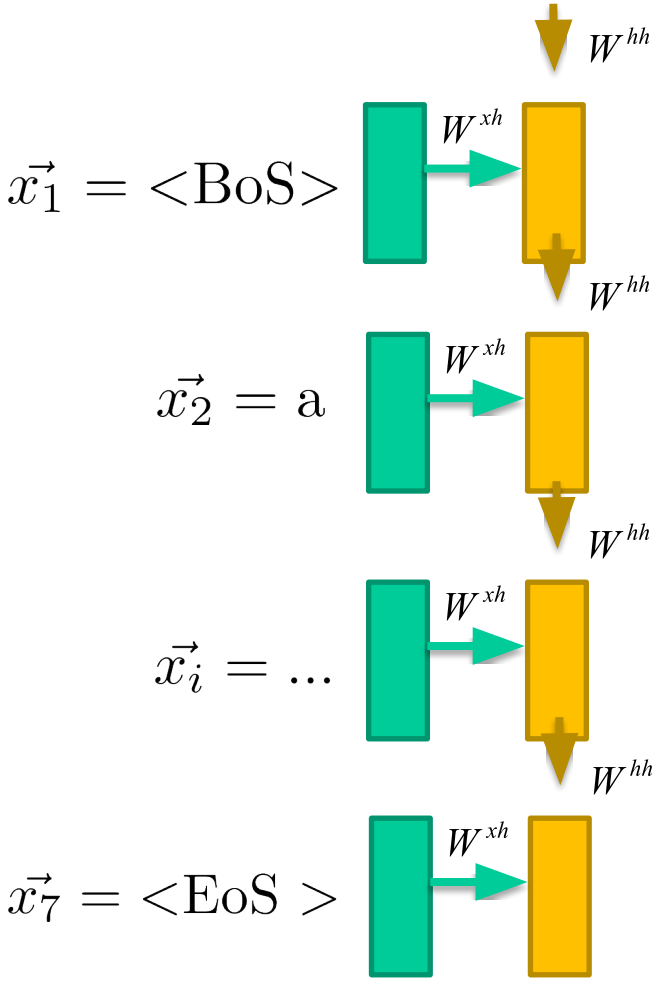
Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**



Autre exemple: **traduction**

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

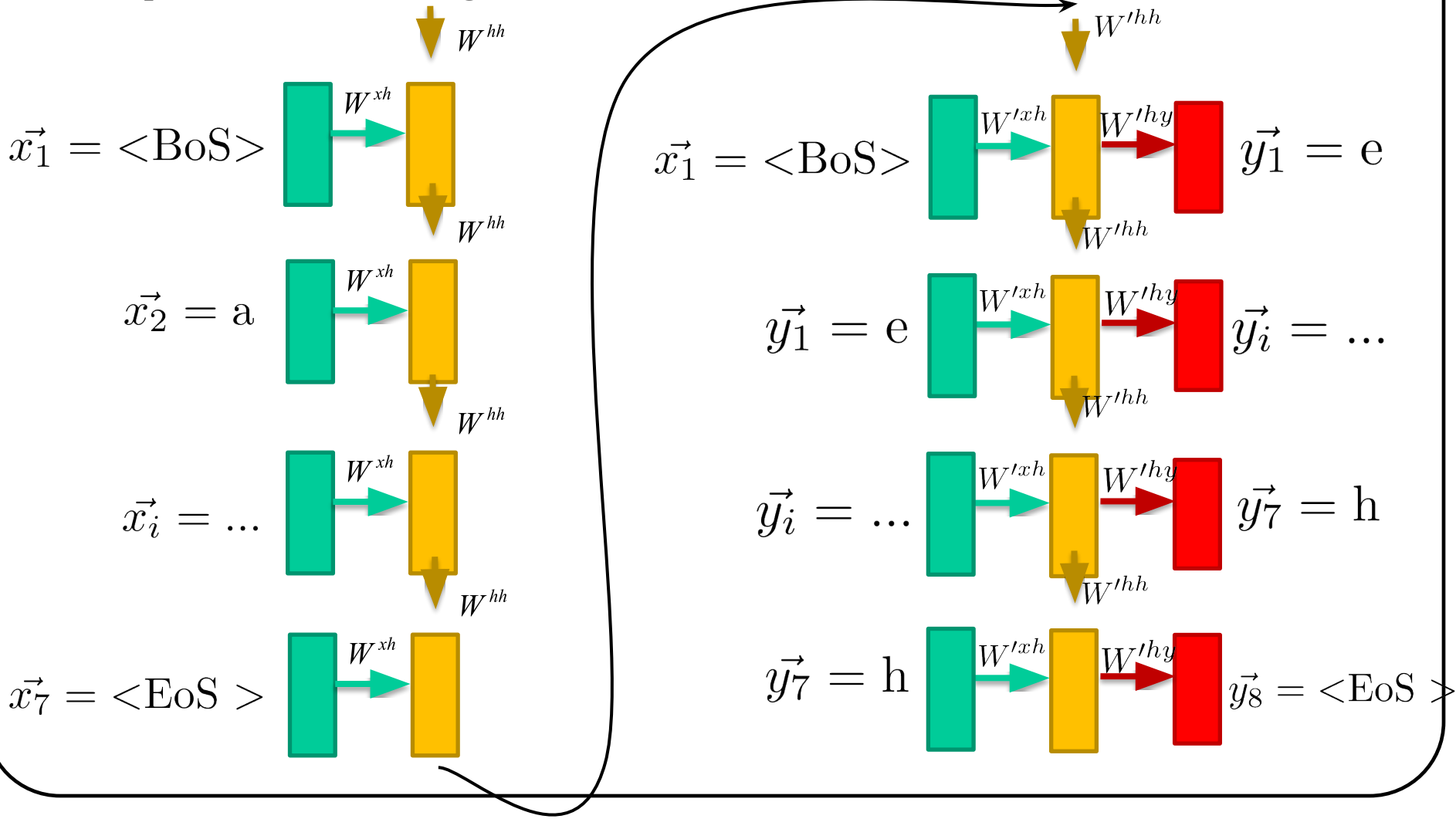
Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**



Autre exemple: **traduction**

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

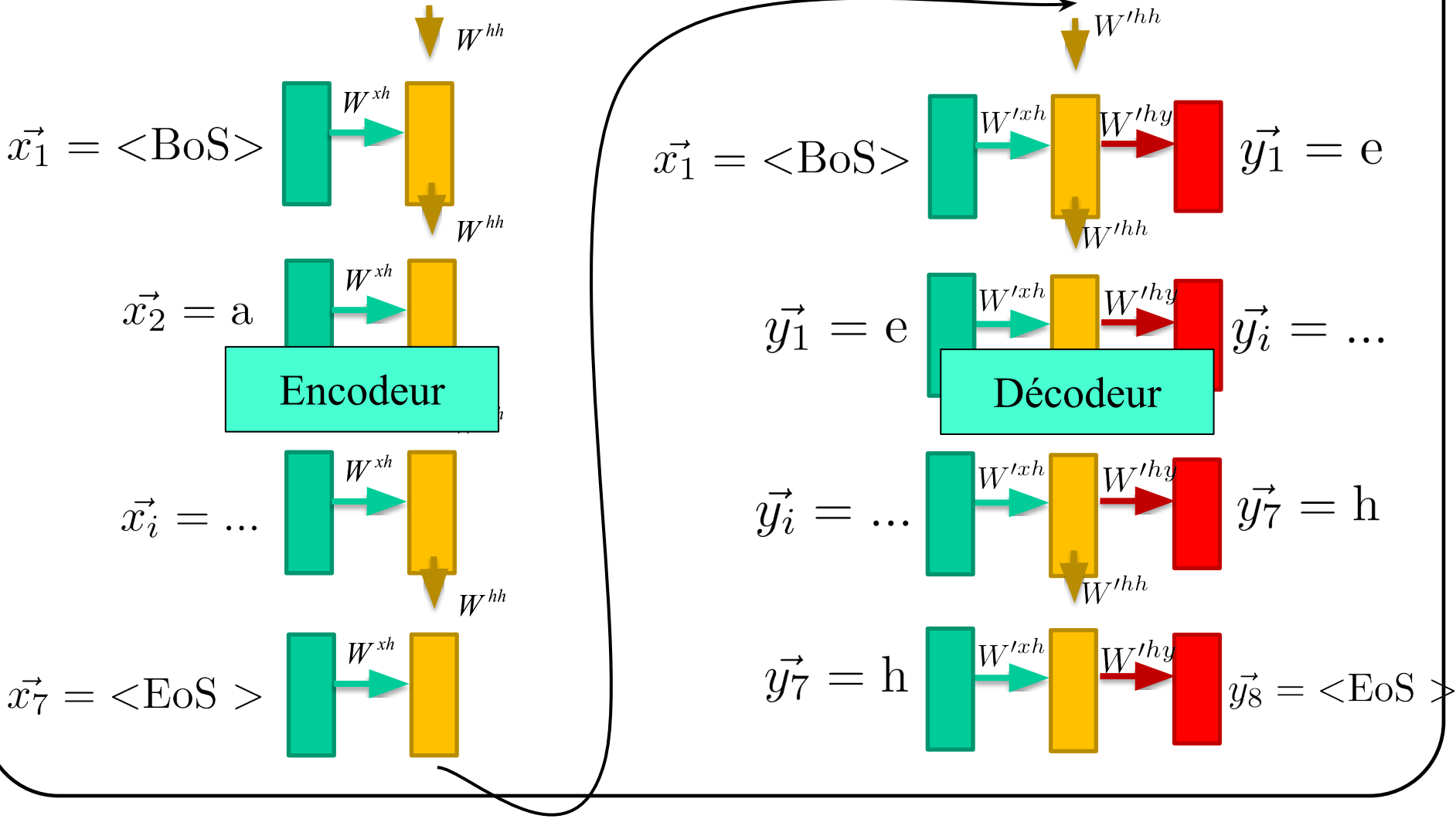
Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**



Autre exemple: **traduction**

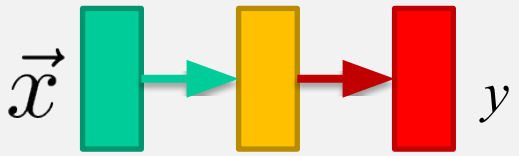
Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**

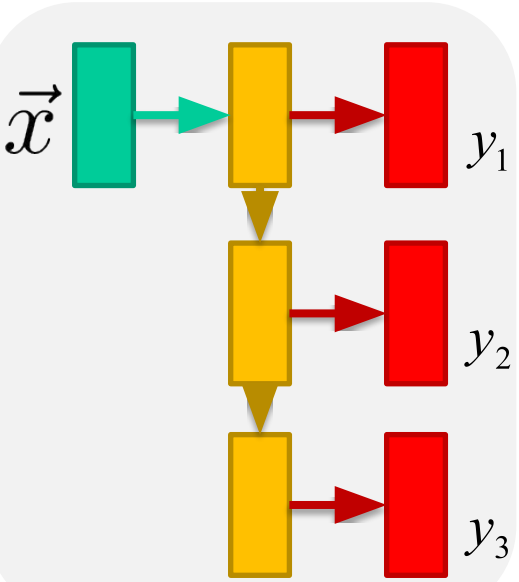


Différentes configurations pour différentes applications

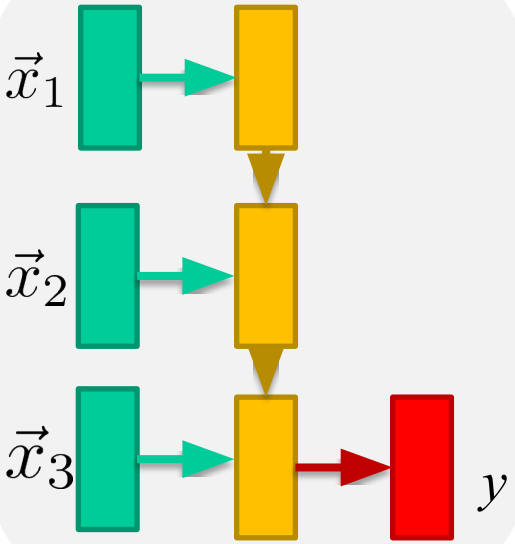
1 entrée et 1 sortie



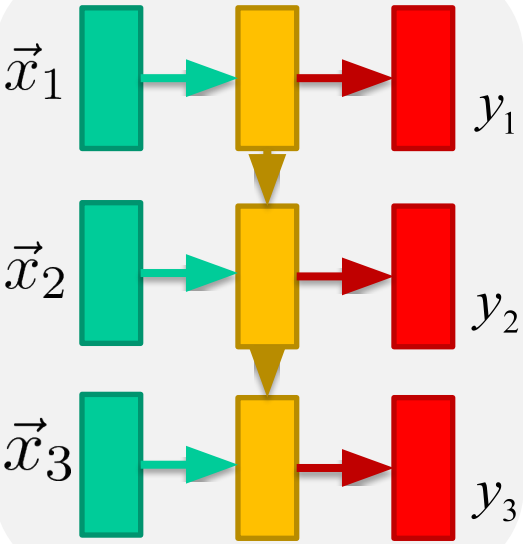
1 entrée et N sorties



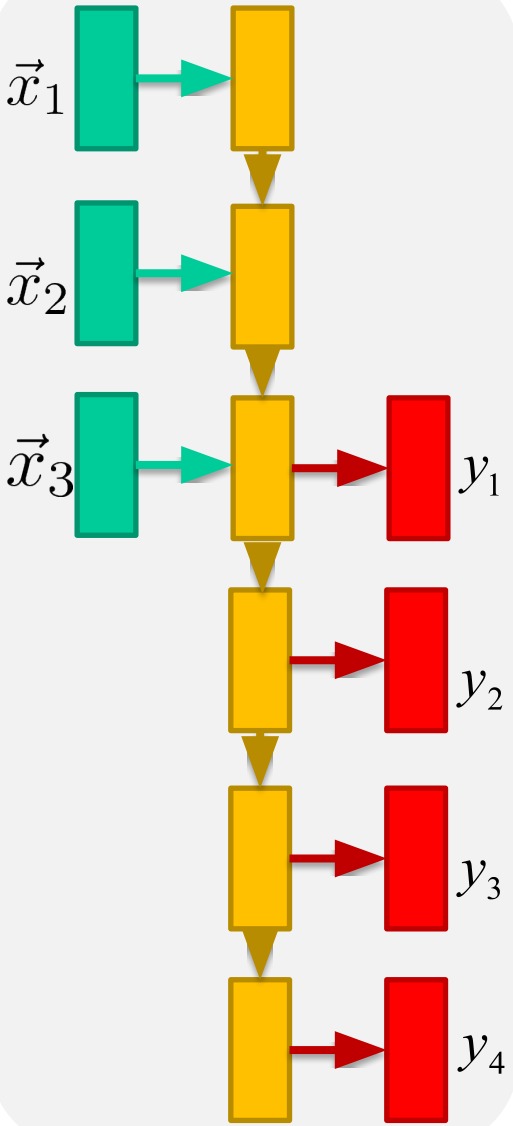
1 entrée et N sorties



N entrées et N sorties



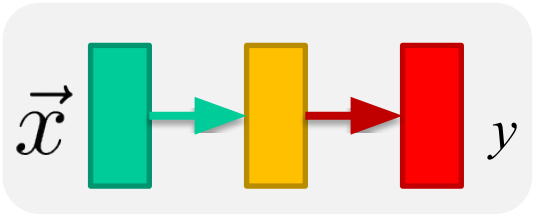
M entrées et N sorties



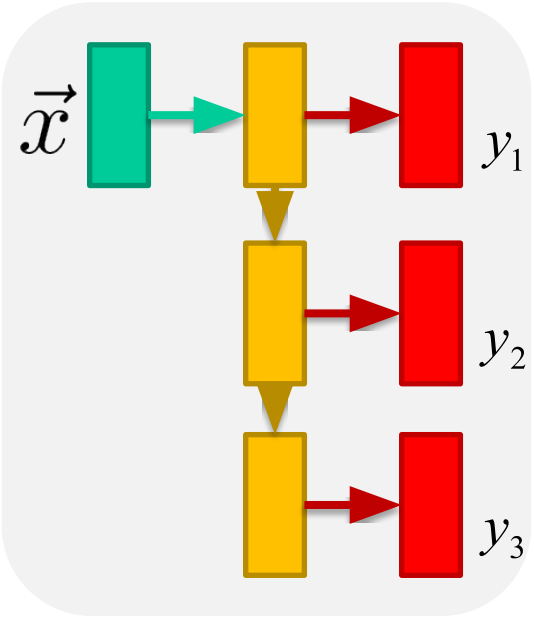
Ex.: Traduction Français-Anglais
N mots => M mots

Différentes configurations pour différentes applications

1 entrée et 1 sortie



1 entrée et N sorties



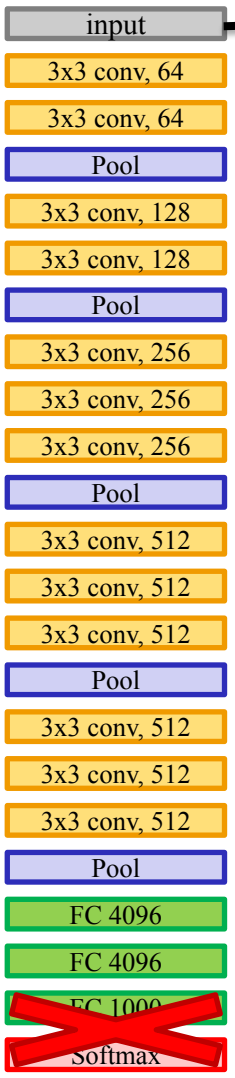
Captioning

input
3x3 conv, 64
3x3 conv, 64
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
FC 4096
FC 4096
FC 1000
Softmax

VGG16

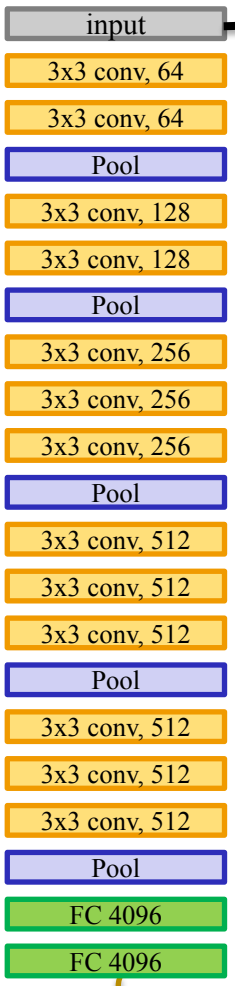
Pré-entraîné sur *ImageNet*

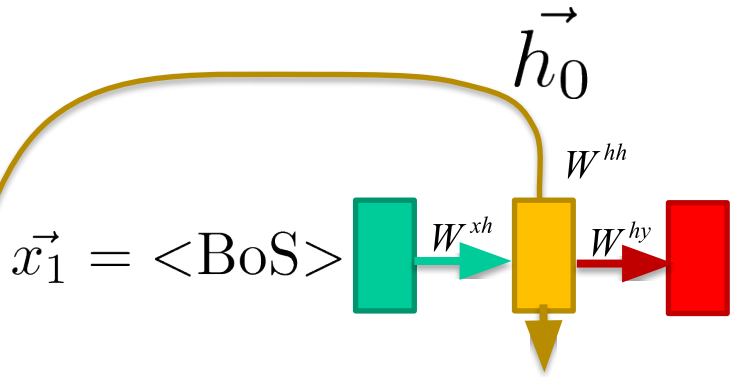
Captioning



VGG16

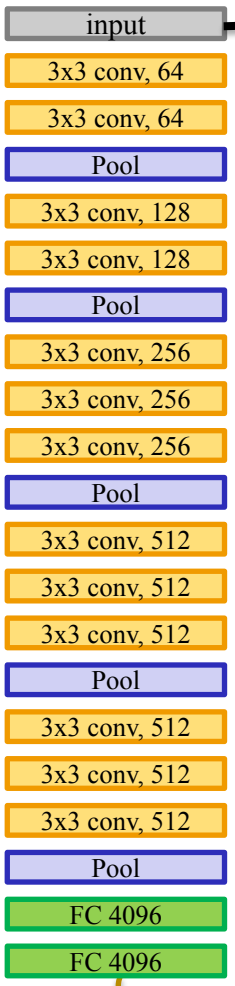
Captioning

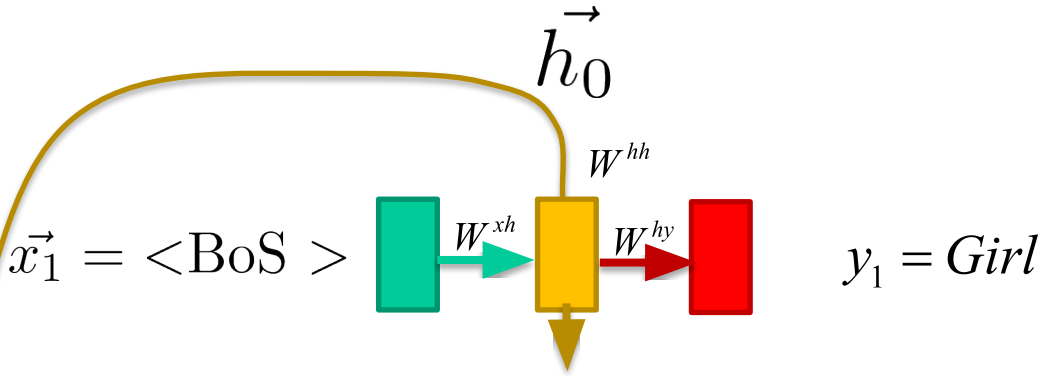




Étape 1 : Init + Propagation avant

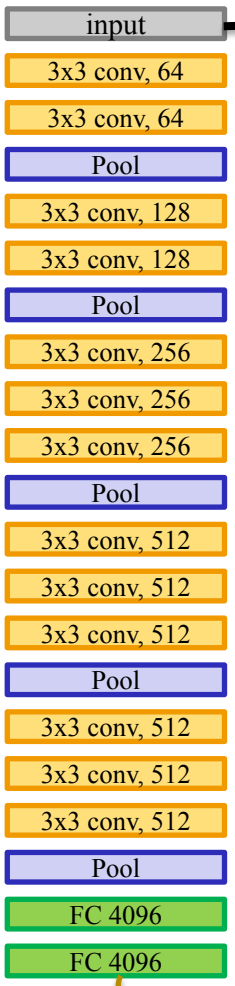
Captioning

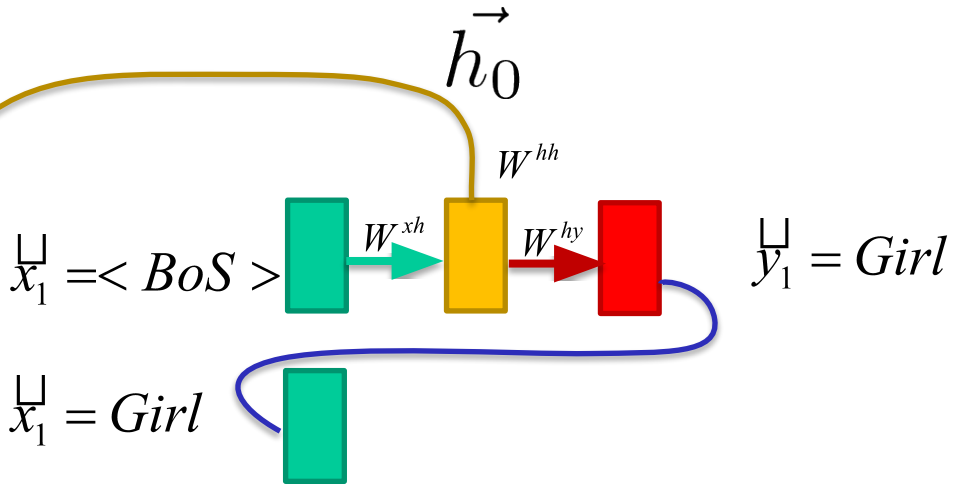




Étape 2 : Échantionnage

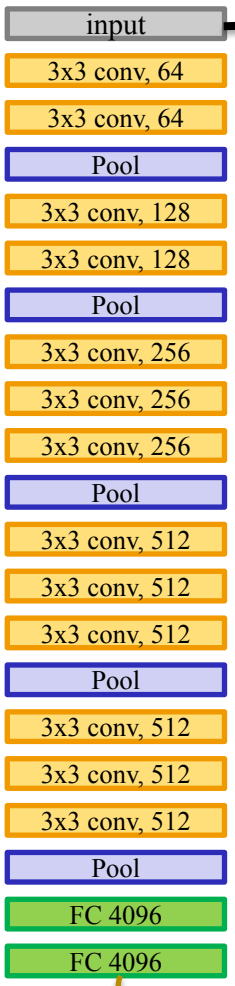
Captioning

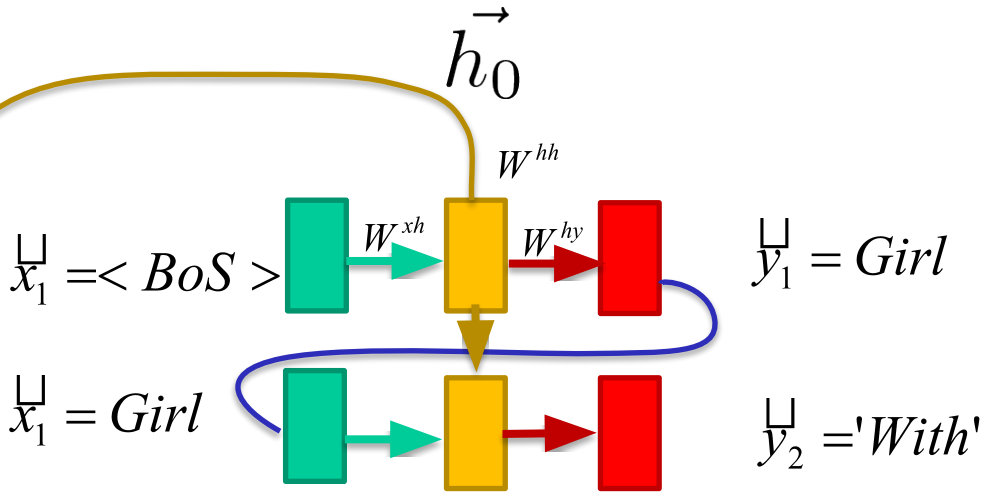




Étape 3 : Copie la sortie vers l'entrée

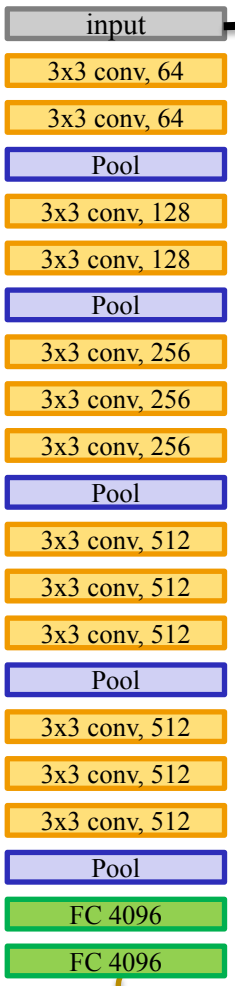
Captioning

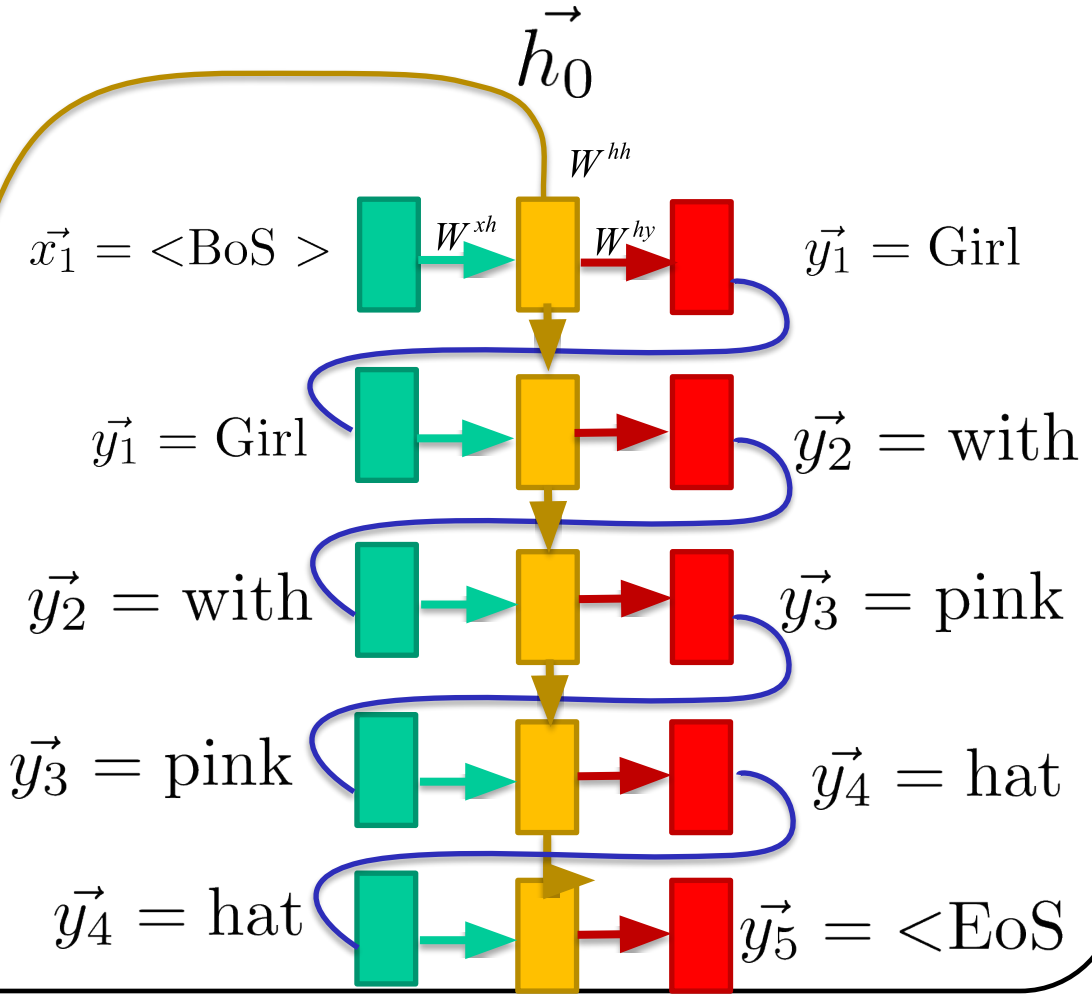




Étape 4 : On recommence!

Captioning


input
3x3 conv, 64
3x3 conv, 64
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
FC 4096
FC 4096


$\vec{h}_0$
W^{hh}
W^{xh}
W^{hy}
$\vec{x}_1 = <\text{BoS}>$
$\vec{y}_1 = \text{Girl}$
$\vec{y}_1 = \text{Girl}$
$\vec{y}_2 = \text{with}$
$\vec{y}_2 = \text{with}$
$\vec{y}_3 = \text{pink}$
$\vec{y}_3 = \text{pink}$
$\vec{y}_4 = \text{hat}$
$\vec{y}_4 = \text{hat}$
$\vec{y}_5 = <\text{EoS}>$

Exemples de résultats

https://github.com/karpathy/neuraltalk2

an elephant standing in a grassy field with trees in the background

a man riding a wave on top of a surfboard

a street sign on a pole in front of a building

a group of people playing a game with nintendo wii controllers

a couple of zebra standing on top of a dirt field

Exemples d'erreurs

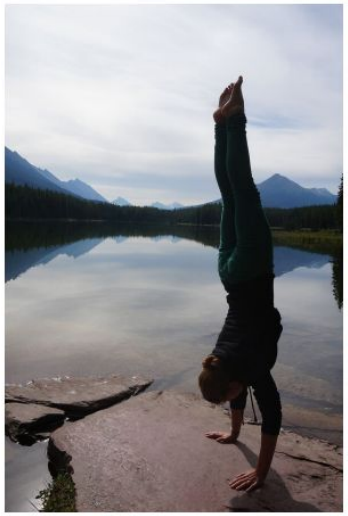
https://github.com/karpathy/neuraltalk2

a man is throwing a frisbee in a park

a man riding a skateboard down a street

a laptop computer sitting on top of a wooden desk

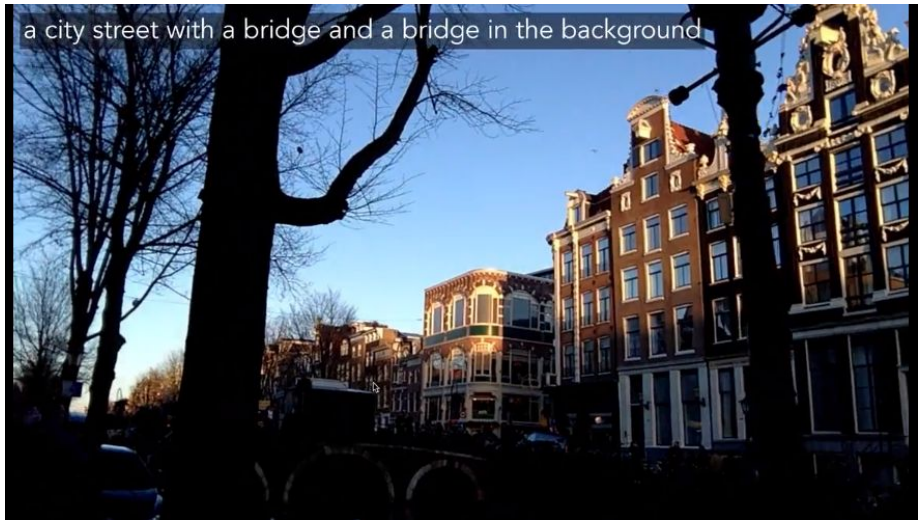
A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

NeuralTalk and Walk

https://vimeo.com/146492001



Limites des « one-hot vectors »

Souvent les modèles de langue utilisent l'encodage « one hot »

Pour des **caractères**…

$$\left.\begin{aligned} 'a' &= [1,0,0,...,0] \\ 'b' &= [0,1,0,...,0] \\ 'c' &= [0,0,1,...,0] \end{aligned}\right\} \in R^{256}$$

...

Limites des « one-hot vectors »

Souvent les modèles de langue utilisent l'encodage « one hot »

Pour des **mots**…

$$
\left.
\begin{array}{l}
\text{'}grand\text{'} = [...,1,0,0,...,0] \\
\text{'}grandement\text{'} = [...,0,1,0,...,0] \\
\text{'}grandeur\text{'} = [...,0,0,1,...,0]
\end{array}
\right\} \in R^{10,000}
$$

Limites des « one-hot vectors »

Bien que simple, cet encodage a plusieurs **inconvénients**

1- Peu efficace en mémoire lorsque non compressé

ex.: 10,000 bits pour encoder le mot « **je** » dans une langue à 10,000 mots!

2- Pas de distance sémantique entre les codes:

Ex.

distance[**one-hot**(‘**bon**’), **one-hot**(‘**bien**’)]= **distance**[**one-hot**(‘**bon**’), **one-hot**(‘**trottoir**’)]

Or, on souhaiterait un **code** tel que

distance[**code**(‘**bon**’), **code**(‘**bien**’)] << **distance**[**code**(‘**bon**’), **code**(‘**trottoir**’)]

distance[**code**(‘**Jean**’), **code**(‘**Chantal**’)] << **distance**[**code**(‘**bon**’), **code**(‘**trottoir**’)]

distance[**code**(‘**Inde**’), **code**(‘**Liban**’)] << **distance**[**code**(‘**bon**’), **code**(‘**trottoir**’)]

Word2Vec s'appuie sur 2 idées fondamentales

Un solution est d'utiliser l'encodage **Word2Vec** de [Mikolov et al. '13]

Idée 1: **Dictionnaire = matrice d'encodage**

Exemple jouet: on veut représenter ces 8 mots par un code à 4 éléments

« one-hot »

'the'	1	0	0	0	0	0	0	0
'quick'	0	1	0	0	0	0	0	0
'brown'	0	0	1	0	0	0	0	0
'fox'	0	0	0	1	0	0	0	0
'jumps'	0	0	0	0	1	0	0	0
'over'	0	0	0	0	0	1	0	0
'lazy'	0	0	0	0	0	0	1	0
'dog'	0	0	0	0	0	0	0	1

Dictionnaire

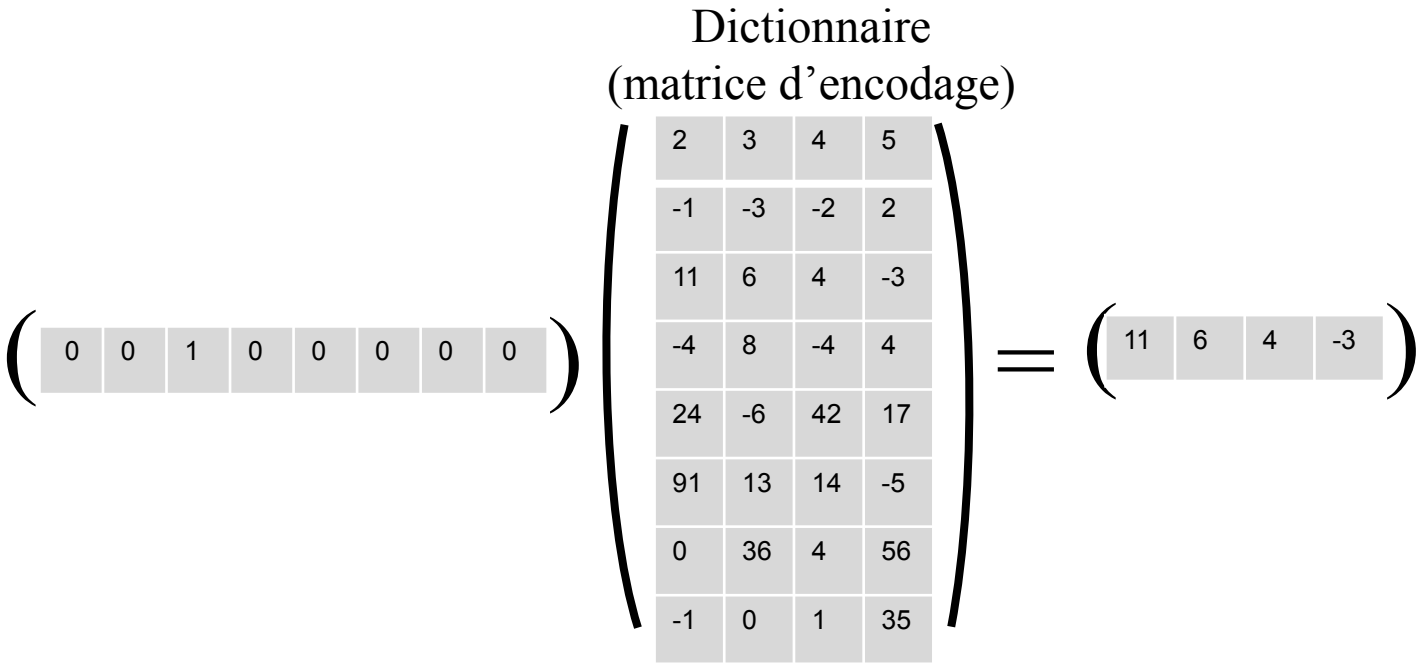
2	3	4	5
-1	-3	-2	2
11	6	4	-3
-4	8	-4	4
24	-6	42	17
91	13	14	-5
0	36	4	56
-1	0	1	35

Word2Vec s'appuie sur 2 idées fondamentales

Idée 1: **Dictionnaire = matrice d'encodage**

Comment sélectionner le code d'un mot? En multipliant son vecteur One-hot par la matrice d'encodage (le dictionnaire!)

