

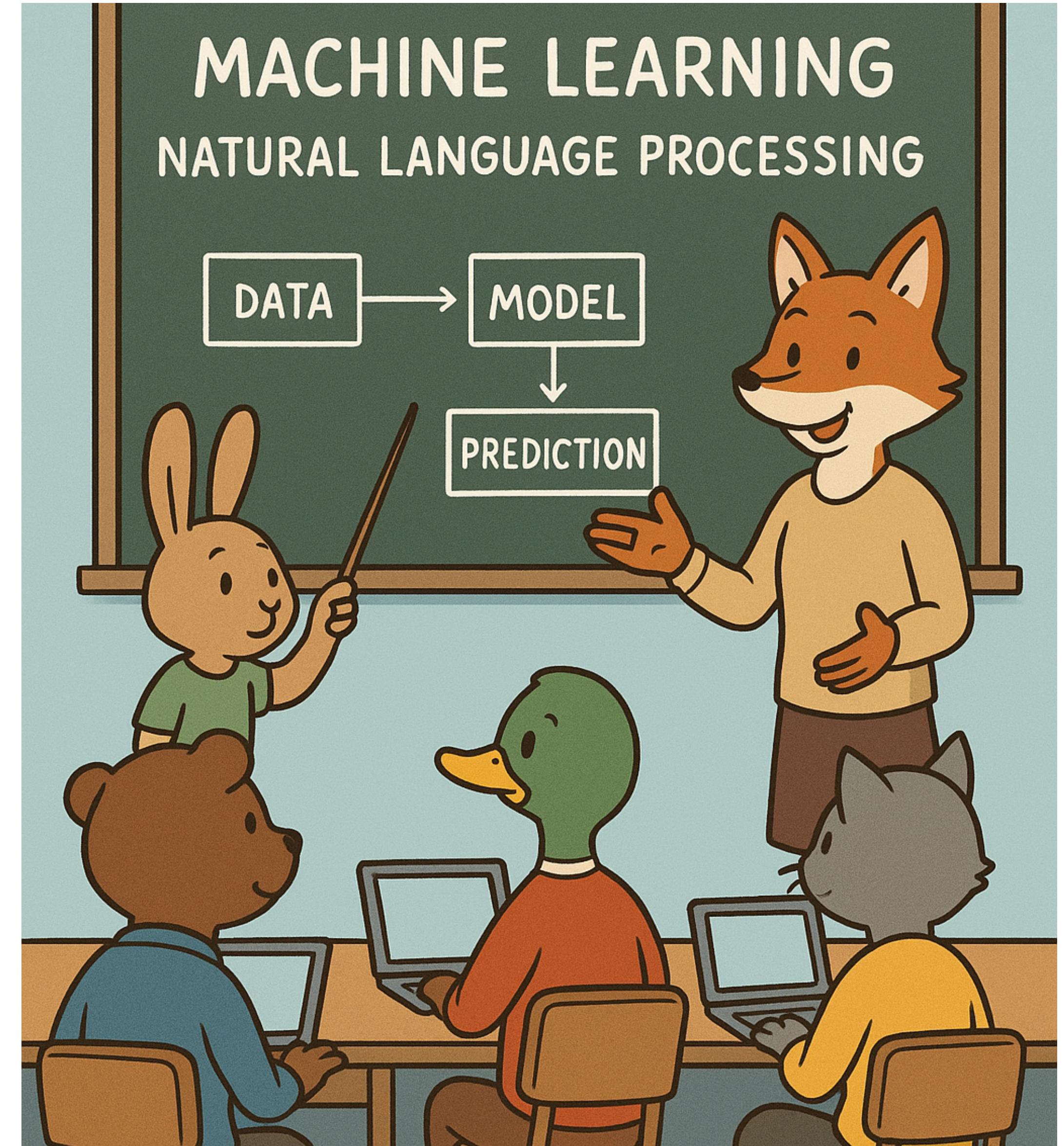
Apprentissage de représentations en TAL

ETAL 2025 - Roscoff - 1-5 sept 2025

Marie Candito

Préambule

- Profils divers
- Ce cours comprend
 - Concepts introductifs d'apprentissage automatique pour le TAL
 - Architectures à la base des IAs génératives



Plan

- | • Notions d'apprentissage pour le TAL | | |
|---------------------------------------|----------------------------------|--------------------------|
| • Réseaux de neurones | Exemple de classifieur | Apprentissage supervisé |
| • Vecteurs de mots statiques | | |
| • Vecteurs de mots contextualisés | | |
| • Transformers | Architecture encoder-only : BERT | |
| • Vecteurs de séquences de mots | | |
| • BERT | SentenceBERT | Apprentissage contrastif |
| • IAs génératives | | |
| • GPT et architecture decoder-only | In-context learning | SFT : instruction tuning |