Ex: sélectionner le code de « brown »



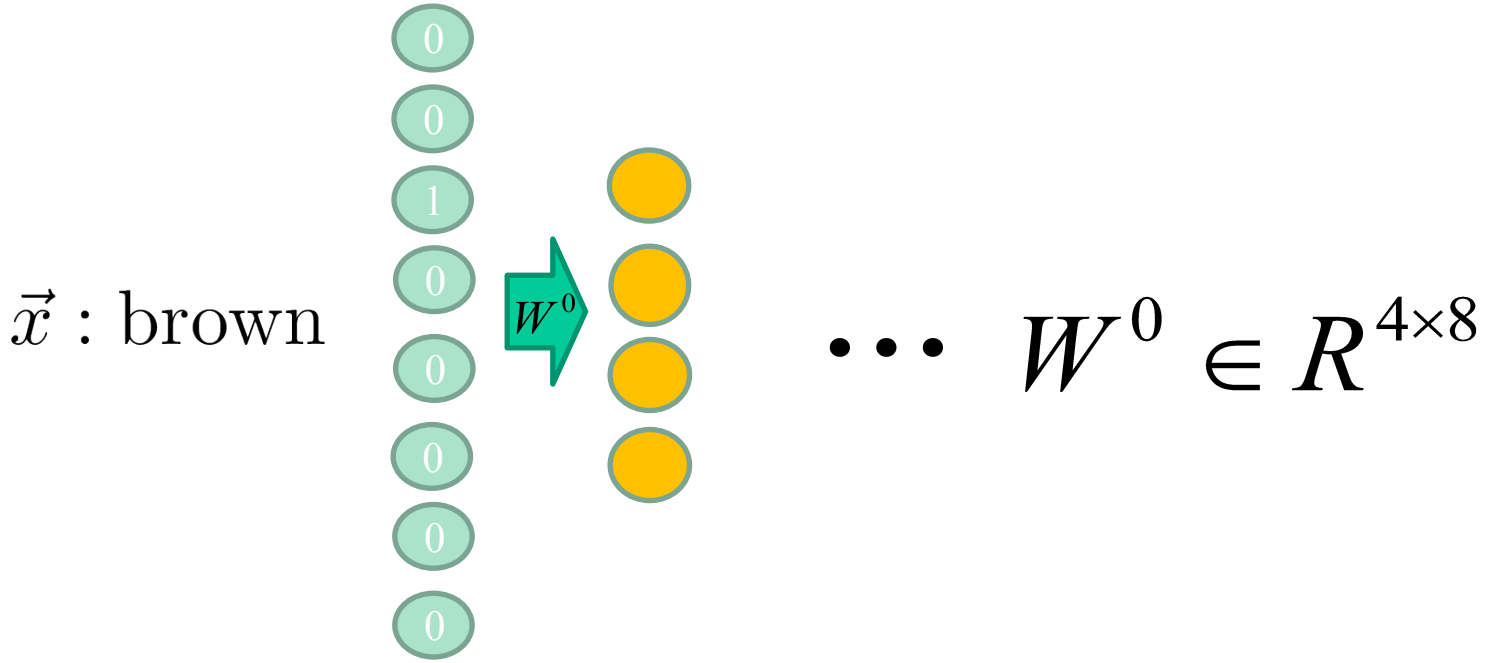
Word2Vec s'appuie sur 2 idées fondamentales

Idée 1: **Dictionnaire = matrice d'encodage**

Première couche d'un réseau de neurones

=

matrice d'encodage



Word2Vec s'appuie sur 2 idées fondamentales

Idée 1: **Dictionnaire = matrice d'encodage**

Première couche d'un réseau de neurones

=

matrice d'encodage

$$code_{\vec{x}} = W^0 \vec{x}$$

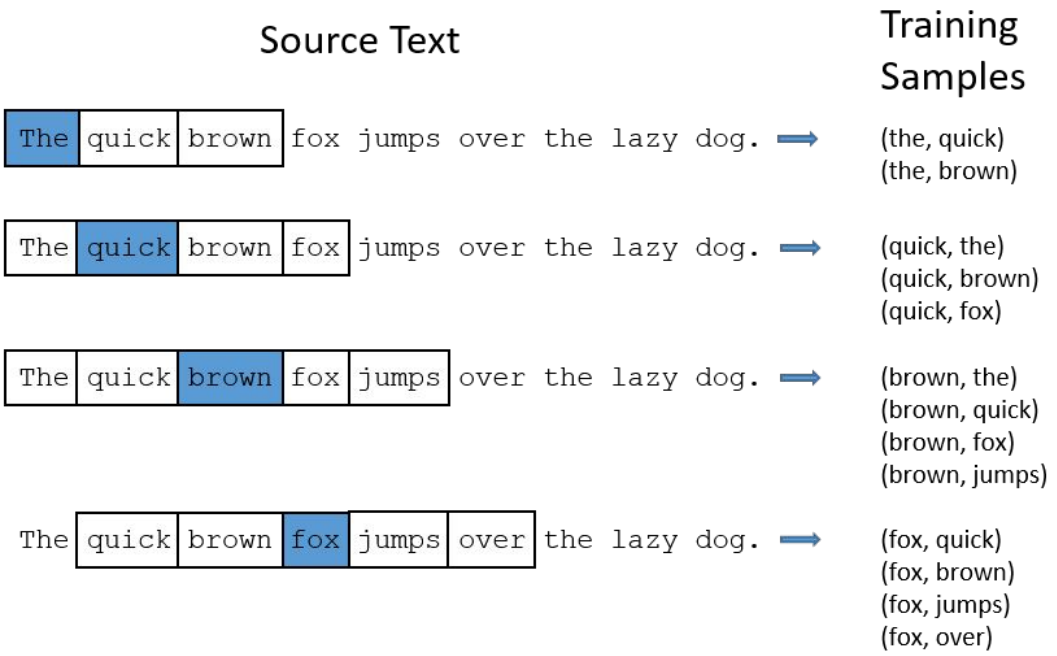
Word2Vec s'appuie sur 2 idées fondamentales

Idée 1: **Dictionnaire = matrice d'encodage**

On pourra donc utiliser un réseau de neurones pour calculer le contenu du dictionnaire

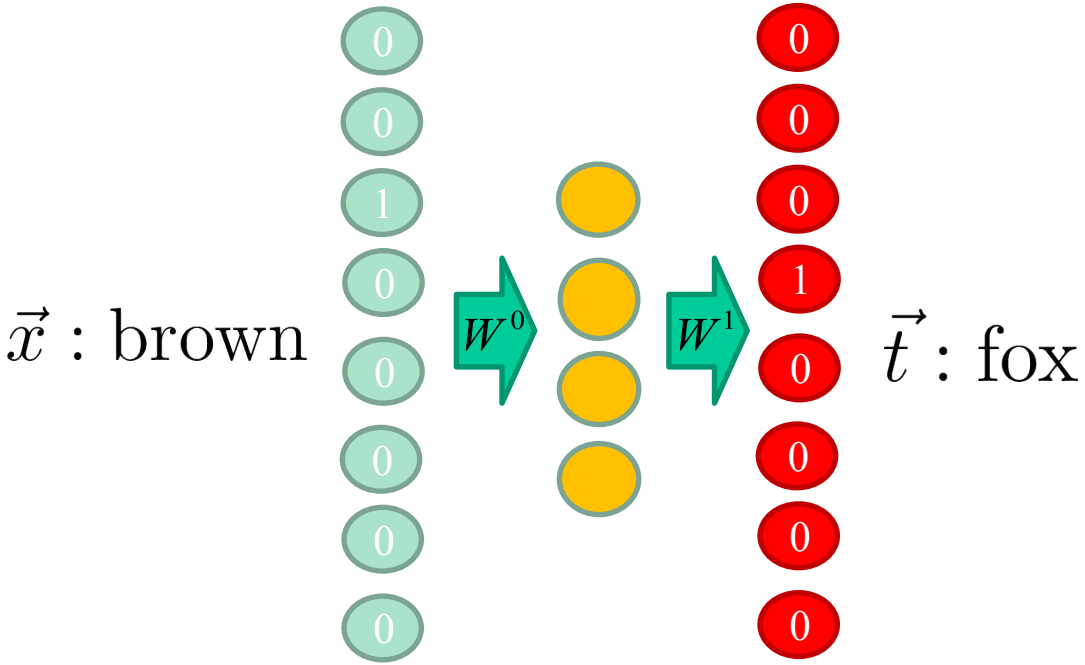
Word2Vec s'appuie sur 2 idées fondamentales

Idée 2: **2 mots proches dans un texte = 2 mots proches sémantiquement**



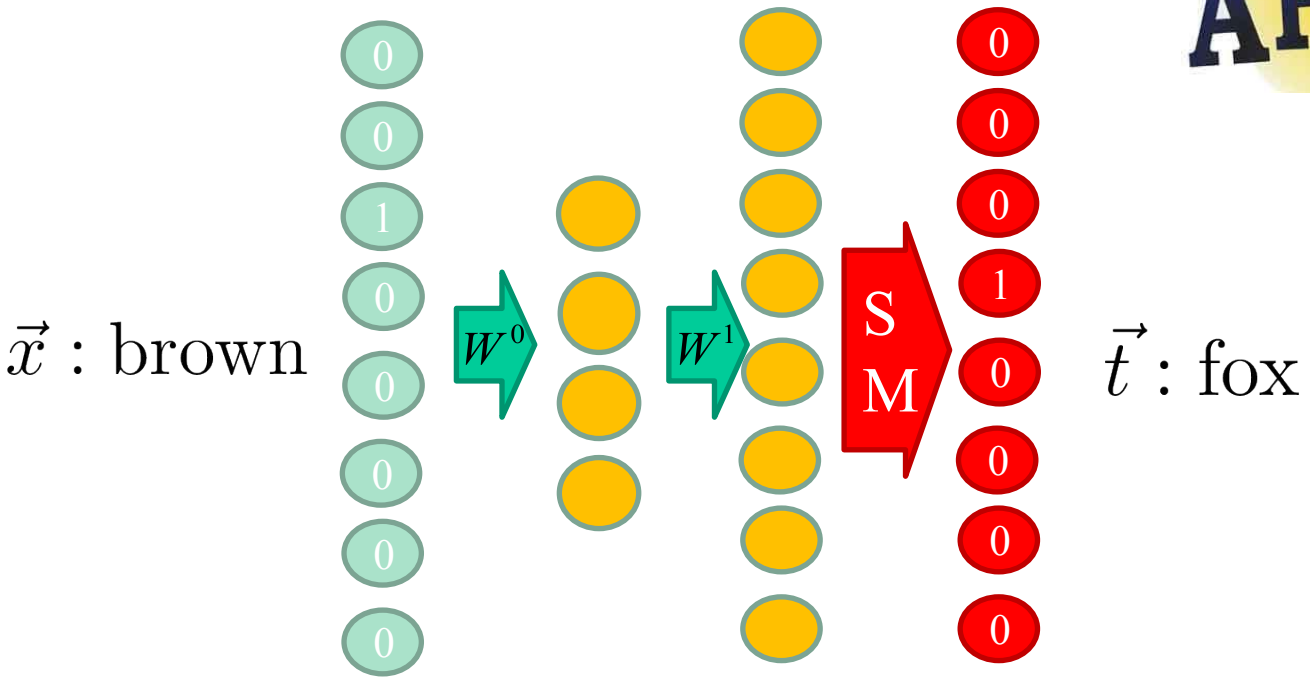
Basé sur un corpus de texte, on va créer des **millions de paires de mots**

Word2Vec [Mikolov et al. '13]



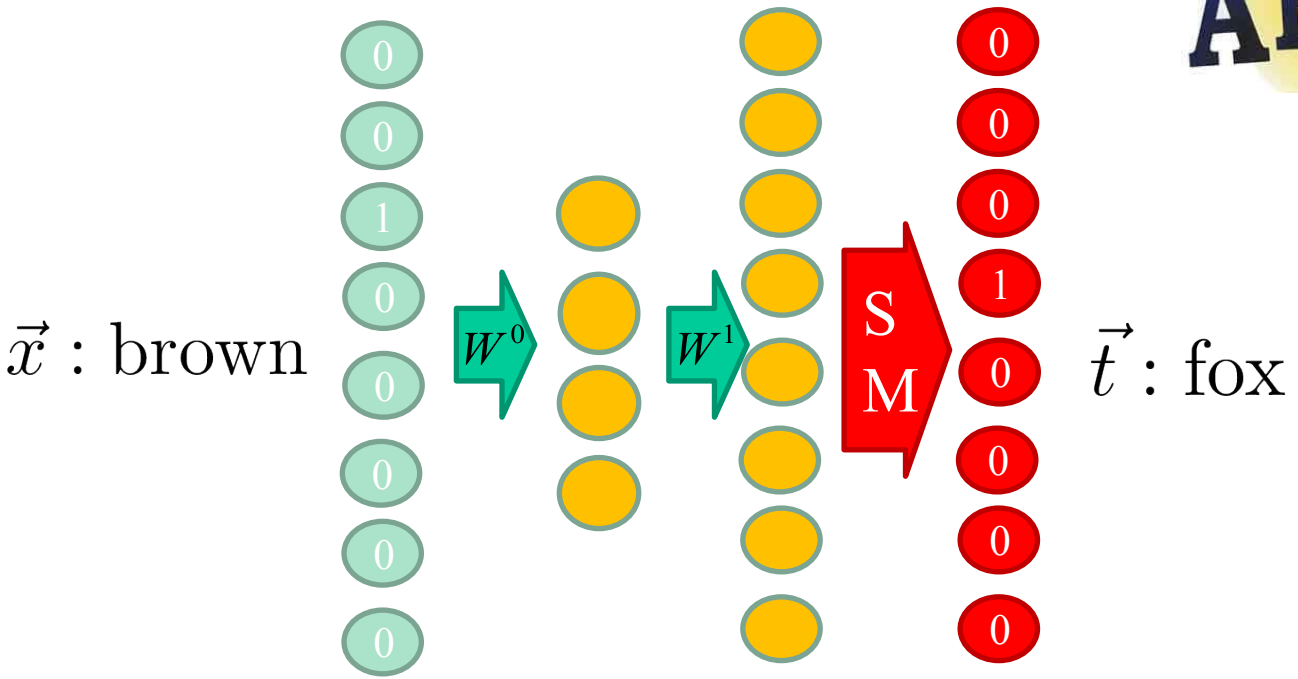
Entraîner un réseau de neurones
à reproduire le 2e mot partant du 1er

Word2Vec [Mikolov et al. '13]



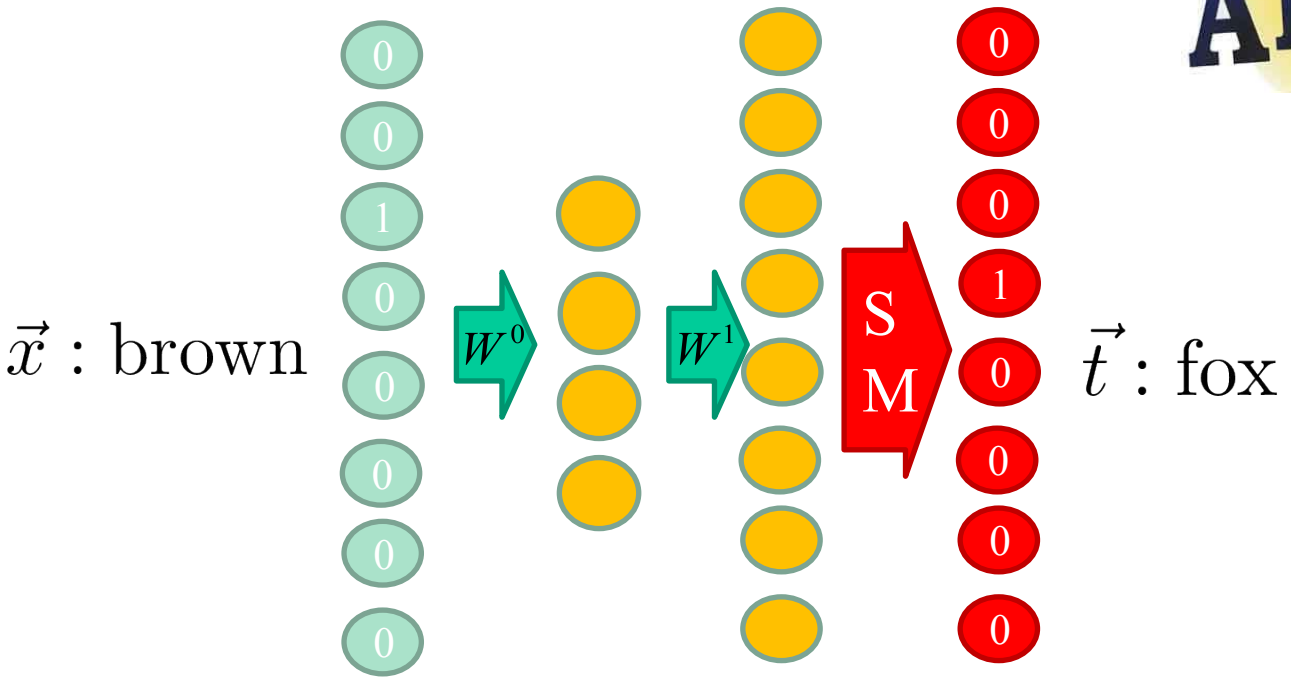
Puisque la sortie est de type « *one-hot* »
on utilise un softmax

Word2Vec [Mikolov et al. '13]



$$y(\vec{x}) = \text{softmax}(W^1 f_a(W^0 \text{x}))$$

Word2Vec [Mikolov et al. '13]



Lorsqu'entraîné, utiliser W^0
comme dictionnaire

Word2Vec [Mikolov et al. '13]

"If two different words have very similar "contexts" (that is, what words are likely to appear around them), then our model needs to output very similar results for these two words. And one way for the network to output similar context predictions for these two words is if the word vectors are similar. So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words! Ta da!

And what does it mean for two words to have similar contexts? I think you could expect that synonyms like "intelligent" and "smart" would have very similar contexts. Or that words that are related, like "engine" and "transmission", would probably have similar contexts as well."

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

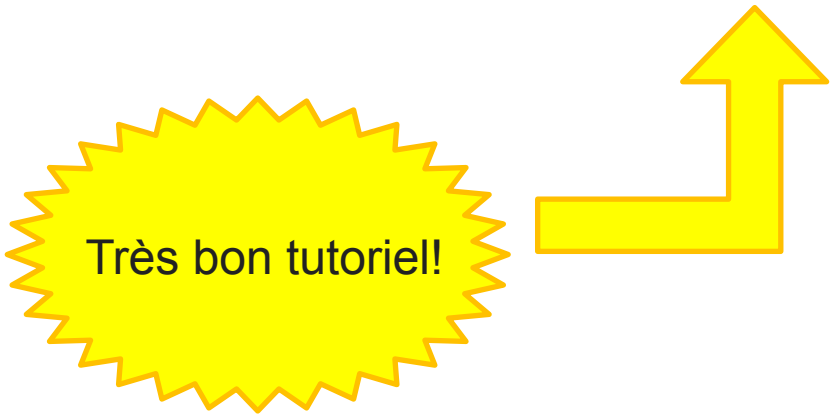
Word2Vec [Mikolov et al. ’13]

Cet algorithme vient avec **d’autres détails**

- Réduire l’occurrence des mots fréquents et sémantiquement faibles (*the, of, for, this, or, and,…*)
- Combiner des mots qui forment une entité (ex: *nations unies*)
- Divers trucs pour simplifier/accélérer l’entraînement

Limites du « one-hot vector »

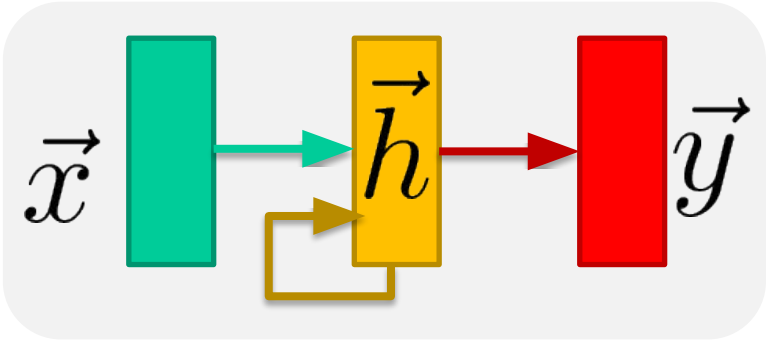
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

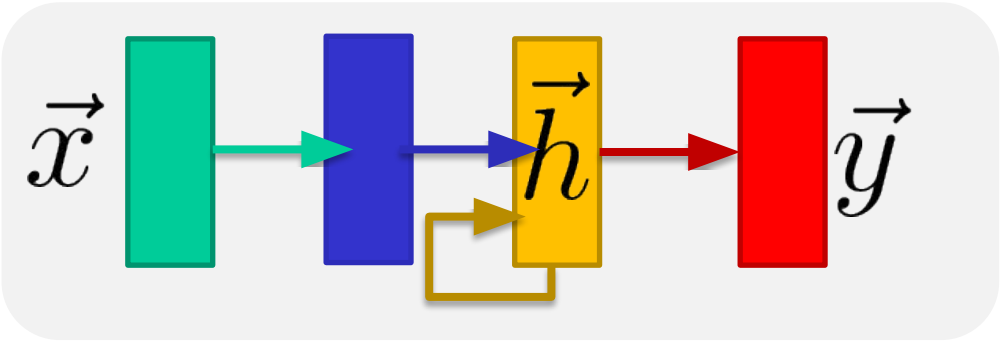


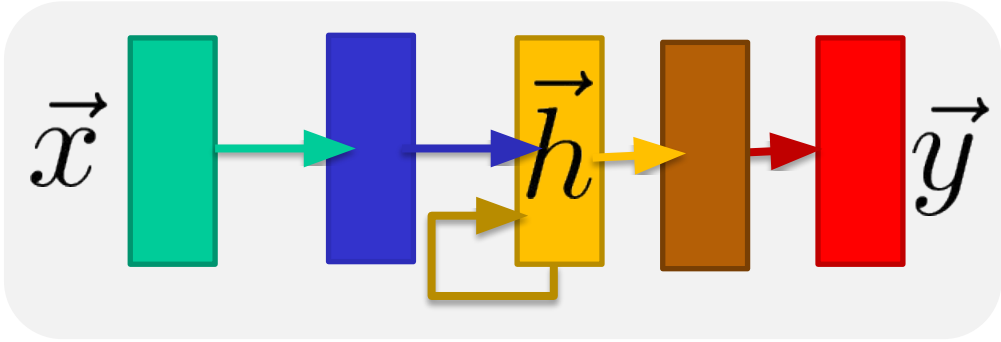
T.Mikolov et al. (2013). "Efficient Estimation of Word Representations in Vector Space", in ICLR 2013

Encodage (*embedding*)

L'encodage peut être appris !







La **couche d'encodage** permet de faire la sélection implicitement et d'ajouter plus de transformations

Prédiction sur des lettres vs. mots

$$\left.\begin{array}{l} 'a' = [1,0,0,...,0] \\ 'b' = [0,1,0,...,0] \\ 'c' = [0,0,1,...,0] \end{array}\right\} \in R^{256}$$

Prédiction sur des lettres

...

...

$$\left.\begin{array}{l} 'grand' = [...,1,0,0,...,0] \\ 'grandement' = [...,0,1,0,...,0] \\ 'grandeur' = [...,0,0,1,...,0] \end{array}\right\} \in R^{10,000}$$

Prédiction sur des mots

...

Prédiction sur des lettres vs. mots

$$\left.\begin{array}{r} \dots \\ 'e' = [0, 0, .., 1, .., 0] \\ \dots \\ 'grand' = [0, 0, .., 1, .., 0] \\ \dots \\ 'ment' = [0, 0, .., 1, .., 0] \\ \dots \end{array}\right\} \in \mathbb{R}^m$$

Prédiction sur les deux !

‘grand’
‘grand’+’e’
‘grand’+’e’+’ment’

Tokenization (jeton-isation ?)

Idée: à partir d'un dictionnaire qui ne contient que des caractères, combiner les séquences fréquentes en jetons (*tokens*)

Les séquences fréquentes (comme les mots ou sous-mots fréquents) se voient attribuer un jeton. Les séquences peu fréquentes peuvent être bâties à partir de jetons.

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

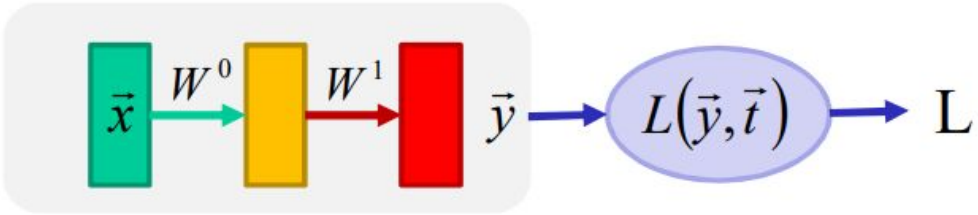
r ·	→	r·
l o	→	lo
lo w	→	low
e r·	→	er·

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909.

Comment entraîner un RNN?

Histoire de gradients

RN de classification avec entropie croisée

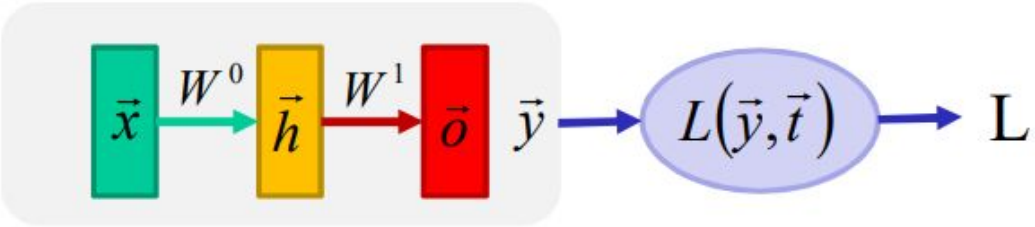


$$\vec{y}(\vec{x}) = S_M\left(W^1 \tanh\left(W^0 \vec{x}\right)\right)$$

$$L = L_{EC}\left(\vec{y},\vec{t}\right)$$

Histoire de gradients

Simple RN de classification avec entropie croisée

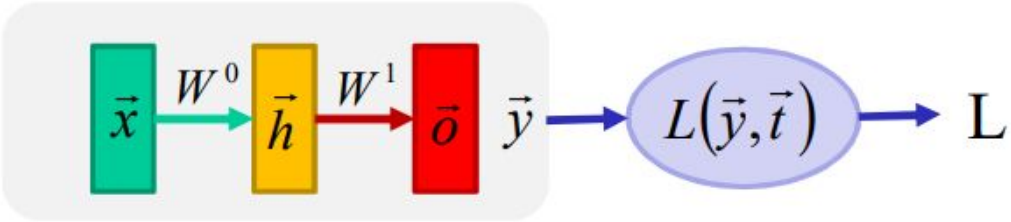


$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y}, \vec{t}\right)$$

Propagation avant

Histoire de gradients

Simple RN de classification avec entropie croisée



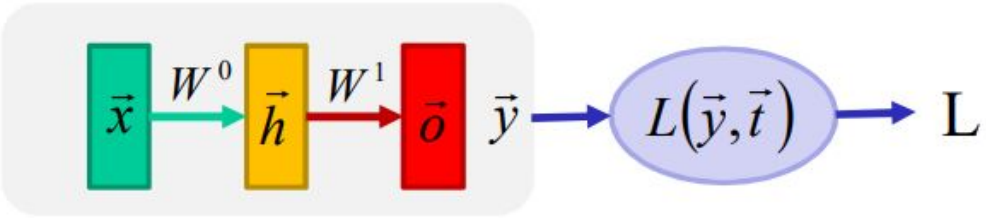
$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y},\vec{t}\right)$$

Pour entraîner le réseau il faut calculer

$$\nabla_{W^o} L \quad \text{et} \quad \nabla_{W^1} L$$

Histoire de gradients

Simple RN de classification avec entropie croisée

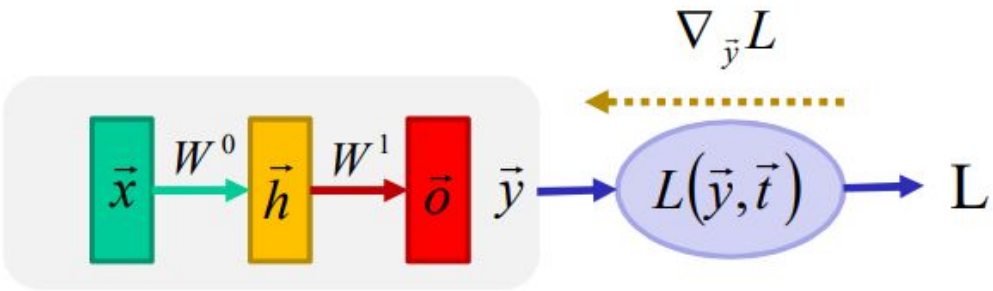


$$\vec{h} = \tanh\left(W^0\vec{x}\right)$$
$$\vec{o} = W^1\vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y},\vec{t}\right)$$

Dérivée en chaîne

$$\nabla_{W^1} L = \nabla_{\vec{y}} L \nabla_{\vec{o}} \vec{y} \nabla_{W^1} \vec{o}$$
$$\nabla_{W^o} L = \nabla_{\vec{y}} L \nabla_{\vec{o}} \vec{y} \nabla_{\vec{h}} \vec{o} \nabla_{W^0} \vec{h}$$

Histoire de gradients

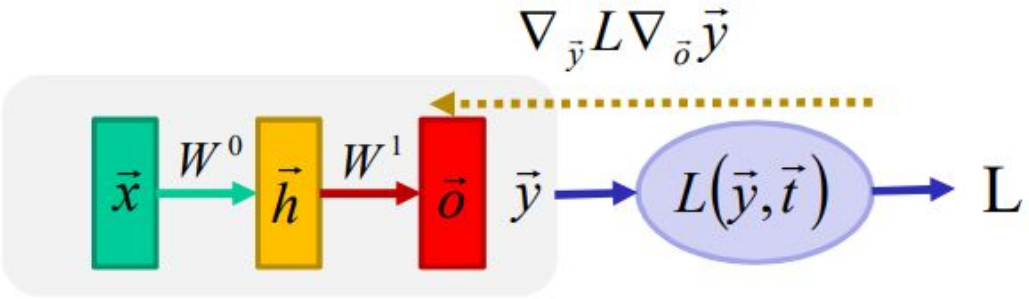


$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y},\vec{t}\right)$$

$$\nabla_{\vec{y}} L = -\frac{\vec{t}}{\vec{y}}$$

Rétro-propagation

Histoire de gradients

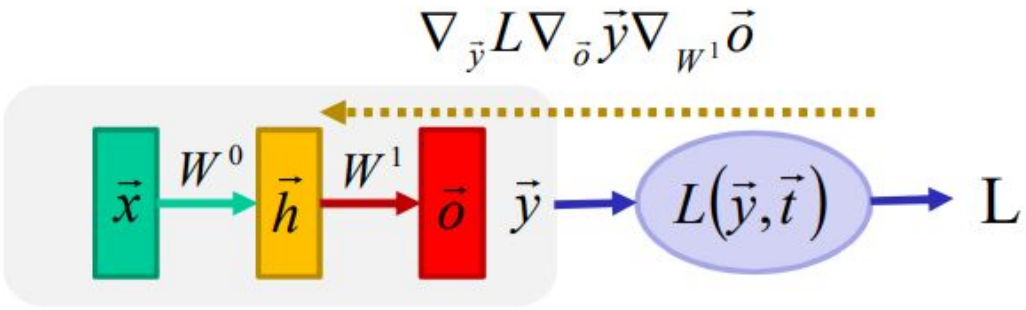


$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y},\vec{t}\right)$$

$$\nabla_{\vec{o}} \vec{y} = \mathbf{I}\vec{y}^T - \vec{y}^T \vec{y}$$
$$\nabla_{\vec{y}} L = -\frac{\vec{t}}{\vec{y}}$$

Rétro-propagation

Histoire de gradients



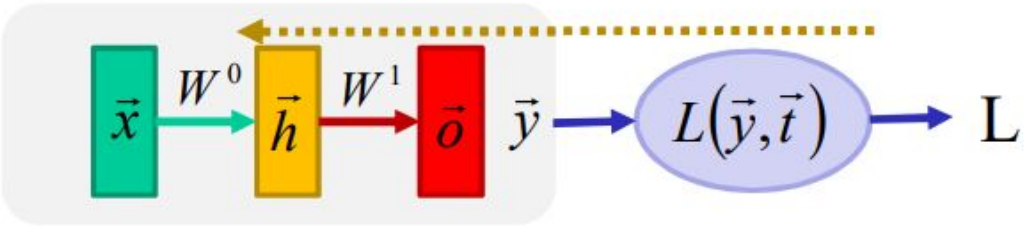
$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y},\vec{t}\right)$$

$$\nabla_{W^1} \vec{o} = \vec{h}$$
$$\nabla_{\vec{o}} \vec{y} = \mathbf{I}\vec{y}^T - \vec{y}^T \vec{y}$$
$$\nabla_{\vec{y}} L = -\frac{\vec{t}}{\vec{y}}$$

Rétro-propagation

Histoire de gradients

$$\nabla_{\vec{y}} L \nabla_{\vec{o}} \vec{y} \nabla_{\vec{h}} \vec{o}$$



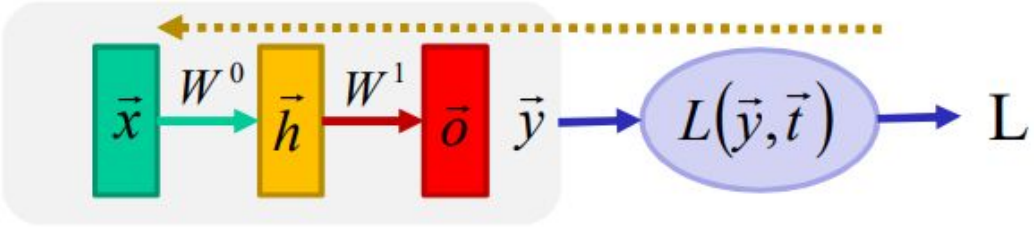
$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y}, \vec{t}\right)$$

$$\nabla_{\vec{h}} \vec{o} = W^1$$
$$\nabla_{W^1} \vec{o} = \vec{h}$$
$$\nabla_{\vec{o}} \vec{y} = \mathbf{I}\vec{y}^T - \vec{y}^T \vec{y}$$
$$\nabla_{\vec{y}} L = -\frac{\vec{t}}{\vec{y}}$$

Rétro-propagation

Histoire de gradients

$$\nabla_{\vec{y}} L \nabla_{\vec{o}} \vec{y} \nabla_{\vec{h}} \vec{o} \nabla_{W^0} \vec{h}$$

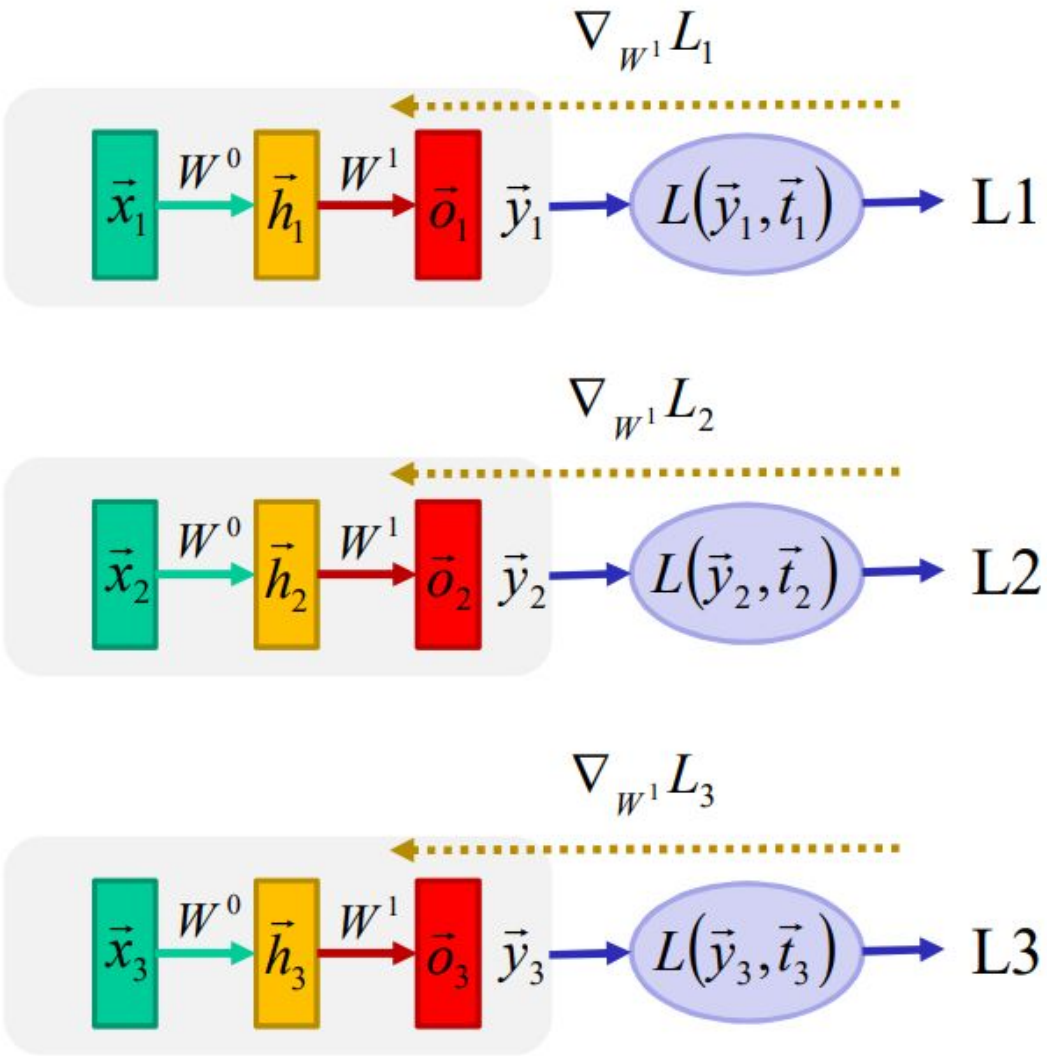


$$\vec{h} = \tanh\left(W^0 \vec{x}\right)$$
$$\vec{o} = W^1 \vec{h}$$
$$\vec{y} = S_M\left(\vec{o}\right)$$
$$L = L_{CE}\left(\vec{y}, \vec{t}\right)$$

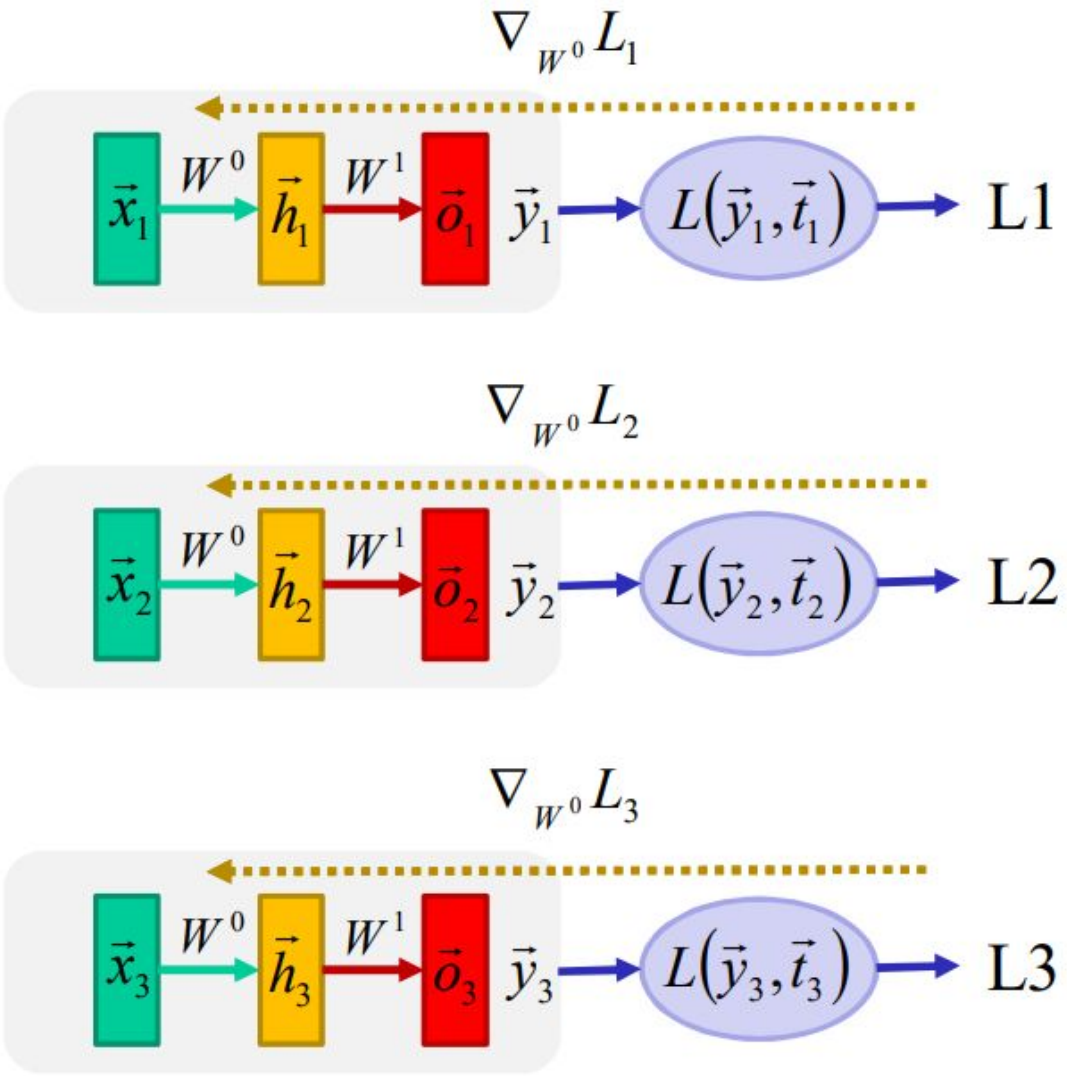
Rétro-propagation

$$\nabla_{W^0} \vec{h} = 1 - \tanh^2(W^0 \vec{x}) \vec{x}$$
$$\nabla_{\vec{h}} \vec{o} = W^1$$
$$\nabla_{W^1} \vec{o} = \vec{h}$$
$$\nabla_{\vec{o}} \vec{y} = \mathbf{I}\vec{y}^T - \vec{y}^T \vec{y}$$
$$\nabla_{\vec{y}} L = -\frac{\vec{t}}{\vec{y}}$$

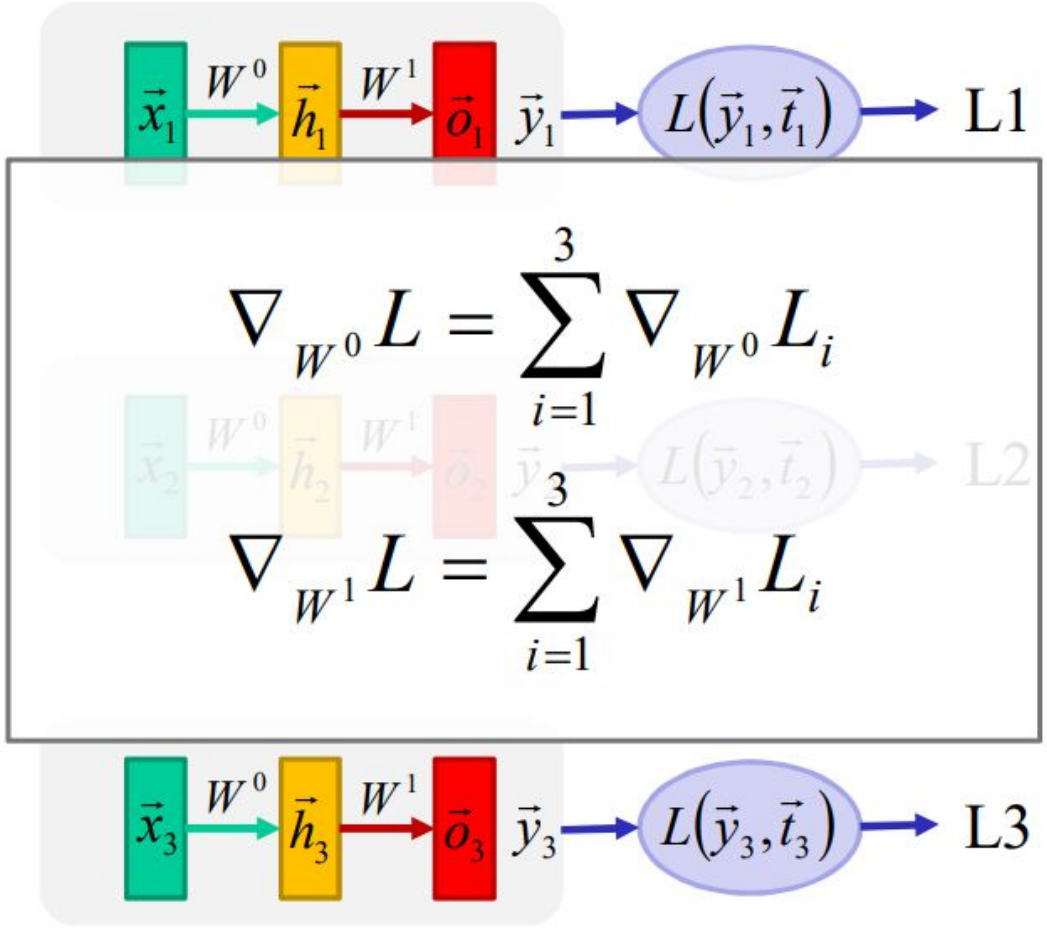
Ex.: 3 données, 3 rétro-propagations



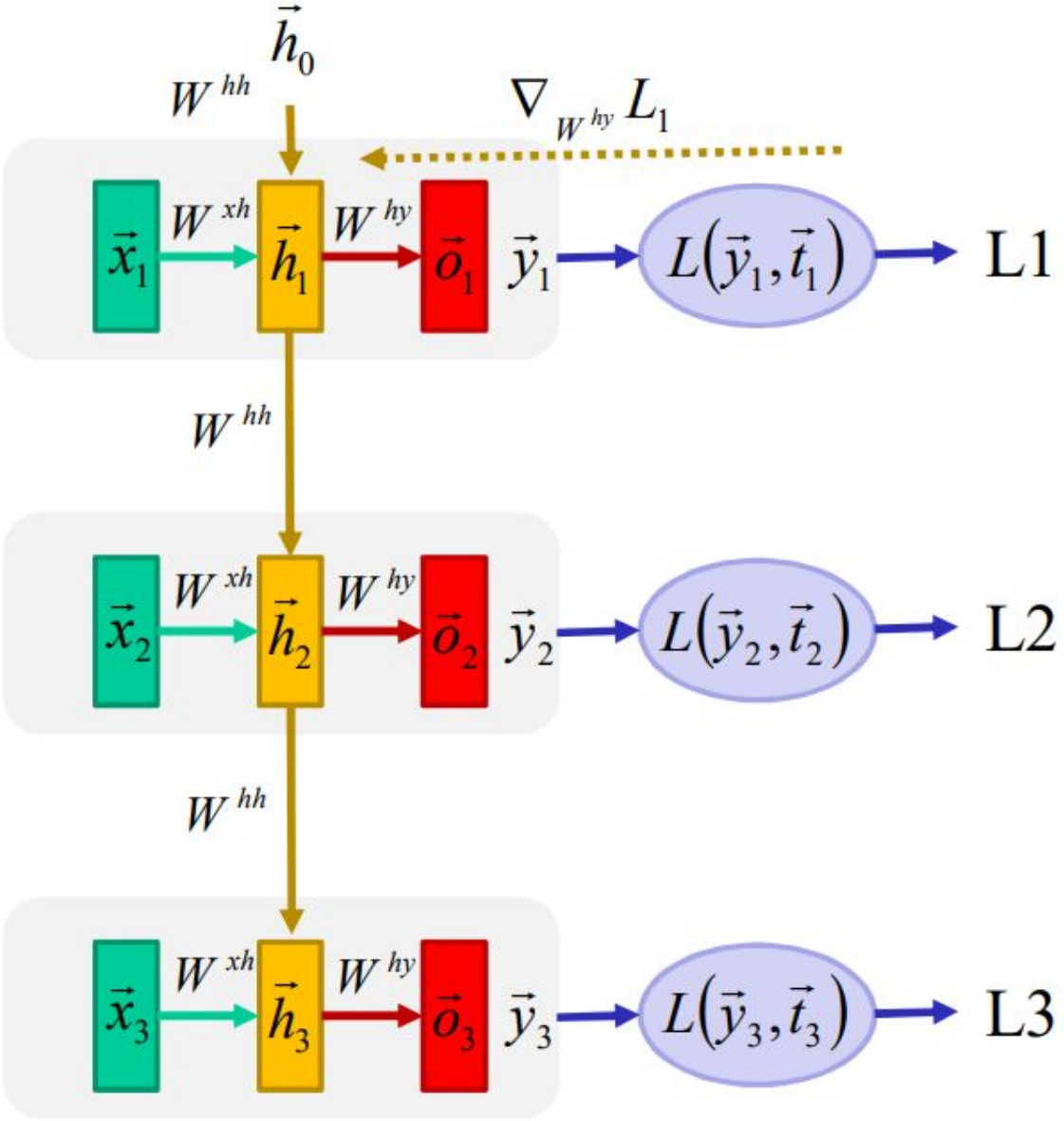
Ex.: 3 données, 3 rétro-propagations



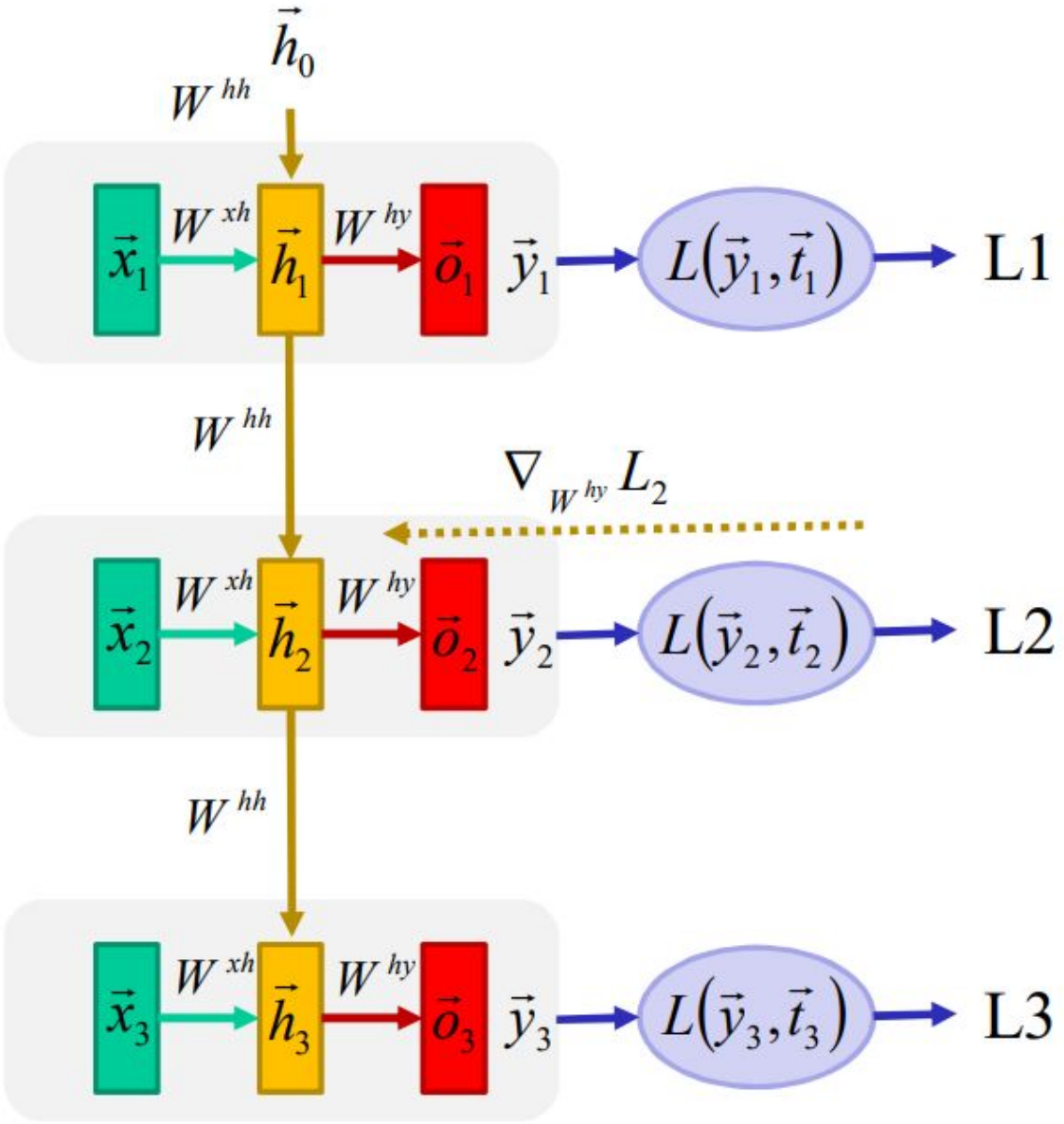
3 rétro-propagations



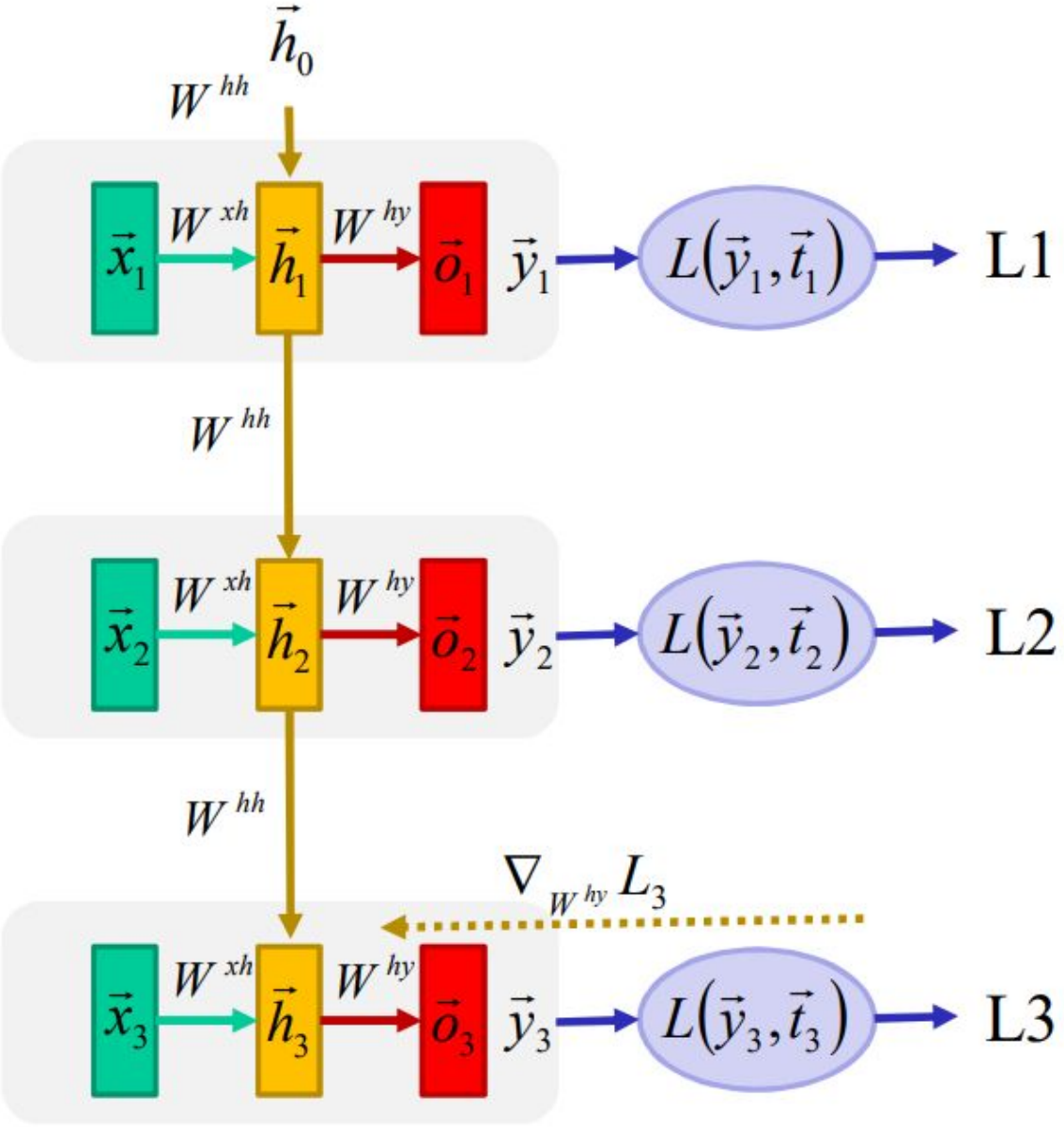
Réseau récurrent: gradient pour W^{hy}



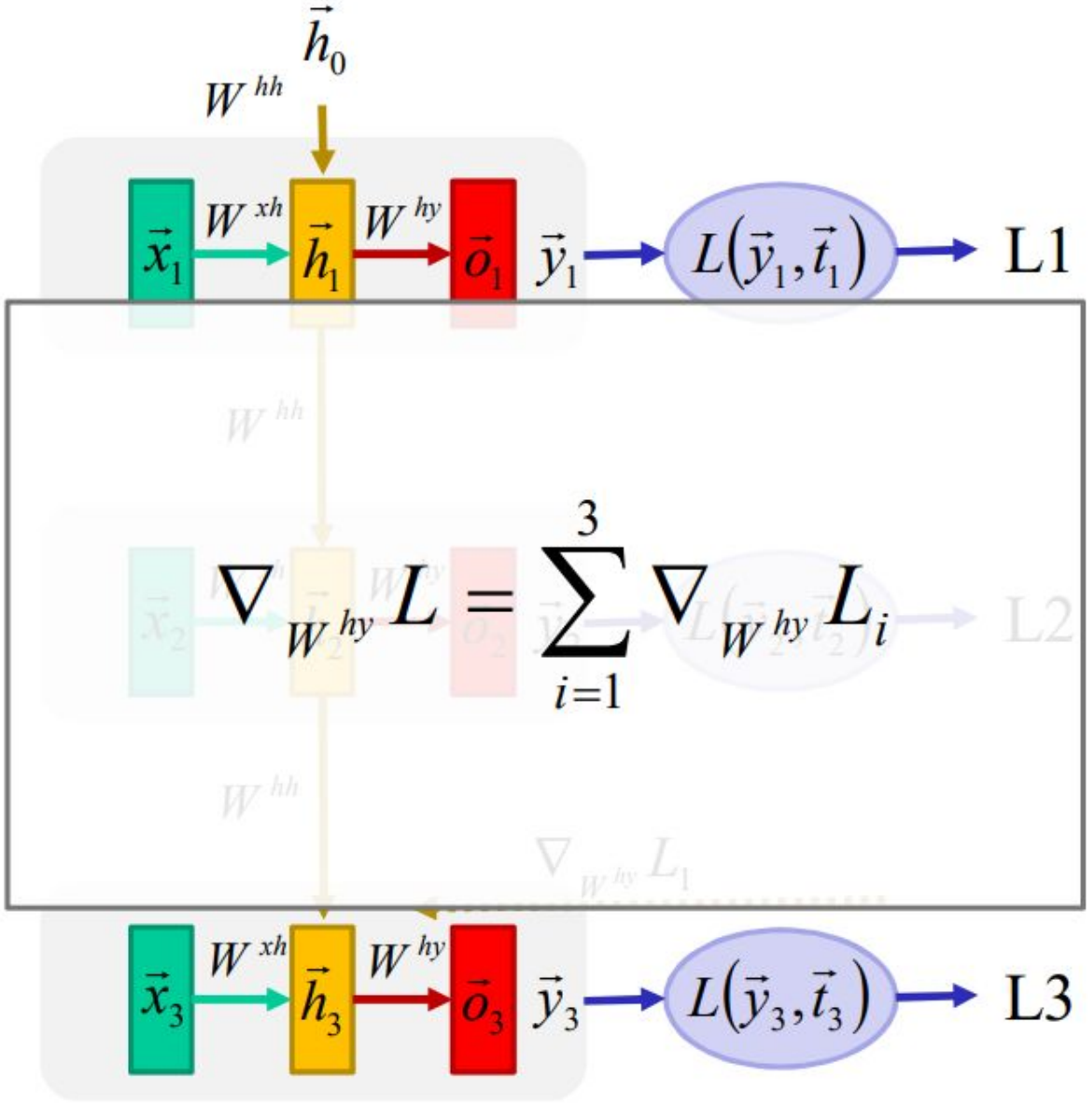
Réseau récurrent: gradient pour W^{hy}



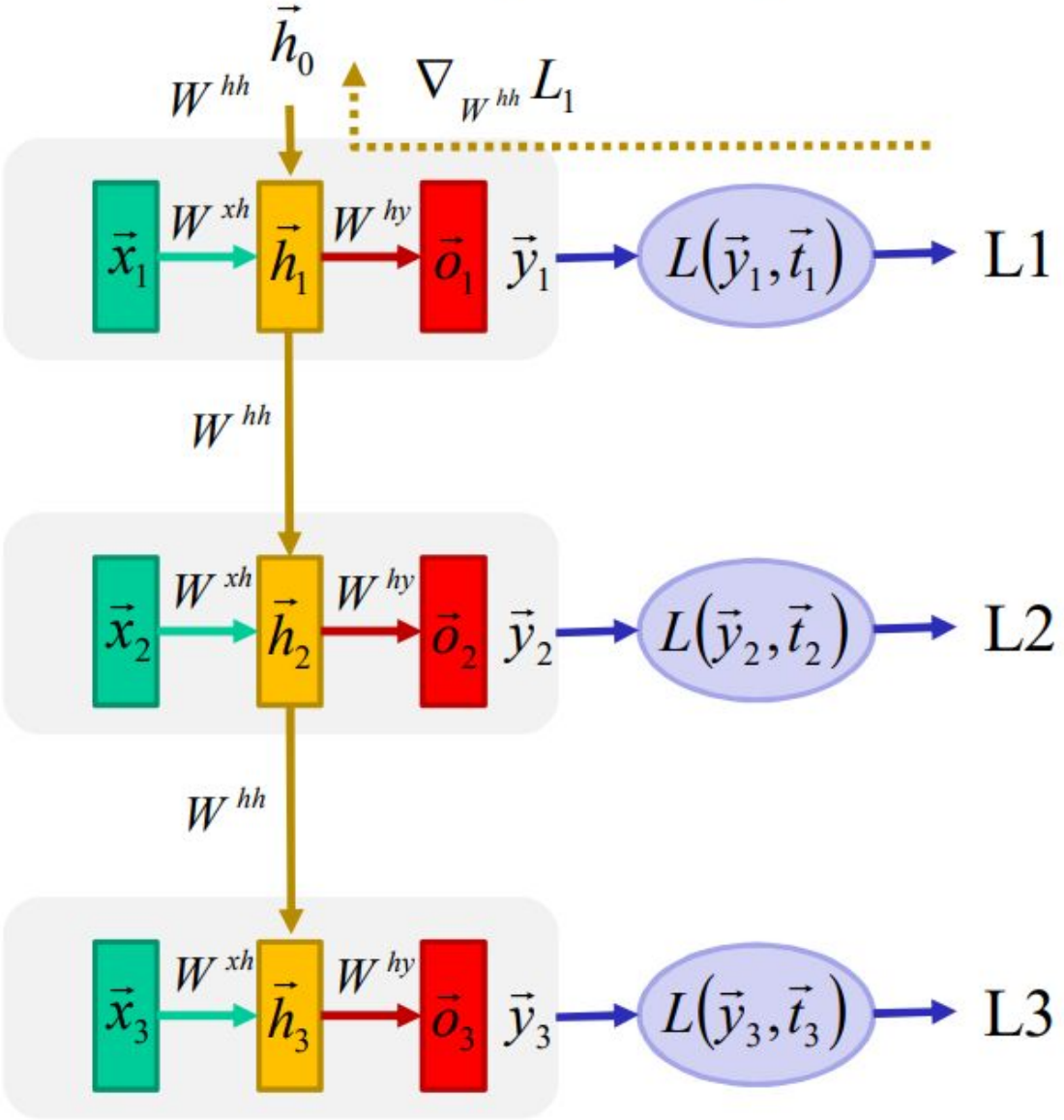
Réseau récurrent: gradient pour W^{hy}



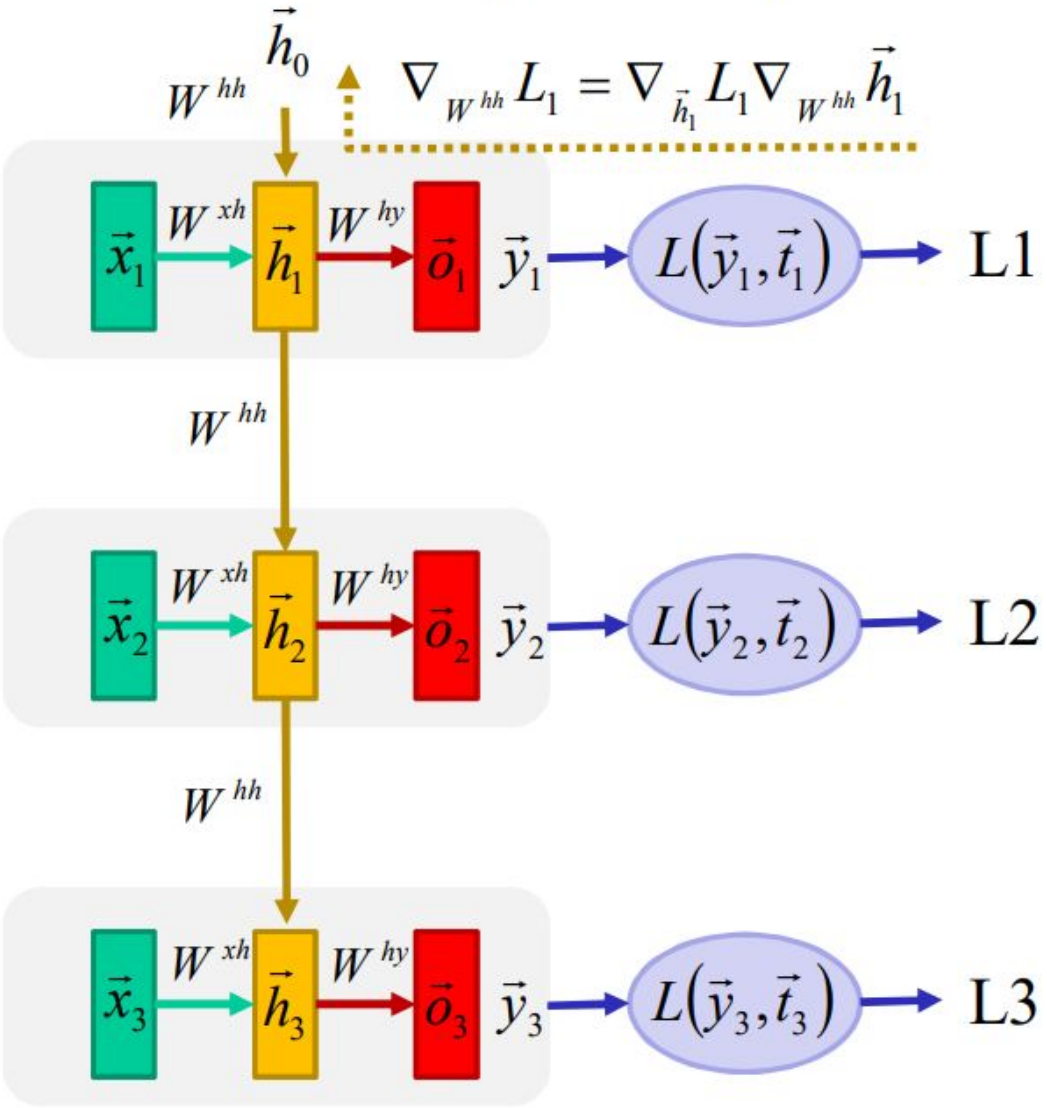
Réseau récurrent: gradient pour W^{hy}



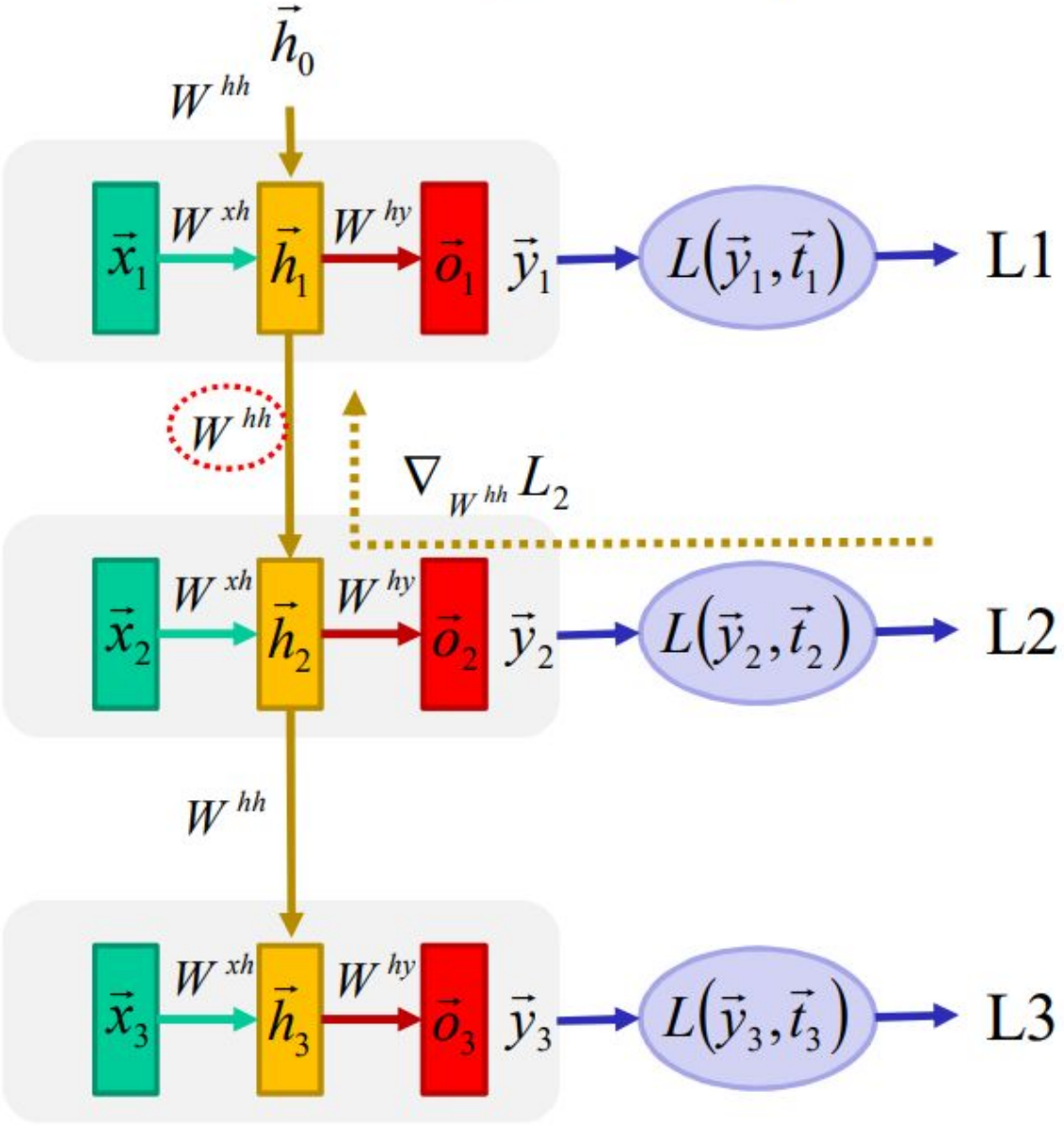
Réseau récurrent: gradient pour W^{hh}



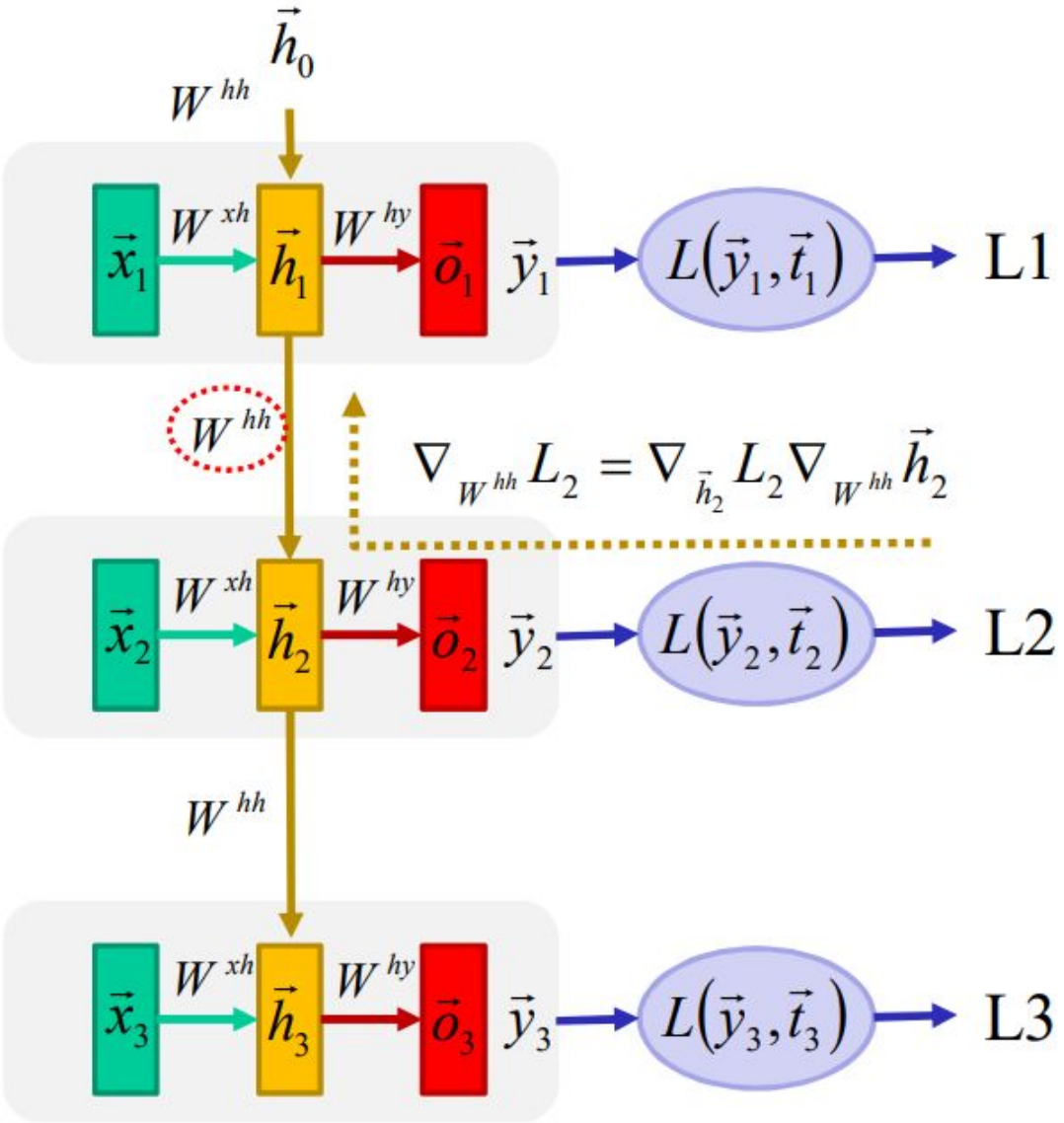
Réseau récurrent: gradient pour W^{hh}



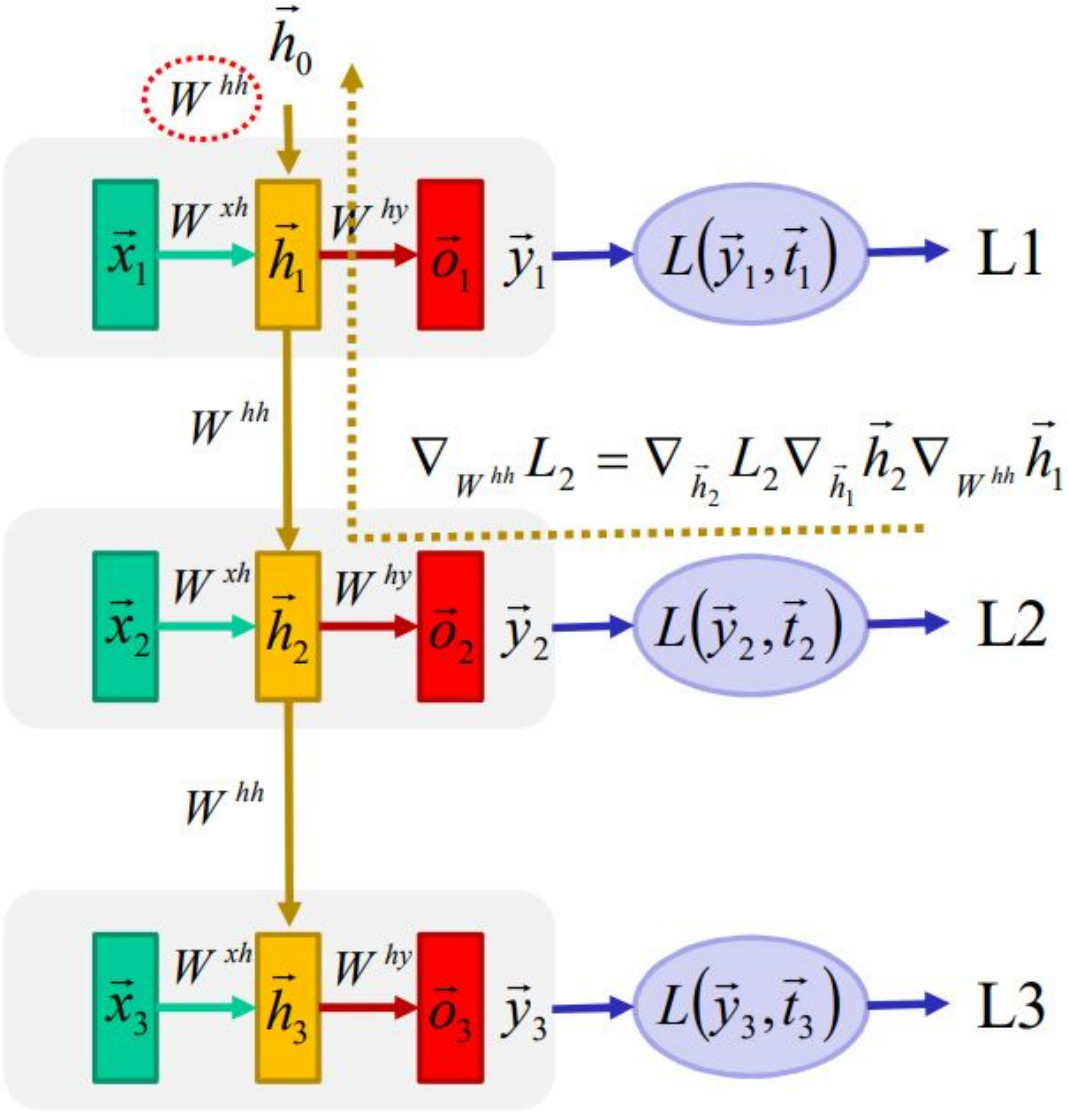
Réseau récurrent: gradient pour W^{hh}



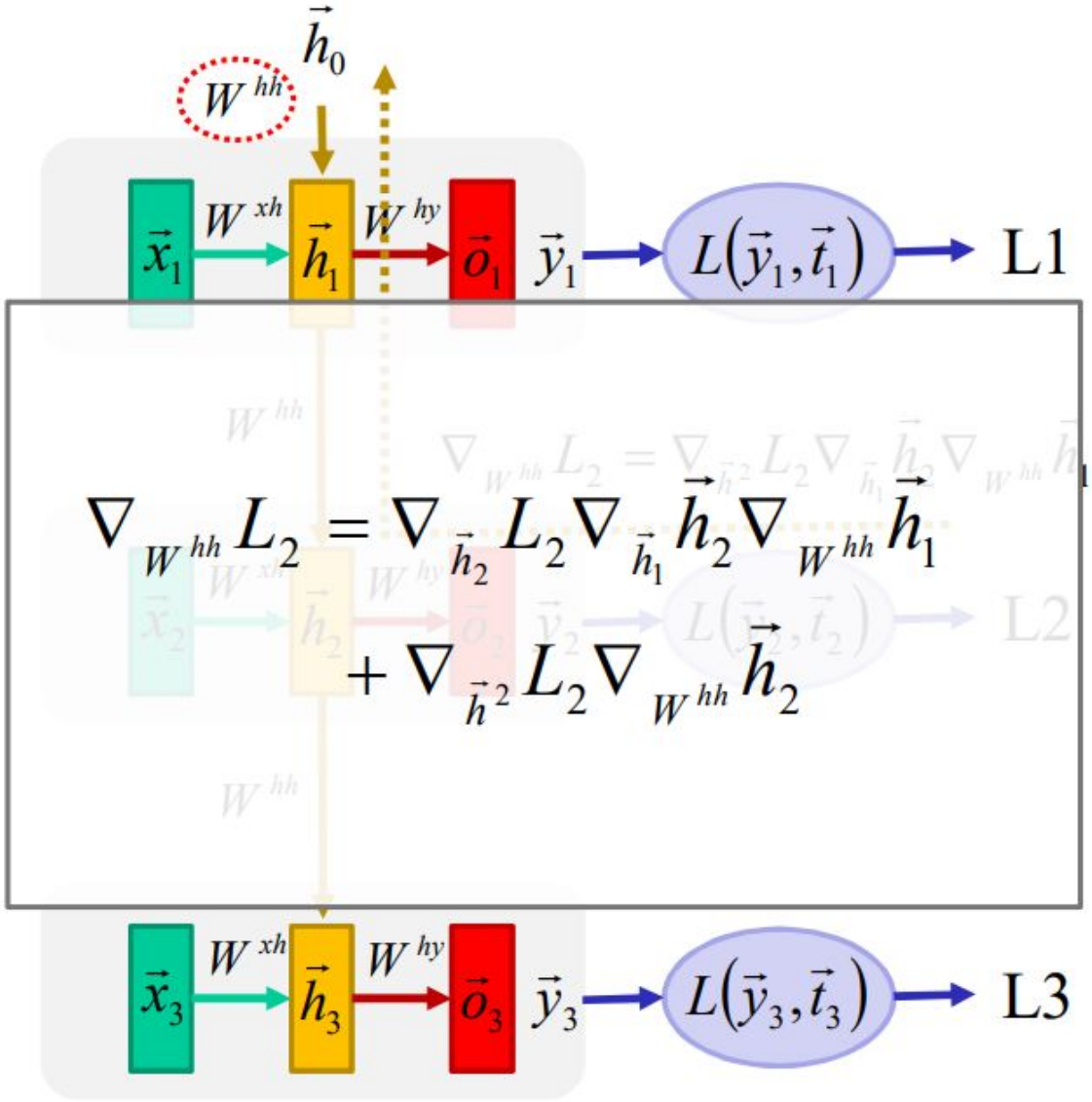
Réseau récurrent: gradient pour W^{hh}



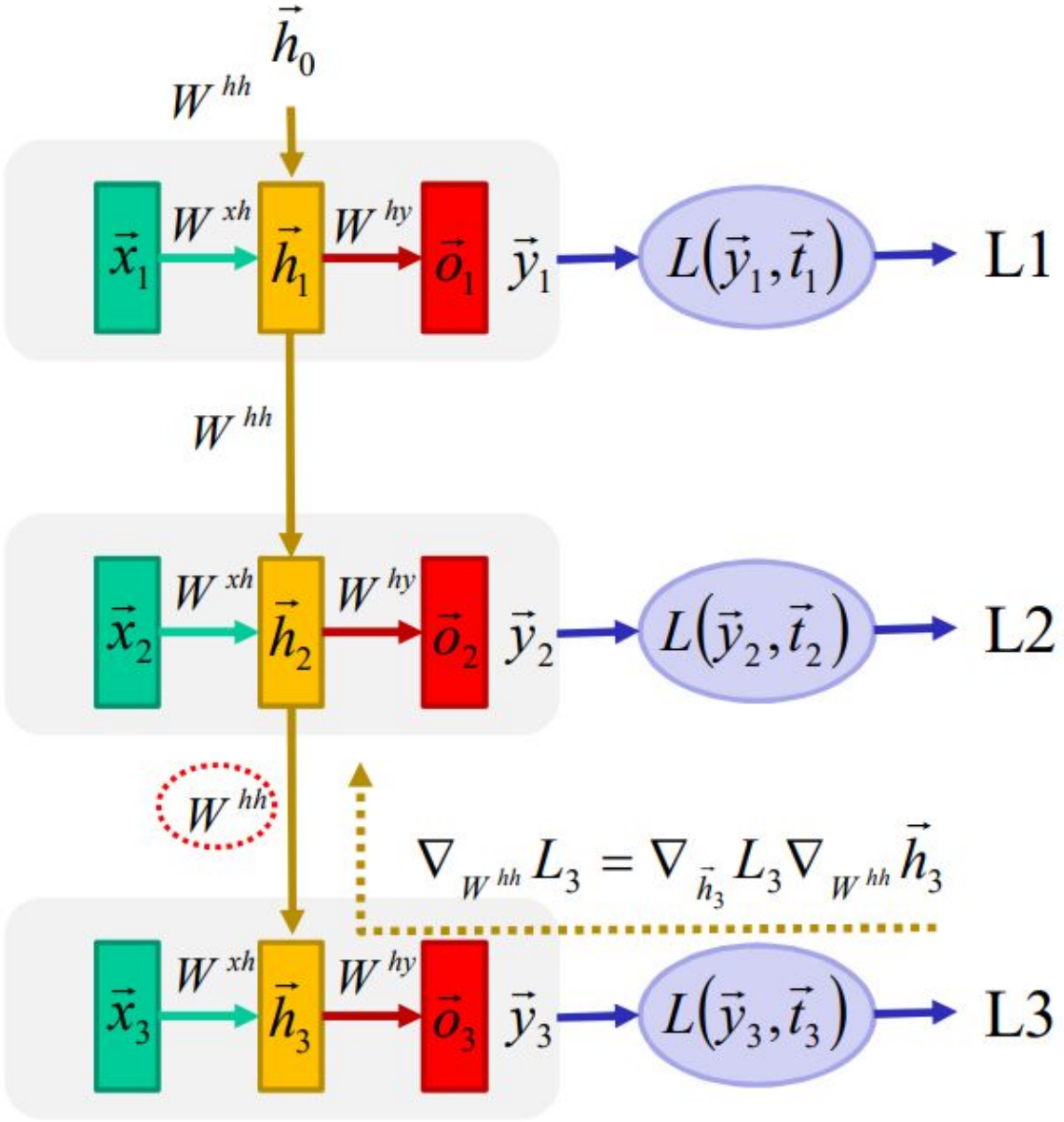
Réseau récurrent: gradient pour W^{hh}



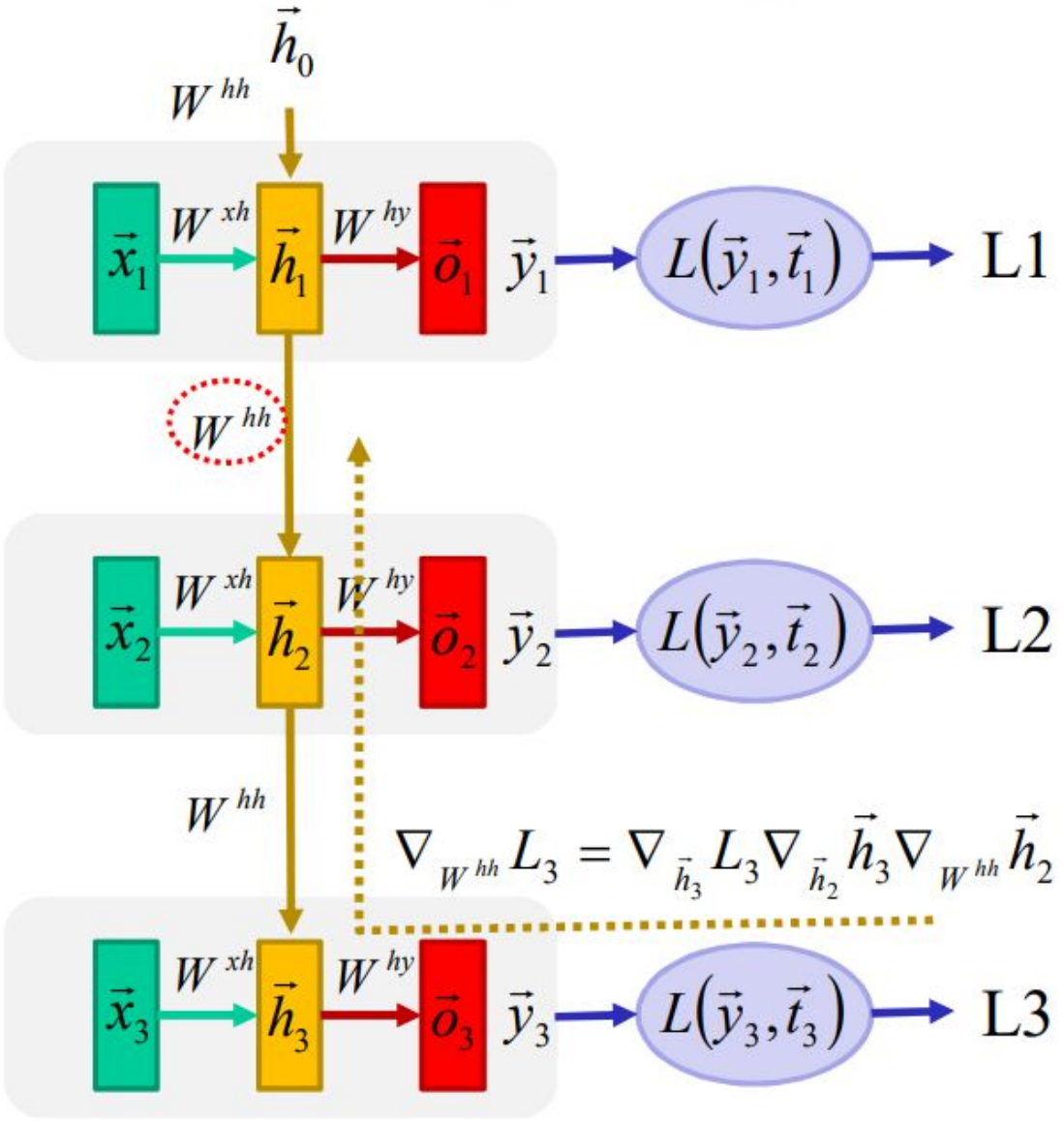
Réseau récurrent: gradient pour W^{hh}



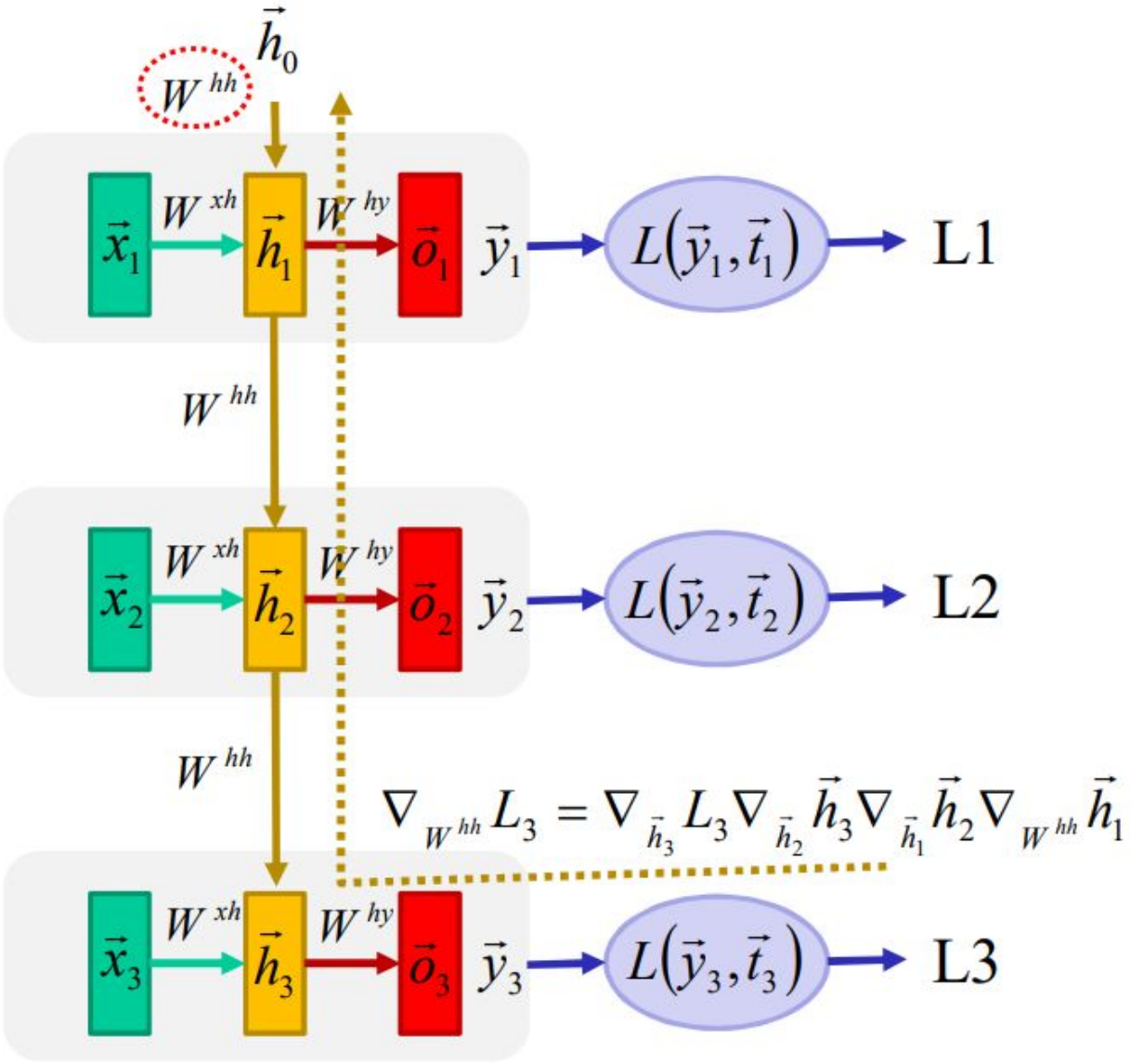
Réseau récurrent: gradient pour W^{hh}



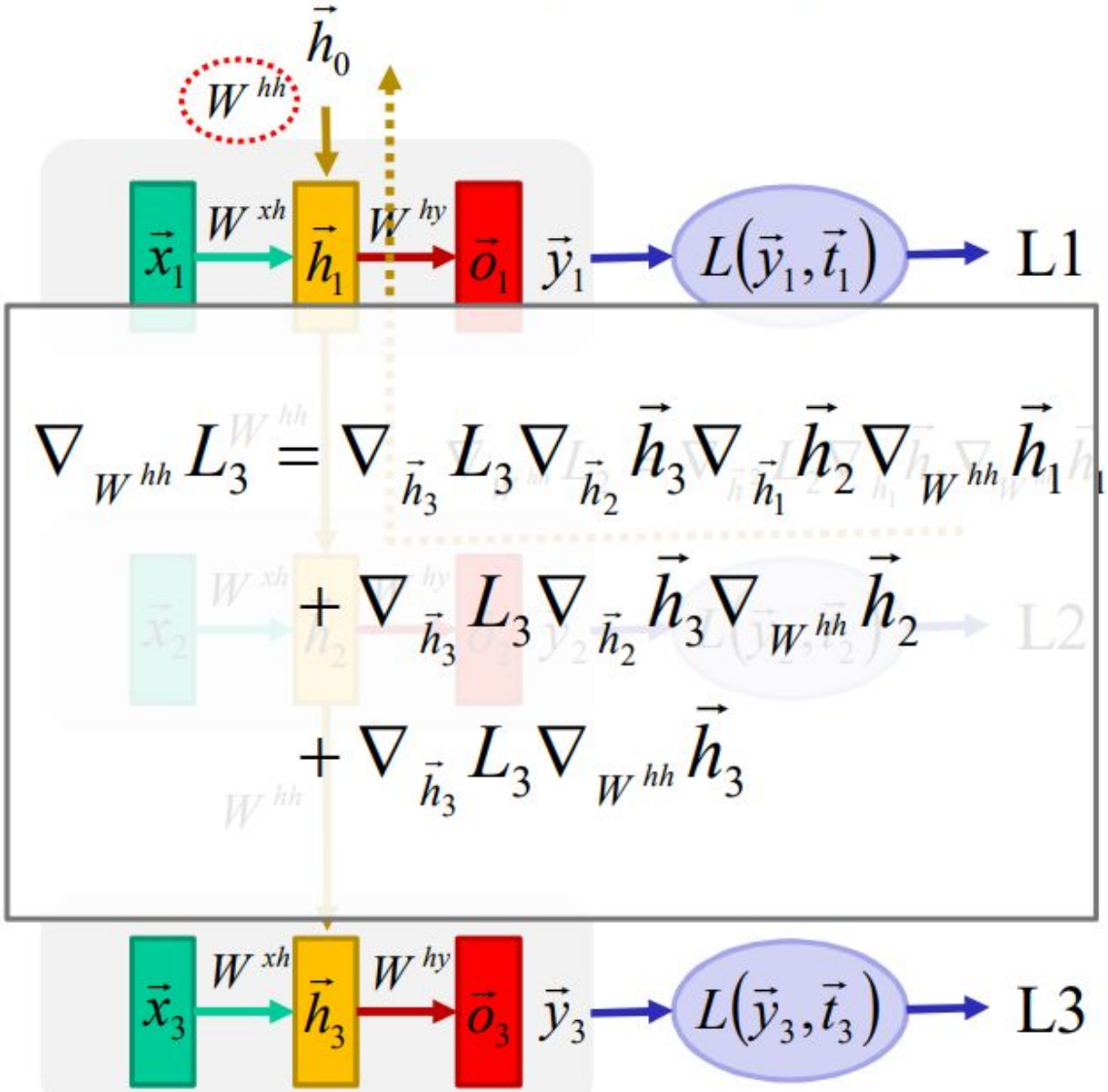
Réseau récurrent: gradient pour W^{hh}



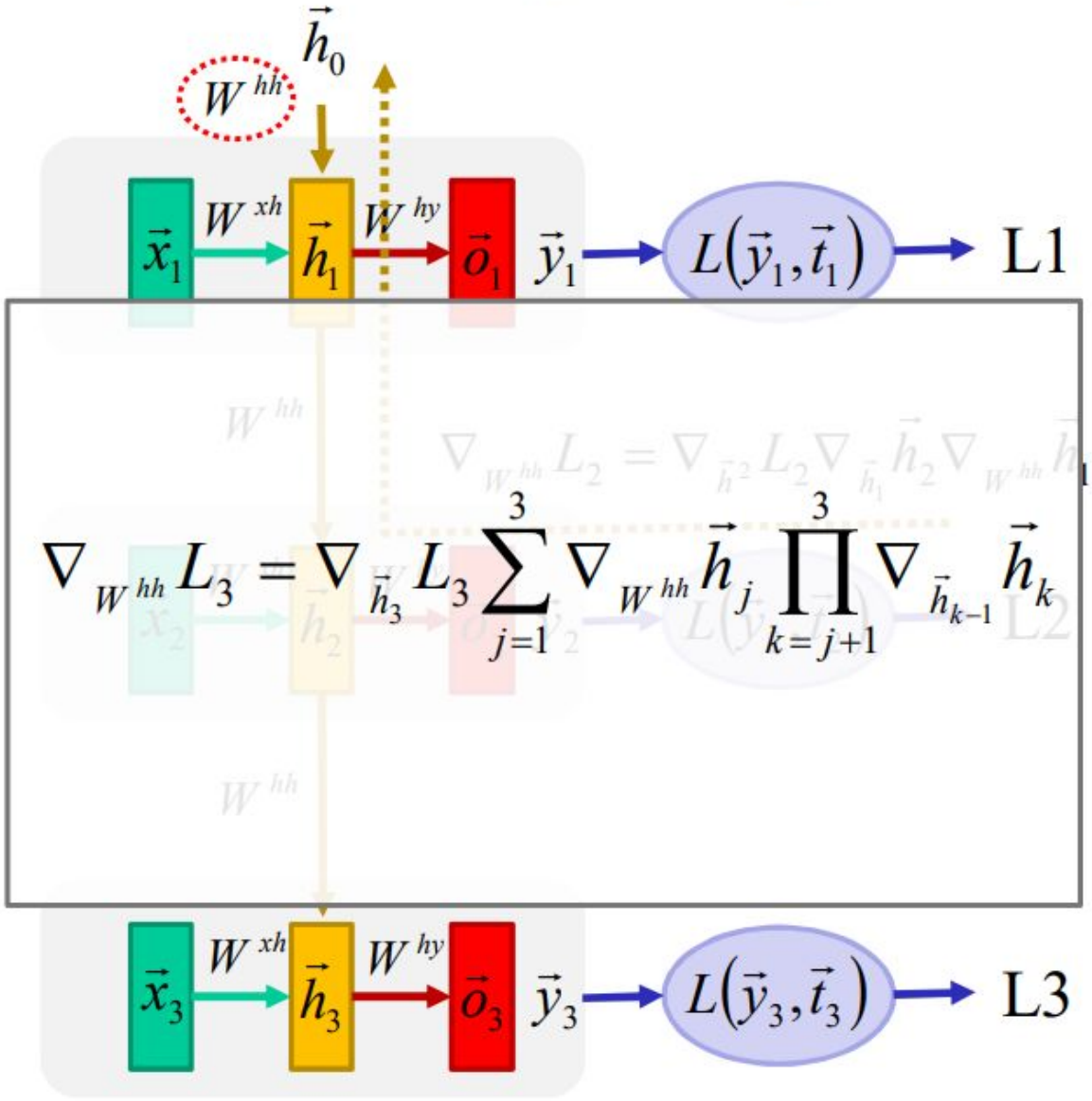
Réseau récurrent: gradient pour W^{hh}



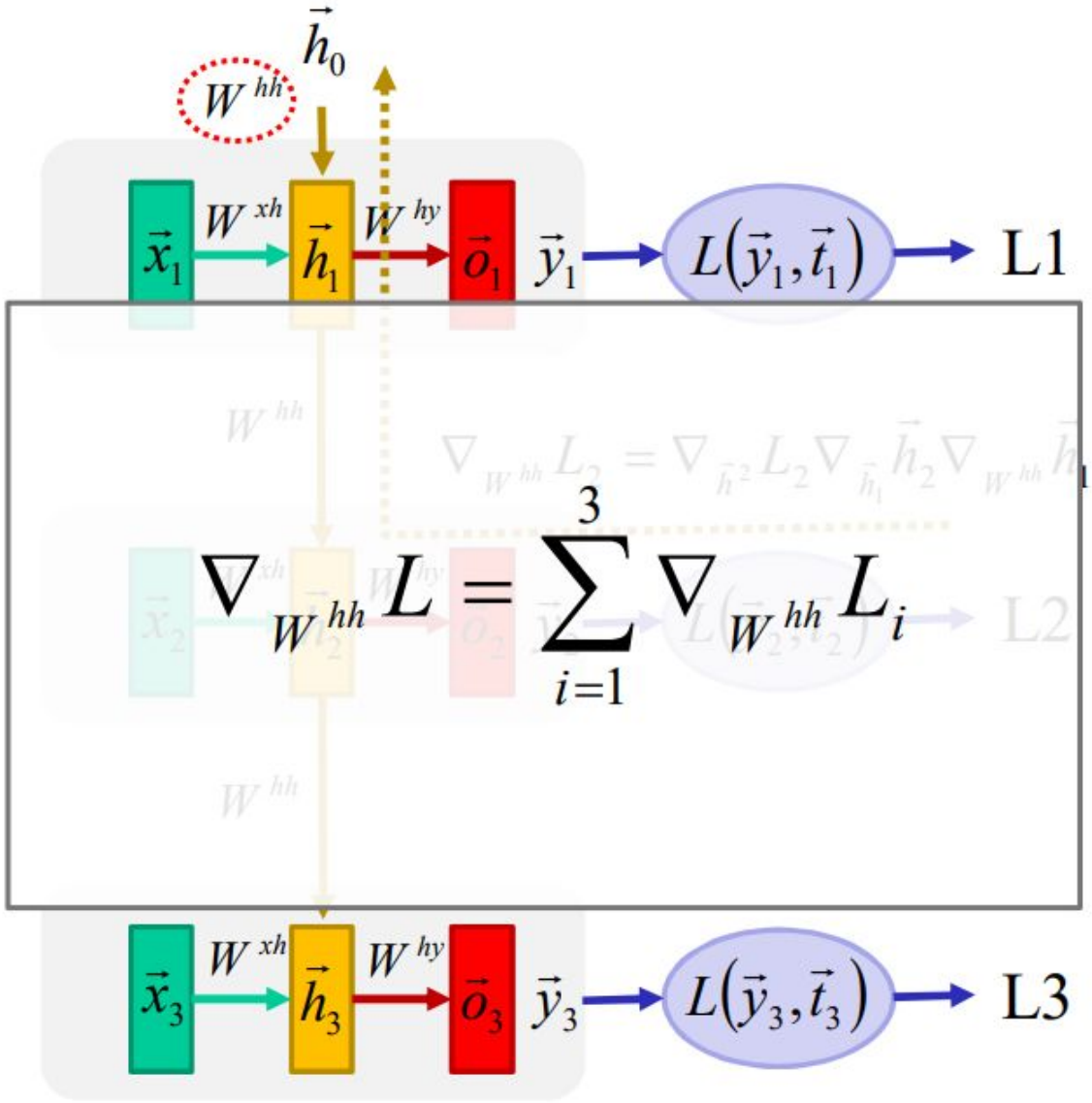
Réseau récurrent: gradient pour W^{hh}



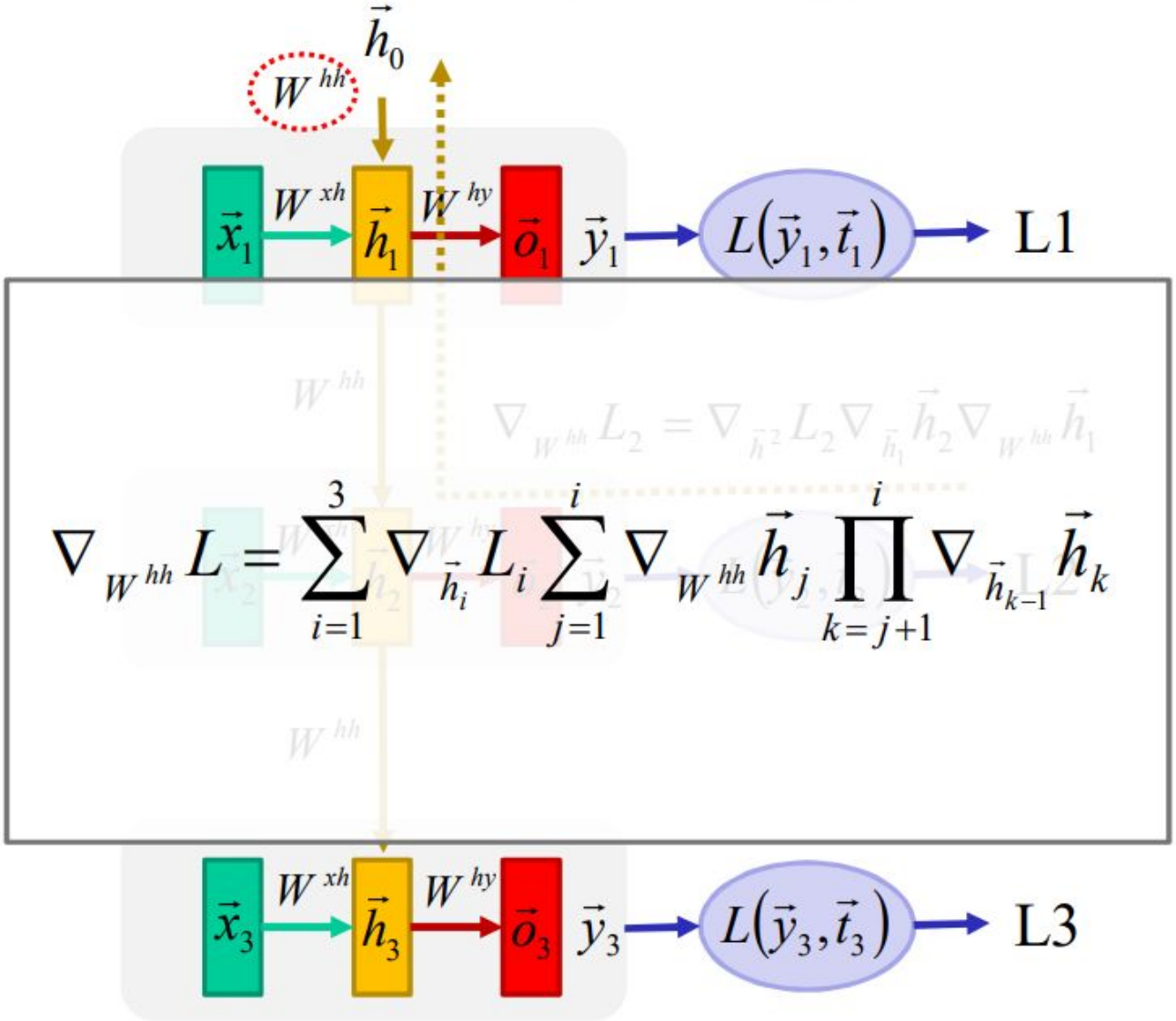
Réseau récurrent: gradient pour W^{hh}



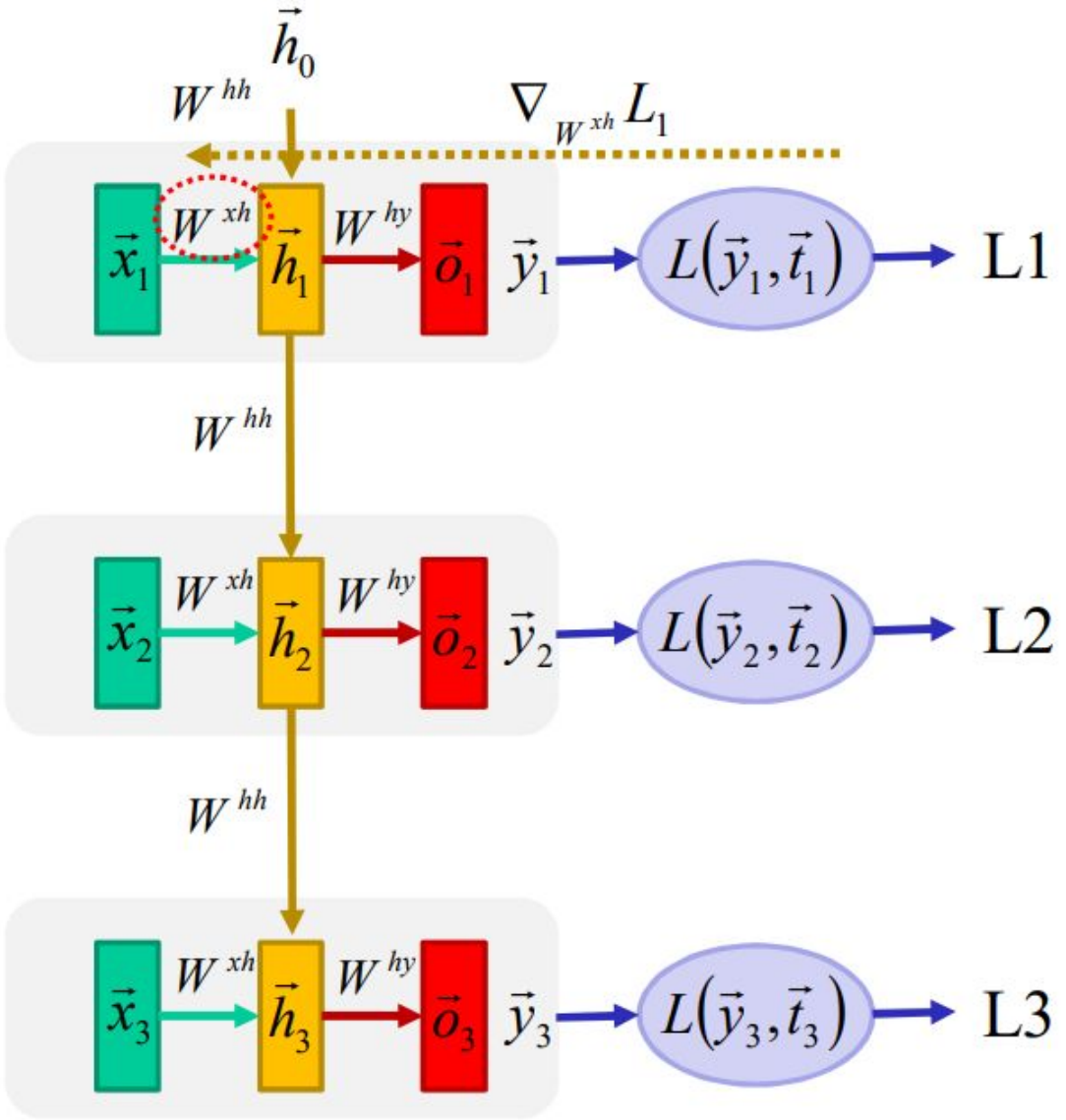
Réseau récurrent: gradient pour W^{hh}



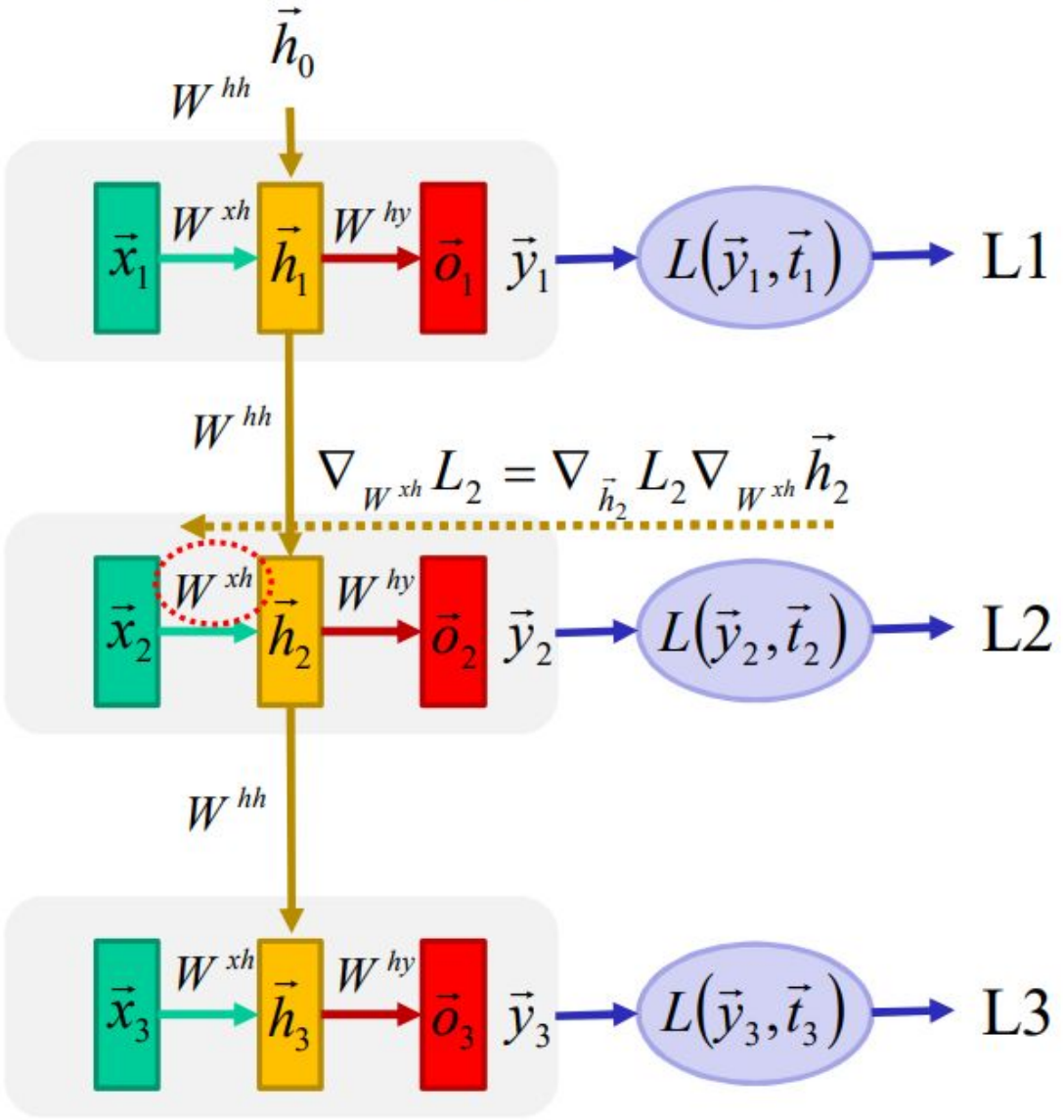
Réseau récurrent: gradient pour W^{hh}



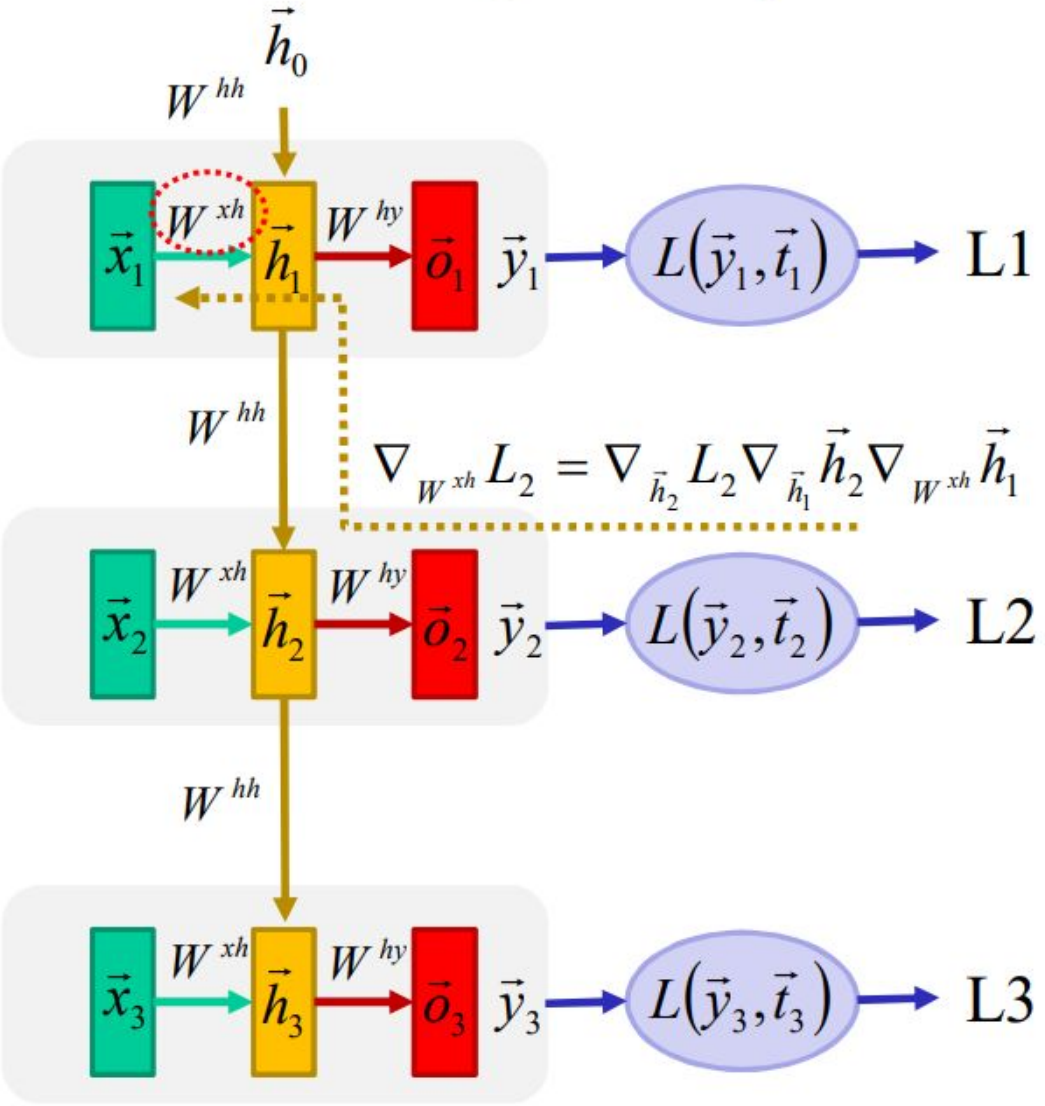
Réseau récurrent: gradient pour W^{xh}



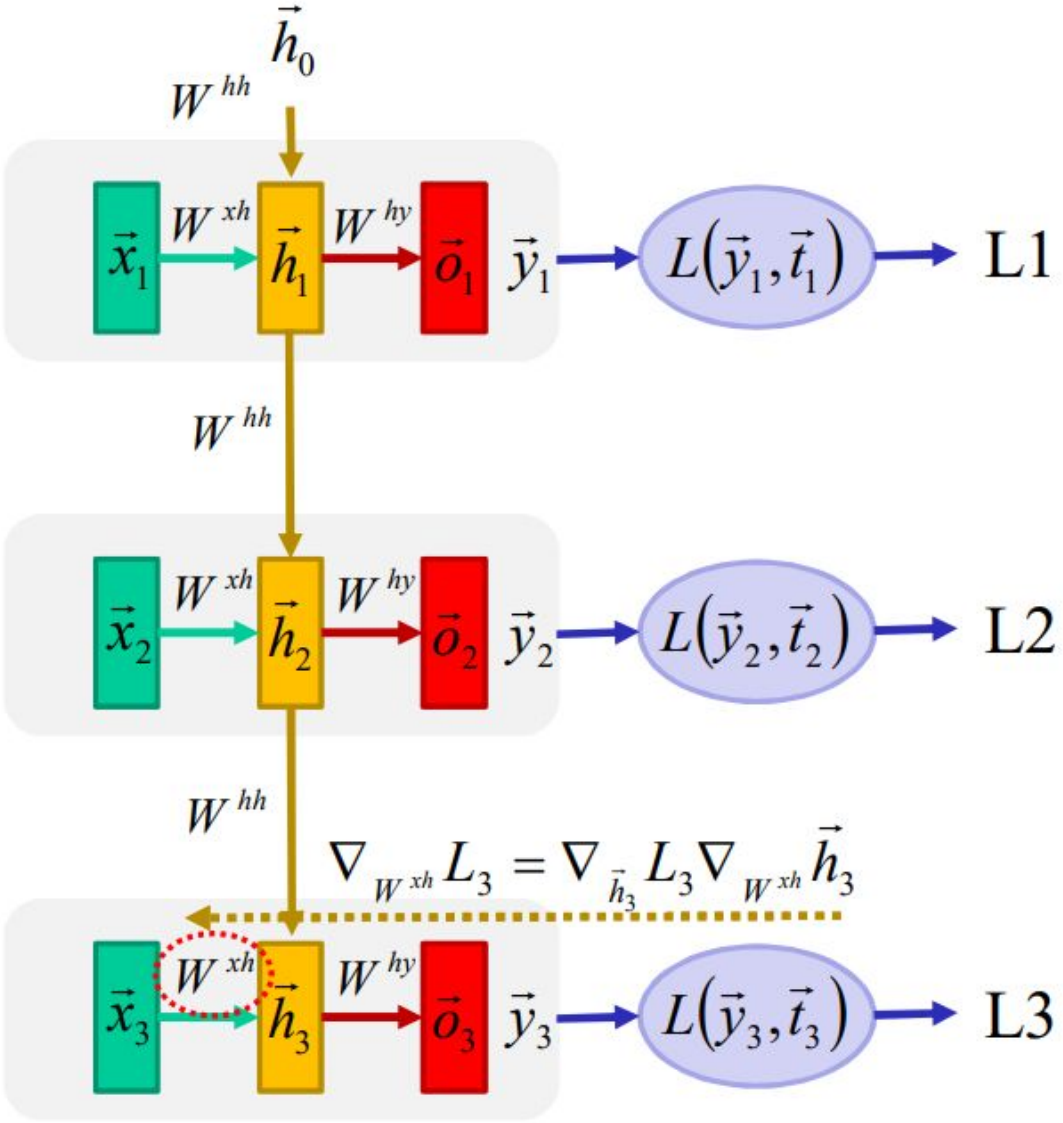
Réseau récurrent: gradient pour W^{xh}



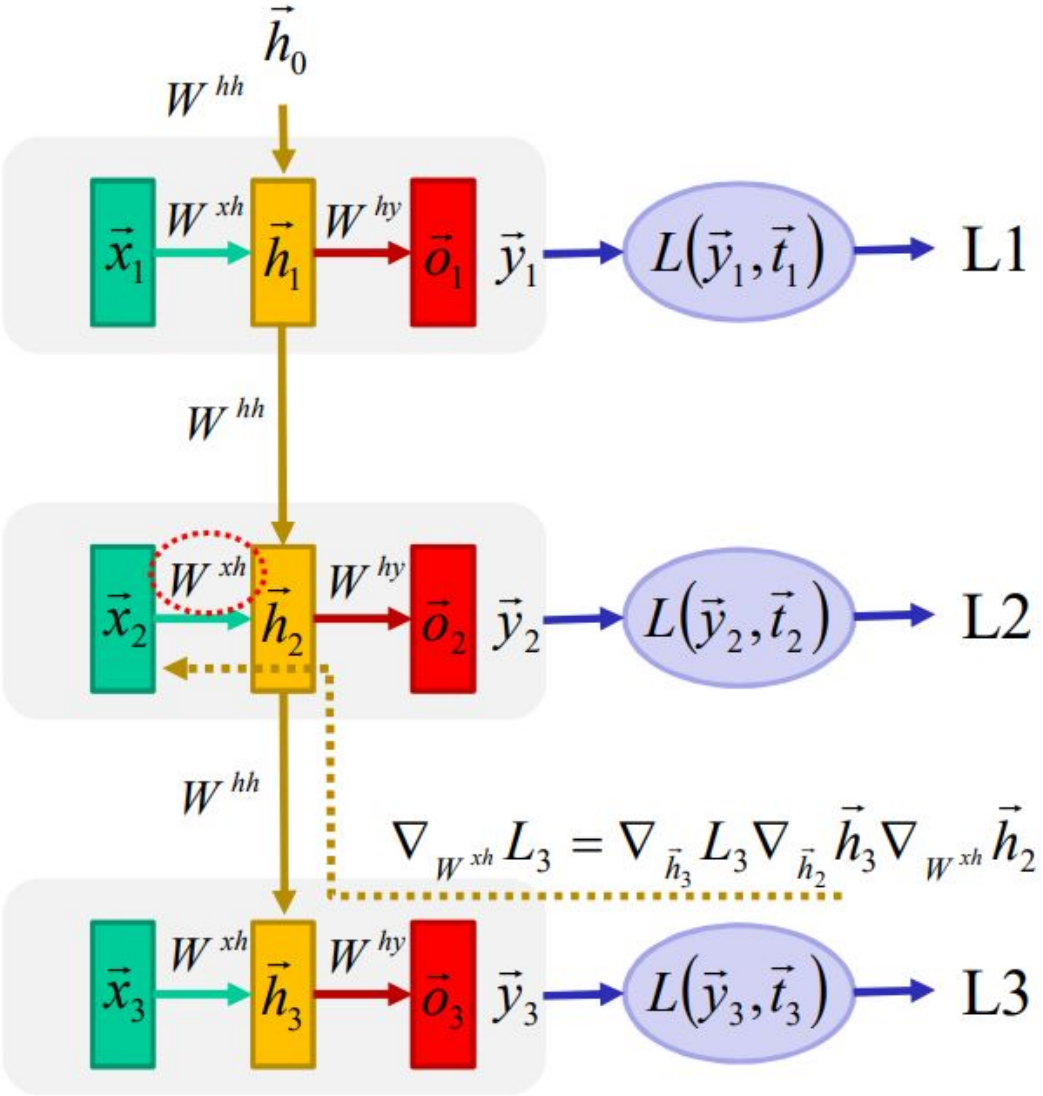
Réseau récurrent: gradient pour W^{xh}



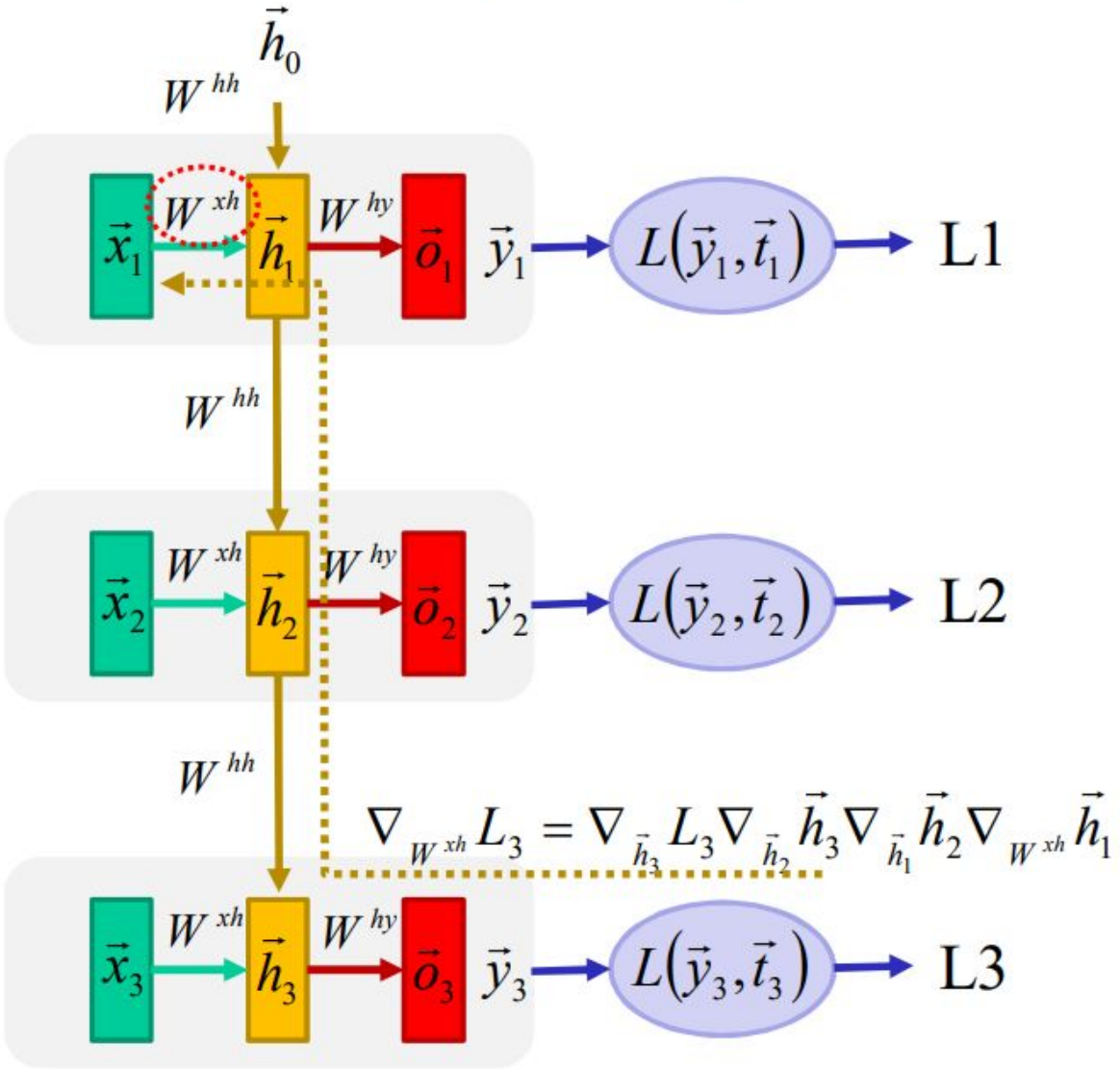