Notions d'apprentissage pour le TAL

Notions d'apprentissage pour le TAL

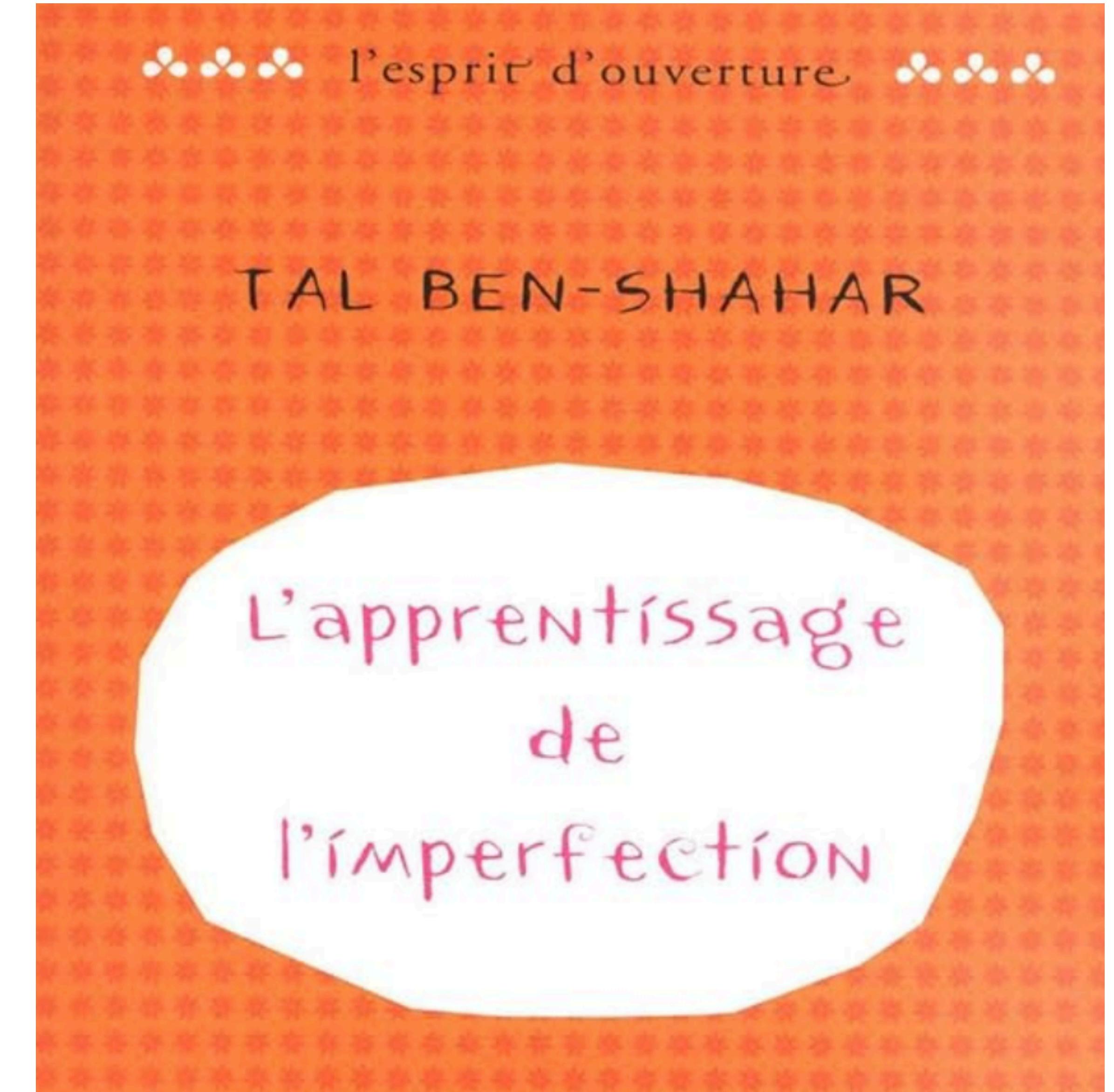
••• l'esprit d'ouverture •••

TAL BEN-SHAHAR

L'apprentissage
du
bonheur

Principes,
préceptes et rituels
pour être heureux

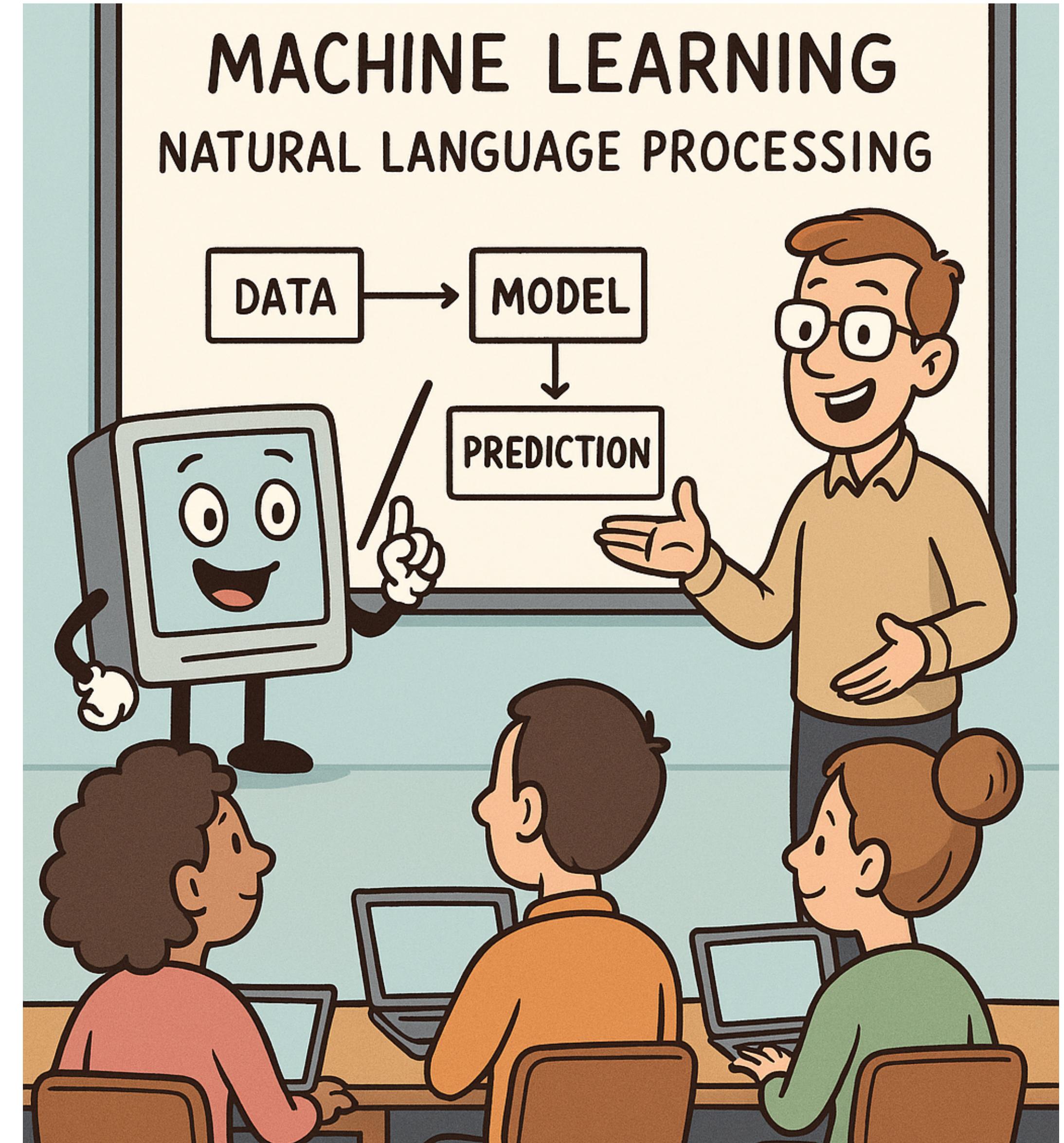
Notions d'apprentissage pour le TAL



Notions d'apprentissage pour le TAL

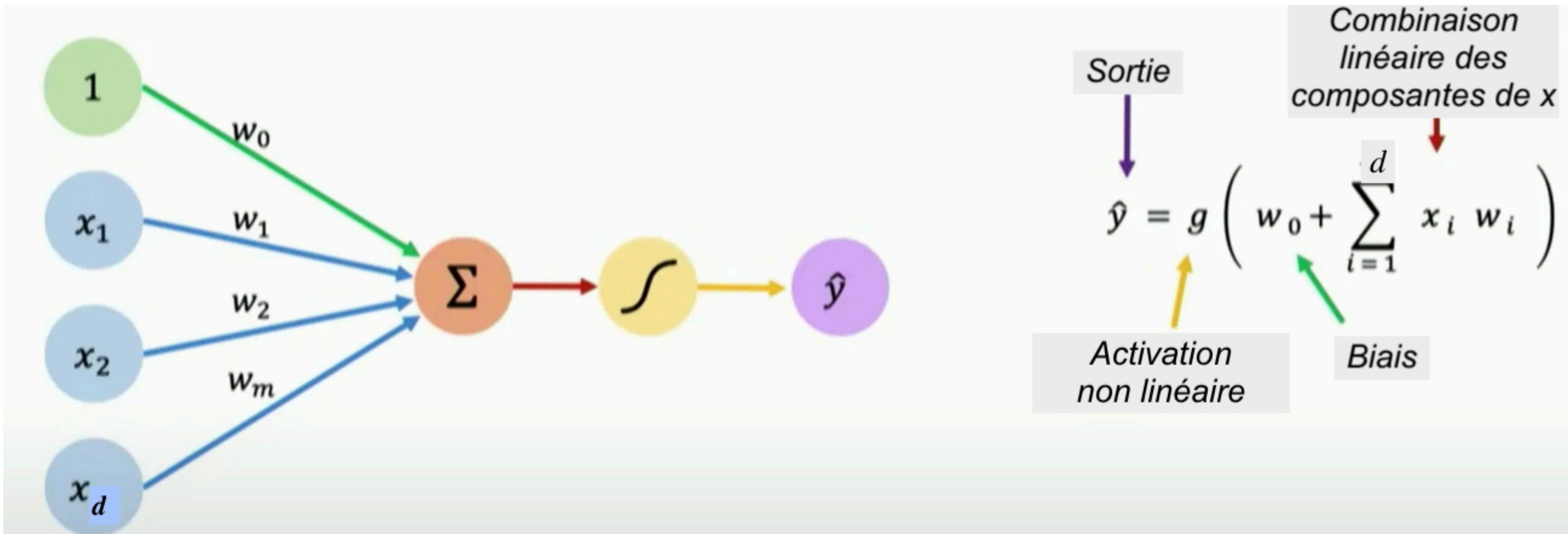
À chatGPT:

Peux-tu générer une image pour illustrer de manière comique un cours d'apprentissage automatique pour le traitement automatique des langues?



Neurone artificiel : le perceptron

Rosenblatt, 1957



Entrée

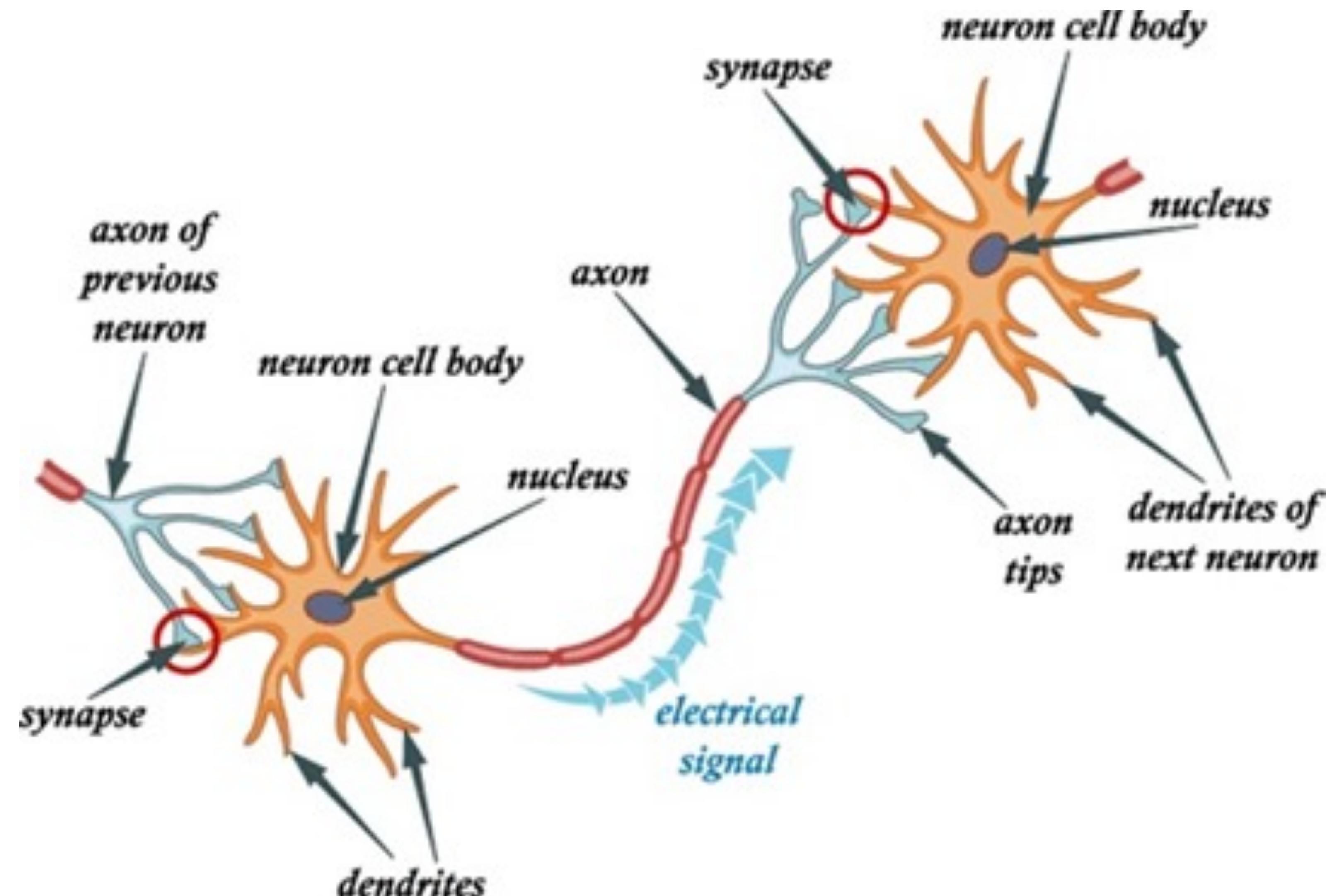
Poids

Somme

Activation

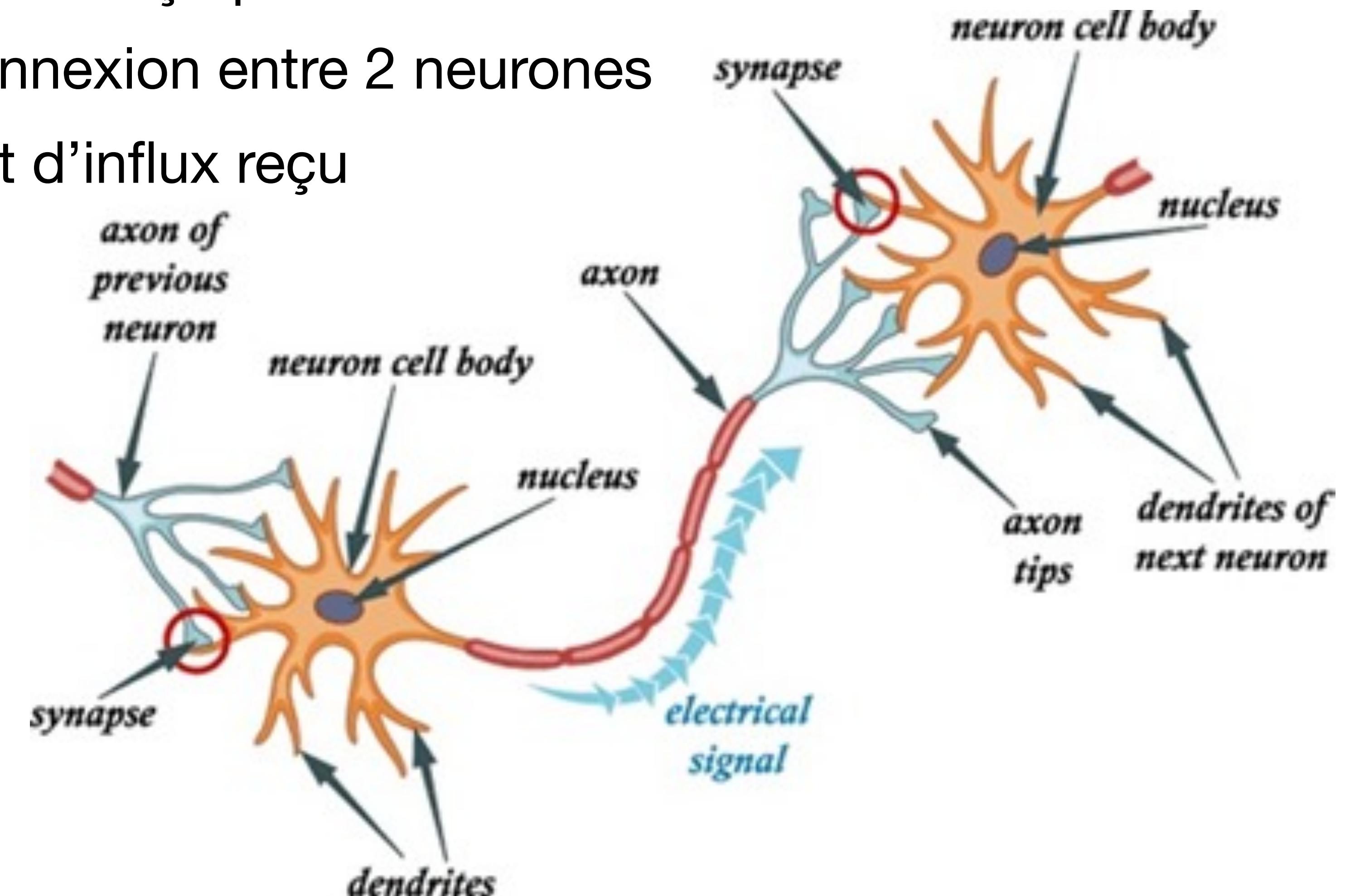
Sortie

Inspiration biologique



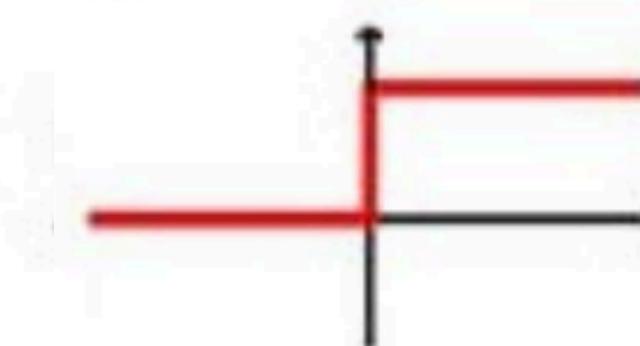
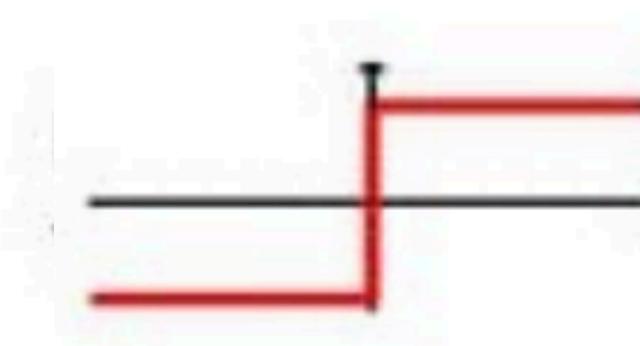
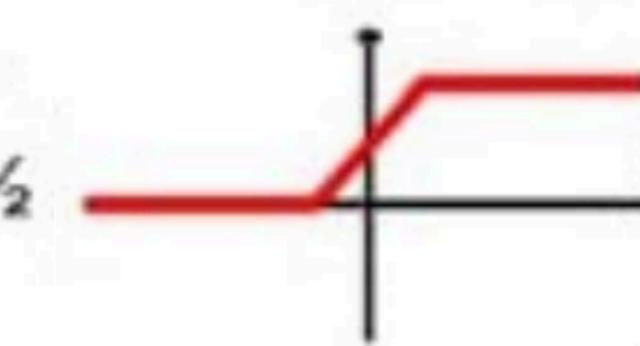
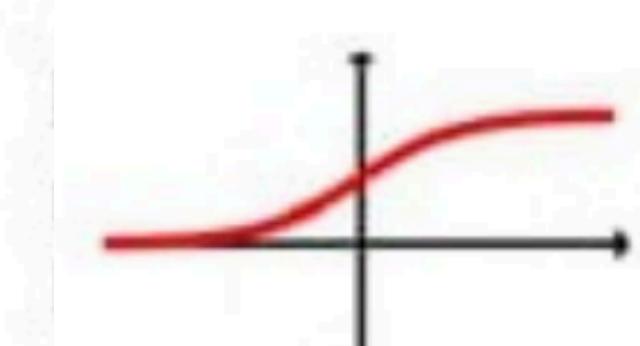
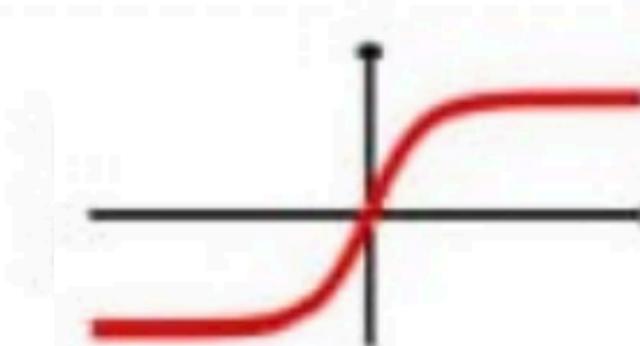
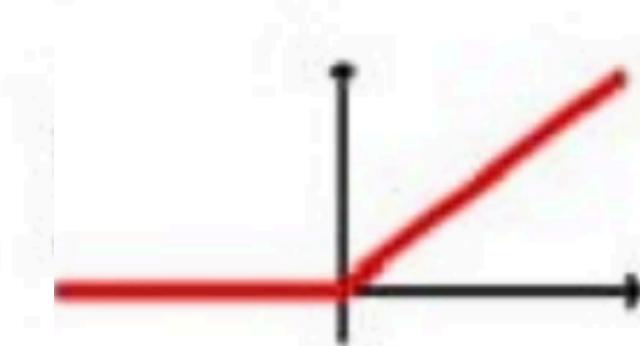
Inspiration biologique

- Composantes d'entrée : influx reçu par les dendrites
- Pondérées par force de connexion entre 2 neurones
- Activation si suffisamment d'influx reçu



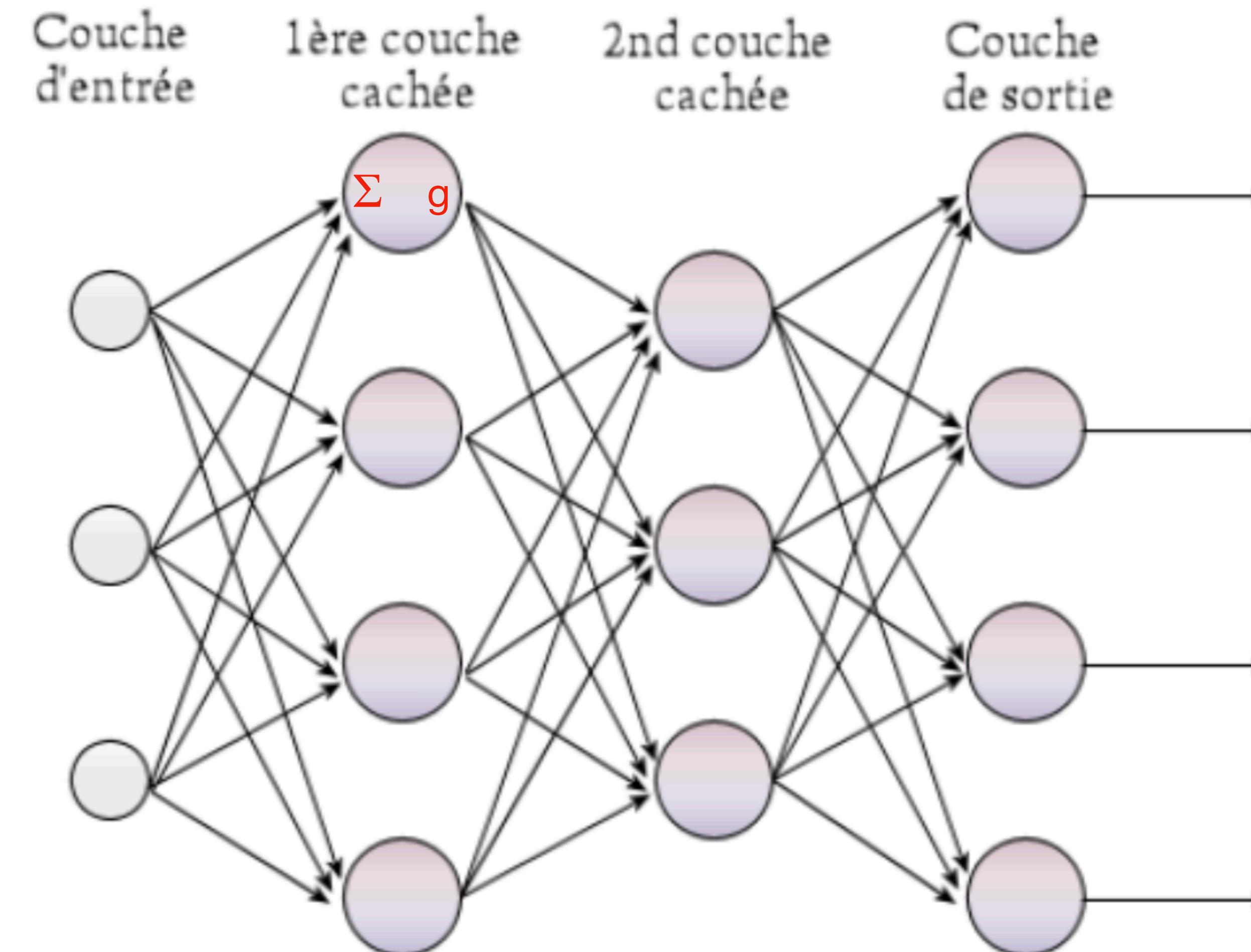
Fonctions d'activation usuelles

- Crucialement :
- Doivent être non-linéaires

Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	

Le perceptron multi-couches

Multi-layer perceptron (MLP) ou Feed-forward neural network (FFN)



Le perceptron multi-couches

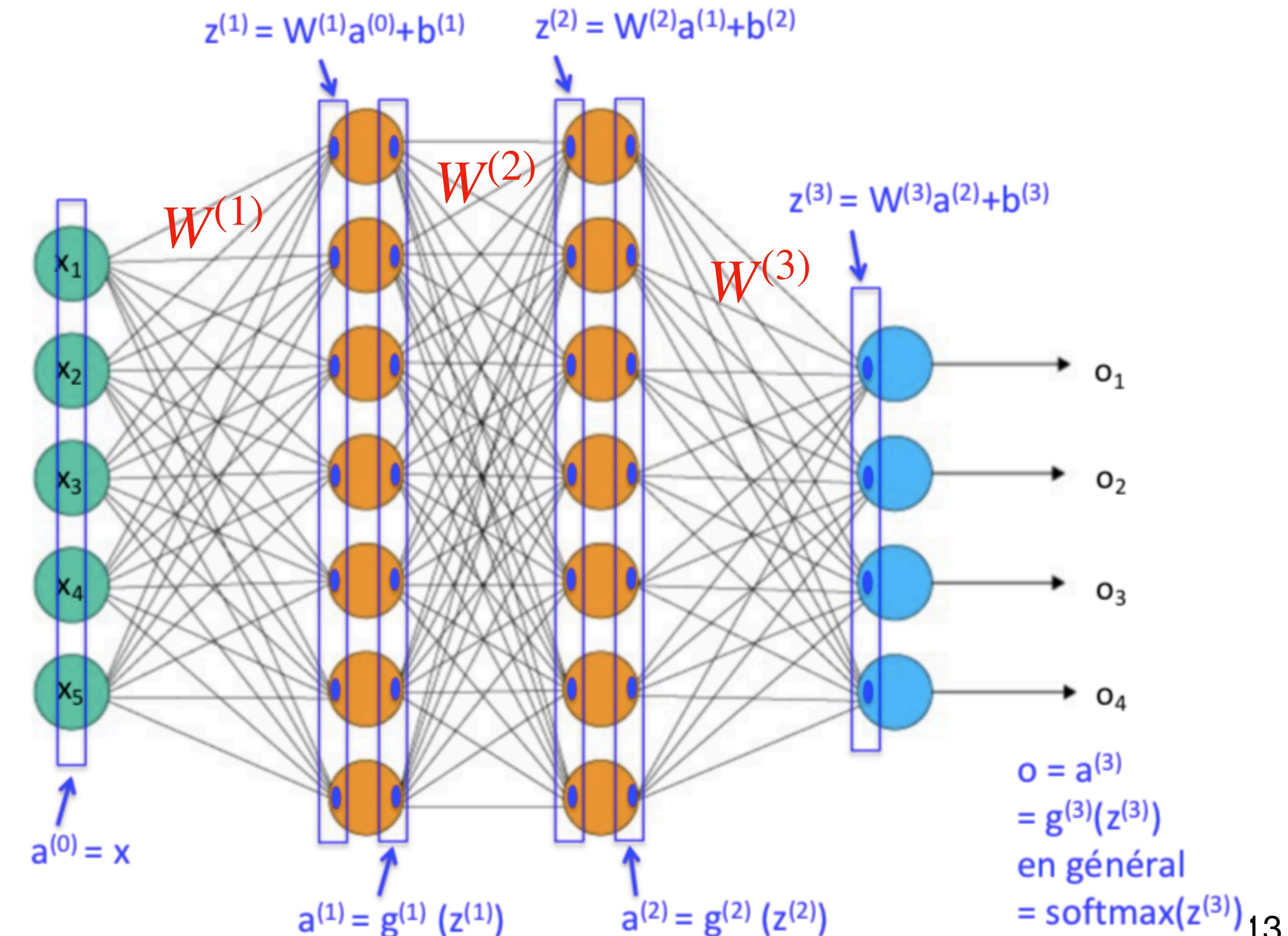
Calcul des valeurs de sortie

= « propagation avant »

= « forward propagation »

(En ignorant les biais)

- NB: un réseau de neurones de type MLP est une fonction $\mathbb{R}^d \rightarrow \mathbb{R}^C$