Réseau récurrent: gradient pour W^{xh}



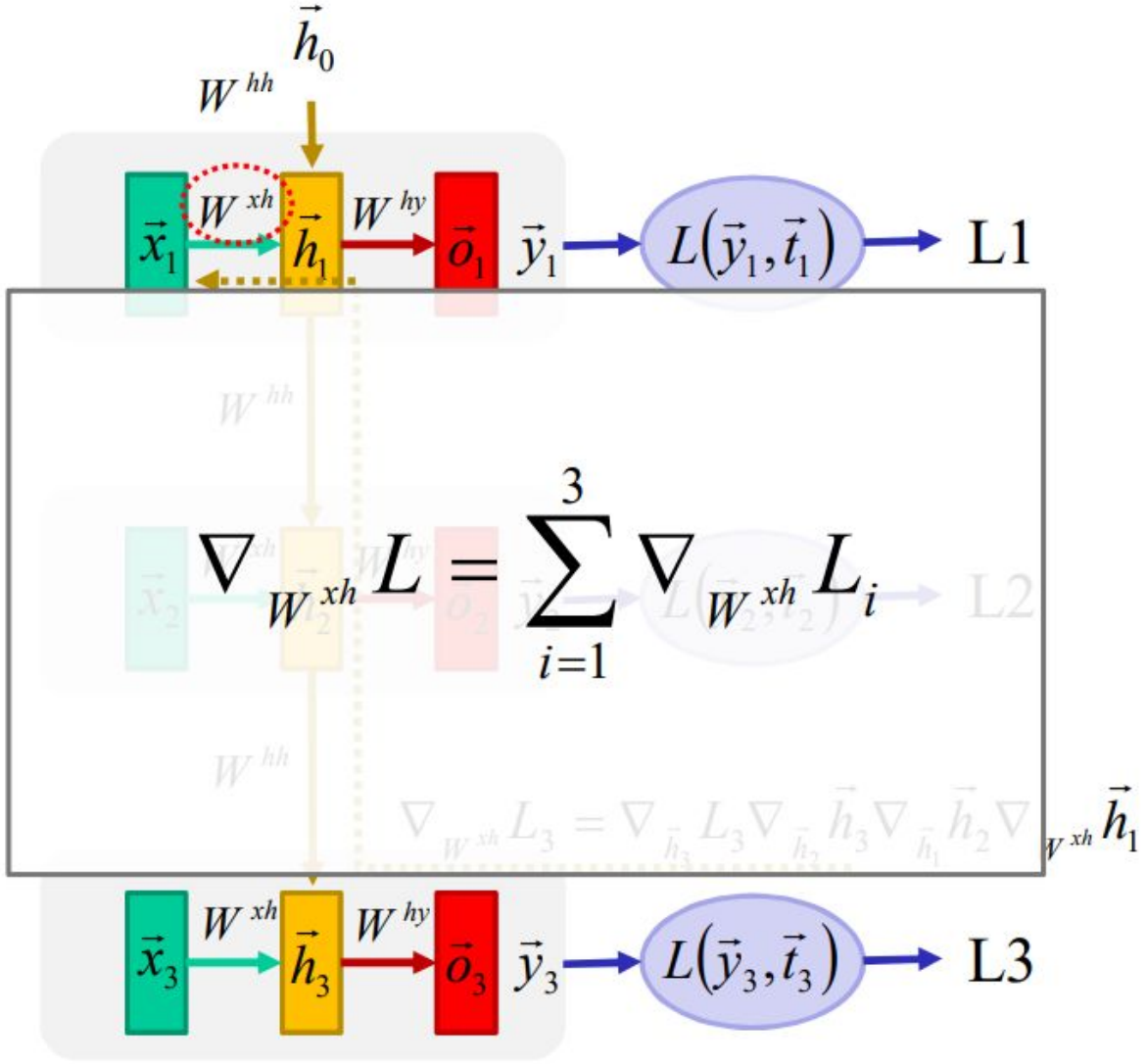
Réseau récurrent: gradient pour W^{xh}



Réseau récurrent: gradient pour W^{xh}



Réseau récurrent: gradient pour W^{xh}



Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !
En réorganisant les termes:

$$\nabla_{W^{xh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{xh}} h_t = \sum_t^T \nabla_{h_t} L \cdot x_t$$

$$\nabla_{W^{hh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{hh}} h_t = \sum_t^T \nabla_{h_t} L \cdot h_{t-1}$$

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !
En réorganisant les termes:

$$\nabla_{W^{xh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{xh}} h_t = \sum_t^T \nabla_{h_t} L \cdot x_t$$

$$\nabla_{W^{hh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{hh}} h_t = \sum_t^T \nabla_{h_t} L \cdot h_{t-1}$$

à définir

obtenus lors de la propagation avant

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !
En réorganisant les termes:

$$\nabla_{W^{xh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{xh}} h_t = \sum_t^T \nabla_{h_t} L \cdot x_t$$

$$\nabla_{W^{hh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{hh}} h_t = \sum_t^T \nabla_{h_t} L \cdot h_{t-1}$$

$$\nabla_{h_t} L = \nabla_{h_{t+1}} L \cdot \nabla_{h_t} h_{t+1} + \nabla_{y_t} L \cdot \nabla_{h_t} y_t = W^{hh} \nabla_{h_{t+1}} L + W^{hy} \nabla_{y_t} L$$

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !
En réorganisant les termes:

$$\nabla_{W^{xh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{xh}} h_t = \sum_t^T \nabla_{h_t} L \cdot x_t$$

$$\nabla_{W^{hh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{hh}} h_t = \sum_t^T \nabla_{h_t} L \cdot h_{t-1}$$

$$\nabla_{h_t} L = \nabla_{h_{t+1}} L \cdot \nabla_{h_t} h_{t+1} + \nabla_{y_t} L \cdot \nabla_{h_t} y_t = W^{hh} \nabla_{h_{t+1}} L + W^{hy} \nabla_{y_t} L$$

facile à calculer

récursivité, oh no !