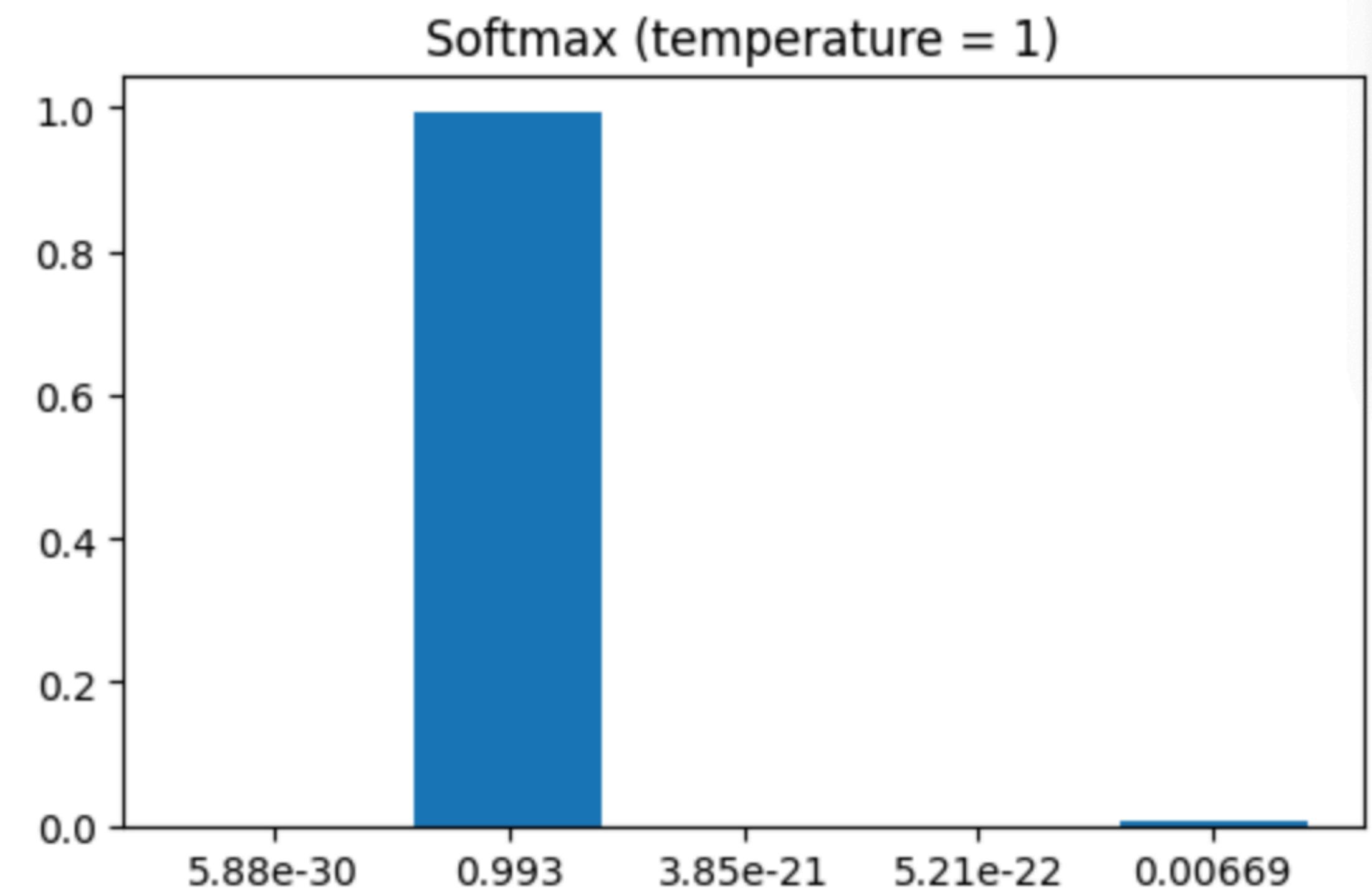
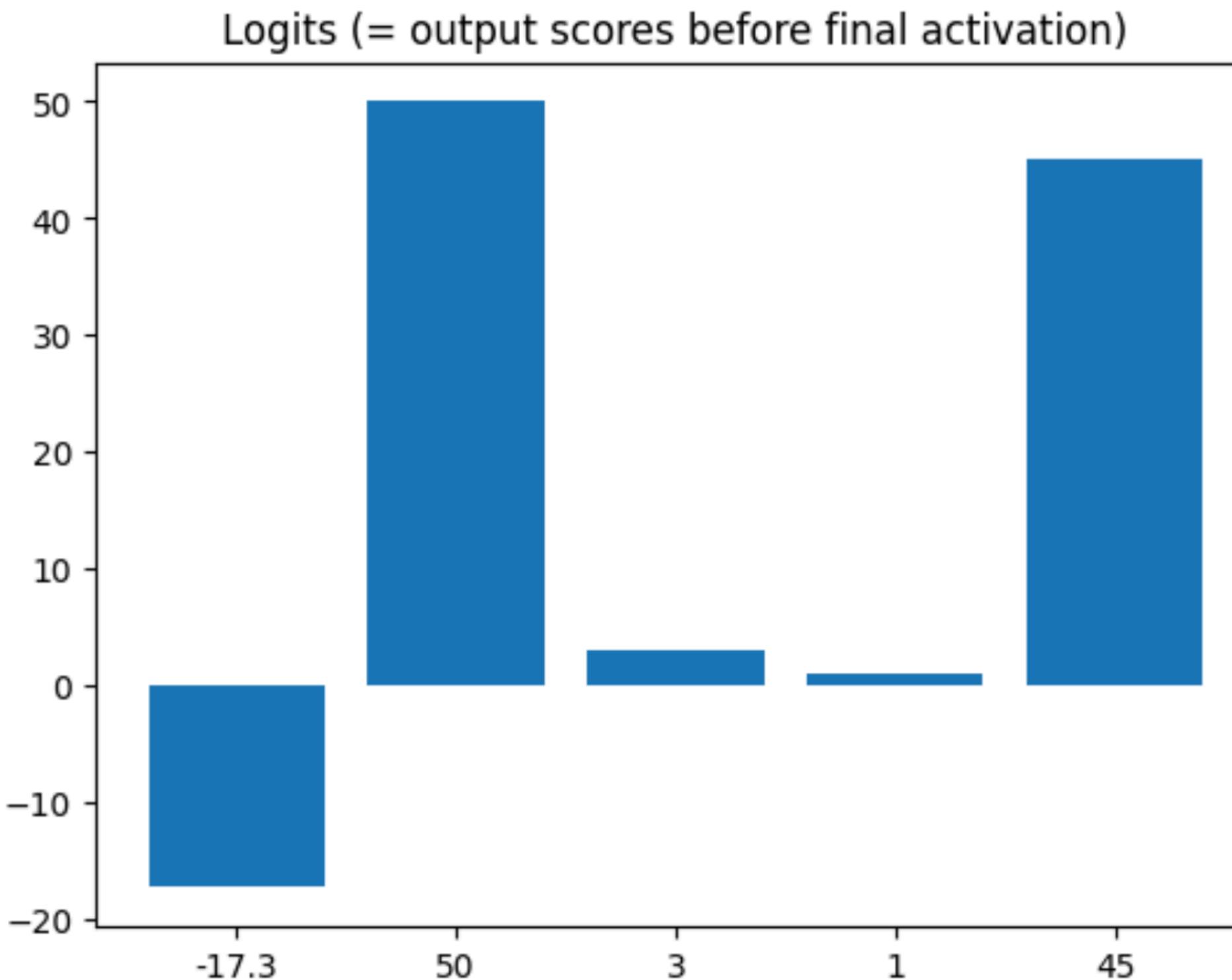
Utilisation d'un MLP comme classifieur

- Réseau avec en sortie un neurone par classe
- Vecteur de sortie o interprété comme vecteur de scores de C classes
- Classe prédite = la classe de plus haut scores : $\hat{y} = \underset{c \in \mathcal{C}}{\operatorname{argmax}} o_c$

Scores probabilistes : activation « softmax »

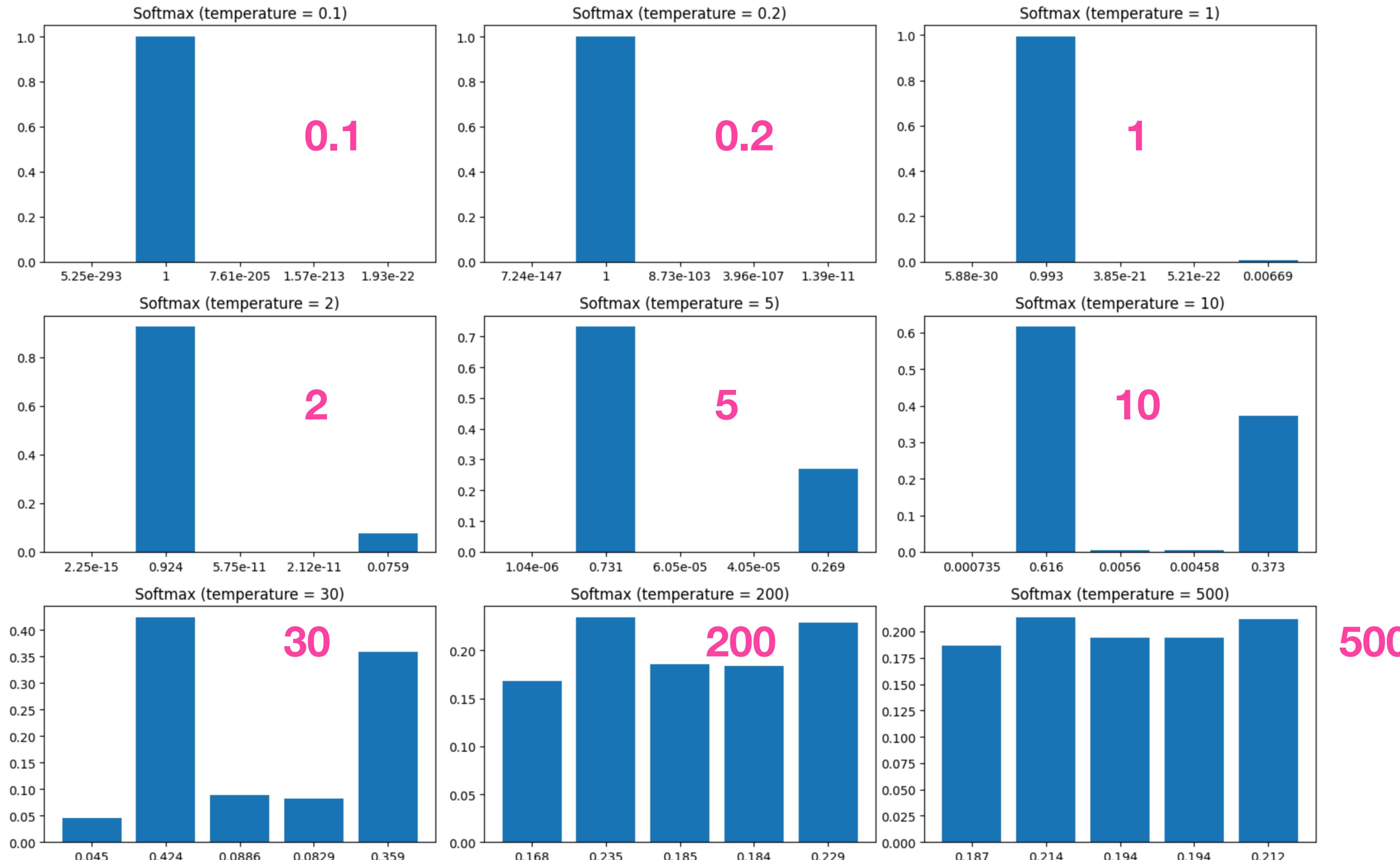
Transforme les scores avant activation finale $z_1 z_2 \dots z_C$ (les « logits ») en distribution de probabilités

$$o_j = (\text{softmax}(z))_j = \frac{e^{z_j/T}}{\sum_{k=1}^C e^{z_k/T}}$$



Softmax : effet de la température

L'entropie de la distribution augmente avec la température T

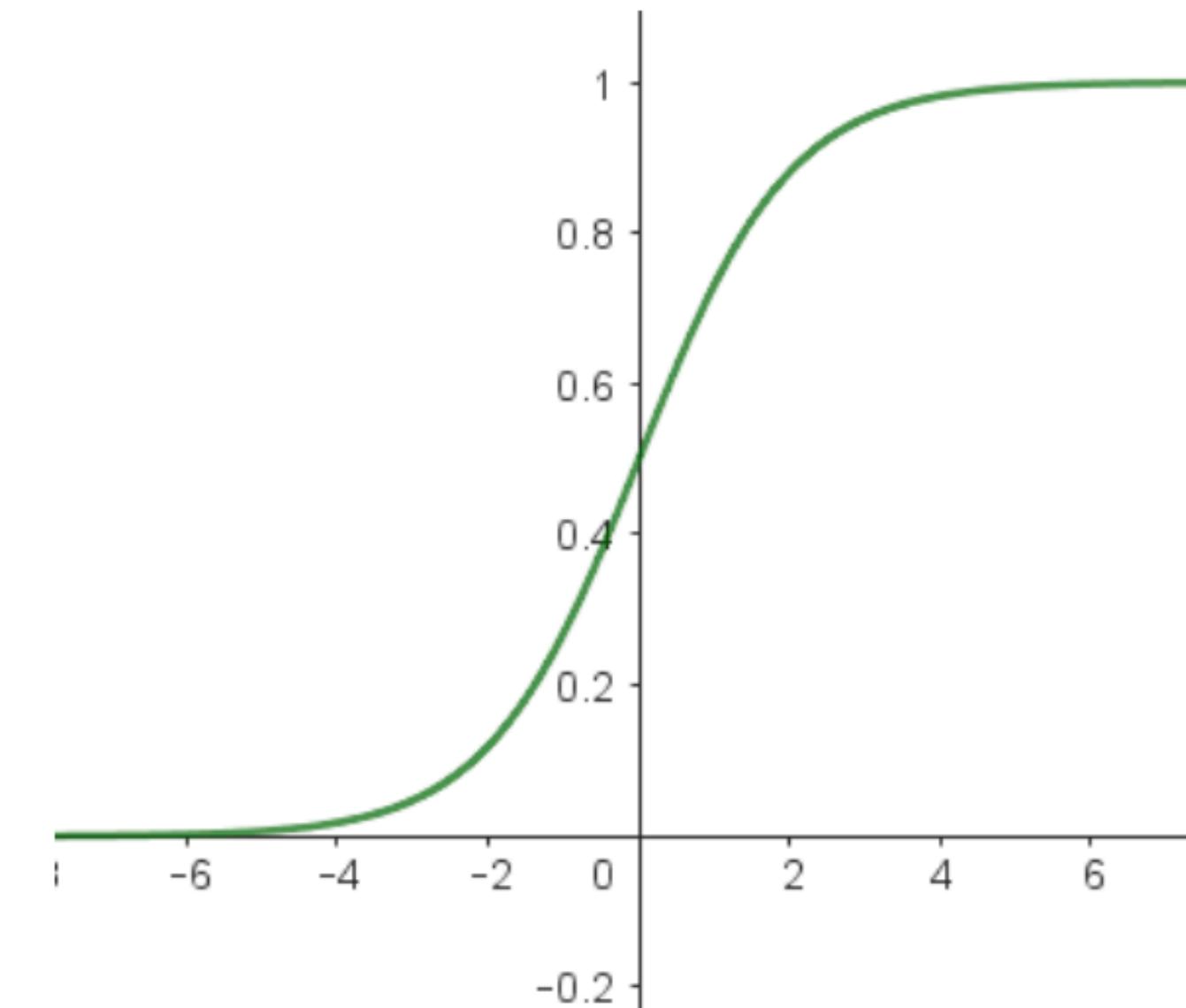


Réseau pour classification binaire

- Possible astuce en cas de classification binaire
 - 2 classes mappées vers 0 et 1 (ou bien -1 et 1)
 - => on utilise un seul neurone de sortie, fournissant le score de la classe +1
 - activation sigmoïde en sortie pour obtenir un score probabiliste