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !
En réorganisant les termes:

$$\nabla_{W^{xh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{xh}} h_t = \sum_t^T \nabla_{h_t} L \cdot x_t$$

$$\nabla_{W^{hh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{hh}} h_t = \sum_t^T \nabla_{h_t} L \cdot h_{t-1}$$

$$\nabla_{h_t} L = \nabla_{h_{t+1}} L \cdot \nabla_{h_t} h_{t+1} + \nabla_{y_t} L \cdot \nabla_{h_t} y_t = W^{hh} \nabla_{h_{t+1}} L + W^{hy} \nabla_{y_t} L$$

$$\nabla_{h_T} L = \nabla_{y_t} L \cdot \nabla_{h_t} y_t = W^{hy} \nabla_{y_T} L$$

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !
En réorganisant les termes:

$$\nabla_{W^{xh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{xh}} h_t = \sum_t^T \nabla_{h_t} L \cdot x_t$$

$$\nabla_{W^{hh}} L = \sum_t^T \nabla_{h_t} L \cdot \nabla_{W^{hh}} h_t = \sum_t^T \nabla_{h_t} L \cdot h_{t-1}$$

$$\nabla_{h_t} L = \nabla_{h_{t+1}} L \cdot \nabla_{h_t} h_{t+1} + \nabla_{y_t} L \cdot \nabla_{h_t} y_t = W^{hh} \nabla_{h_{t+1}} L + W^{hy} \nabla_{y_t} L$$

$$\nabla_{h_T} L = \nabla_{y_t} L \cdot \nabla_{h_t} y_t = W^{hy} \nabla_{y_T} L$$

On peut donc partir de la fin T et calculer des sommes courantes !

Cas de base facile à calculer !

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !

```
44    # backward pass: compute gradients going backwards
45    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47    dhnext = np.zeros_like(hs[0])
48    for t in reversed(xrange(len(inputs))):
49      dy = np.copy(ps[t])
50      dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-study/#grad if confused here
51      dWhy += np.dot(dy, hs[t].T)
52      dby += dy
53      dh = np.dot(Why.T, dy) + dhnext # backprop into h
54      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55      dbh += dhraw
56      dWxh += np.dot(dhraw, xs[t].T)
57      dWhh += np.dot(dhraw, hs[t-1].T)
58      dhnext = np.dot(Whh.T, dhraw)
59    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
60      np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

Pas besoin de recalculer tous les termes intermédiaires pour chaque *timestep*

Voir https://d2l.ai/chapter_recurrent-neural-networks/bptt.html pour plus d'informatioms

Réseau récurrent: calcul du gradient

Semble complexe, mais ne l'est pas vraiment !

```
44    # backward pass: compute gradients going backwards
45    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47    dhnext = np.zeros_like(hs[0])
48    for t in reversed(xrange(len(inputs))):
49      dy = np.copy(ps[t])
50      dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-study/#grad if confused here
51      dWhy += np.dot(dy, hs[t].T)
52      dby += dy
53      dh = np.dot(Why.T, dy) + dhnext # backprop into h
54      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55      dbh += dhraw
56      dWxh += np.dot(dhraw, xs[t].T)
57      dWhh += np.dot(dhraw, hs[t-1].T)
58      dhnext = np.dot(Whh.T, dhraw)
59    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
60      np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

Pas besoin de recalculer tous les termes intermédiaires pour chaque *timestep*

Voir https://d2l.ai/chapter_recurrent-neural-networks/bptt.html pour plus d'informatioms

Exemples: analyse grammaticale

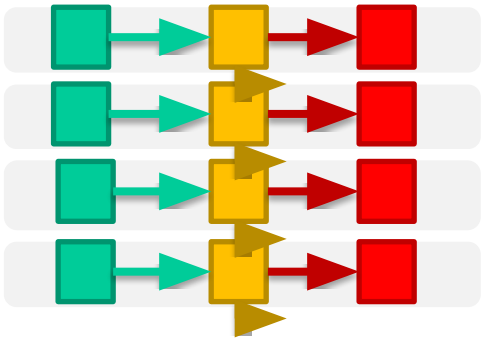
Entraîner un réseau à détecter des erreurs grammaticales

« La » **OK**

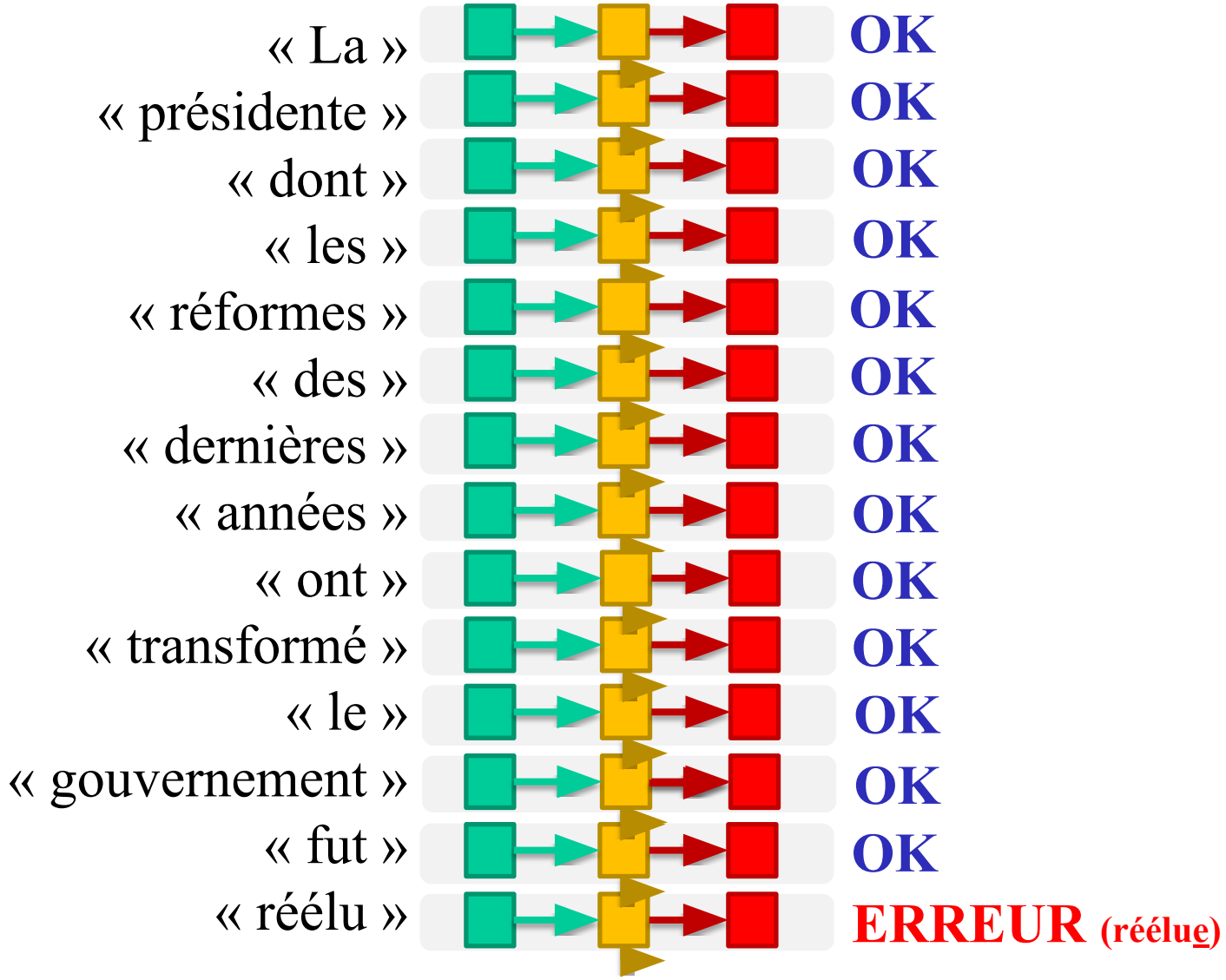
« présidente » **OK**

« fut » **OK**

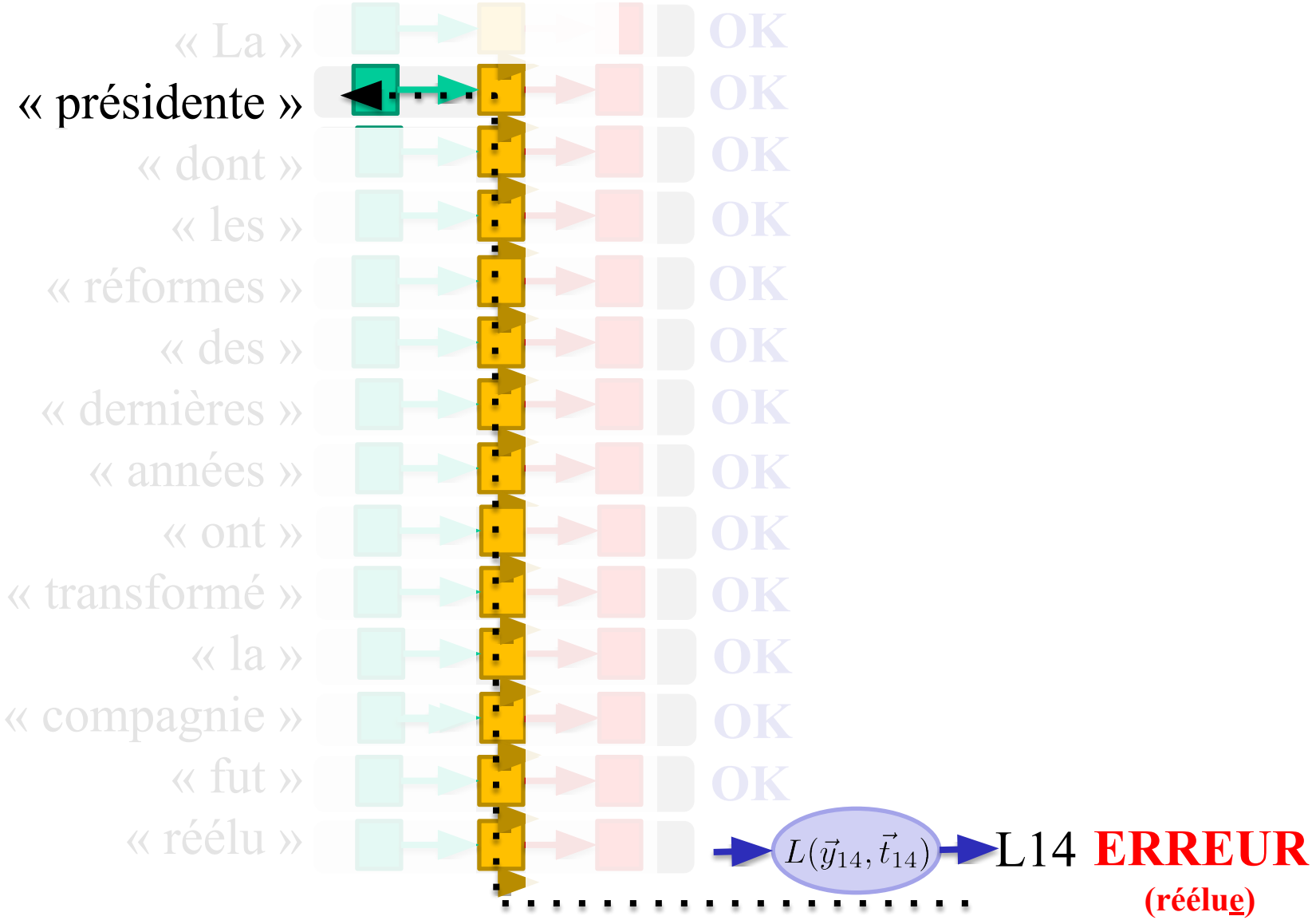
« réélu » **ERREUR (réélue)**



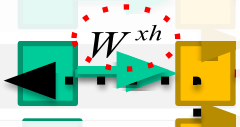
Exemple d'une relation à **courte distance**
(1 mot sépare « présidente » de « réélu »)



Exemple d'une relation à **longue distance**
(12 mots séparent « présidente » de « réélu »)


« La »
« présidente »
« dont »
« les »
« réformes »
« des »
« dernières »
« années »
« ont »
« transformé »
« la »
« compagnie »
« fut »
« réélu »
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
$L(\vec{y}_{14}, \vec{t}_{14})$
L14 ERREUR
(réélue)

« La »
« présidente »
« dont »
« les »
« réformes »
« dernières »
« années »
« transformé »
« la »
« compagnie »
« fut »
« réélu »



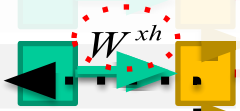
OK
OK
OK
OK
OK
OK
OK
OK
OK

$$\nabla_{W^{xh}} L_{14} = \nabla_{\vec{h}^{14}} L_{14} \nabla_{\vec{h}_{13}} \vec{h}_{14} \nabla_{\vec{h}_{12}} \vec{h}_{13} \nabla_{\vec{h}_{11}} \vec{h}_{12} \nabla_{\vec{h}_{10}} \vec{h}_{11} \nabla_{\vec{h}_{9}} \vec{h}_{10} \nabla_{\vec{h}_{8}} \vec{h}_{9} \nabla_{\vec{h}_{7}} \vec{h}_{8} \nabla_{\vec{h}_{6}} \vec{h}_{7} \nabla_{\vec{h}_{5}} \vec{h}_{6} \nabla_{\vec{h}_{4}} \vec{h}_{5} \nabla_{\vec{h}_{3}} \vec{h}_{4} \nabla_{\vec{h}_{2}} \vec{h}_{3} \nabla_{\vec{h}_{1}} \vec{h}_{2} \nabla_{W^{xh}} \vec{h}_{1}$$

$L(\vec{y}_{14}, \vec{t}_{14})$ → L14 **ERREUR**

(réélue)

« présidente »



$$\nabla_{W^{xh}} L_{14} = \nabla_{h^{14}} L_{14} \Big[\underbrace{\prod_{j=14}^{2} \nabla_{h_{j-1}} h_j}_{\text{Disparition du gradient}} \Big] \nabla_{W^{xh}} h_1$$

si

$$-1 < \nabla_{h_{j-1}} h_j < 1$$

$L(\vec{y}_{14}, \vec{t}_{14})$ → L14 **ERREUR**

(réélue)

« présidente »

$$\nabla_{W^{xh}} L_{14} = \nabla_{h^{14}} L_{14} \Big[\prod_{j=14}^{2} \nabla_{h_{j-1}} h_j \Big] \nabla_{W^{xh}} h_1$$

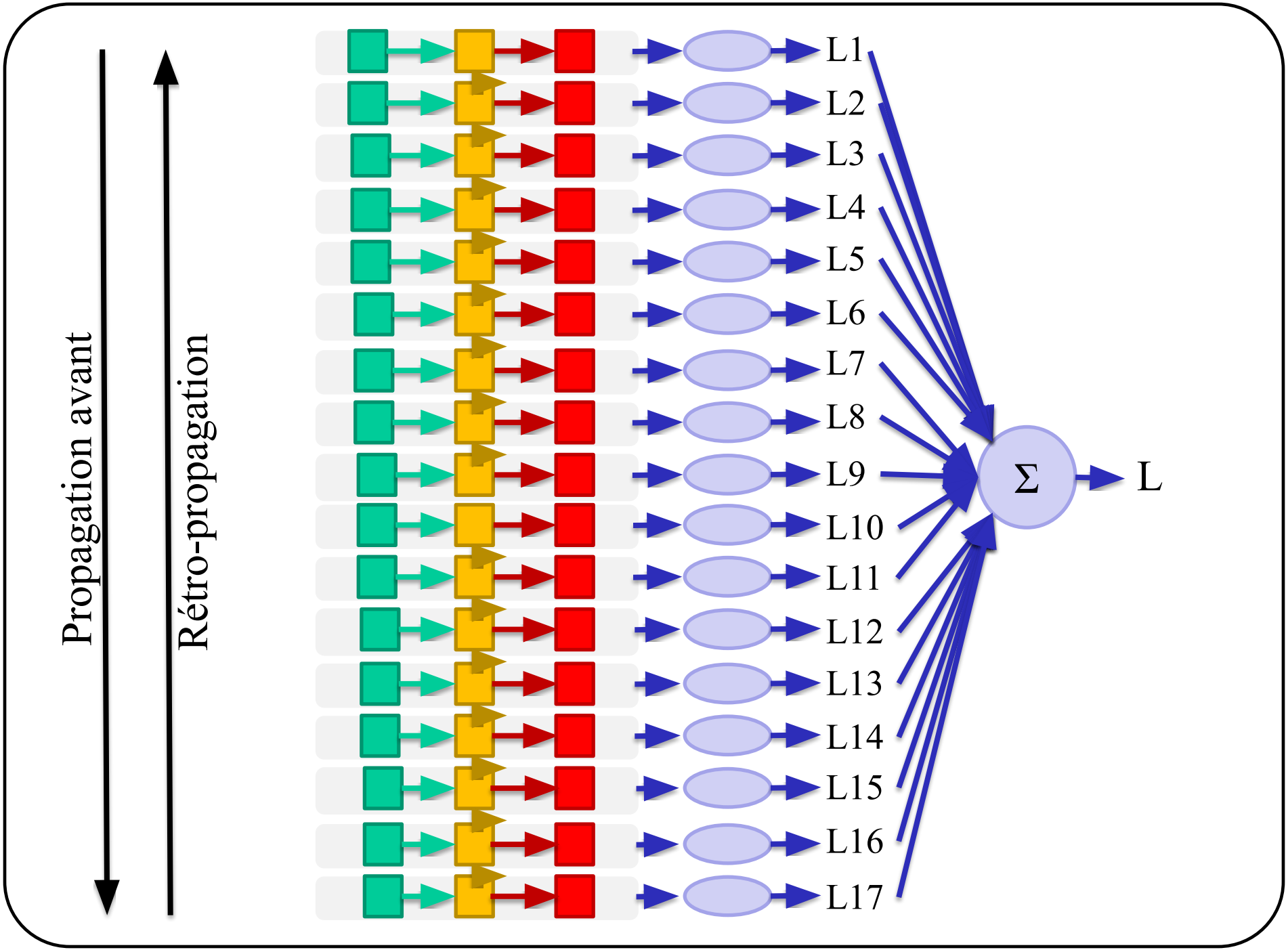
Explosion du gradient

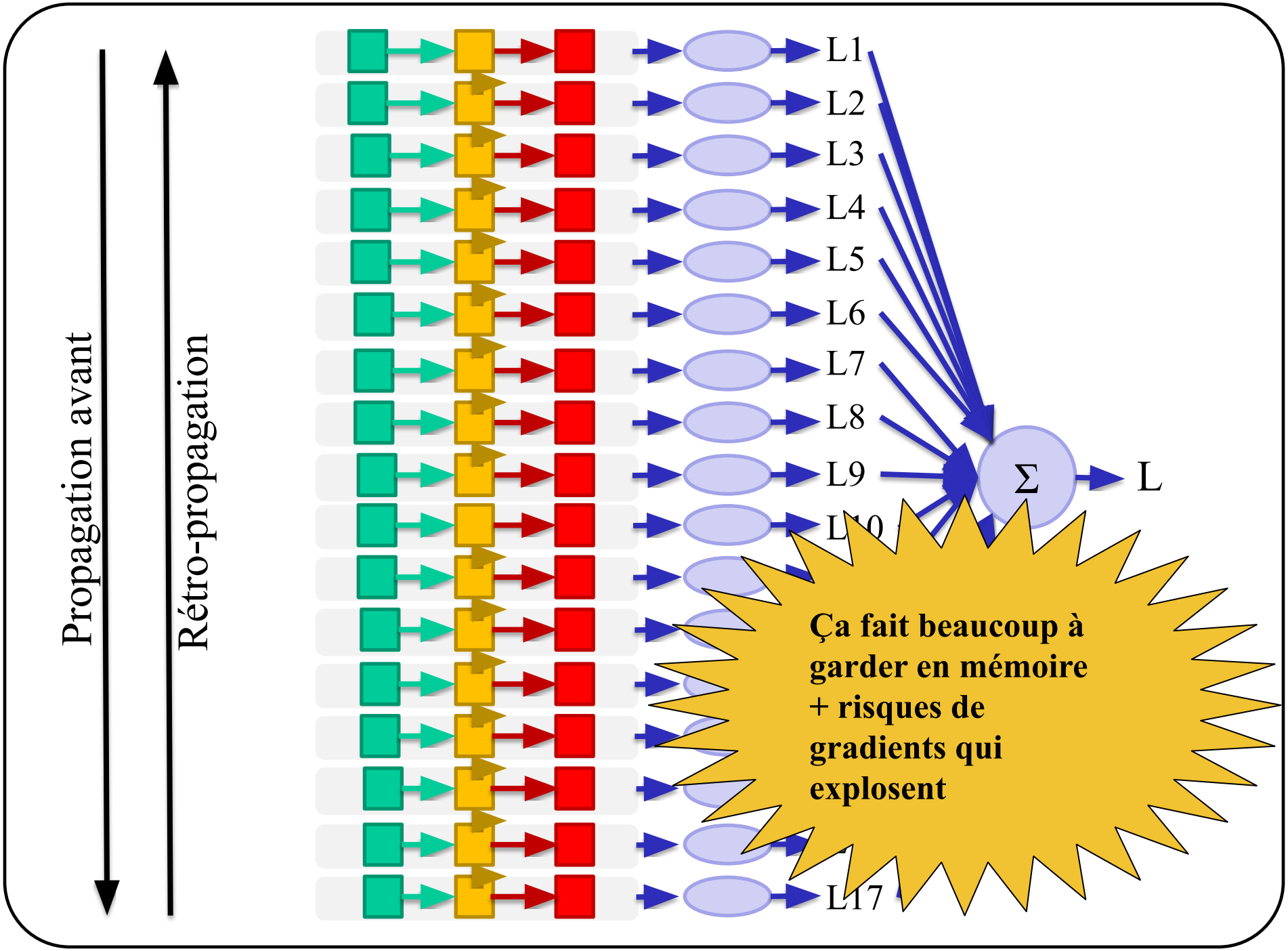
si

$$-1 > \nabla_{h_{j-1}} h_j > 1$$

$L(\vec{y}_{14}, \vec{t}_{14})$ → L14 **ERREUR**

(réélue)


Propagation avant
Rétro-propagation
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
L16
L17
Σ
L

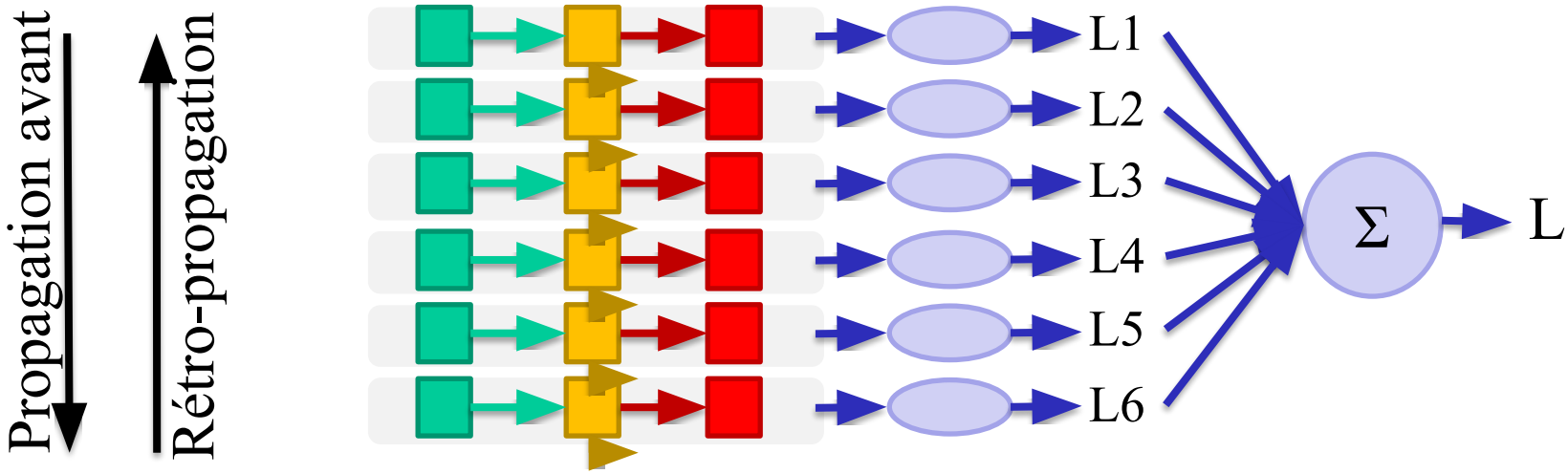

Propagation avant
Rétro-propagation
L1
L2
L3
L4
L5
L6
L7
L8
L9
L17
Σ
L
Ça fait beaucoup à garder en mémoire + risques de gradients qui explosent

Les réseaux récurrents ont un inconvénient majeur:

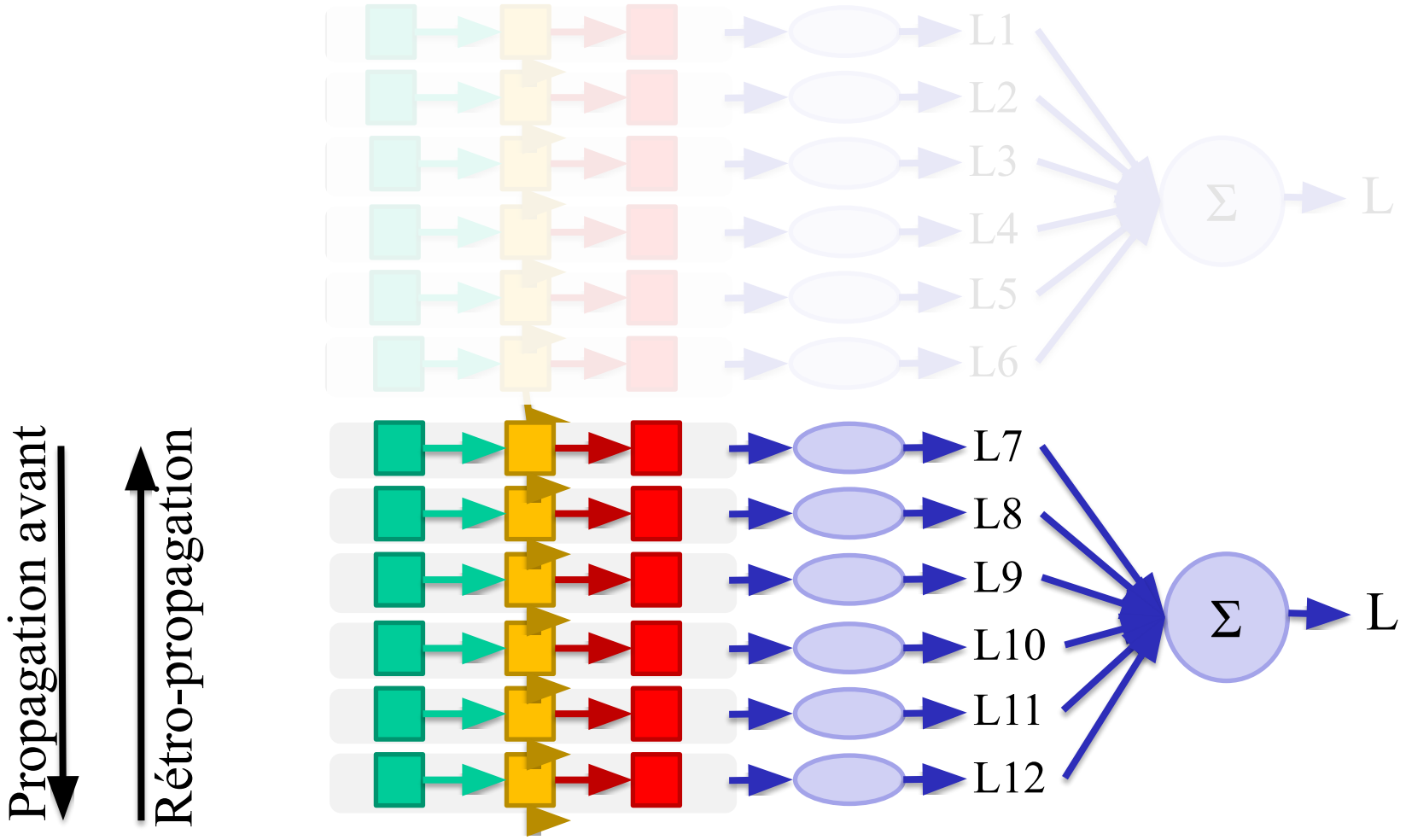
difficile à établir des
relations à longue distance

Solutions:

Rétropropagation tronquée
Gated recurrent unit : GRU
Long-Short Term Memory : LSTM



Lorsque les séquences sont trop longues
On entraîne le réseau par fenêtres coulissantes



on propage le contenu de la couche cachée
d'une section à l'autre.

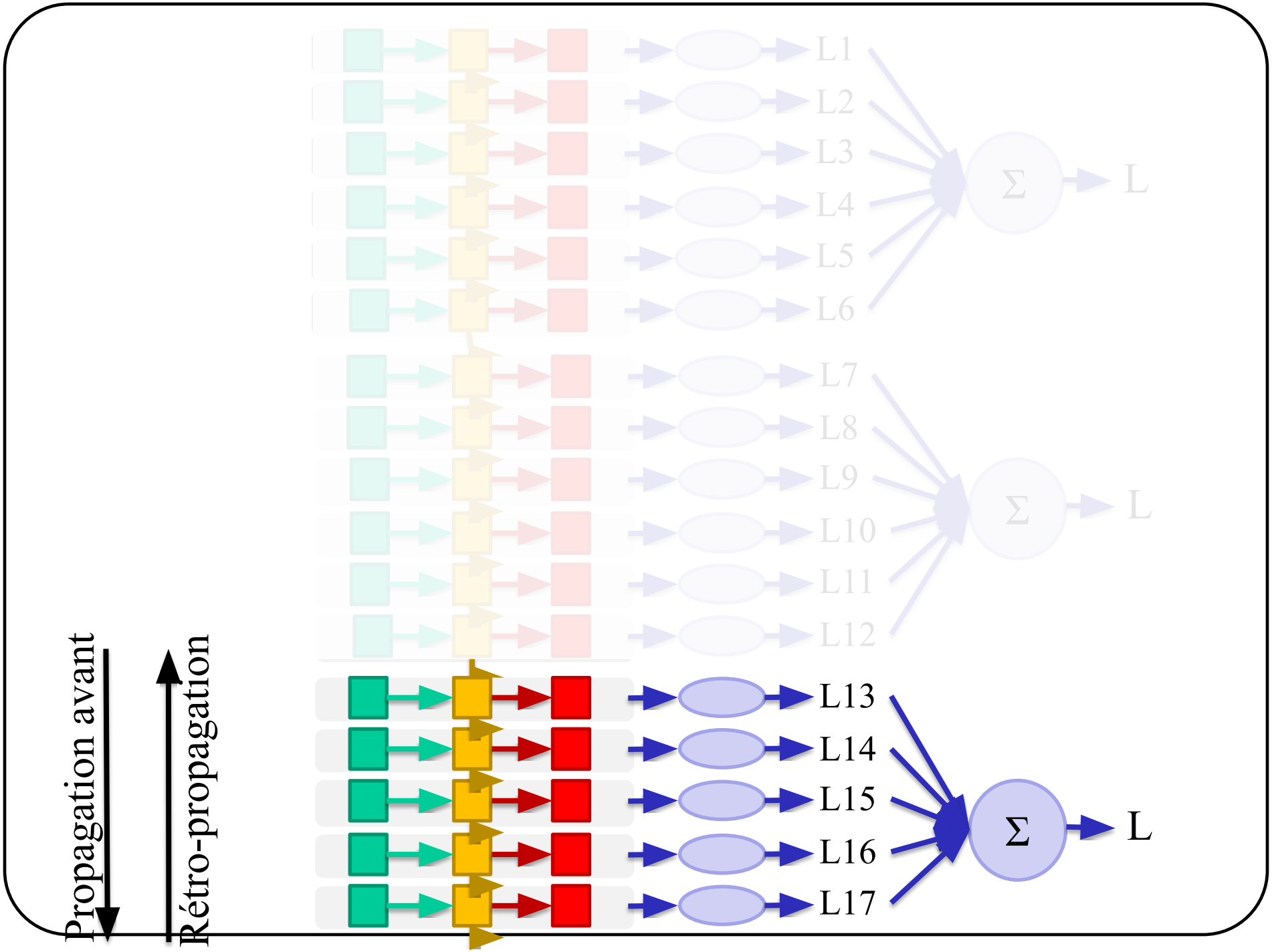
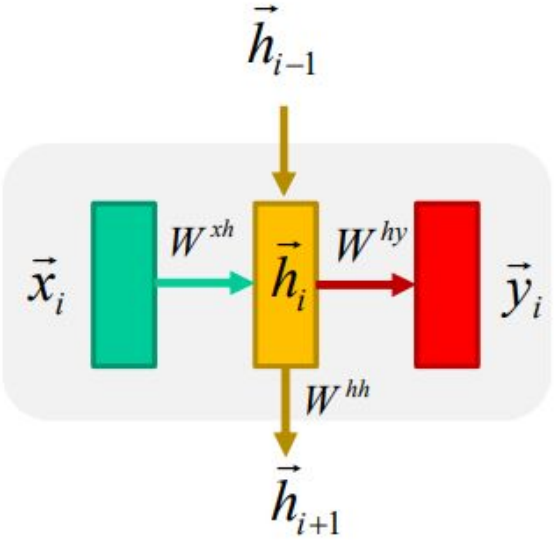

L1
L2
L3
L4
L5
L6
Σ
L
L7
L8
L9
L10
L11
L12
Σ
L
L13
L14
L15
L16
L17
Σ
L
Propagation avant
Rétro-propagation

Illustration + formulation d'un RNN

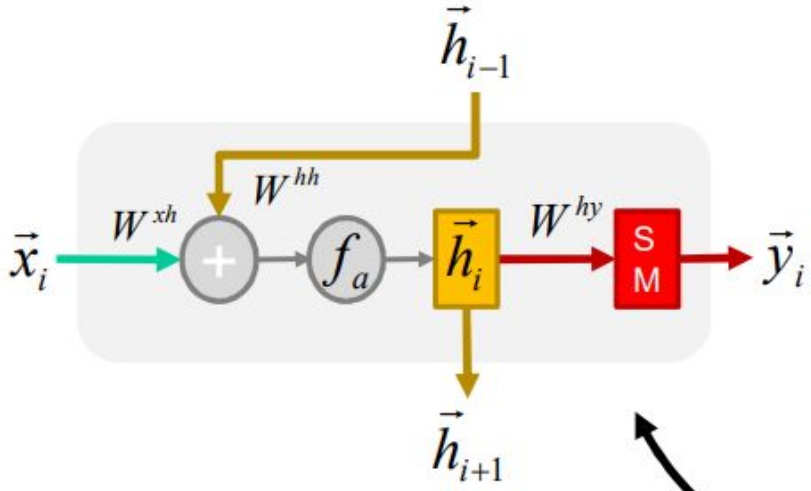


$$\vec{h}_i = f_a\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)$$

$$\hat{y}_i = W^{hy}\vec{h}_i$$

$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

Autre illustration du même RNN



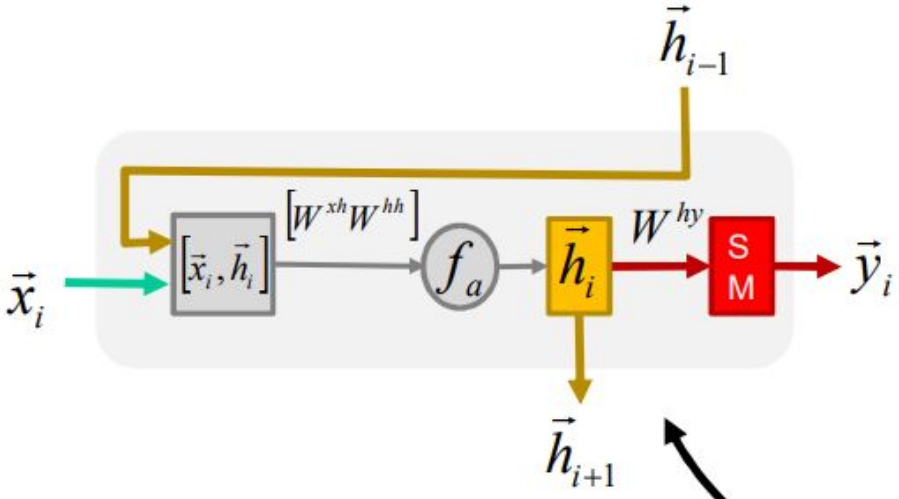
$$\vec{h}_i = f_a\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)$$

$$\hat{y}_i = W^{hy}\vec{h}_i$$

$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

Identique

Autre illustration du même RNN



$$\vec{h}_i = f_a\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)$$

$$\hat{y}_i = W^{hy}\vec{h}_i$$

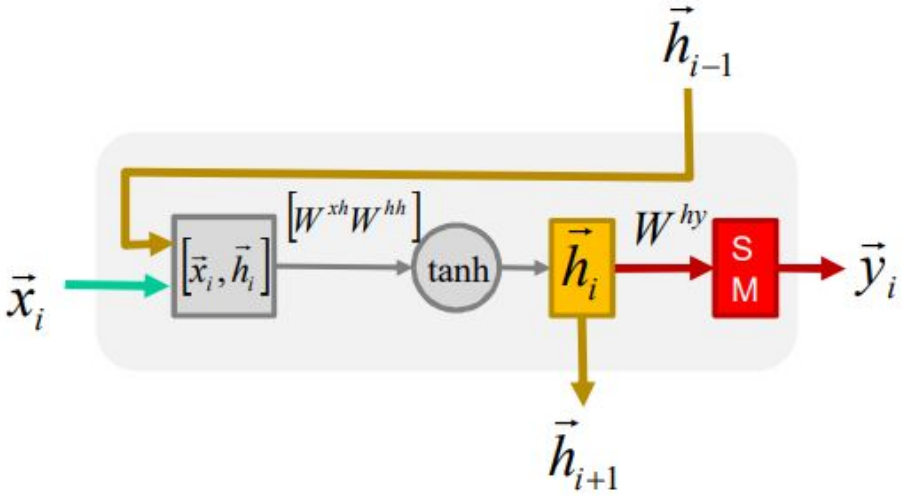
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

Encore identique

GRU (Gated Recurrent Unit)

Modif 1

$$f_a = \tanh$$



$$\vec{h}_i = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)$$

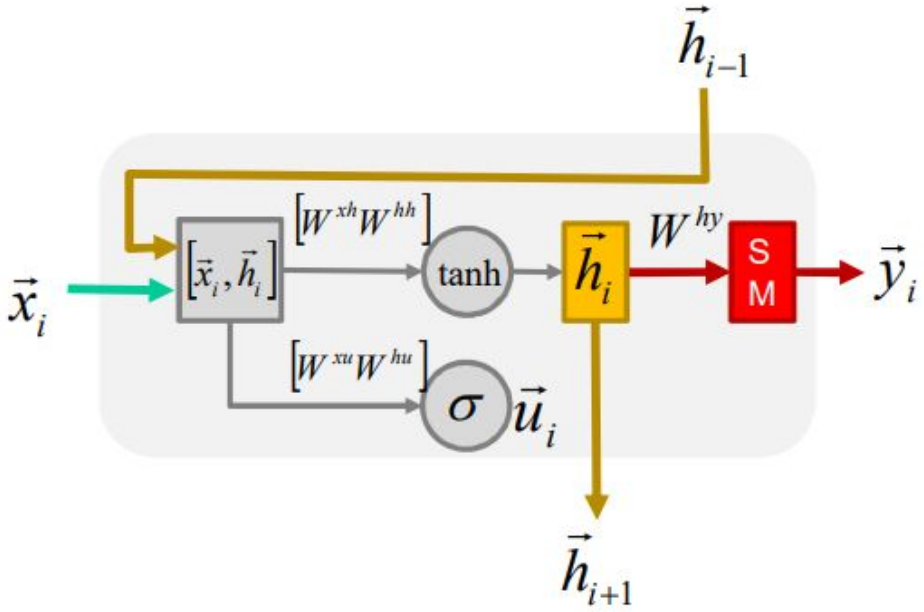
$$\hat{y}_i = W^{cy}\vec{h}_i$$

$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

GRU (Gated Recurrent Unit)

Modif 2

$$\sigma = \text{sigmoid}$$



$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$

$$\vec{h}_i = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)$$

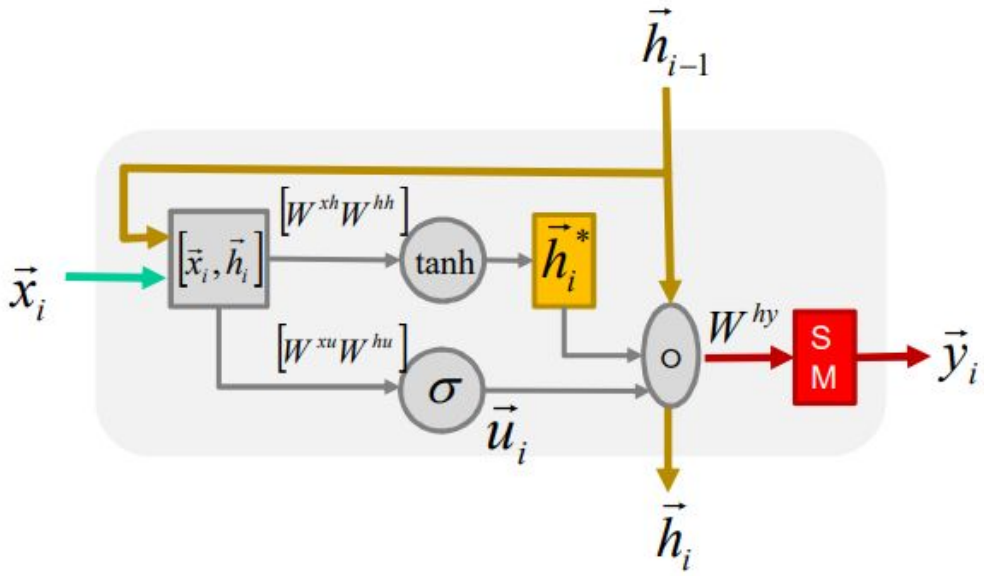
$$\hat{y}_i = W^{cy}\vec{h}_i$$

$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

GRU (Gated Recurrent Unit)

Modif 2

Update gate



$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$

$$\vec{h}_i^* = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)$$

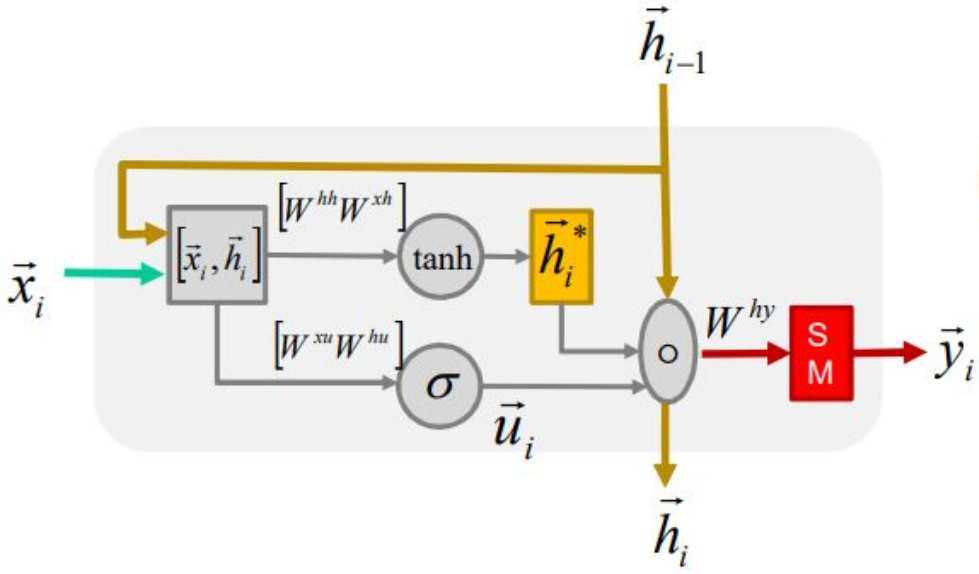
$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^* + \left(1 - \vec{u}_i\right) \circ \vec{h}_{i-1}$$

$$\hat{y}_i = W^{cy}\vec{h}_i$$

$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

GRU (Gated Recurrent Unit)