$$o = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

- sortie o interprétée comme $P(\text{classe} = 1 | x)$



Théorème d'approximation universelle

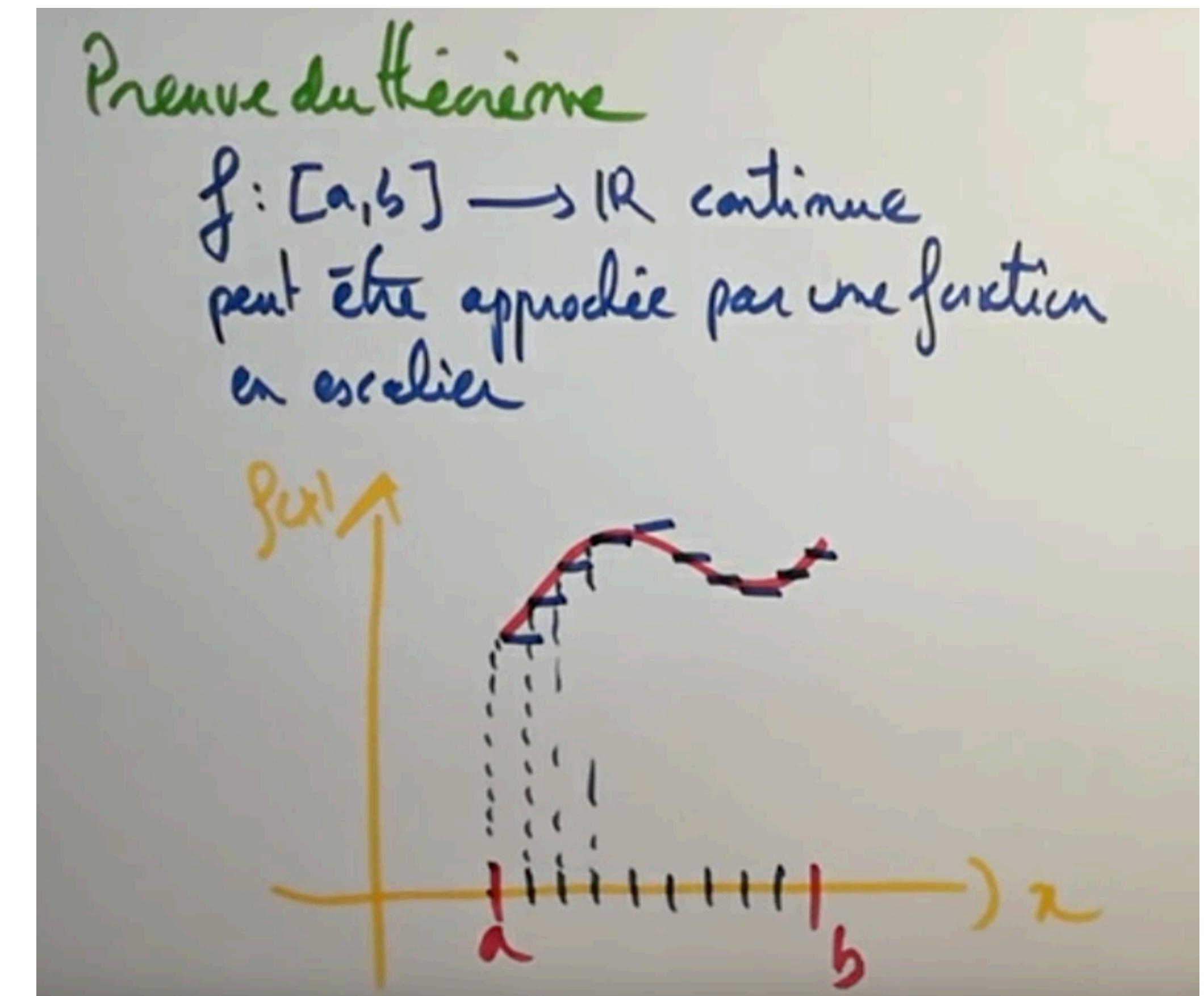
- Pour toute fonction continue $f : I^d \rightarrow \mathbb{R}$ avec I un intervalle de \mathbb{R} ,
- \exists un MLP à une seule couche et activation non linéaire approximant f
Version simplifiée (Cybenko, 89; Hornik, 91...)
- Vaut également pour $f : I^d \rightarrow \mathbb{R}^C$

Théorème d'approximation universelle

- Pour toute fonction continue $f: I^d \rightarrow \mathbb{R}$ avec I un intervalle de \mathbb{R} ,
- \exists un MLP à une seule couche et activation non linéaire approximant f

Version simplifiée (Cybenko, 89; Hornik, 91...)

- Vaut également pour $f: I^d \rightarrow \mathbb{R}^C$



Théorème d'approximation universelle

- Pour toute fonction continue $f: I^d \rightarrow \mathbb{R}$ avec I un intervalle de \mathbb{R} , il existe un MLP à une seule couche et activation non linéaire approximant f
Version simplifiée (Cybenko, 89; Hornik, 91...)
- NB:
 - Ne donne pas la taille de la couche cachée
 - Ne garantit pas l'apprenabilité
- Mais reste extrêmement puissant
 - Contrairement aux classifieurs linéaires, la frontière de décision n'a pas à être un hyperplan

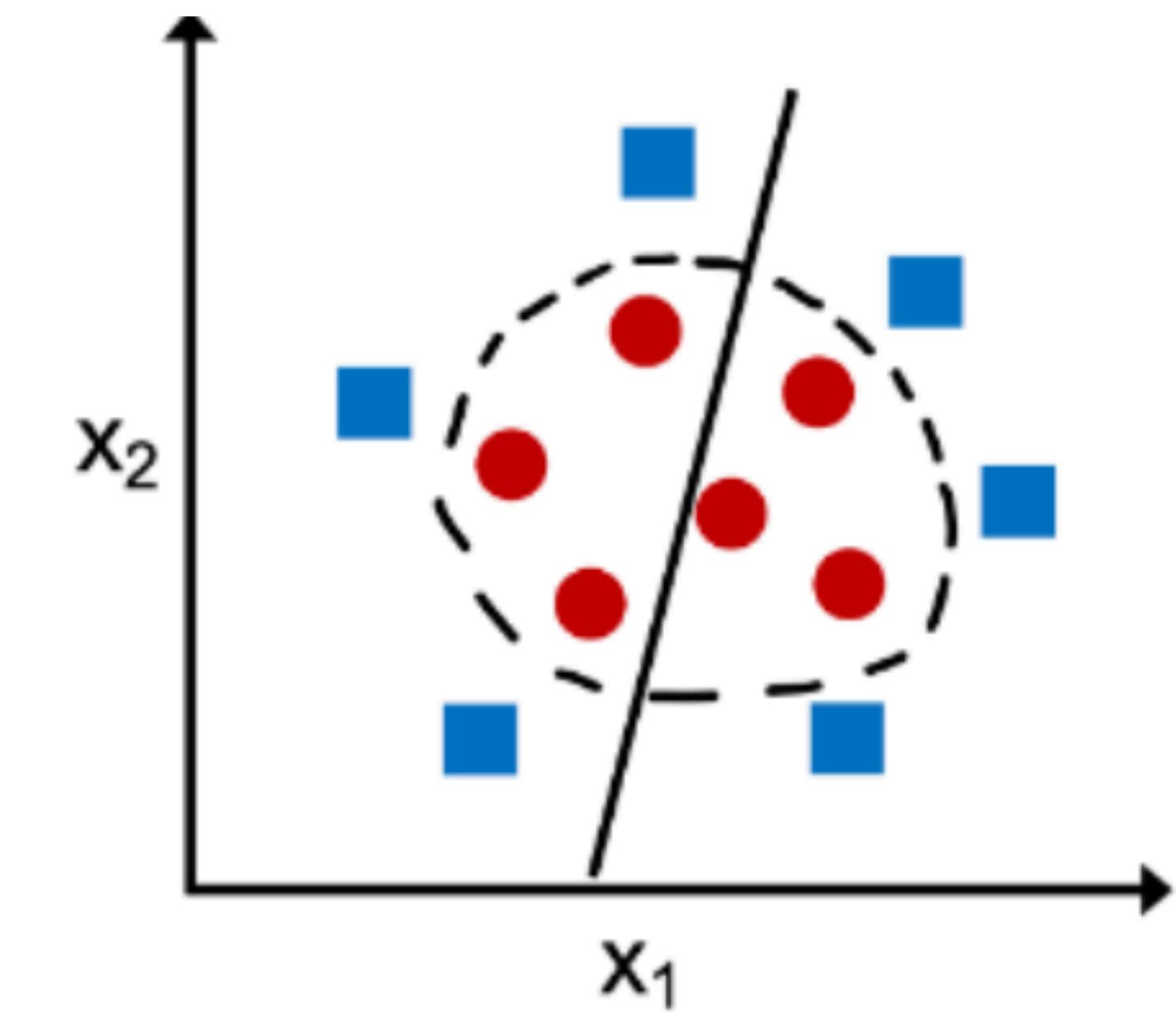
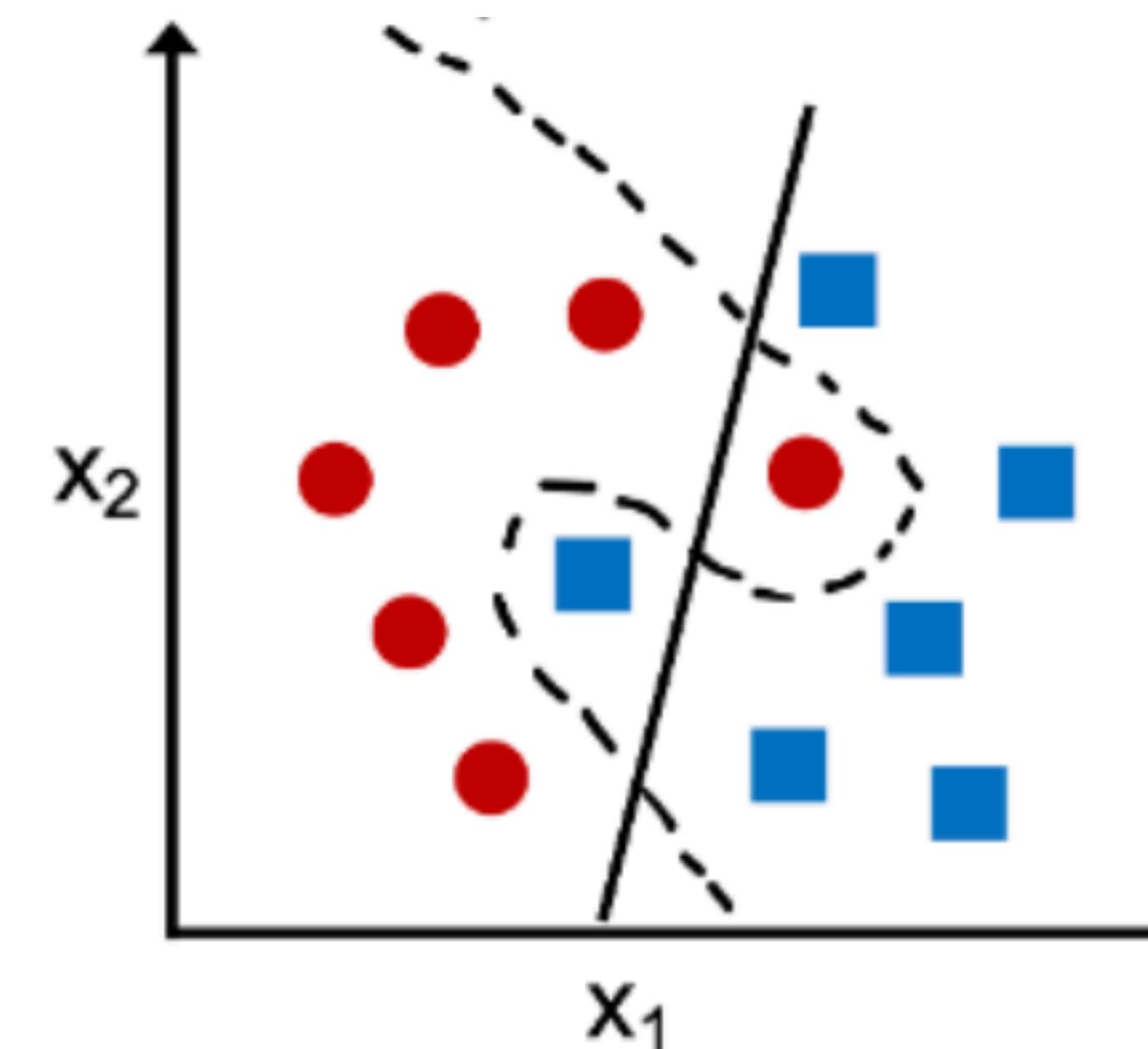
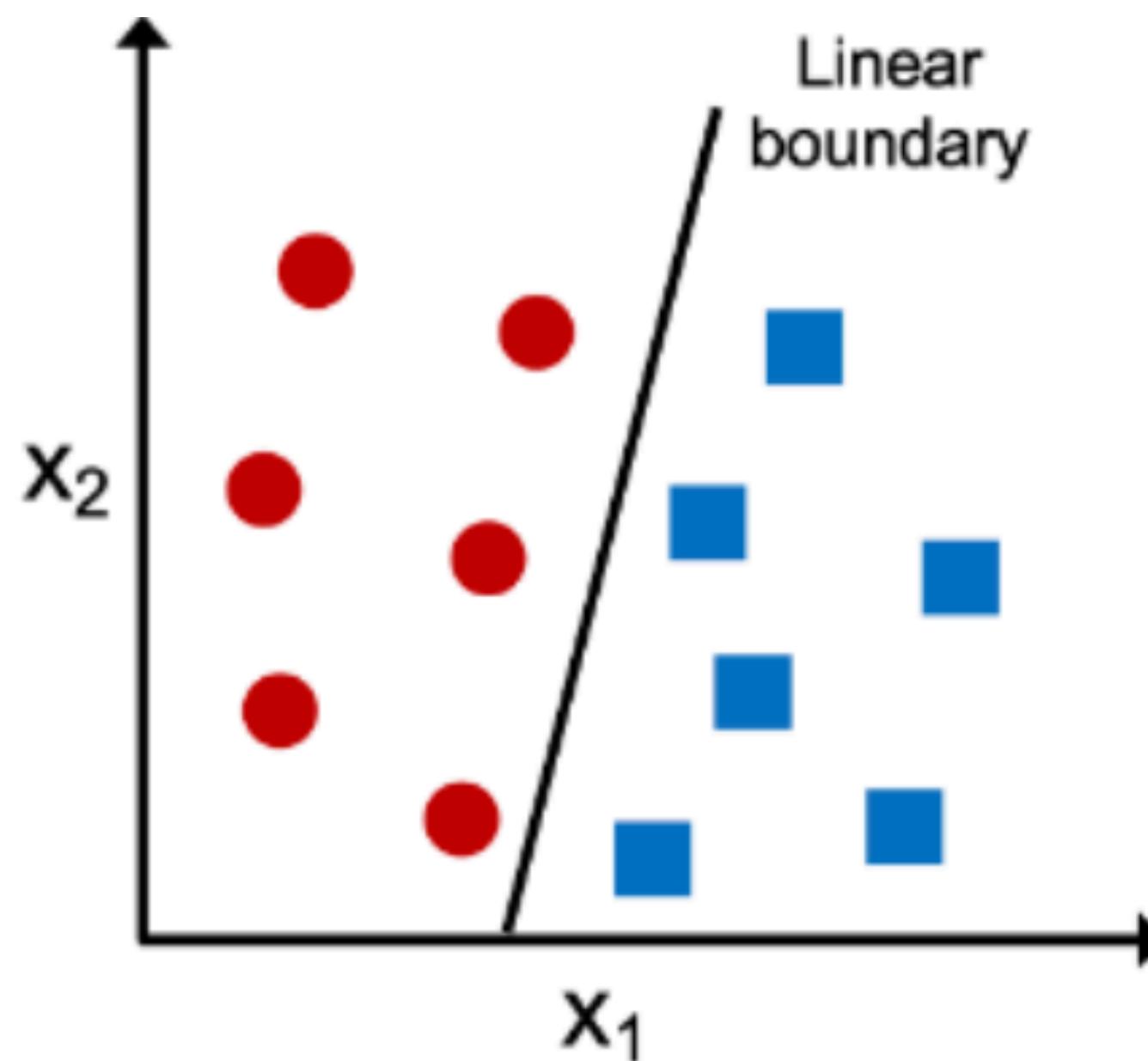
Frontières de décision linéaires vs. non linéaires

Exemple : classifieur binaire

Fonction de score $f: \mathbb{R}^2 \rightarrow \mathbb{R}$

Décision : classe ● si $f(x_1, x_2) \geq 0$ et classe ■ sinon

La **frontière de décision** est l'ensemble des points tels que $f(x_1, x_2) = 0$

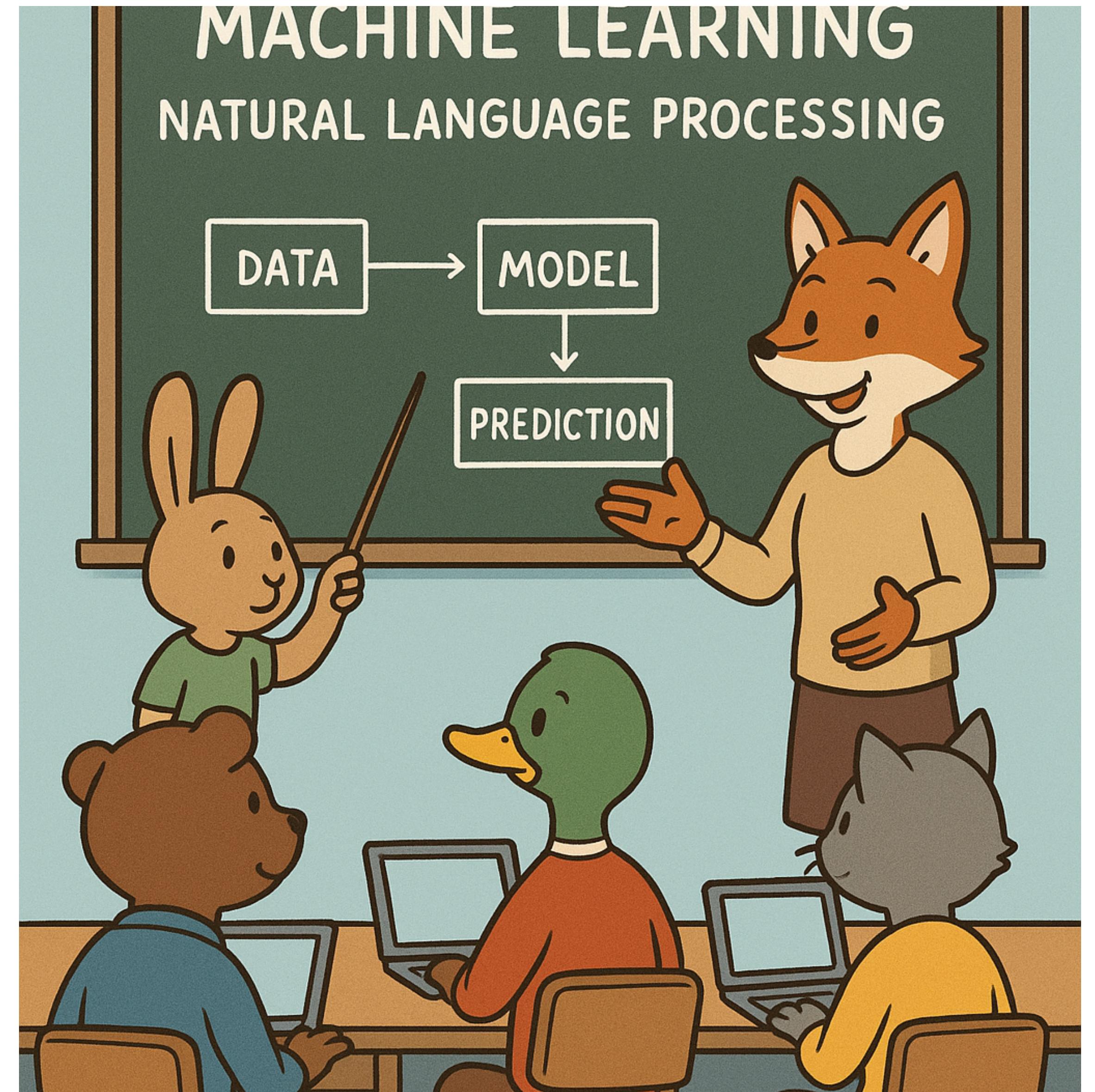


Résumé jusqu'ici

- Un réseau de neurones de type MLP permet d'assigner un **score** à une ou plusieurs classes, étant donné un objet à classer représenté par un vecteur
 - C'est simplement une fonction $f: \mathbb{R}^d \longrightarrow \mathbb{R}^C$
 - Ou $f: \mathbb{R}^d \longrightarrow [0,1]^C$ si scores probabilistes
- On peut « **prédir** » la classe d'un objet simplement en choisissant la classe de plus haut score
- Même si 2 objets à vecteurs proches $x^{(1)}$ et $x^{(2)}$ sont de classes distinctes, il existera toujours un MLP permettant de donner des scores différents pour $x^{(1)}$ et $x^{(2)}$
- Cette puissance expressive provient de la ou des **fonctions d'activation non linéaires** utilisées dans les couches cachées

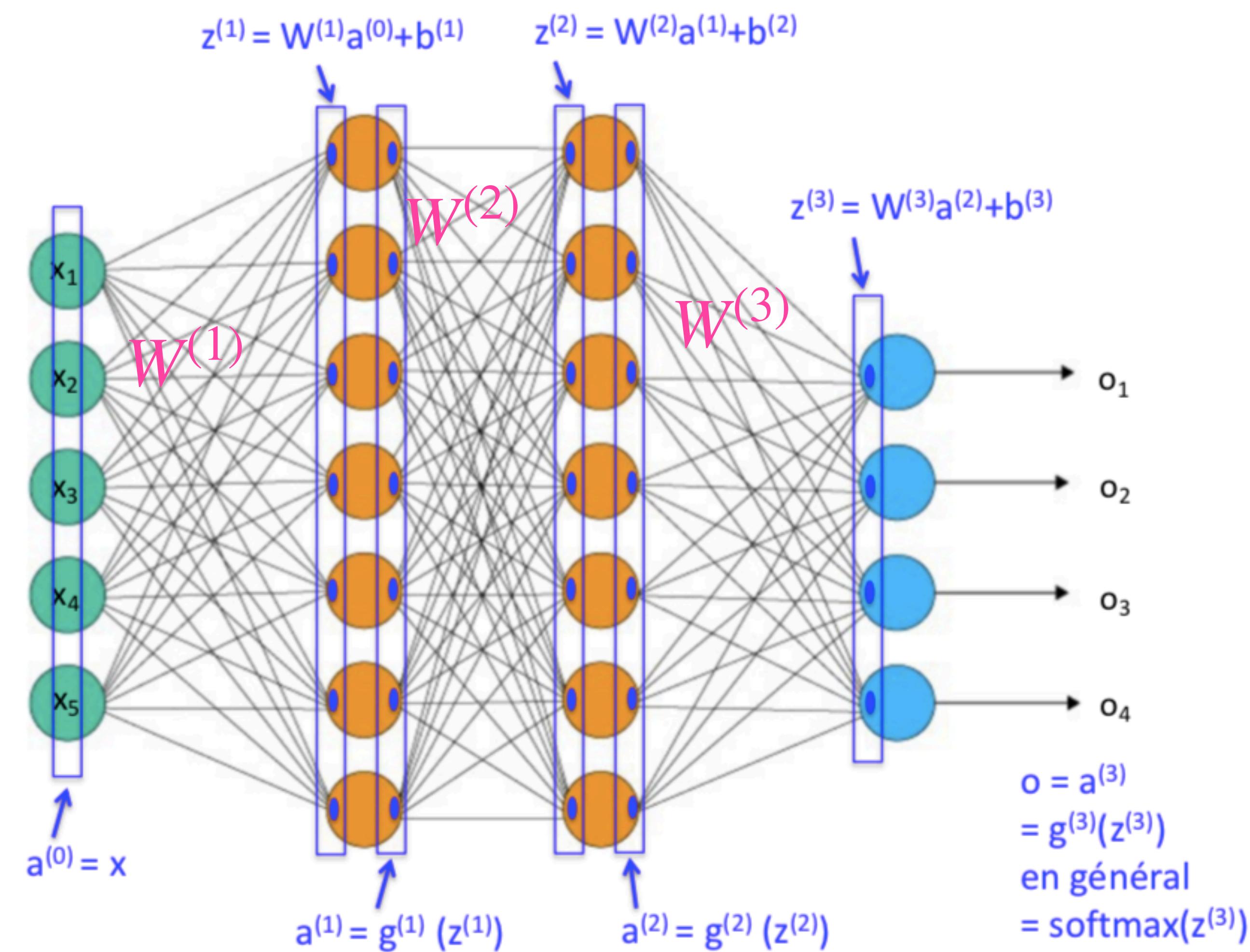
Notions d'apprentissage supervisé

Ici pour la classification
multiclasse



Apprentissage des paramètres

- Une fois fixée l'architecture du réseau
 - Nb et taille des couches cachées
 - Fonctions d'activation
- Pour pouvoir prédire la classe d'objets on doit connaître les **paramètres θ**
 - Ici les matrices $W^{(1)}, W^{(2)}, W^{(3)}$
 - Et vecteurs de biais $b^{(1)}, b^{(2)}, b^{(3)}$