Modif 2



« element-wise product »

ou

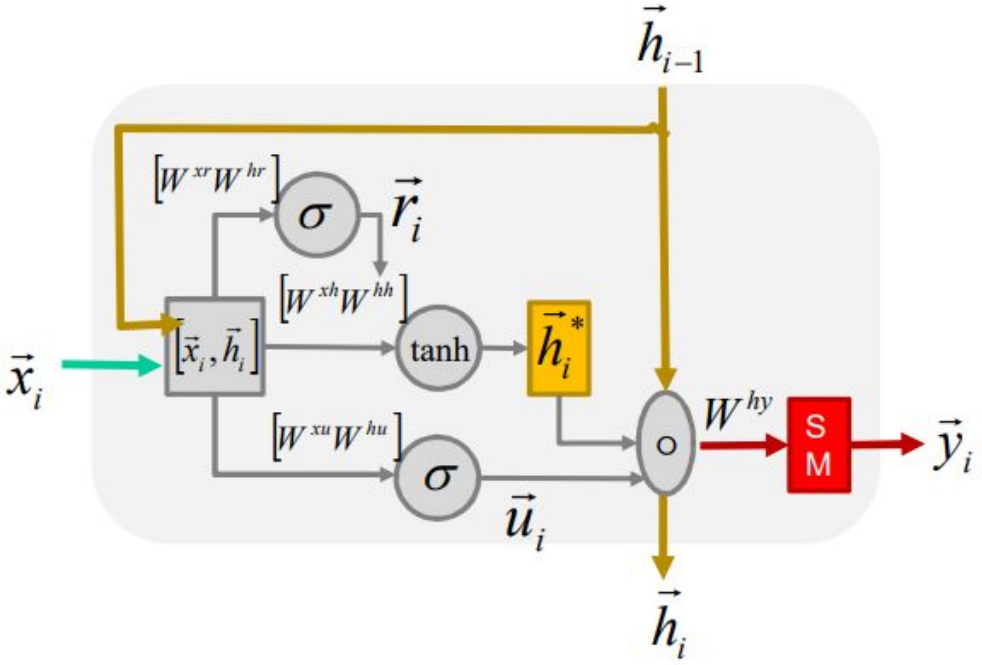
« Hadamard product »

$$(a,b,c)\circ(x,y,z)=(ax,by,cz)$$

GRU (Gated Recurrent Unit)

Modif 3

Reset gate



$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$

$$\vec{r}_i = \sigma\left(W^{xr}\vec{x}_i + W^{hr}\vec{h}_{i-1}\right)$$

$$\vec{h}_i^* = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\left(\vec{r}_i \circ \vec{h}_{i-1}\right)\right)$$

$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^* + \left(1 - \vec{u}_i\right) \circ \vec{h}_{i-1}$$

$$\hat{y}_i = W^{cy}\vec{h}_i$$

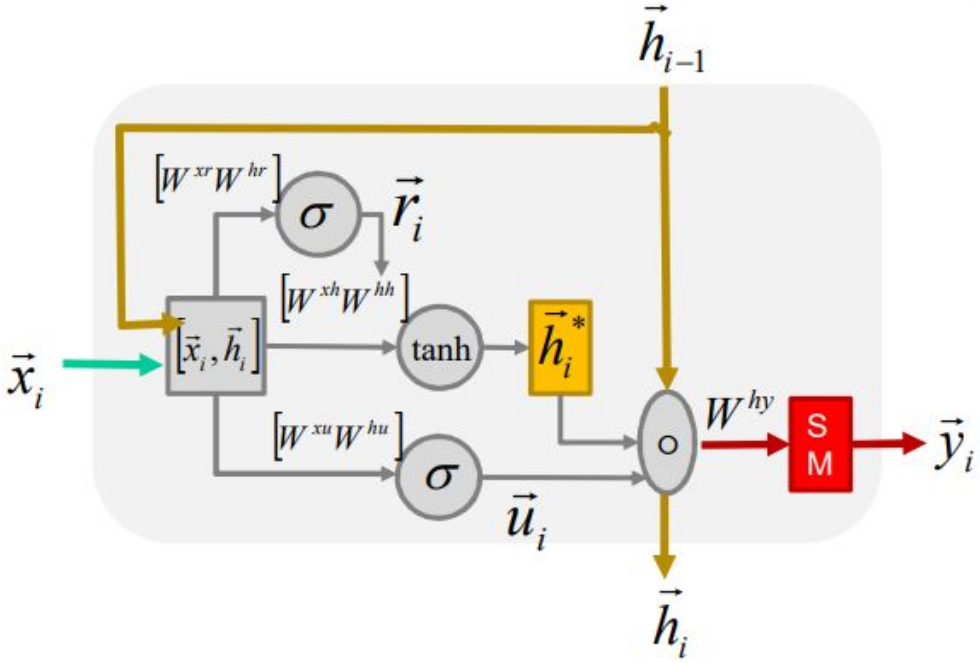
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

Comprendre les gates

$$\textit{SI} \begin{cases} u_i = 1 \\ r_i = 1 \end{cases}$$

$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$
$$\vec{r}_i = \sigma\left(W^{xr}\vec{x}_i + W^{hr}\vec{h}_{i-1}\right)$$
$$\vec{h}_i^{*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\left(\vec{r}_i \circ \vec{h}_{i-1}\right)\right)$$
$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^{*} + \left(1 - \vec{u}_i\right) \circ \vec{h}_{i-1}$$
$$\hat{y}_i = W^{cy}\vec{h}_i$$
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$



Comprendre les gates

$$\textbf{SI} \begin{cases} u_i = 1 \\ r_i = 1 \end{cases}$$

$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$

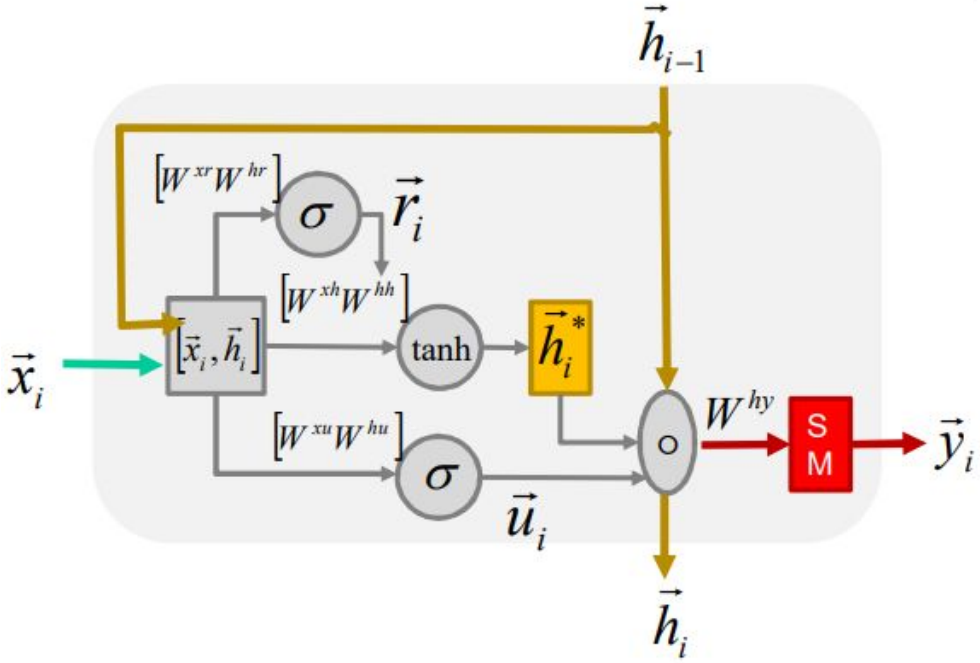
$$\vec{r}_i = \sigma\left(W^{xr}\vec{x}_i + W^{hr}\vec{h}_{i-1}\right)$$

$$\vec{h}_i^{*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\left(\vec{r}_i \circ \vec{h}_{i-1}\right)\right)$$

$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^{*} + (1 - \vec{u}_i) \circ \vec{h}_{i-1}$$

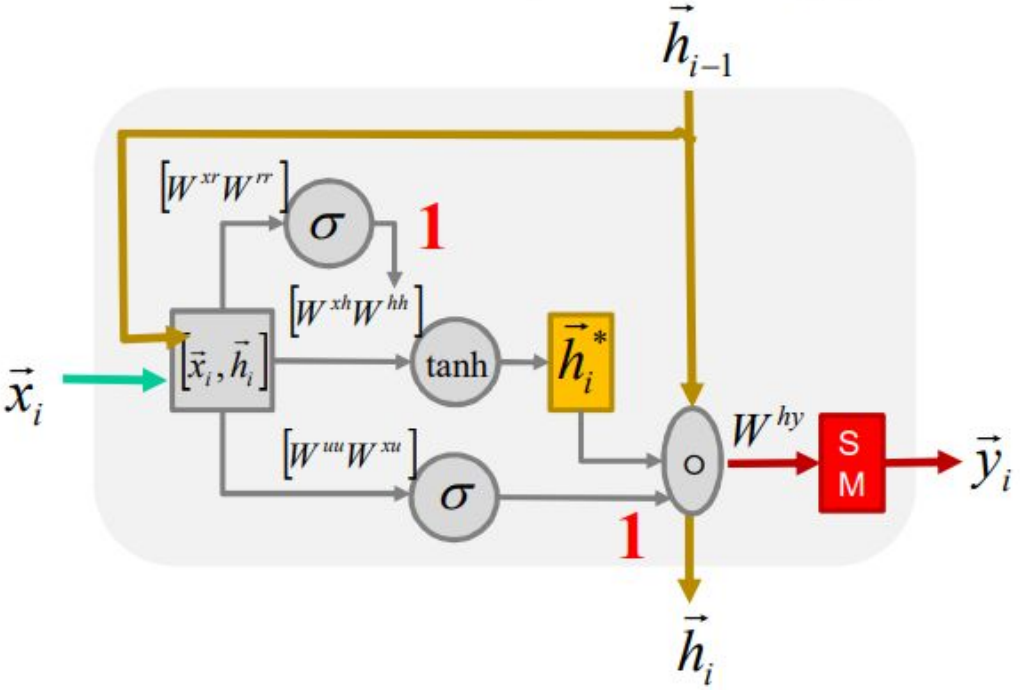
$$\hat{y}_i = W^{cy}\vec{h}_i$$

$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$



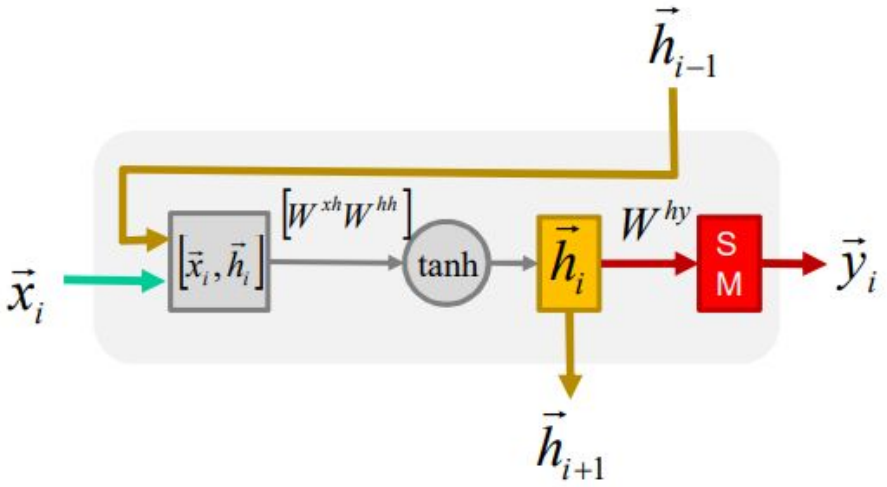
Comprendre les gates

$$\textbf{SI}\quad \left.\begin{array}{l}\vec{u}_i = 1\\ \vec{r}_i = 1\end{array}\right\}\left\{\begin{array}{l}\vec{h}_i^{\,*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)\\ \vec{h}_i = \vec{h}_i^{\,*}\\ \hat{y}_i = W^{cy}\vec{h}_i\\ \vec{y}_i = SMAX\left(\hat{y}_i\right)\end{array}\right.$$



Comprendre les gates

$$\textbf{SI}\quad \begin{matrix}\vec{u}_i = 1\\ \vec{r}_i = 1\end{matrix}\Bigg\{\Bigg\{ \begin{matrix}\vec{h}_i^{\,*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1}\right)\\ \vec{h}_i = \vec{h}_i^{\,*}\\ \hat{y}_i = W^{cy}\vec{h}_i\\ \vec{y}_i = SMAX\left(\hat{y}_i\right)\end{matrix}$$

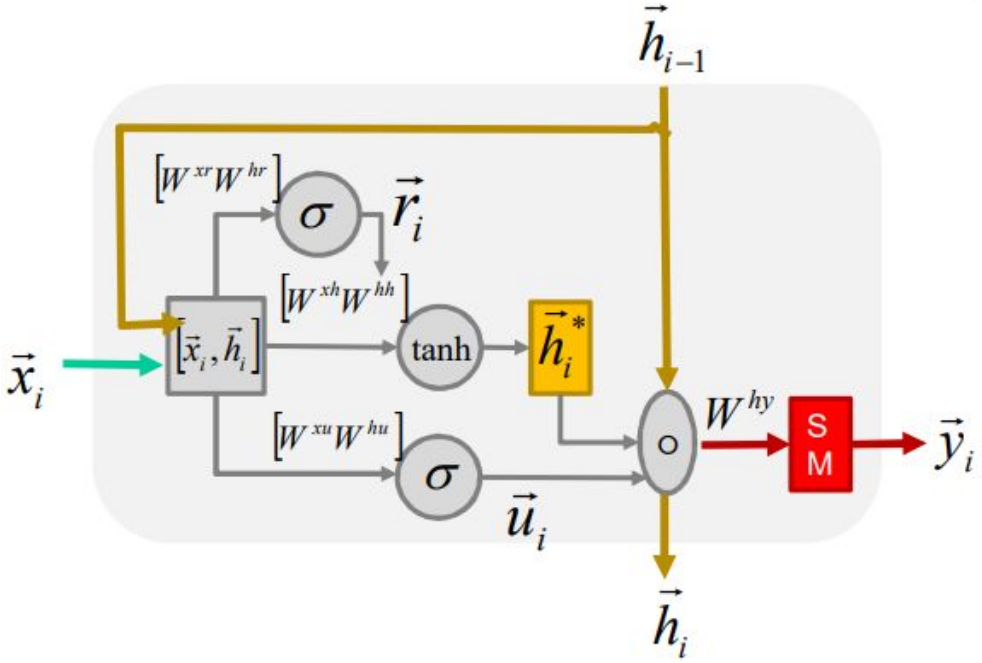


RNN de base!

Comprendre les gates

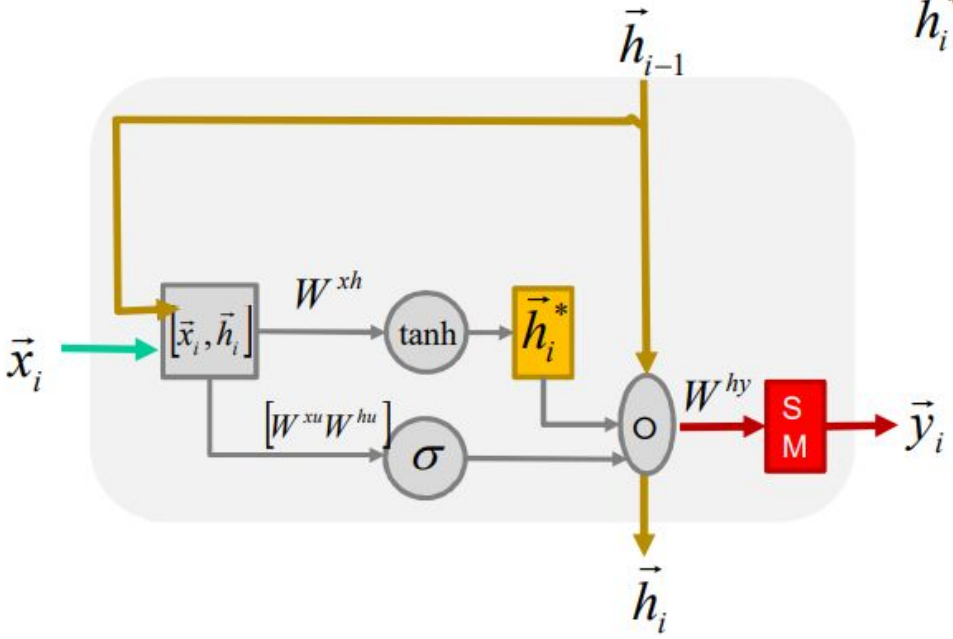
$$\textbf{SI} \quad \begin{matrix} u_i & \neq 1 \\ r_i & = 0 \end{matrix} \Big\} \Big\{$$

$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$
$$\vec{r}_i = \sigma\left(W^{xr}\vec{x}_i + W^{hr}\vec{h}_{i-1}\right)$$
$$\vec{h}_i^{*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\left(\vec{r}_i \circ \vec{h}_{i-1}\right)\right)$$
$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^{*} + \left(1 - \vec{u}_i\right) \circ \vec{h}_{i-1}$$
$$\hat{y}_i = W^{cy}\vec{h}_i$$
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$



SI $\vec{u}_i \neq 1$, $\vec{r}_i = 0$

$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$
$$\vec{h}_i^{*} = \tanh\left(W^{xh}\vec{x}_i\right)$$
$$\vec{h}_i = \vec{u}_i \otimes \vec{h}_i^{*} + \left(1-\vec{u}_i\right)\otimes \vec{h}_{i-1}$$
$$\hat{y}_i = W^{cy}\vec{h}_i$$
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

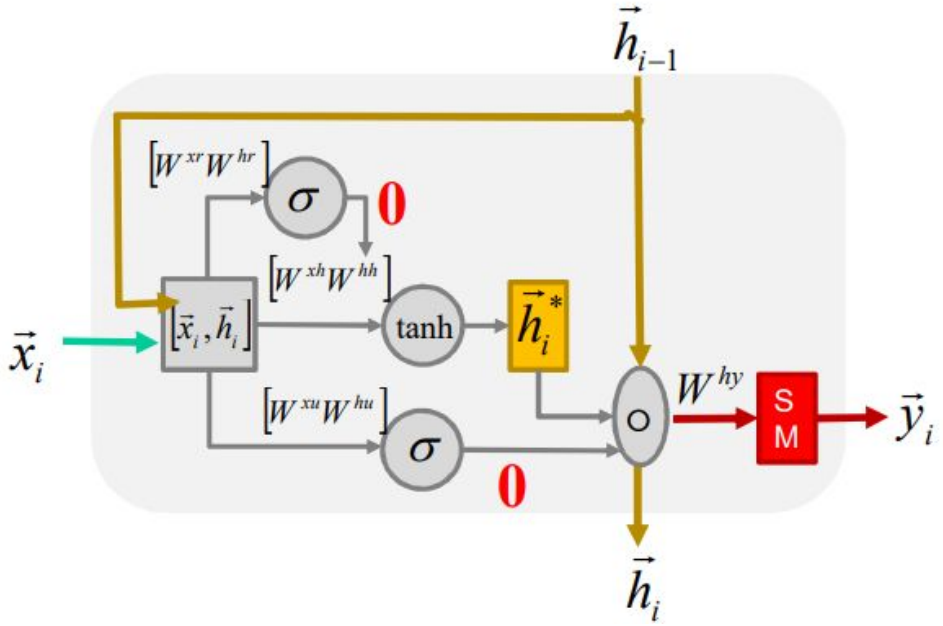


$\vec{h}_i^{*}$ ne dépend que de $\vec{x}_i$ et non de $\vec{h}_{i-1}$

SI $\vec{u}_i = 0$, $\vec{r}_i = 0$

$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$
$$\vec{r}_i = \sigma\left(W^{xr}\vec{x}_i + W^{hr}\vec{h}_{i-1}\right)$$
$$\vec{h}_i^{*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\left(\vec{r}_i \circ \vec{h}_{i-1}\right)\right)$$
$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^{*} + \left(1 - \vec{u}_i\right) \circ \vec{h}_{i-1}$$
$$\hat{y}_i = W^{cy}\vec{h}_i$$
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

1

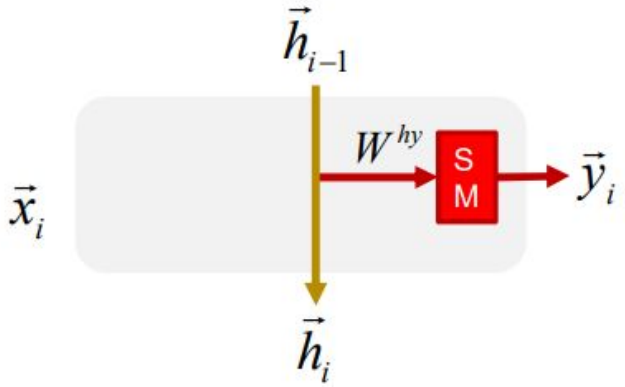


$$\textbf{SI} \quad \left.\begin{array}{l} \vec{u}_i = 0 \\ \vec{r}_i = 0 \end{array}\right\} \left\{\begin{array}{l} \vec{h}_i = \vec{h}_{i-1} \\ \hat{y}_i = W^{cy}\vec{h}_i \\ \vec{y}_i = SMAX(\hat{y}_i) \end{array}\right.$$

$\vec{h}_{i-1}$ **est recopié**

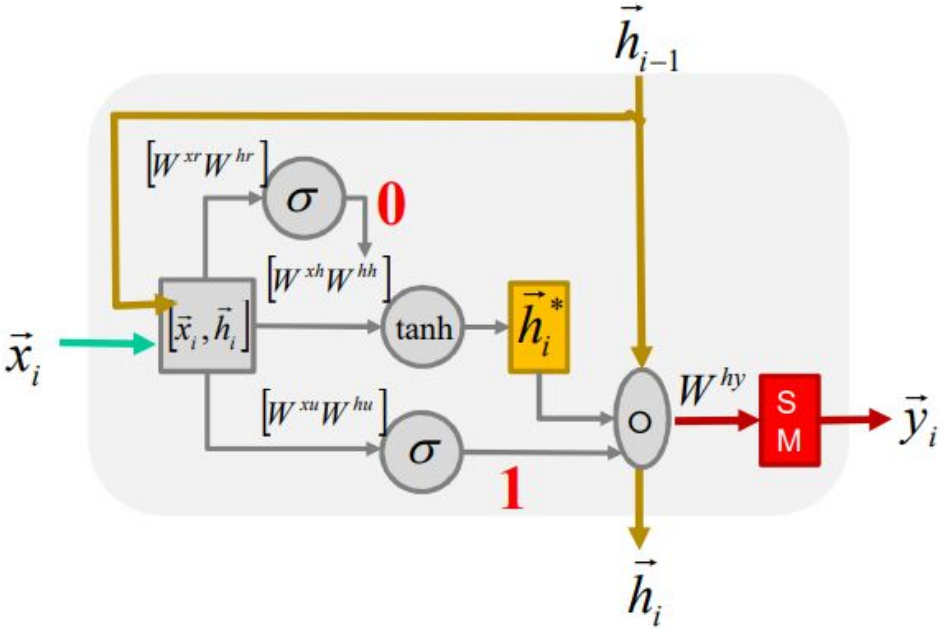
$\vec{x}_i$ **est ignoré**

Aucune disparition de gradient



SI $\vec{u}_i = 1$, $\vec{r}_i = 0$

$$\vec{u}_i = \sigma\left(W^{xu}\vec{x}_i + W^{hu}\vec{h}_{i-1}\right)$$
$$\vec{r}_i = \sigma\left(W^{xr}\vec{x}_i + W^{hr}\vec{h}_{i-1}\right)$$
$$\vec{h}_i^{*} = \tanh\left(W^{xh}\vec{x}_i + W^{hh}\left(\vec{r}_i \circ \vec{h}_{i-1}\right)\right)$$
$$\vec{h}_i = \vec{u}_i \circ \vec{h}_i^{*} + \left(1 - \vec{u}_i\right) \circ \vec{h}_{i-1}$$
$$\hat{y}_i = W^{cy}\vec{h}_i$$
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$



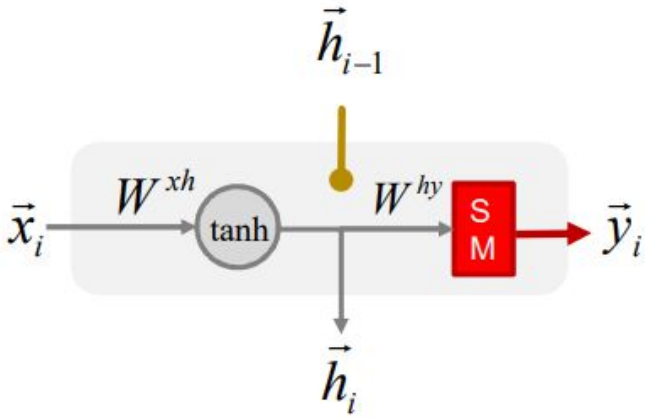
SI $\vec{u}_i = 1$, $\vec{r}_i = 0$

$$\vec{h}_i = \tanh\left(W^{xh}\vec{x}_i\right)$$
$$\hat{y}_i = W^{cy}\vec{h}_i$$
$$\vec{y}_i = SMAX\left(\hat{y}_i\right)$$

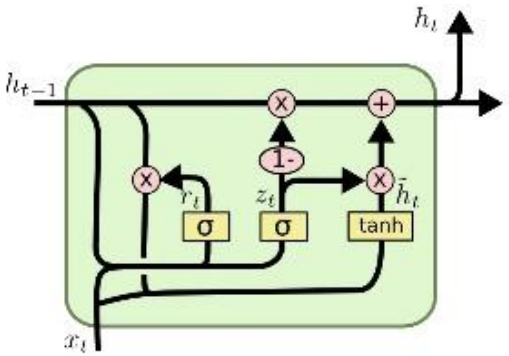
$\vec{h}_{i-1}$ **est ignoté**

$\vec{x}_i$ **est le seul utilisé**

Gradient temporel bloqué



Gated Recurrent Unit (GRU)



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

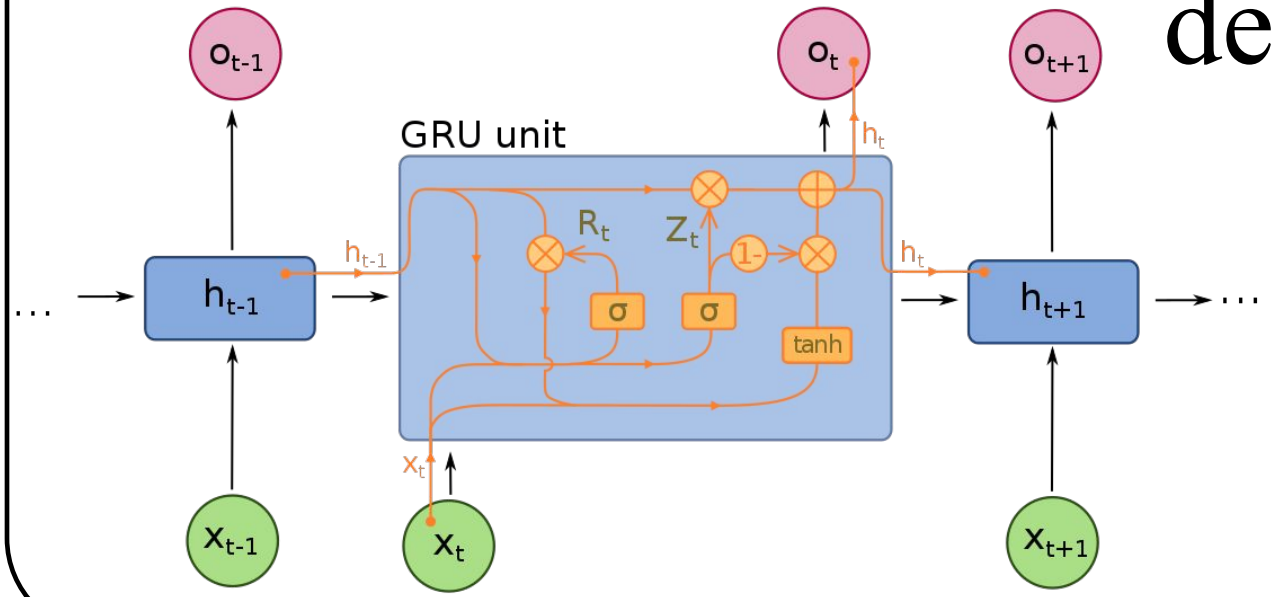
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

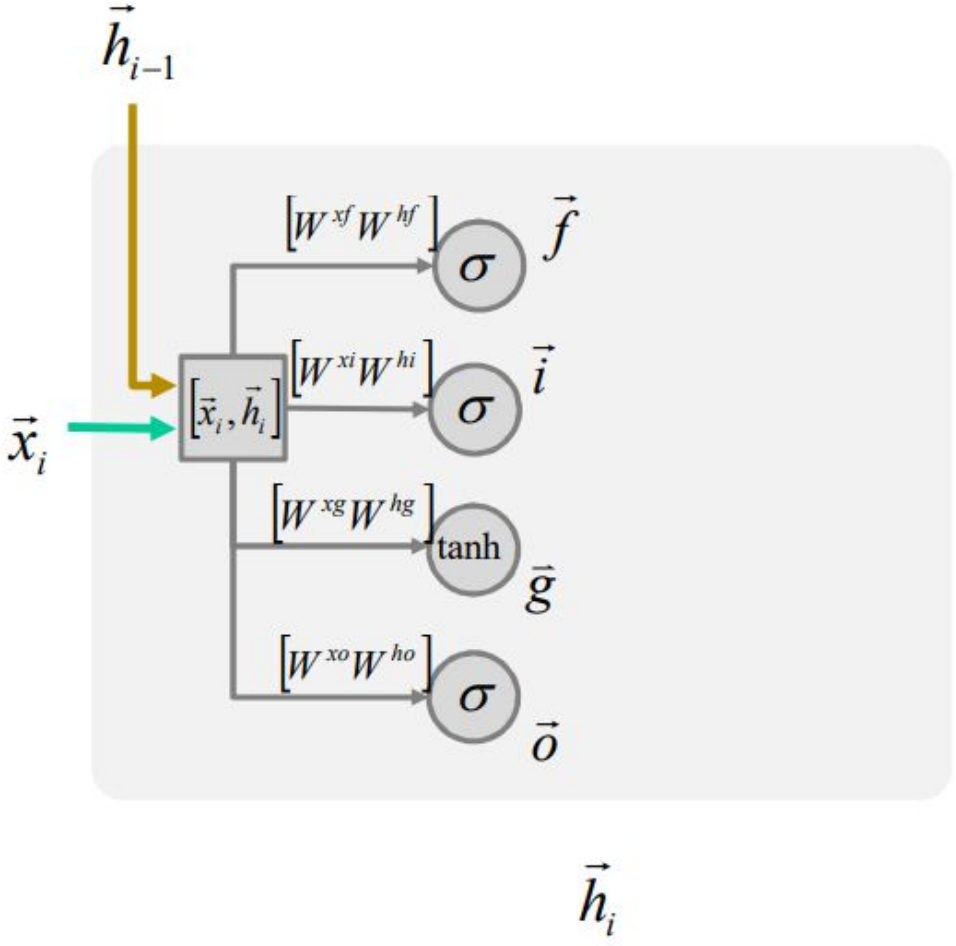
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure: Cristopher Olah, "Understanding LSTM Networks" (2015) / Slide: Alberto Montes

Le web regorge d'illustrations de GRU!



LSTM (Long Short Term Memory)



$\vec{f}$: forget gate

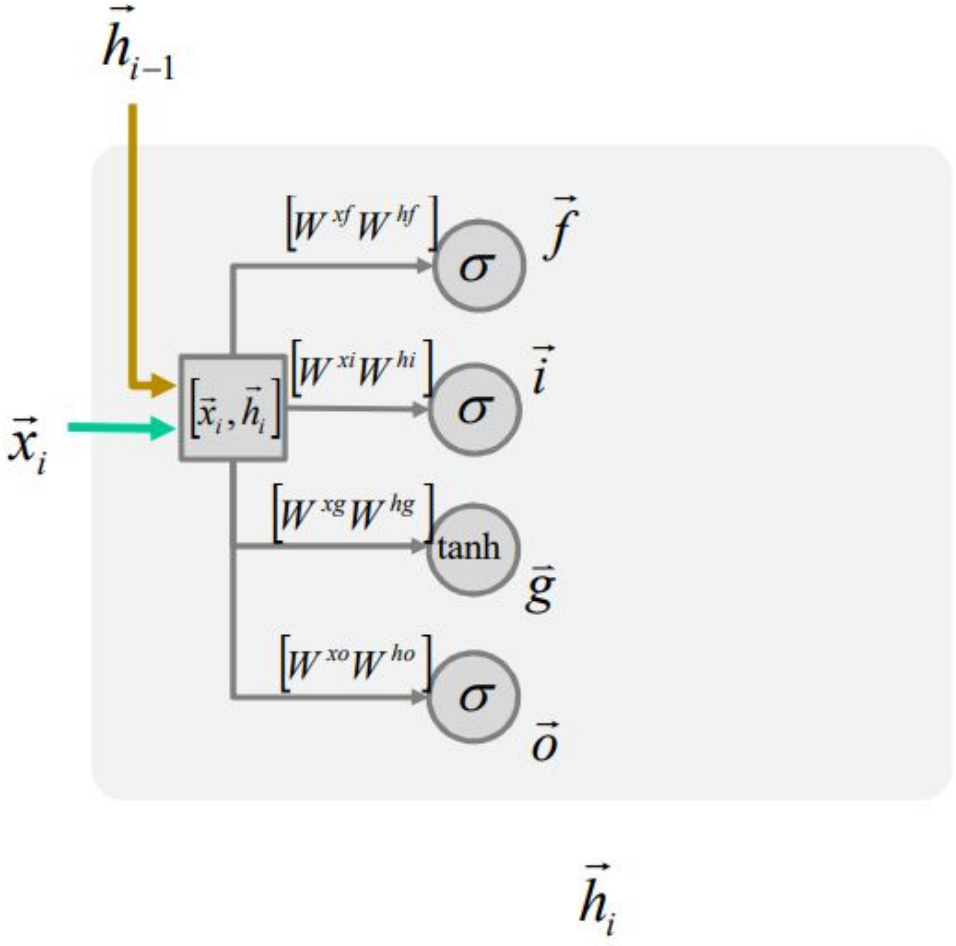
$\vec{i}$: input gate

$\vec{g}$: gate gate

$\vec{o}$: output gate

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation,1997

LSTM (Long Short Term Memory)



$\vec{f}$: forget gate

$\vec{i}$: input gate

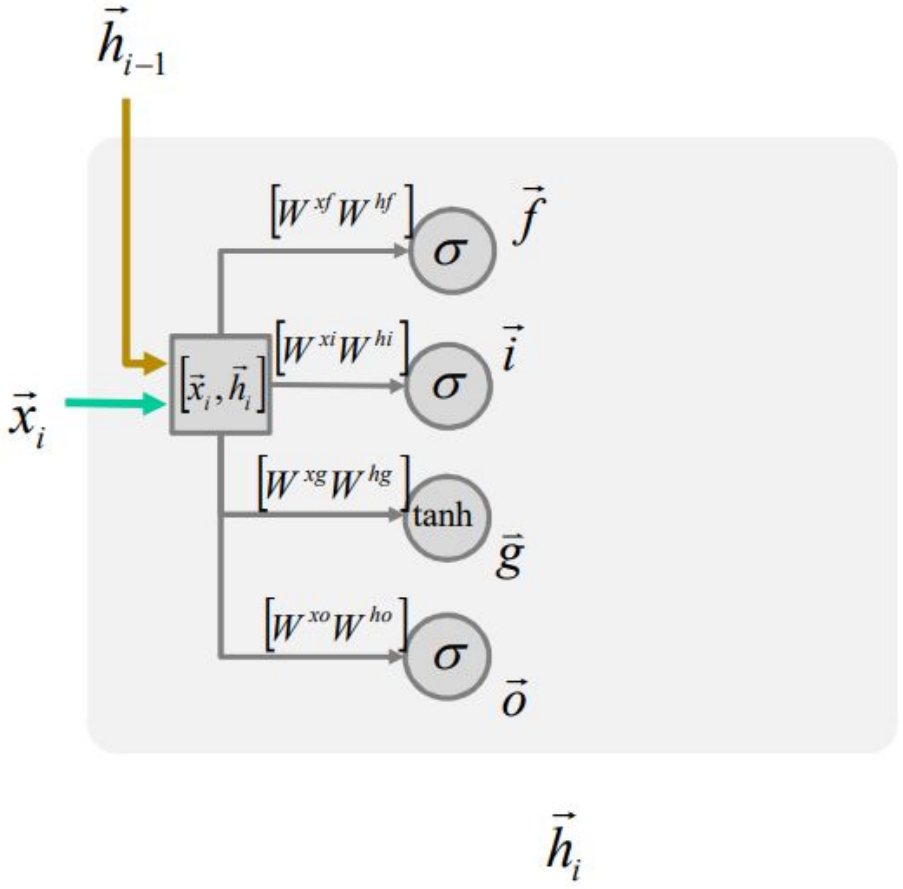
$\vec{g}$: gate gate

t gate

Modèle récurrent le plus utilisé. À bien comprendre !

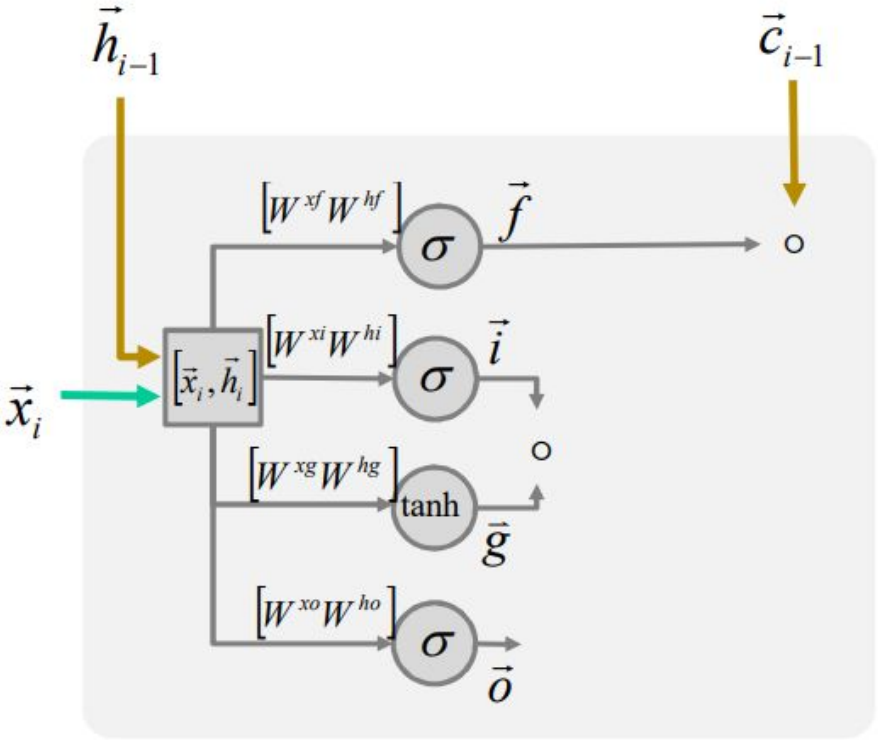
Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation,1997

LSTM (Long Short Term Memory)



$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$

LSTM (Long Short Term Memory)

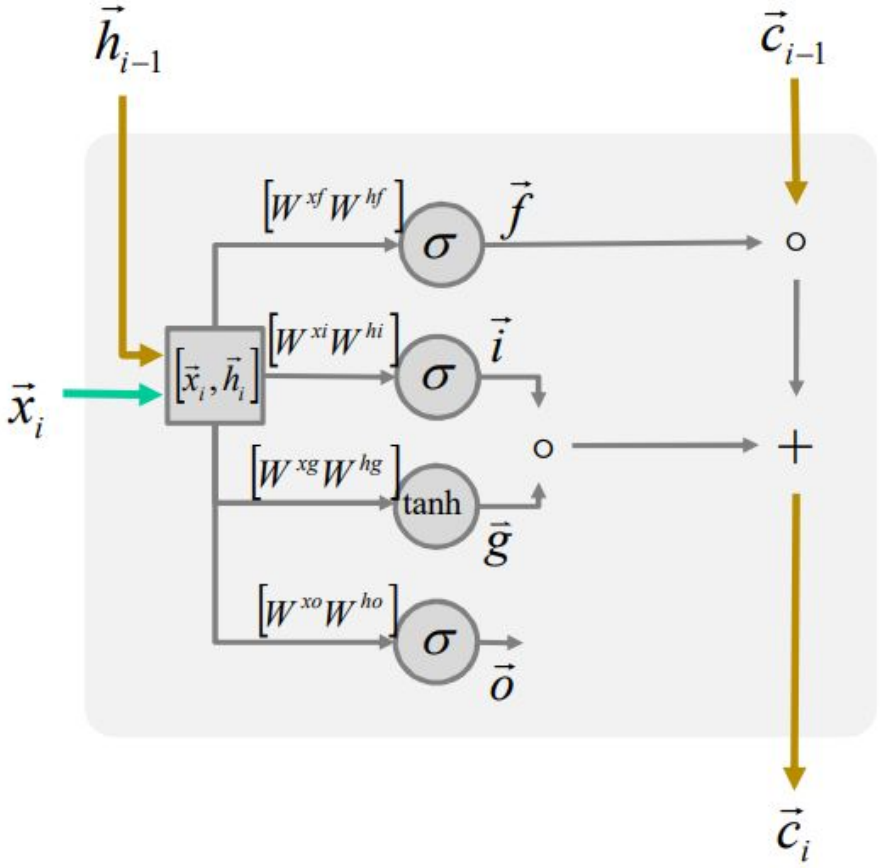


$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$

$$\vec{f} \circ \vec{c}_i$$
$$\vec{i} \circ \vec{g}$$

LSTM (Long Short Term Memory)

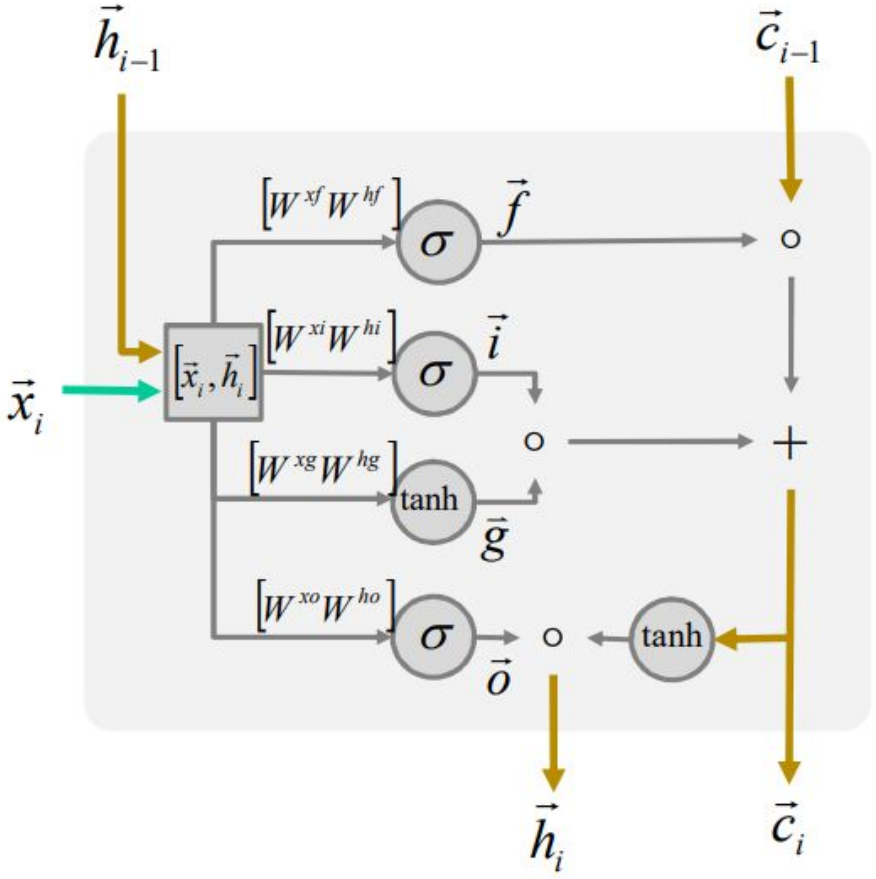
$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$



$$\vec{c}_i = \vec{f} \circ \vec{c}_i + \vec{i} \circ \vec{g}$$

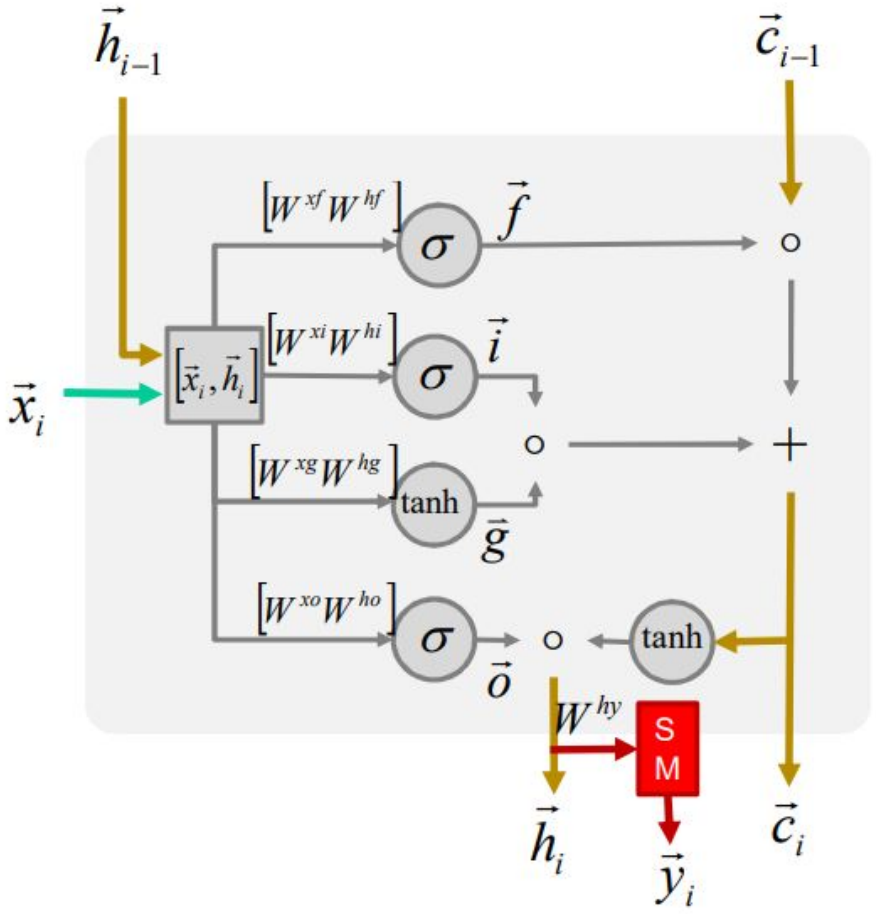
LSTM (Long Short Term Memory)

$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$



$$\vec{c}_i = \vec{f} \circ \vec{c}_i + \vec{i} \circ \vec{g}$$
$$\vec{h}_i = \vec{o} \circ \tanh\left(\vec{c}_i\right)$$

LSTM (Long Short Term Memory)

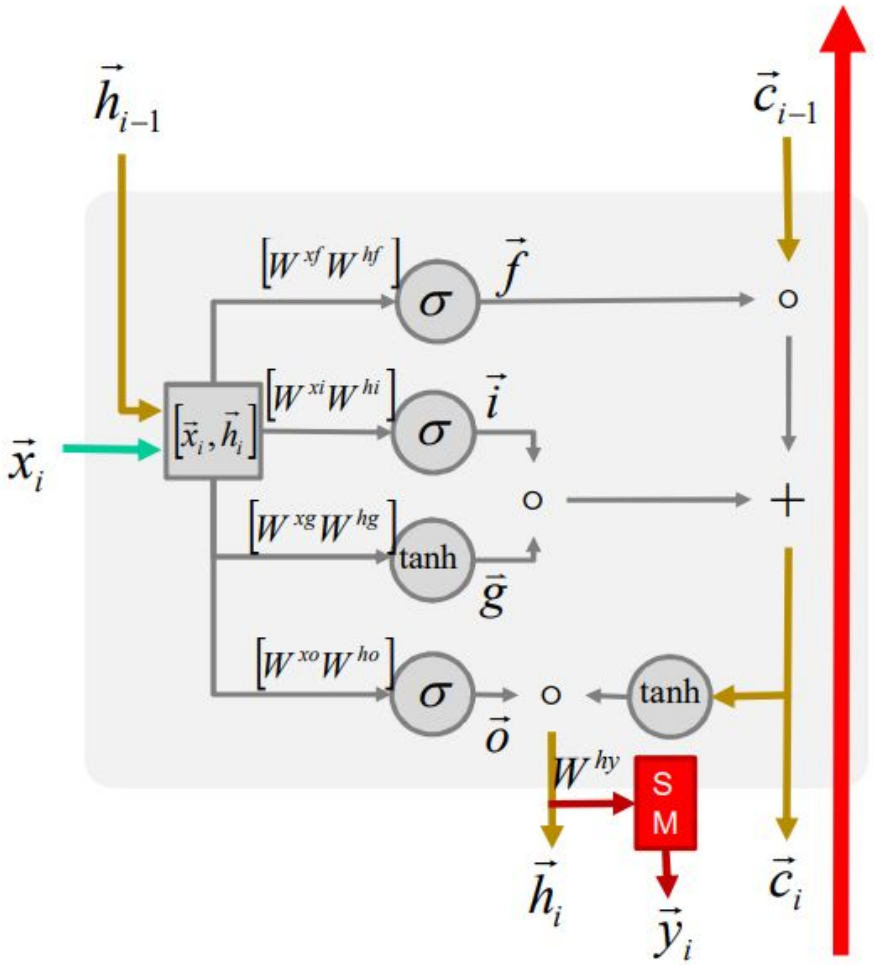


$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$

$$\vec{c}_i = \vec{f} \circ \vec{c}_i + \vec{i} \circ \vec{g}$$
$$\vec{h}_i = \vec{o} \circ \tanh\left(\vec{c}_i\right)$$
$$\vec{y}_i = SMAX\left(W^{hy}\vec{h}_i\right)$$

LSTM (Long Short Term Memory)

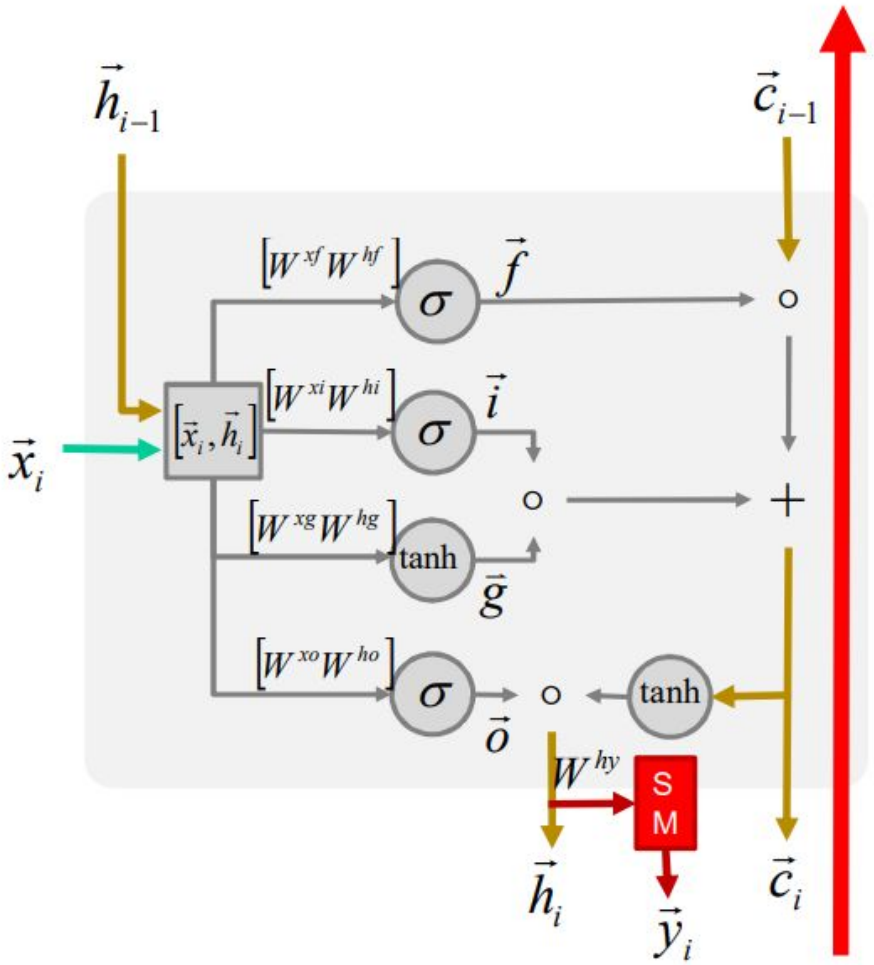
$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$



Gradient simplifié

LSTM (Long Short Term Memory)

$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$

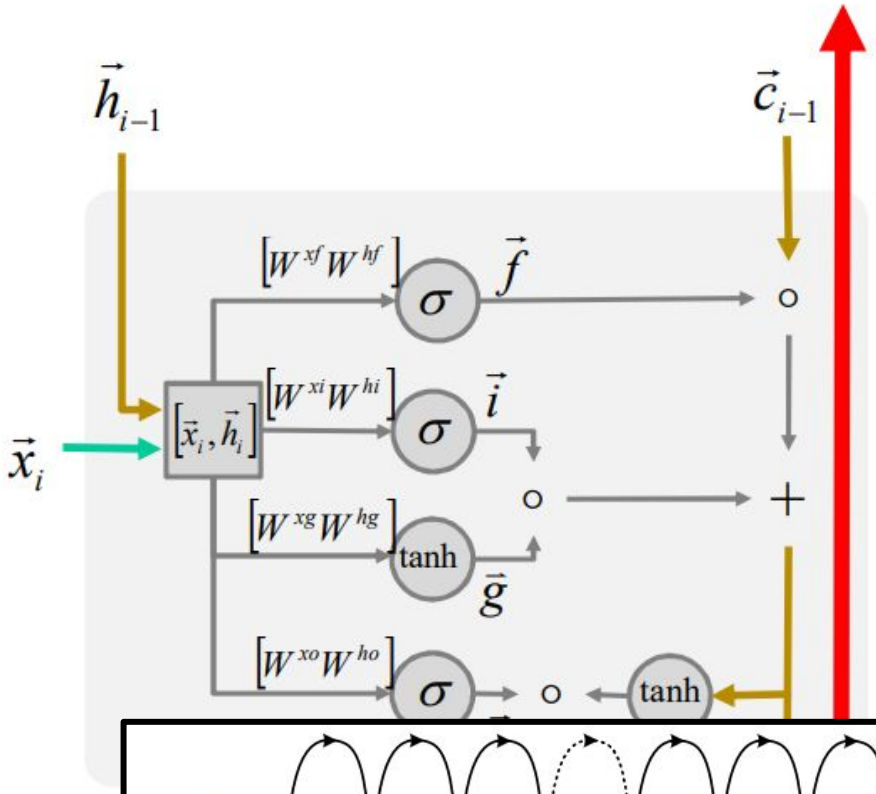


Gradient simplifié

Ça vous rappelle quelque chose ?

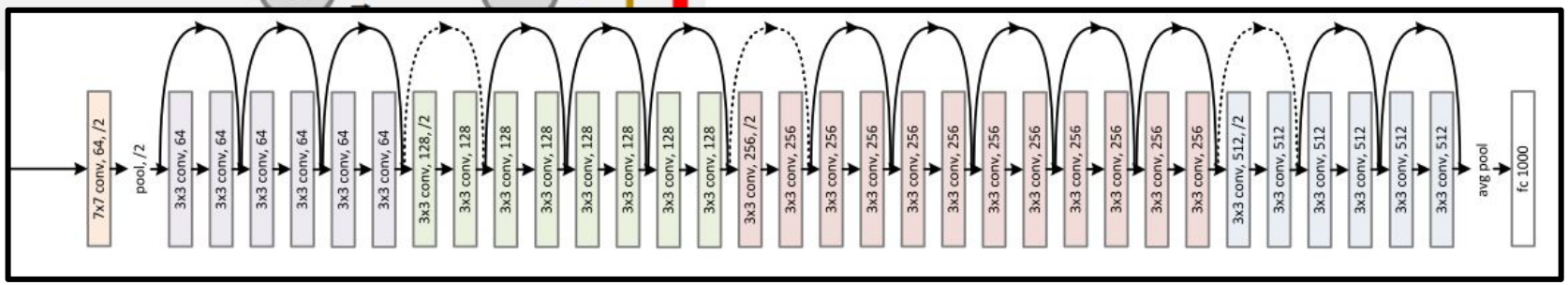
LSTM (Long Short Term Memory)

$$\vec{f} = \sigma\left(W^{xf}\vec{x}_i + W^{hf}\vec{h}_{i-1}\right)$$
$$\vec{i} = \sigma\left(W^{xi}\vec{x}_i + W^{hi}\vec{h}_{i-1}\right)$$
$$\vec{g} = \tanh\left(W^{xg}\vec{x}_i + W^{hg}\vec{h}_{i-1}\right)$$
$$\vec{o} = \sigma\left(W^{xo}\vec{x}_i + W^{ho}\vec{h}_{i-1}\right)$$

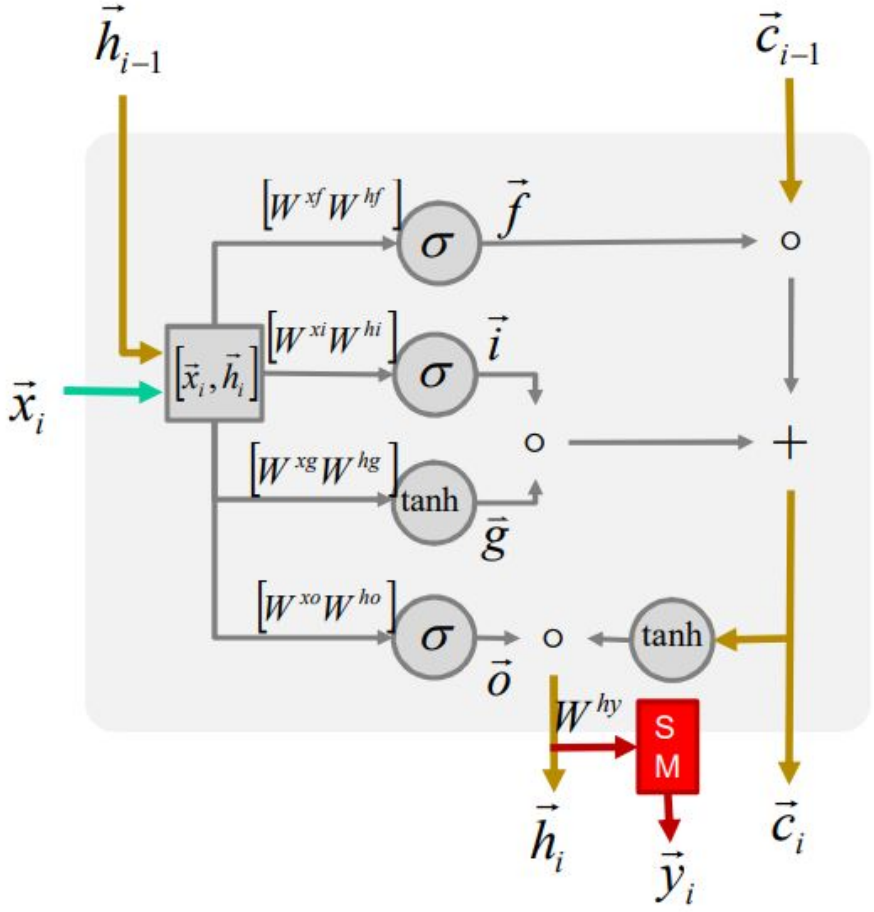


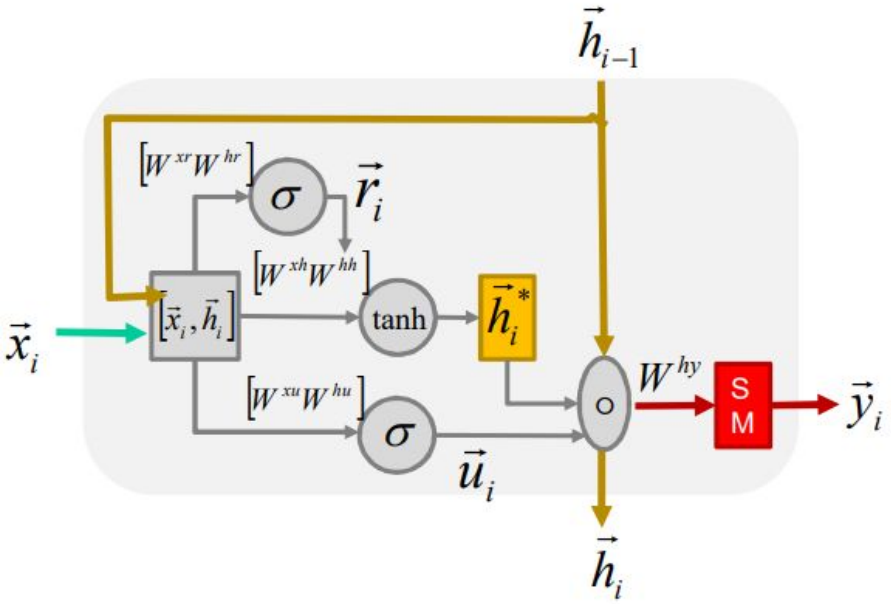
Gradient simplifié

Ça vous rappelle quelque chose ?
Le Resnet !



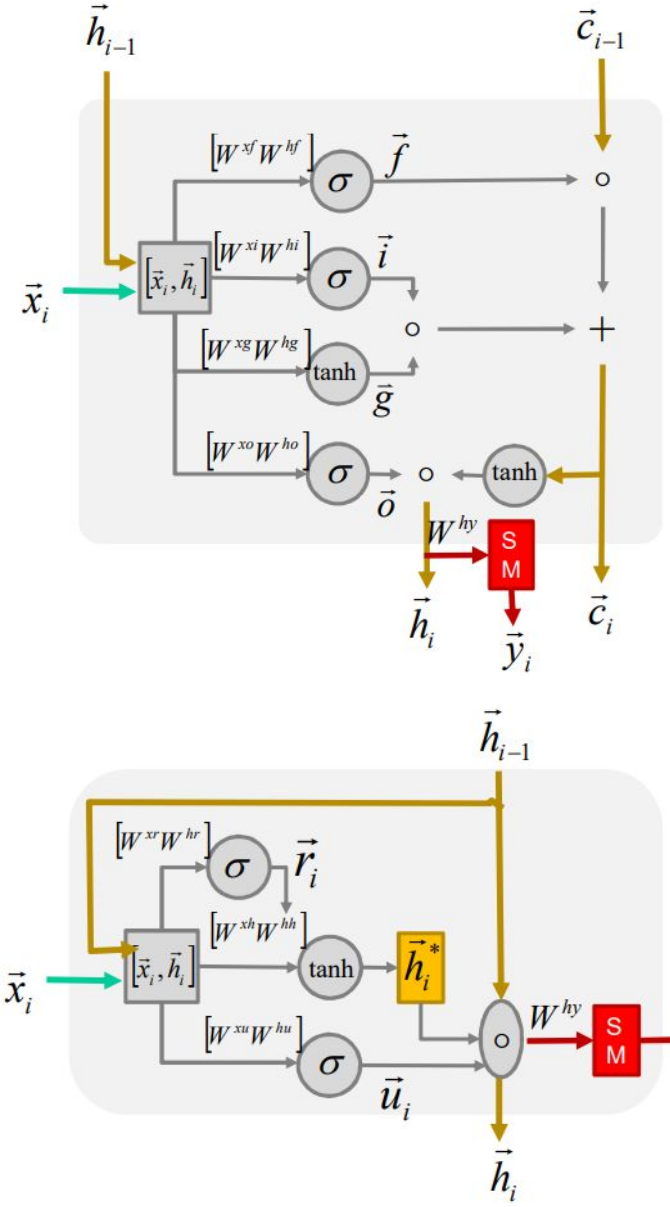
LSTM et GRU


$\vec{h}_{i-1}$
$\vec{c}_{i-1}$
$[W^{xf}W^{hf}]$
σ
$\vec{f}$
$[W^{xi}W^{hi}]$
$\vec{i}$
$\vec{x}_i$
$[\vec{x}_i, \vec{h}_i]$
$[W^{xg}W^{hg}]$
tanh
$\vec{g}$
$[W^{xo}W^{ho}]$
$\vec{o}$
tanh
W^{hy}
S
M
$\vec{h}_i$
$\vec{y}_i$
$\vec{c}_i$

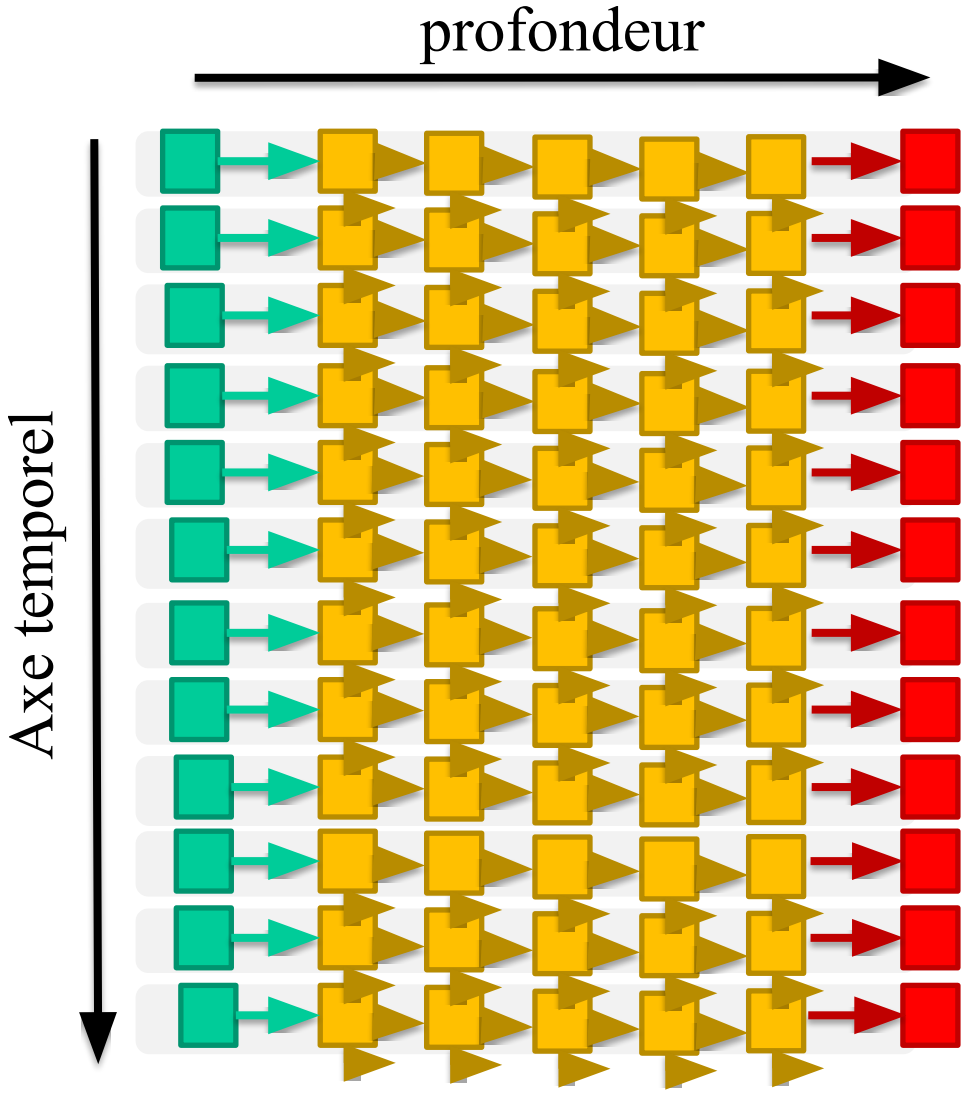

$\vec{h}_{i-1}$
$[W^{xr}W^{hr}]$
σ
$\vec{r}_i$
$\vec{x}_i$
$[\vec{x}_i, \vec{h}_i]$
$[W^{xh}W^{hh}]$
tanh
$\vec{h}_i^*$
$[W^{xu}W^{hu}]$
σ
$\vec{u}_i$
W^{hy}
S
M
$\vec{y}_i$
$\vec{h}_i$

LSTM et GRU

- Servent à protéger le gradient
- Conçus empiriquement
- GRU légèrement plus simple
- Les “gates” ne servent qu’à bloquer ou permettre à l’information (données ou temporelle) de passer



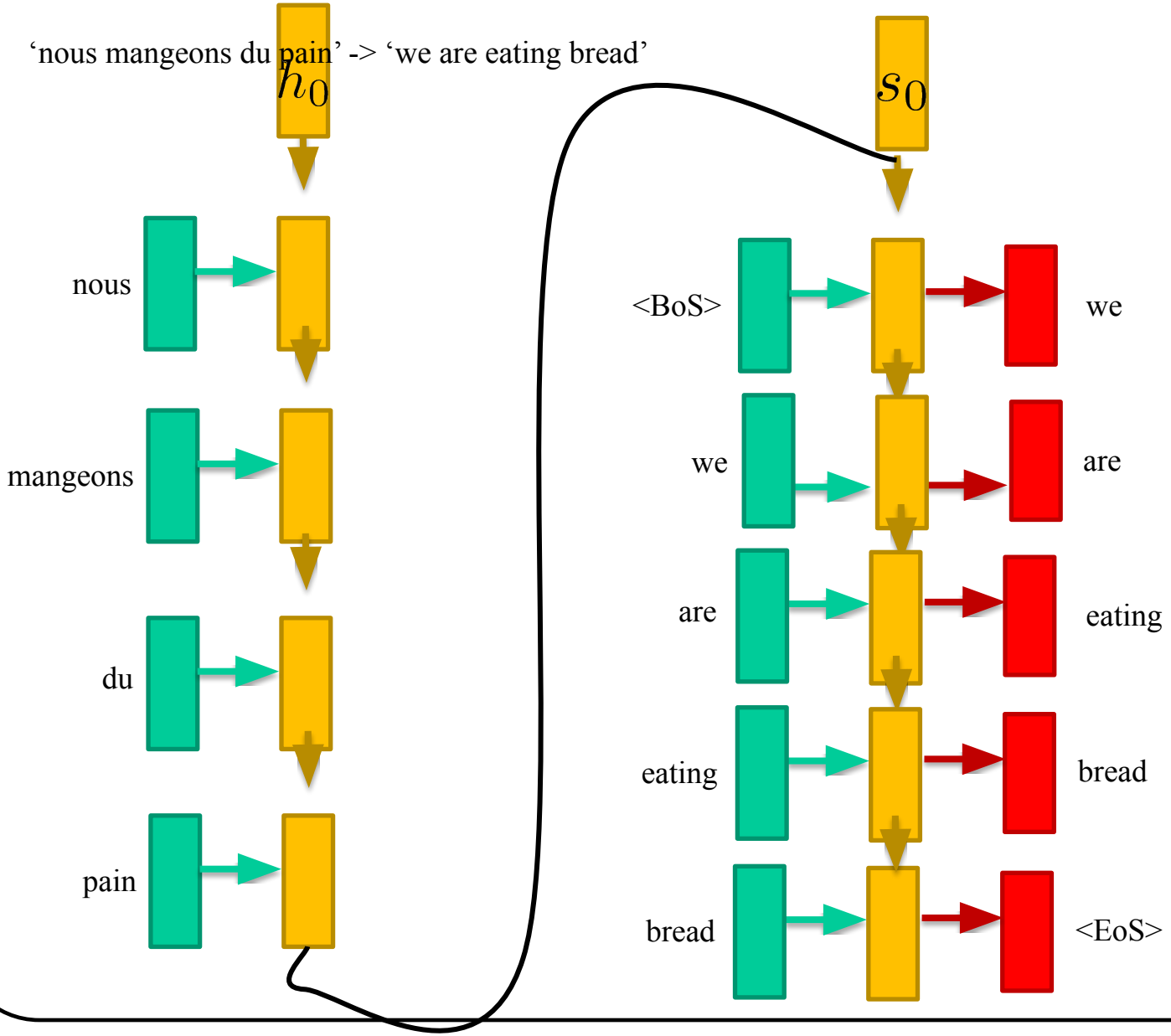
RNN multi-couches



Modèles d’attention

Seq2Seq:

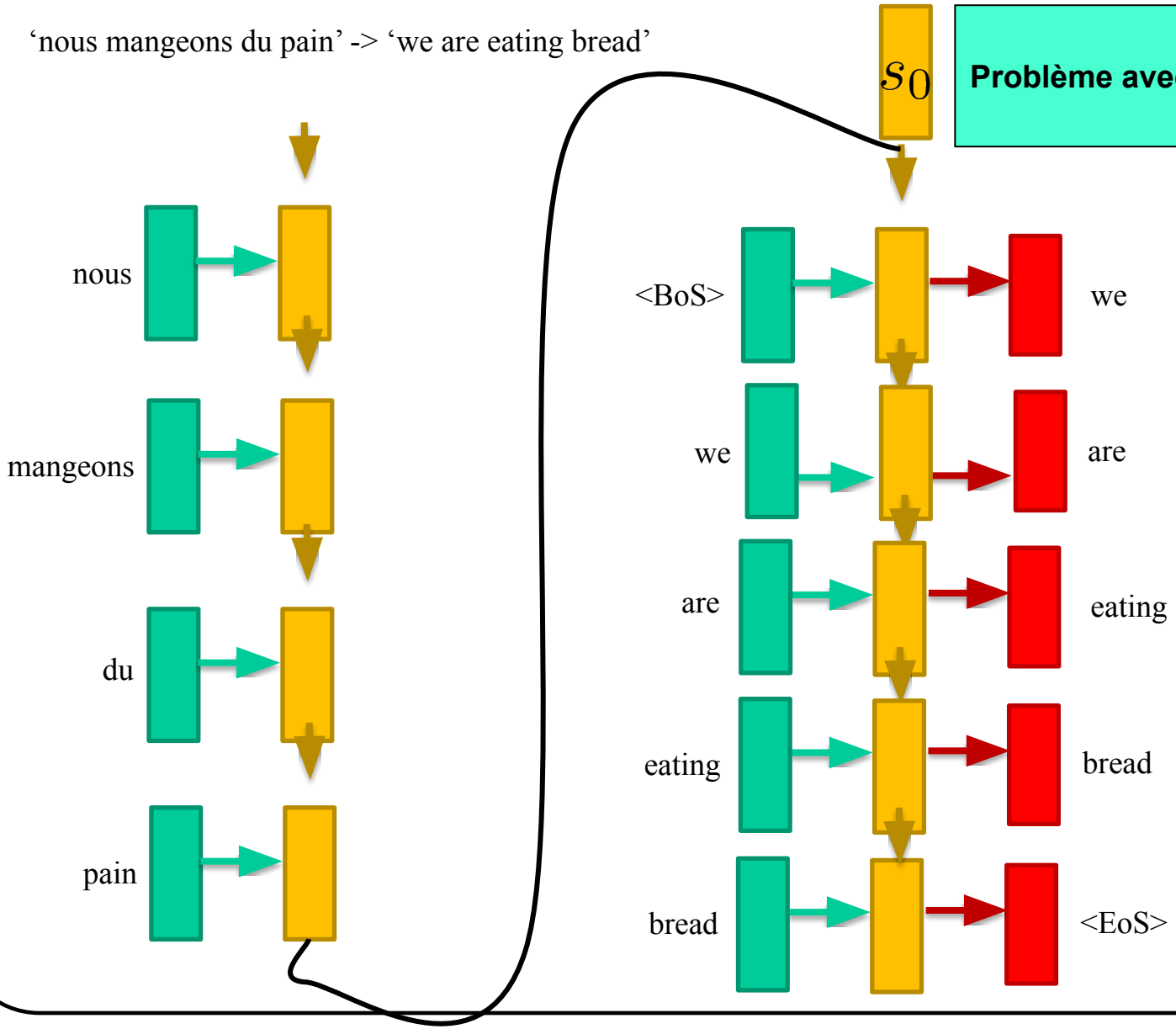
Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.



Seq2Seq:

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

‘nous mangeons du pain’ -> ‘we are eating bread’

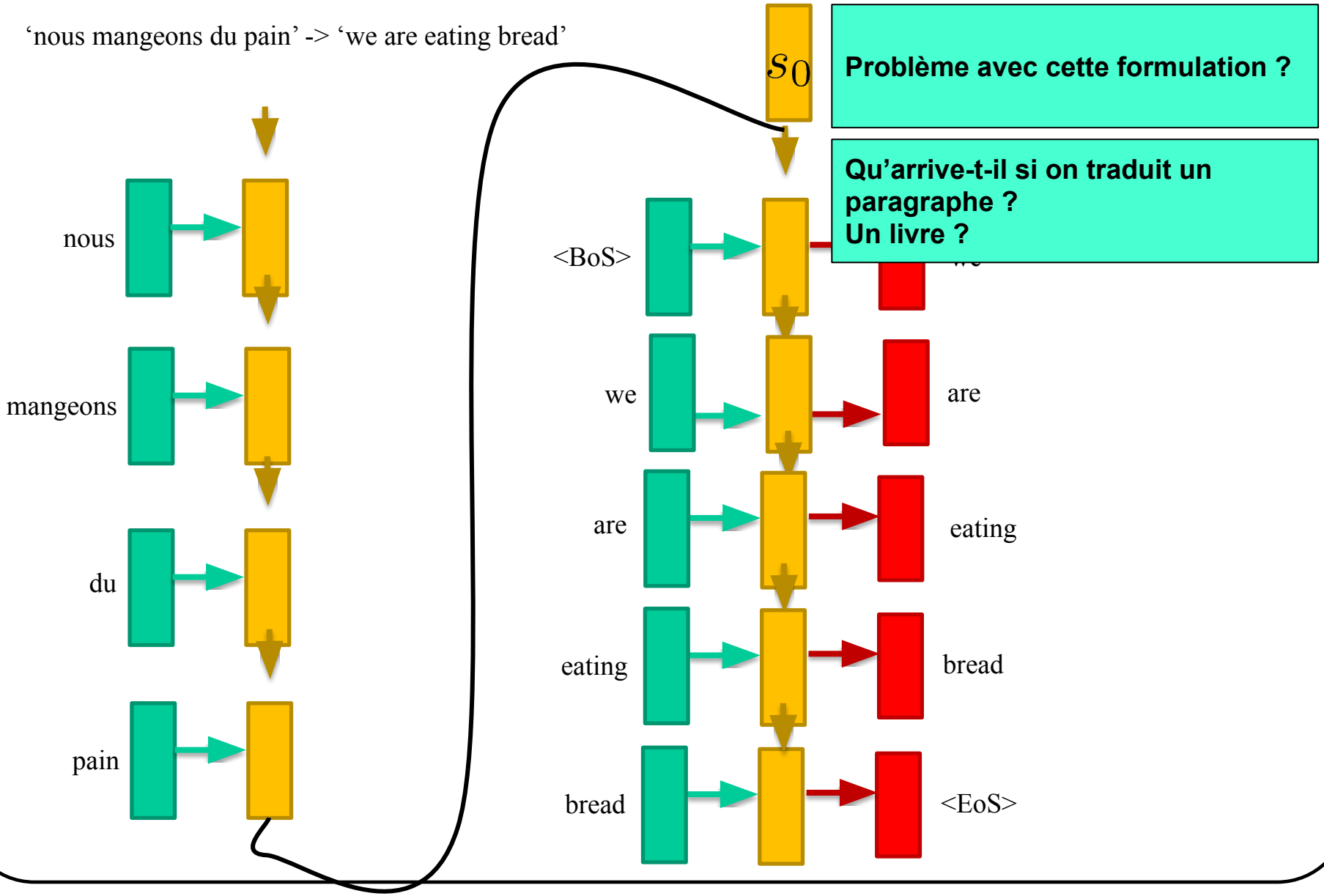


Problème avec cette formulation ?

Seq2Seq:

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

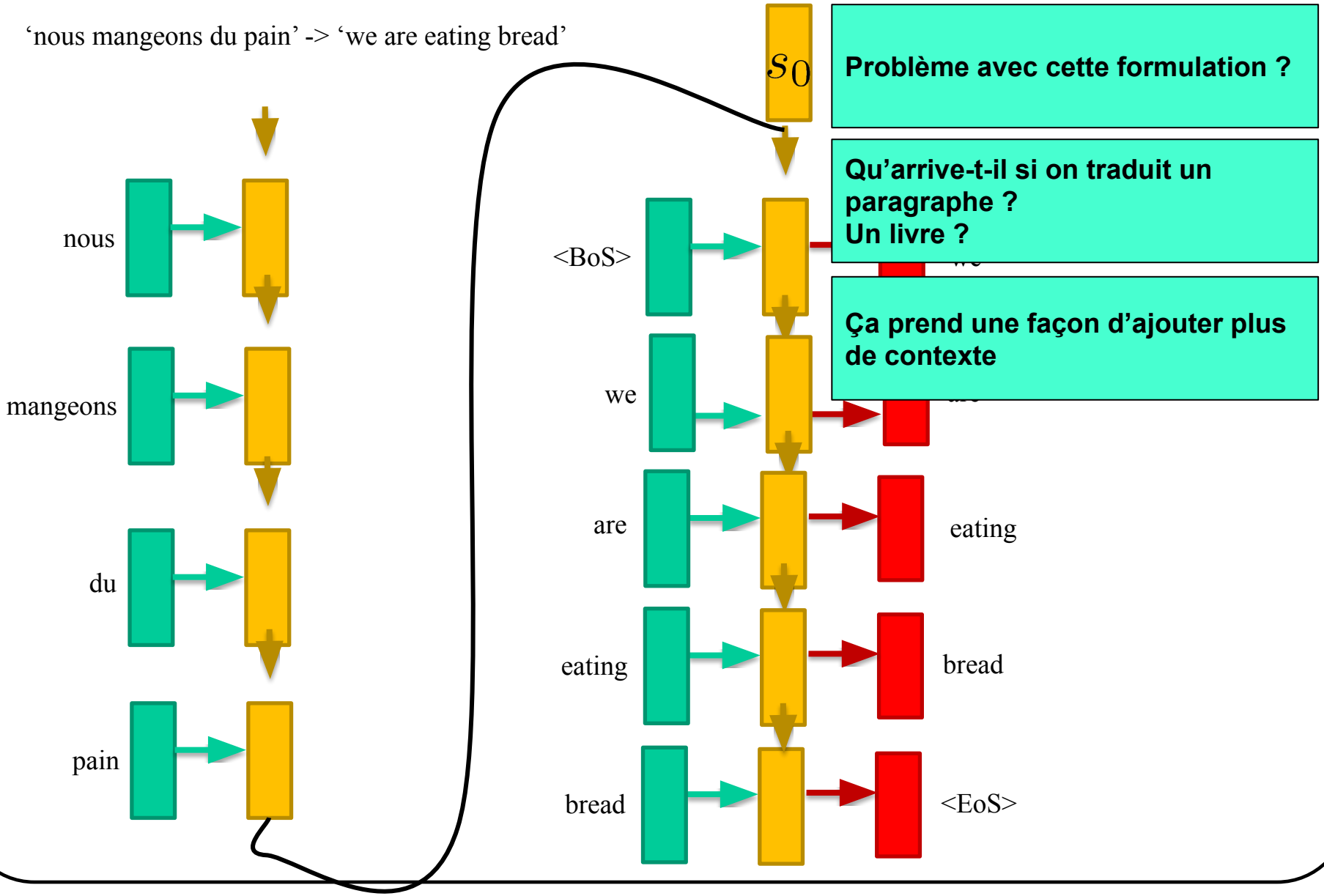
‘nous mangeons du pain’ -> ‘we are eating bread’



Seq2Seq:

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

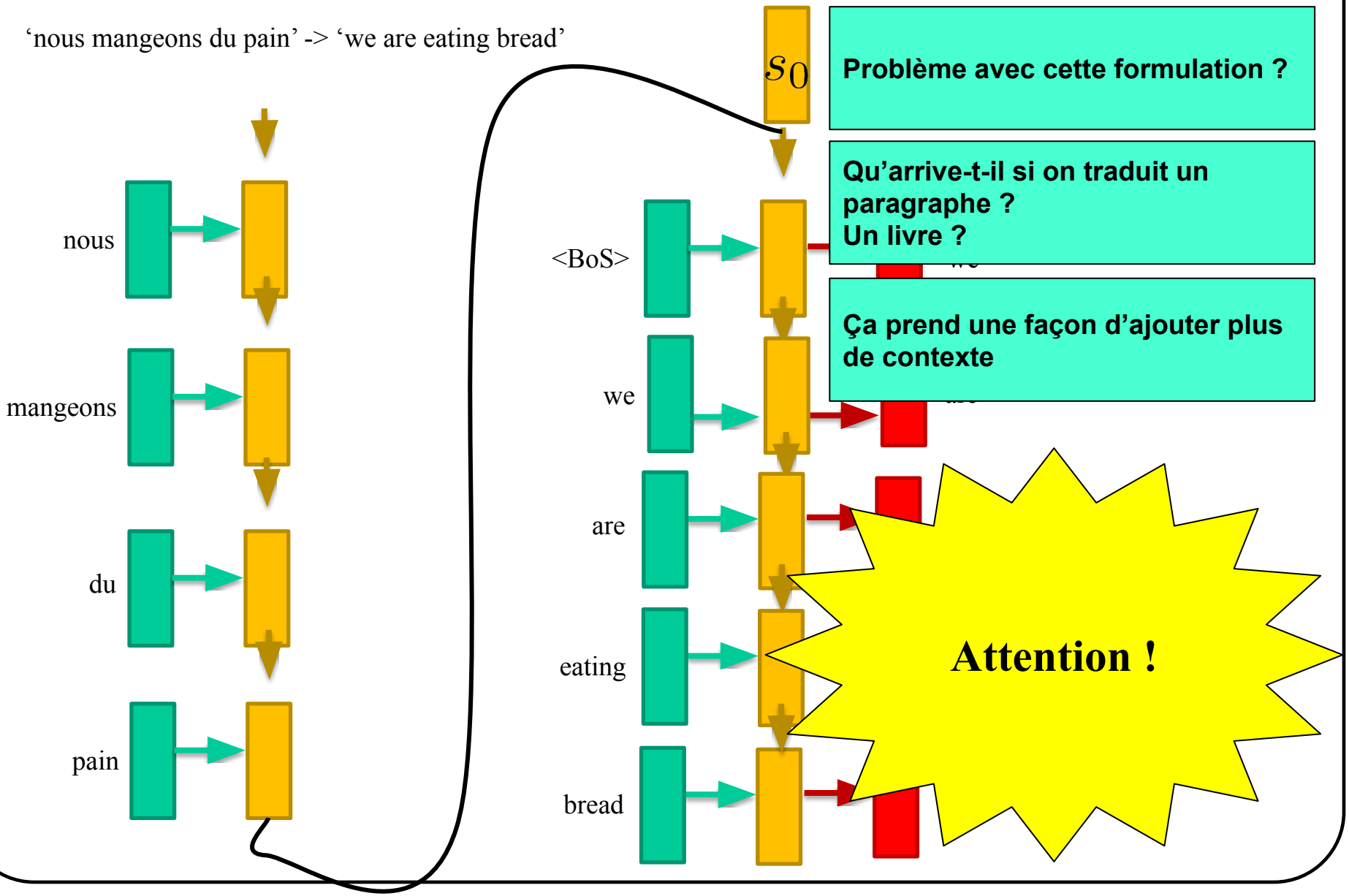
‘nous mangeons du pain’ -> ‘we are eating bread’