Apprentissage supervisé: exemples d'apprentissage

- Apprentissage supervisé pour la classification dans C classes: on dispose d'exemples classés
 - Ensemble d'apprentissage $\mathcal{T} = \{(x^{(1)}, y^{(1)}), (x^{(1)}, y^{(1)}), \dots, (x^{(T)}, y^{(T)})\}$
 - $x^{(t)}$ vecteur représentant un objet à classer,
 - $y^{(t)}$ sa classe « gold » (nb de 1 à C)
 - Notons
 - $f: \mathbb{R}^d \times \mathbb{R}^n \longrightarrow \mathbb{R}^C$ la fonction de scores, intégrant les paramètres $\theta \in \mathbb{R}^n$
 - $f(x, \theta)$ est le vecteur de scores des C classes, pour l'objet x et les paramètres θ

Apprentissage supervisé: fonction de perte

- On veut que les classes gold reçoivent le score le plus haut
- Notion de **marge** de classification : écart entre le score de la classe gold et le plus haut score hors classe gold
 - $\text{marge}(f(x, \theta), y) > 0 \iff \text{exemple bien classé} \iff \text{la classe gold y reçoit le score maximal}$
- On numérise la notion d'erreur vs succès de classification via une fonction de **perte**
- Perte d'autant plus petite/grande que la marge est grande/petite
 - Exemple : perte de Hinge

$$l^{hinge}(f(x, \theta), y) = \max(0, 1 - \text{marge}(f(x, \theta), y))$$

- **Tracez la courbe de la perte en fonction de la marge**
- **À quelle condition la perte est-elle nulle?**

Apprentissage supervisé: perte entropie croisée

= cross-entropy

= log-vraisemblance négative

= negative log-likelihood NLL

- Utilisée si les scores sont probabilistes
- score de la classe c : $f(x, \theta)_c = P(\text{classe} = c \mid \text{objet} = x)$
- $l^{NLL}(f(x, \theta), y) = -\log(f(x, \theta)_y) = -\log(P(\text{classe} = y \mid \text{objet} = x))$
- = -log du score probabiliste de la classe gold
- L'objectif de minimiser la NLL correspond à maximiser la proba de la classe gold

Apprentissage supervisé : principe général

Minimisation de la perte

- Fonction de scores intégrant les paramètres $f: \mathbb{R}^d \times \mathbb{R}^n \longrightarrow \mathbb{R}^C$
- $f(x, \theta)$ est le vecteur de scores des C classes pour l'objet x et les paramètres $(\theta \in \mathbb{R}^n)$
- Objectif : trouver les valeurs des paramètres minimisant la perte l sur l'ensemble d'apprentissage

$$\theta^* = \underset{\theta \in \mathbb{R}^n}{argmin} \sum_{t=1}^T l(f(x^{(t)}, \theta), y^{(t)})$$

Apprentissage supervisé: descente de gradient

Algorithme itératif, sur paquets (batchs) d'exemples

- **Input** : fonction f , pas d'apprentissage η , ensemble d'apprentissage \mathcal{T} , taille des batchs k , nb d'époques E
 - **Initialiser** les paramètres θ au hasard
 - **Boucler** sur E époques
 - Mélanger les exemples dans \mathcal{T}
 - **Boucler** sur chaque paquet B (« batch ») de k exemples
 - **Forward**: Calculer la perte sur les exemples de B : $L(f, \theta, B) = \sum_{(x,y) \in B} l(f(x, \theta), y)$
 - **Backpropagation** : Calcul efficace de la mise à jour g à opérer :
 - le **gradient** de la perte par rapport aux paramètres $g = \nabla_{\theta} L(f, \theta, B)$
 - **Mettre à jour** : $\theta \leftarrow \theta - \eta g$
 - Retourner θ

Le gradient c'est koi

- Vecteur des dérivées partielles par rapport à chaque paramètre individuel

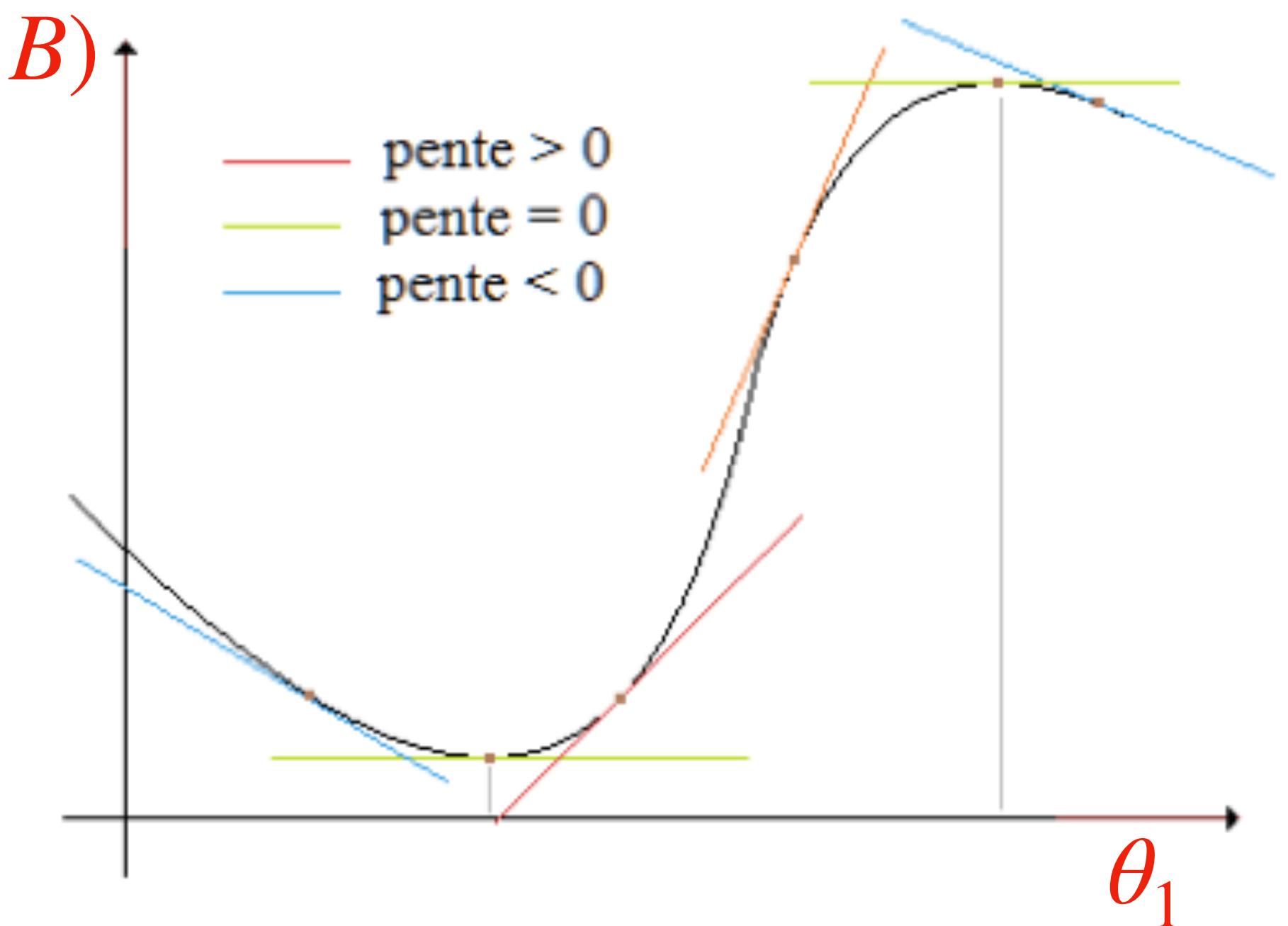
$$\nabla_{\theta} L(f, \theta, B) = \left(\frac{\partial L}{\partial \theta_1}(f, \theta, B), \dots, \frac{\partial L}{\partial \theta_n}(f, \theta, B) \right)$$

- Chaque paramètre est mis à jour dans le sens opposé à la dérivée partielle de la perte par rapport à ce paramètre

$$\theta_i \leftarrow \theta_i - \eta \frac{\partial L}{\partial \theta_i}(f, \theta, B)$$

Intuition du sens de mise à jour

- Simplifions dans d'une fonction $L(f, \theta_1, B)$ à un seul paramètre θ_1
- Pour un f et un batch B fixés, la dérivée partielle $\frac{\partial L}{\partial \theta_1}(f, \theta_1, B)$ est la **pente** de la tangente à la courbe de L en fonction de θ_1
- Modifier θ_1 dans le sens opposé à la pente va diminuer L
 - ... à moins que l'on passe le minimum...



Méthodologie minimale : validation et test

- Si suffisamment de données
- Découpage de \mathcal{T} en 3 ensembles **train / dev=validation / test**
- Apprentissage sur « **train** »
- Réglage des hyperparamètres d'après performance sur « **dev** »
 - En particulier sortie prématurée de la boucle des époques (« **early stopping** »)
- Evaluation finale sur « **test** »
 - Sur exemples pas du tout utilisés
 - Mesurant le niveau de généralisation

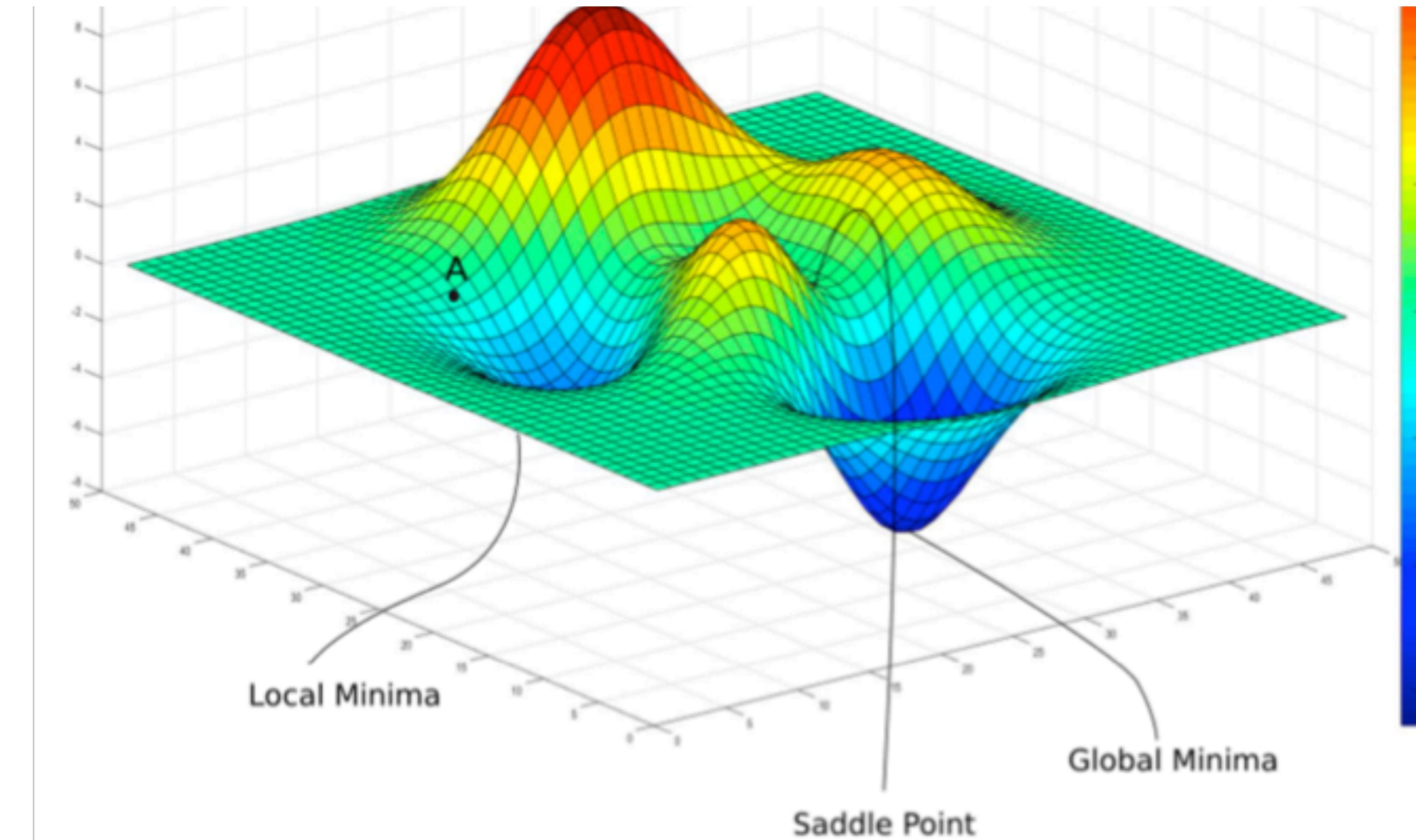
Apprentissage supervisé: descente de gradient

Avec early stopping

- **Input** : fonction f , pas d'apprentissage η , ensemble d'apprentissage \mathcal{T} , taille des batchs k , nb d'époques E
 - **Initialiser** les paramètres θ au hasard
 - Boucler sur E époques
 - Mélanger les exemples dans \mathcal{T}
 - **Boucler** sur chaque paquet B (« batch ») de k exemples
 - **Forward** : Calculer la perte sur les exemples de B
 - **Backpropagation** : Calcul du **gradient** $g = \nabla_{\theta}L(f, \theta, B)$
 - **Mettre à jour** : $\theta \leftarrow \theta - \eta g$
 - **Si perte sur ensemble de dev a augmenté par rapport à époque précédente**
 - **Stop**
 - Retourner θ

Descente de gradient : convergence?