Seq2Seq:

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

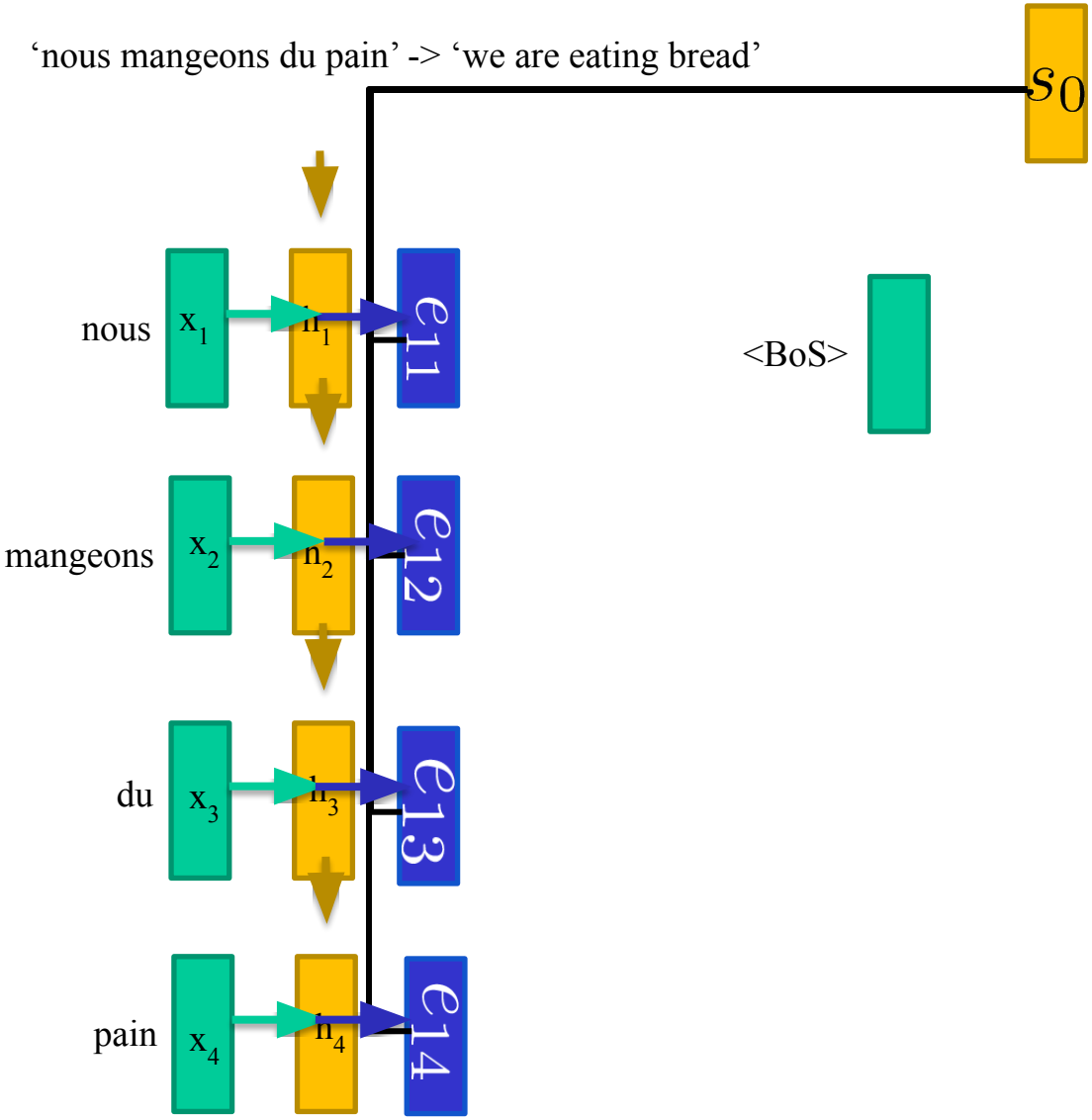
‘nous mangeons du pain’ -> ‘we are eating bread’



Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’

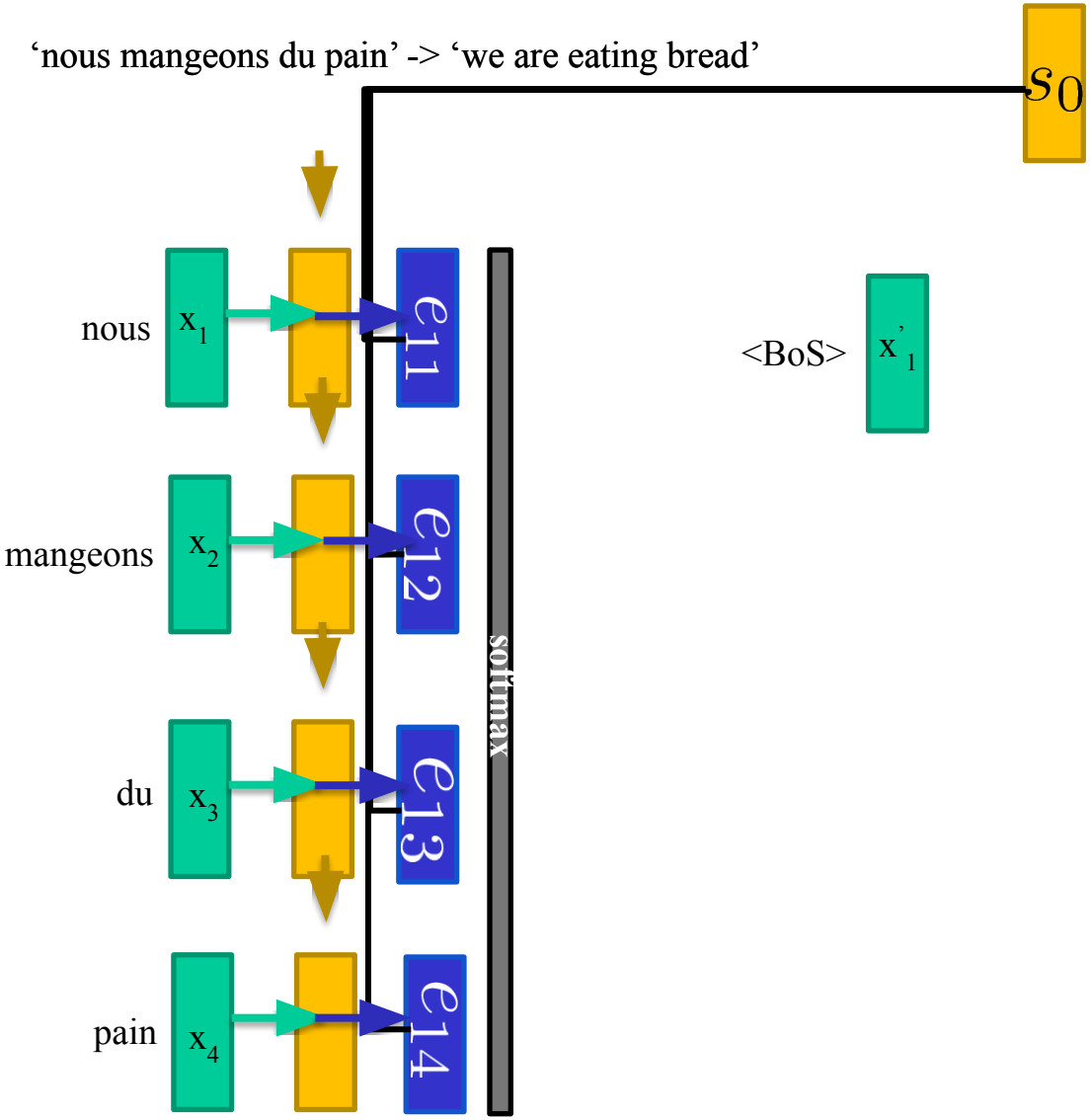


$$e_{ti} = f(s_{t-1}, h_i)$$

f est un ‘MLP’ à une couche

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.



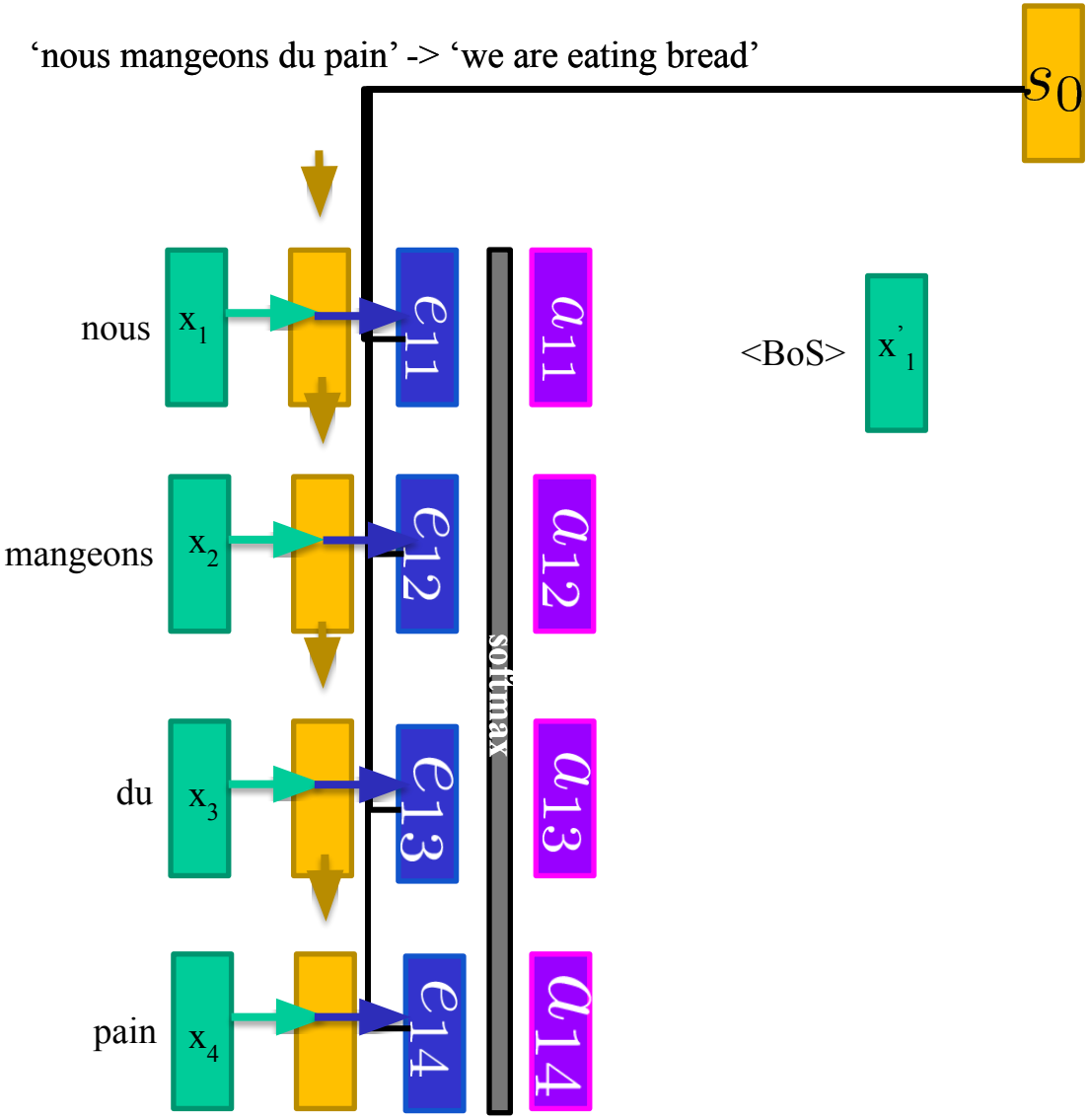
$$e_{ti} = f(s_{t-1}, h_i)$$

f est un ‘MLP’ à une couche

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’



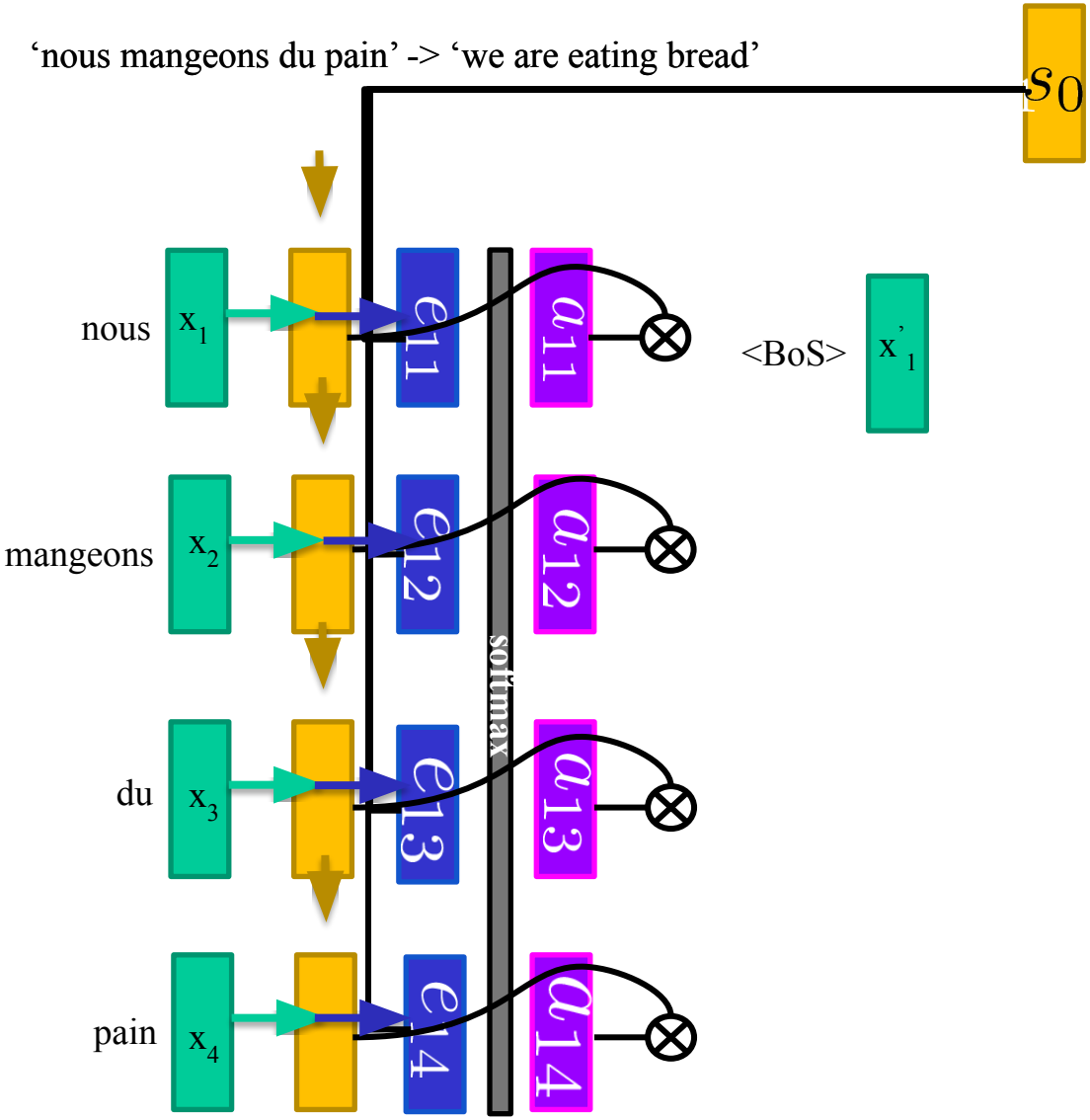
$$e_{ti} = f(s_{t-1}, h_i)$$

f est un ‘MLP’ à une couche

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})}$$

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.



$$e_{ti} = f(s_{t-1}, h_i)$$

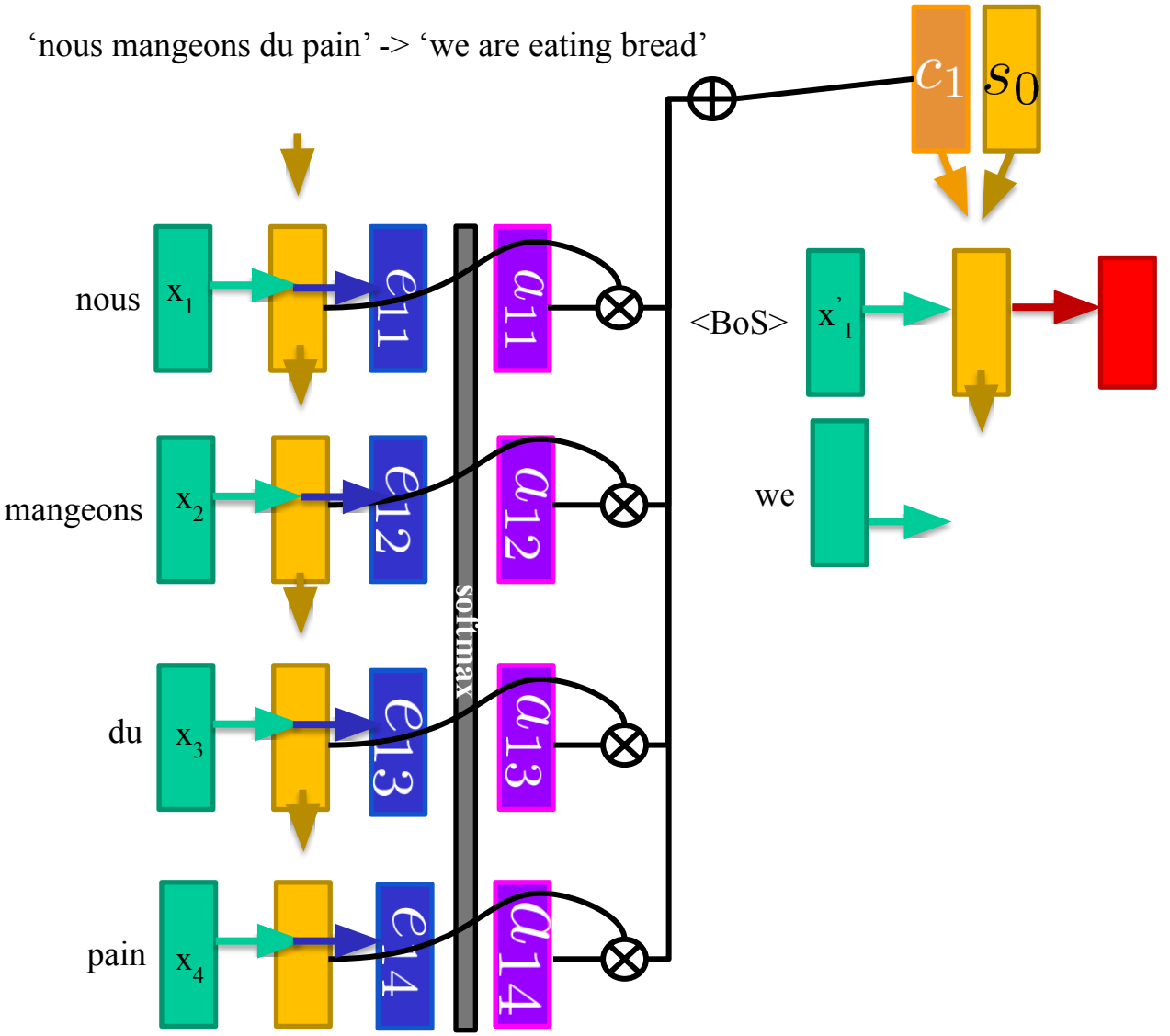
f est un ‘MLP’ à une couche

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})}$$

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’



$$e_{ti} = f(s_{t-1}, h_i)$$

f est un ‘MLP’ à une couche

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})}$$

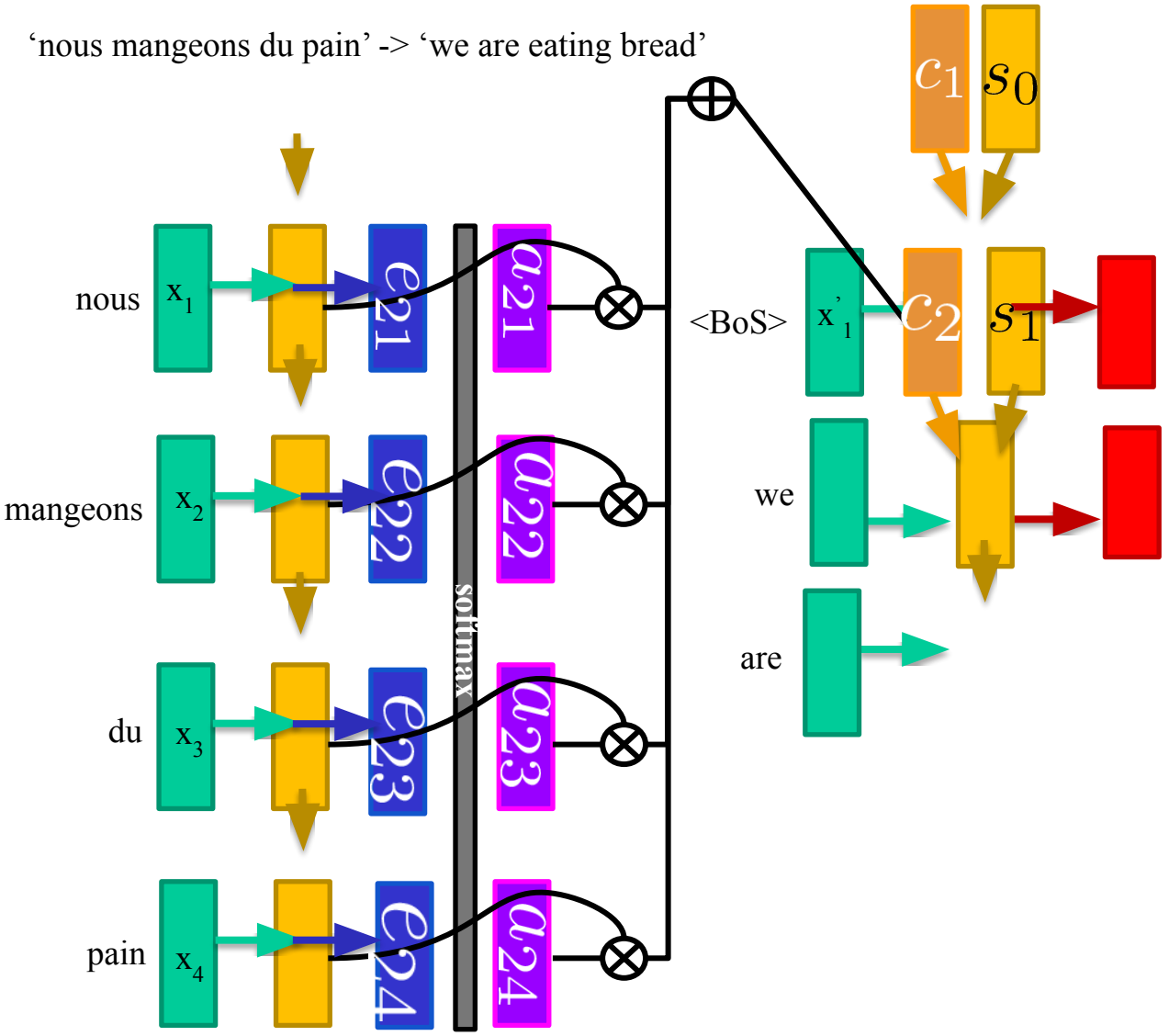
$$c_t = \sum_{i=1}^{T} a_{ti} h_i$$

c_t est une somme pondérée est couches cachées de l’encodeur

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’



$$e_{ti} = f(s_{t-1}, h_i)$$

f est un ‘MLP’ à une couche

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})}$$

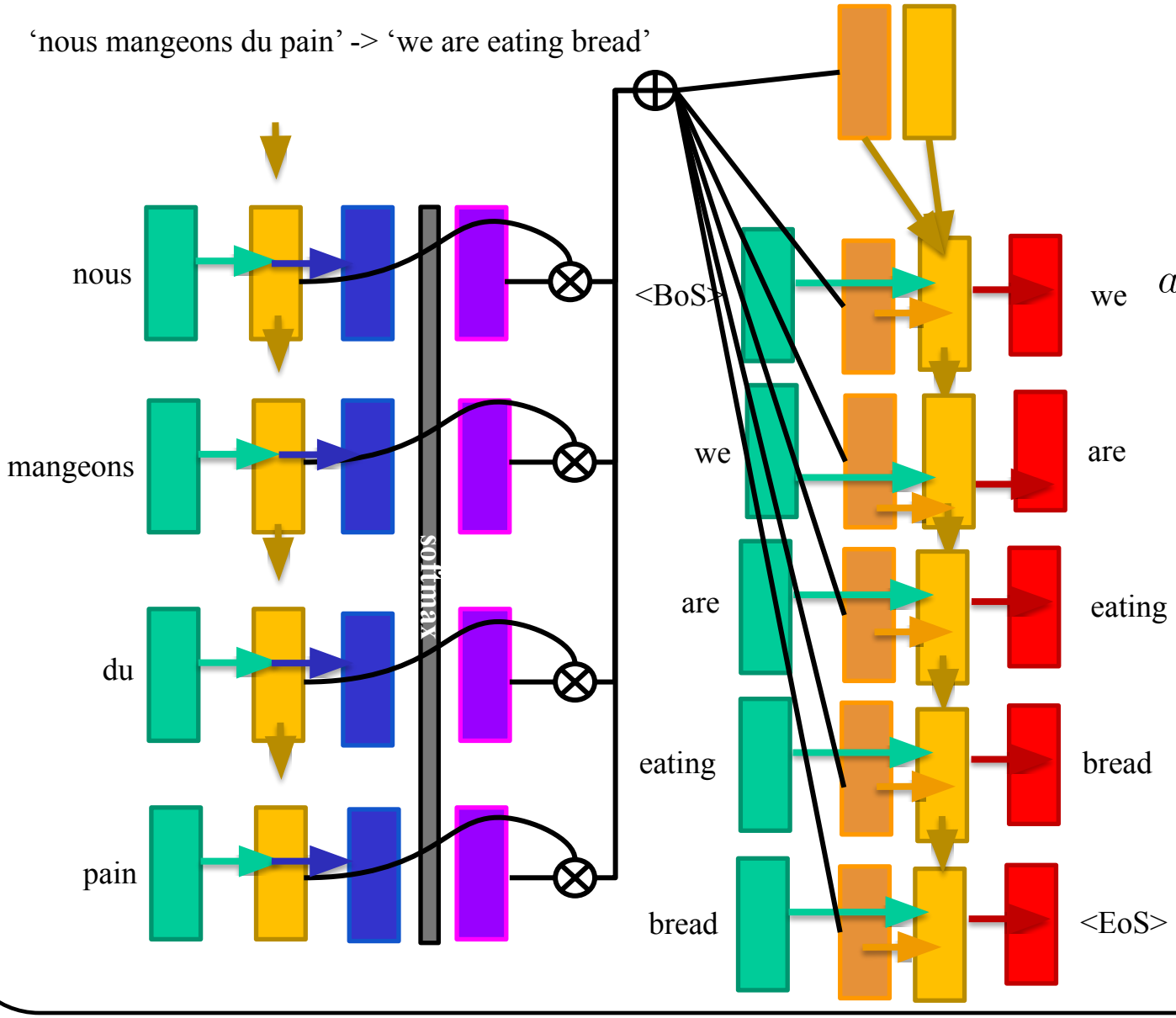
$$c_t = \sum_{i=1}^{T} a_{ti} h_i$$

c_t est une somme pondérée est couches cachées de l’encodeur

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’



$$e_{ti} = f(s_{t-1}, h_i)$$

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})}$$

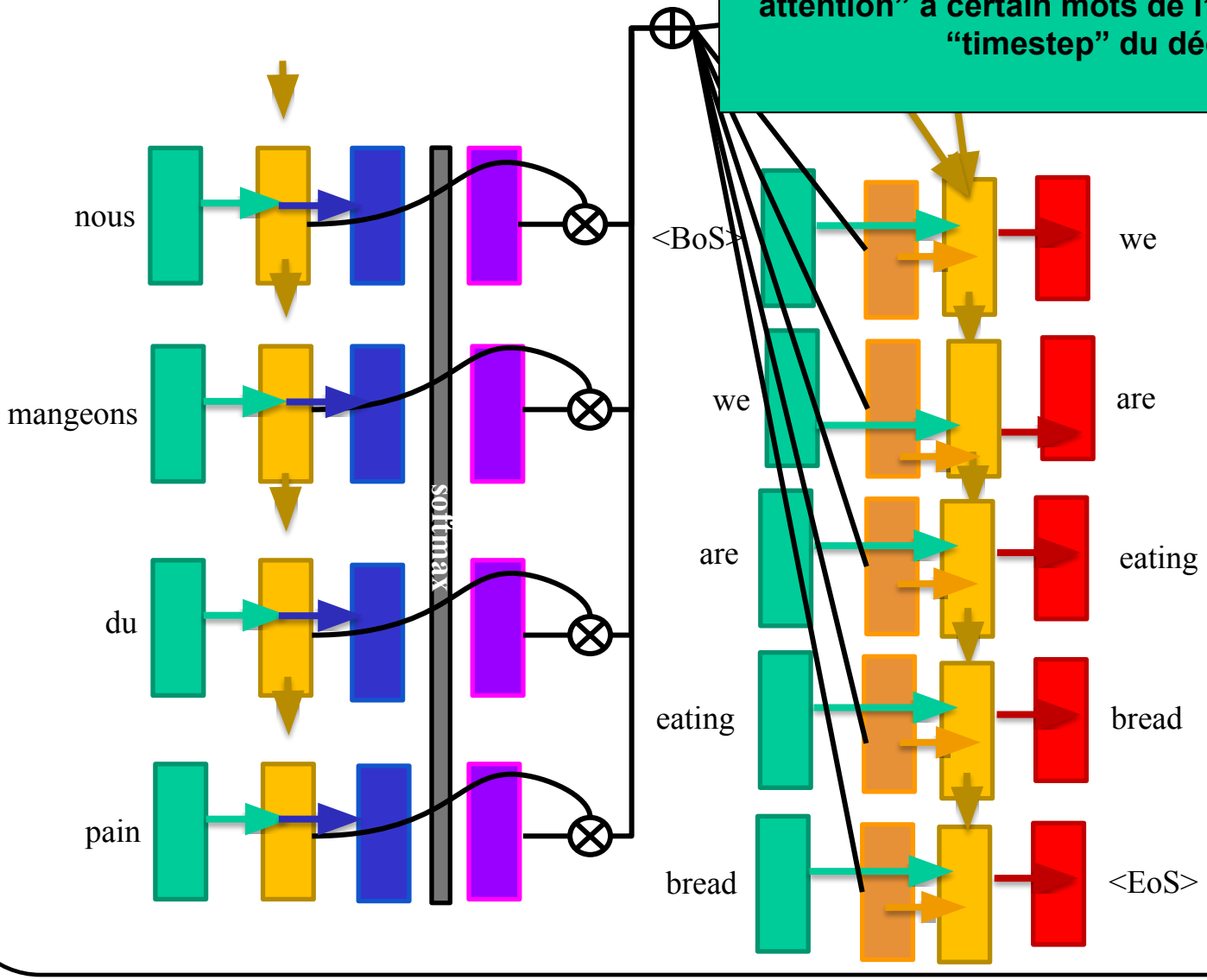
$$c_t = \sum_{i=1}^{T} a_{ti} h_i$$

Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

‘nous mangeons du pain’ -> ‘we are eating bread’

Le modèle apprend par lui-même à “porter attention” à certain mots de l’encodeur à chaque “timestep” du décodage



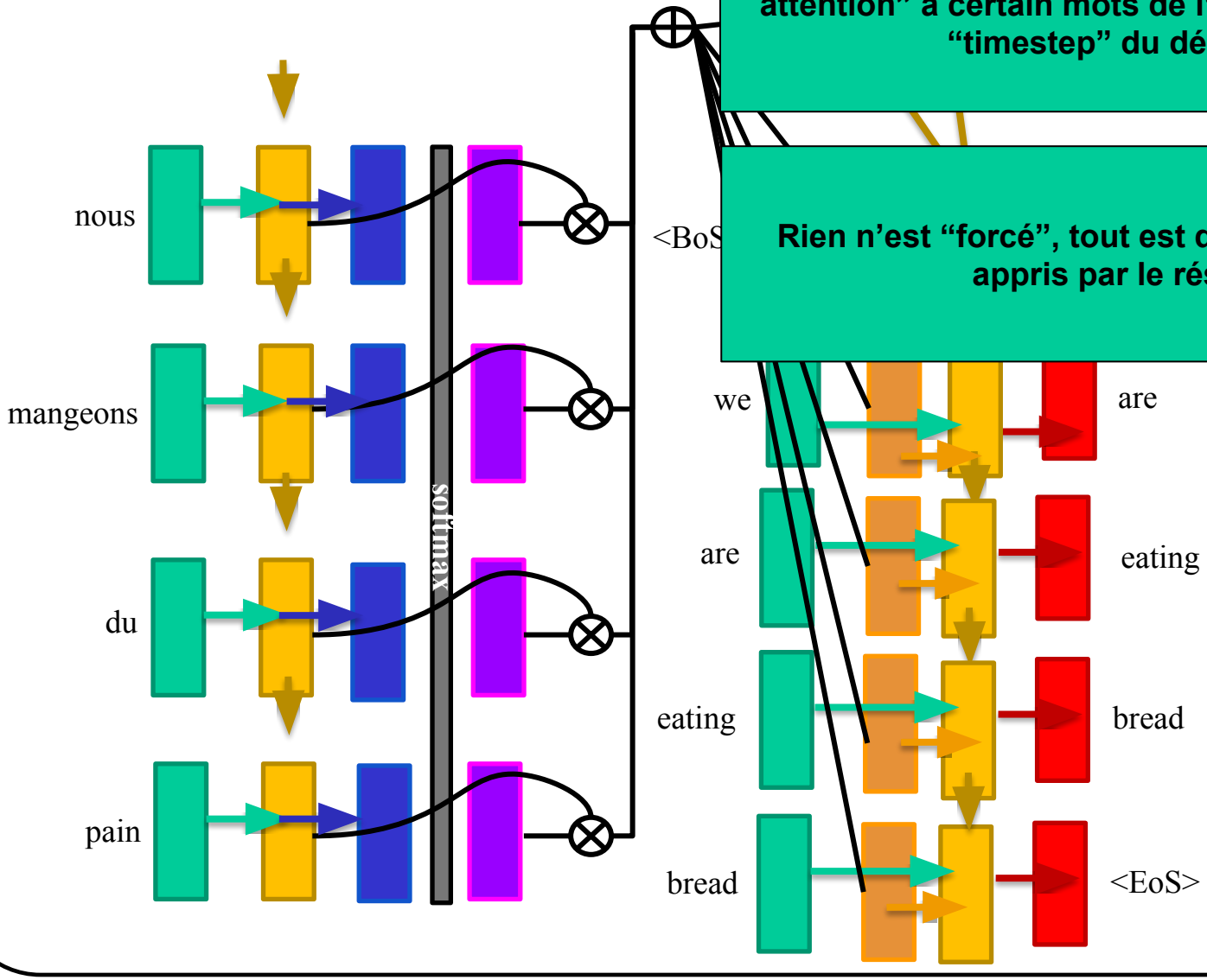
Seq2Seq avec attention

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

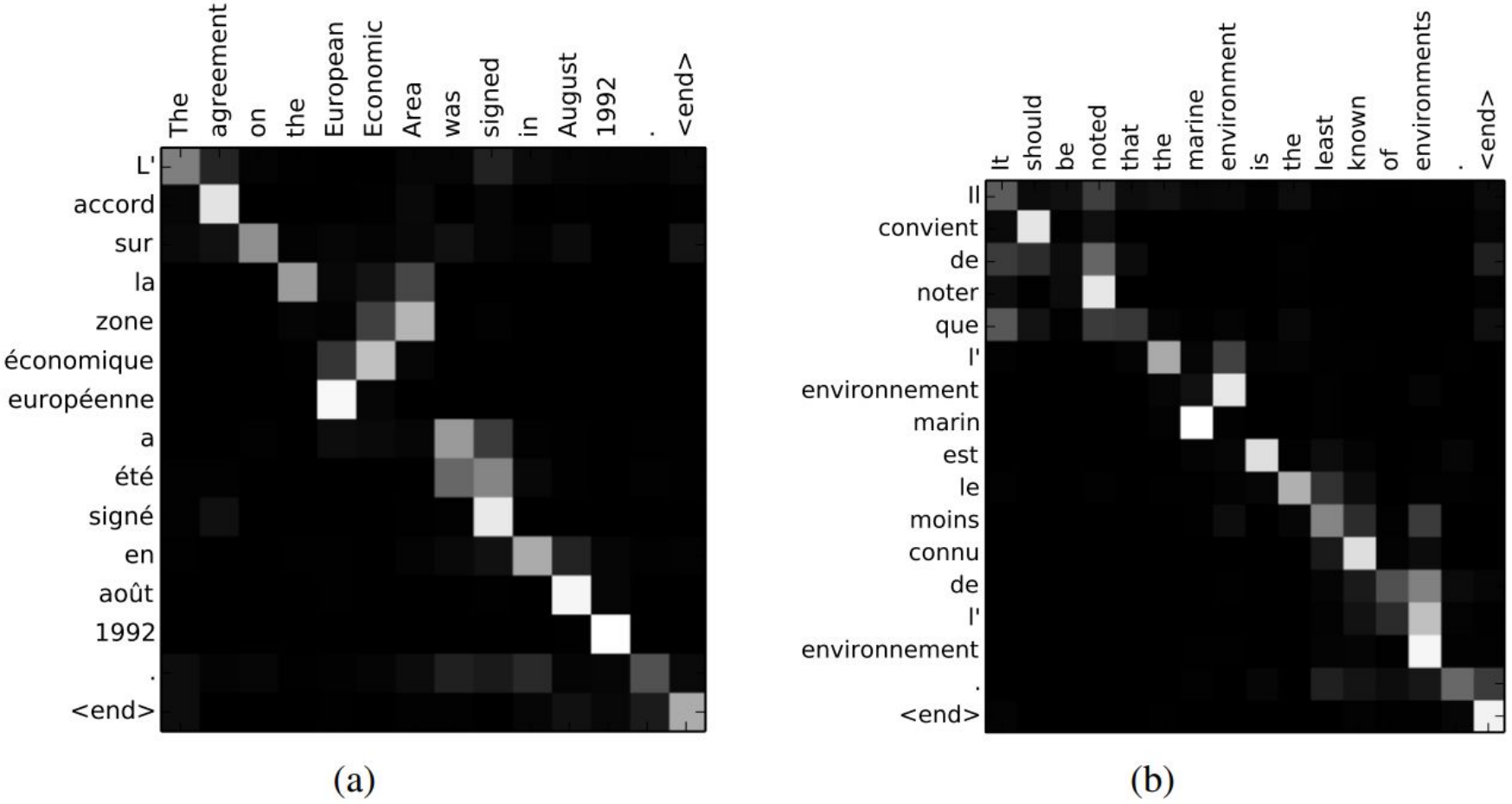
‘nous mangeons du pain’ -> ‘we are eating bread’

Le modèle apprend par lui-même à “porter attention” à certain mots de l’encodeur à chaque “timestep” du décodage

Rien n’est “forcé”, tout est différentiable, donc appris par le réseau !

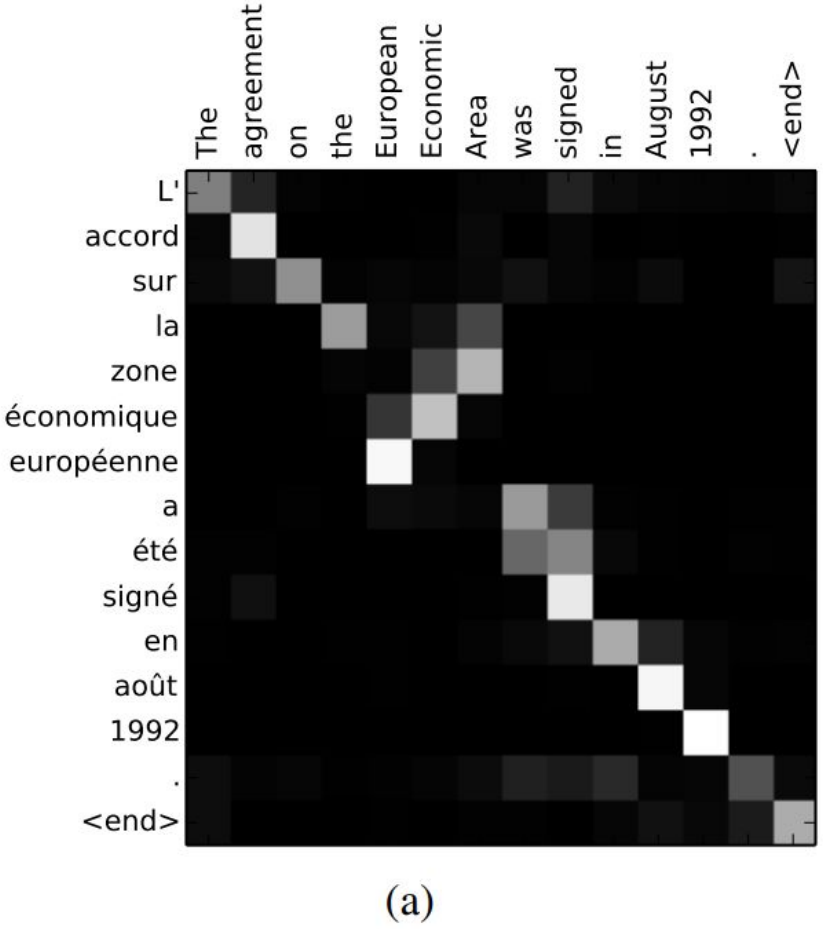


Seq2Seq avec attention

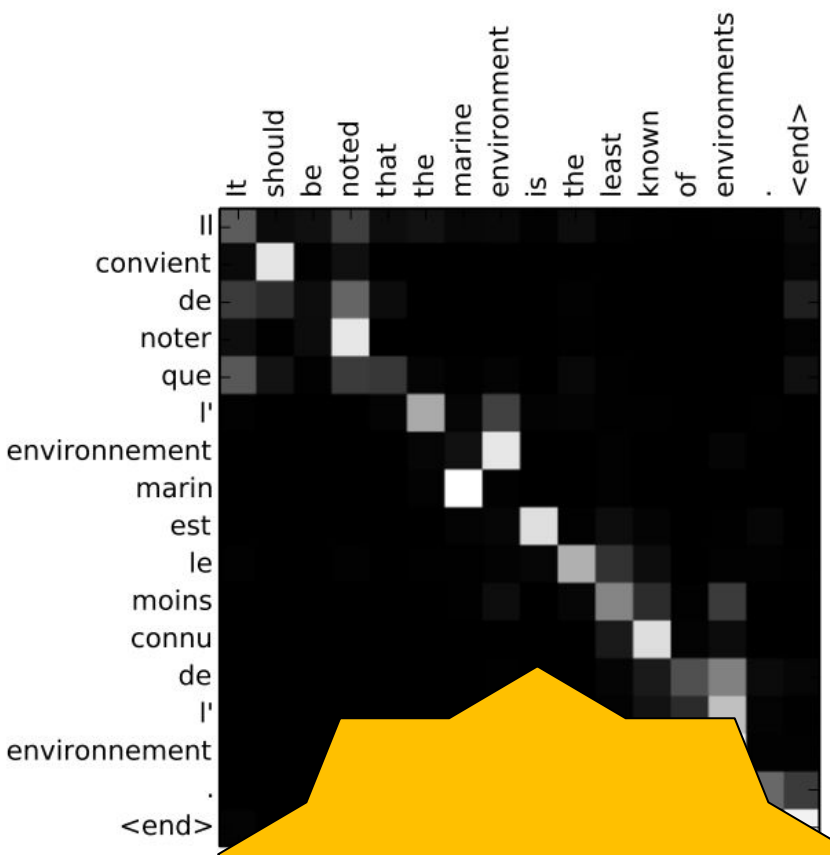


Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Seq2Seq avec attention



(a)



Ajoute de l'interprétabilité au modèle !

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning
arXiv:1409.0473.