- Si η suffisamment petit, la descente de gradient converge vers un minimum local



<https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/>

- Si L est une fonction **convexe**, le minimum local est aussi global

Descente de gradient : convergence?

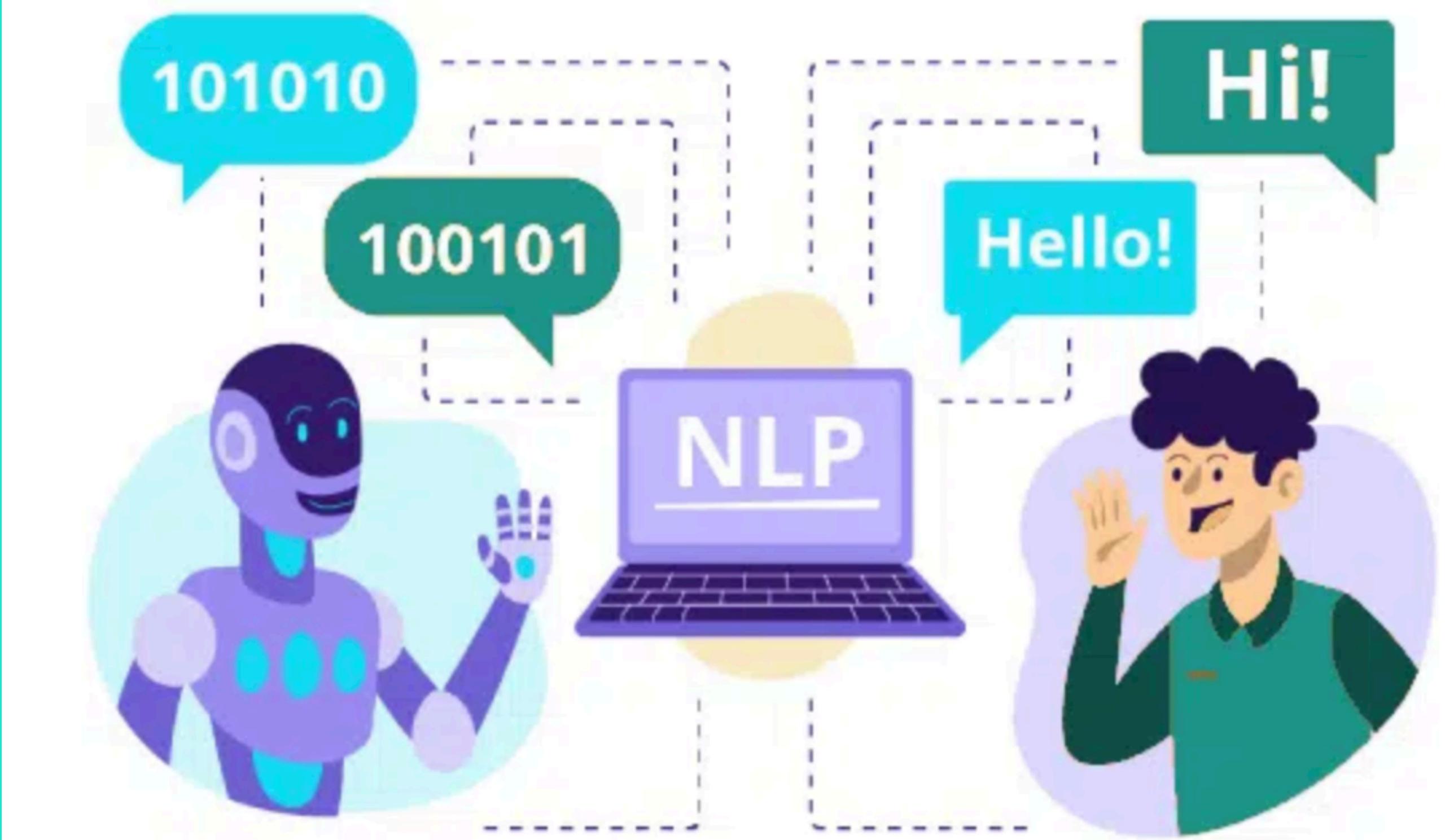
- Avec un MLP + perte NLL : L n'est pas convexe
 - Mais résultats empiriques et théoriques ([Choromanska et al. 2015](#))
 - « *For large-size networks, most local minima are equivalent and yield similar performance on a test set.* »
 - (Tests avec 1 couche cachée, nb neurones 250, 500)
 - « *The probability of finding a “bad” (high value) local minimum is non-zero for small-size networks and decreases quickly with network size.* »

Backpropagation

Calcul efficace du gradient par programmation dynamique

- Voir TP pour ceux intéressés

Et le TAL dans tout ça?



<https://ledigitalizeur.fr/ia/traitement-langage-naturel/>

Représentations de mots et de phrases

Avant...: représentation one-hot et bag-of-word

- On dispose d'un **vocabulaire** de mots (ou symboles), associés à un **rang**
- Créable par ex. à partir d'un corpus

```
documents = [ 'First, this is the first sentence.' ,  
              'And this is the second .' ,  
              'Is this is the third sentence?' ]
```



```
[ 'and' 'first' 'is' 'second' 'sentence' 'the' 'third' 'this' ]
```

Représentations de mots et de phrases

Avant...: représentation one-hot et bag-of-word

- Chaque **mot** est représentable par un **vecteur « one-hot »** : un seul 1

Vocabulaire

[‘and’ ‘first’ ‘is’ ‘second’ ‘sentence’ ‘the’ ‘third’ ‘this’]

Vecteur one-hot

“is” —————→ [0, 0, 1, 0, 0, 0, 0, 0]

Représentations de mots et de phrases

Avant...: représentation one-hot et bag-of-word

- Pour une séquence de mots : on somme les one-hot de chaque occurrence

Vocabulaire

['and' 'first' 'is' 'second' 'sentence' 'the' 'third' 'this']

Vecteurs

« **Bag-of-Word** »

« **BOW** »

First, this is the first sentence

[0, 2, 1, 0, 1, 1, 0, 1]

"and" appears zero times

"first" appears twice

And this is the second

[1, 0, 1, 1, 0, 1, 0, 1]

"and" appears once

"first", "sentence" , "third"
appears zero times

Représentations de mots et de phrases

Avant...: comptes de co-occurrence

Corpus:

- Apples are green and red.
- Red apples are sweet.
- Green oranges are sour.

Matrice de co-occurrence de mots
au sein d'une fenêtre
de taille fixe autour du mot
ou au sein de la même phrase

	apples	are	green	and	red	sweet	oranges	sour
apples	2	2	1	1	2	1	0	0
are	2	3	1	1	2	1	1	1
green	1	1	2	1	1	0	1	1
and	1	1	1	1	1	0	0	0
red	2	2	1	1	2	1	0	0
sweet	1	1	0	0	1	1	0	0
oranges	0	1	1	0	0	0	1	1
sour	0	1	1	0	0	0	1	1

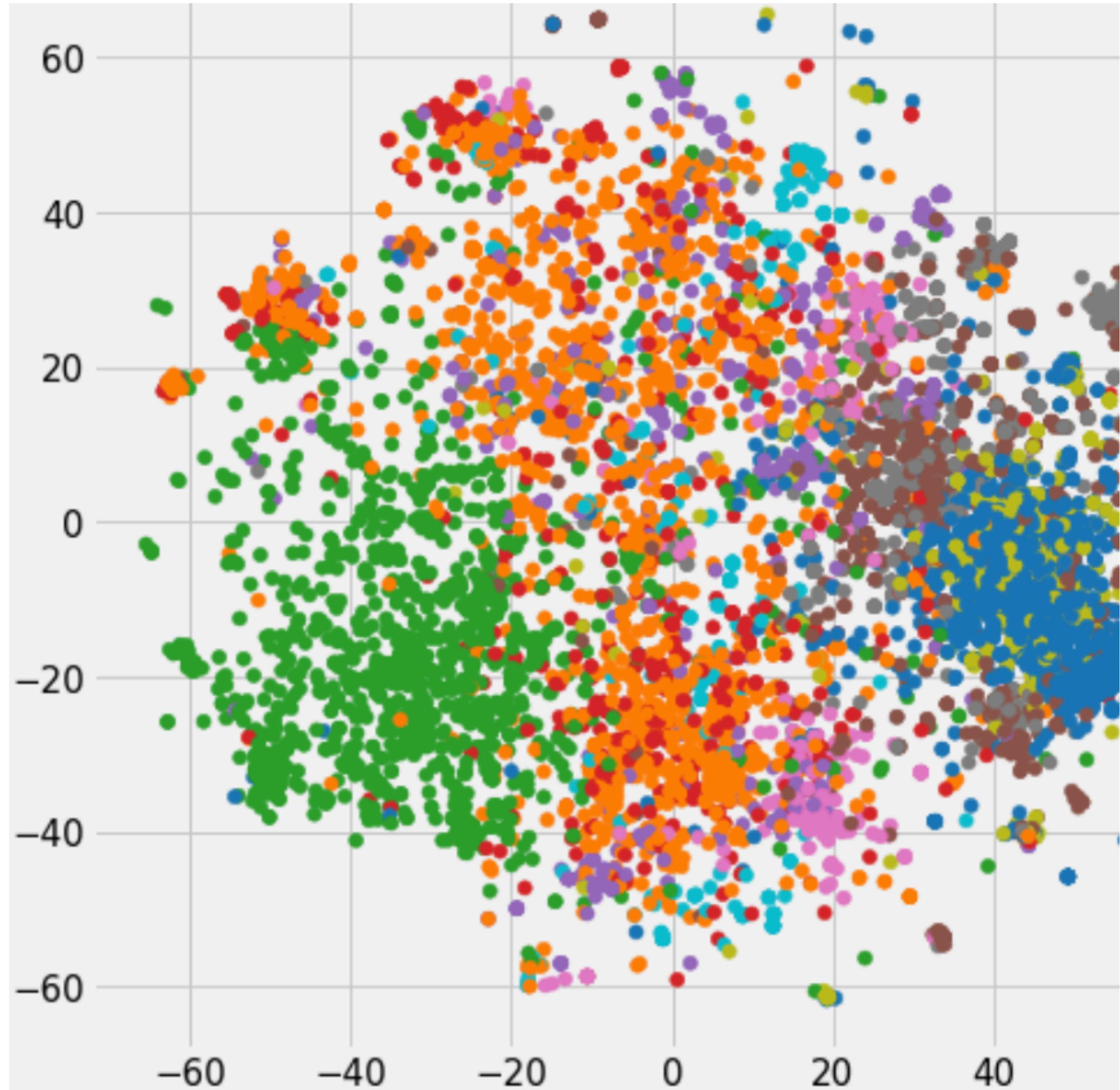
Représentations de mots et de phrases

Avant...: inconvénients

- Représentations très creuses
 - bcp de mots
 - peu de co-occurrences
- Aucune similarité entre mots
 - similarité non nulle entre BOW uniquement si mots exacts en commun
- ⇒ depuis la fin des années 2000 : apprentissage de vecteurs de mots denses, comme paramètres de réseaux neuronaux

Vecteurs de mots statiques, denses, obtenus par apprentissage

- Historique: modèle de langue neuronal
- Word2vec
- Apprentissage par transfert



Modèle de langue neuronal

Bengio et al. 2003

- « **Modèle de langue** » = associer une probabilité à une séquence de mots

- Règle de multiplication (chain rule)

$$P(t_1 t_2 t_3 \dots t_n) = P(t_1) P(t_2 | t_1) P(t_3 | t_1 t_2) P(t_4 | t_1 t_2 t_3) \dots P(t_n | t_1 \dots t_{n-2} t_{n-1})$$

- Eventuelle hypothèse d'indépendance (Markov)

$$P(t_1 t_2 t_3 \dots t_n) = P(t_1) P(t_2 | \textcolor{blue}{t}_1) P(t_3 | \textcolor{blue}{t}_1 \textcolor{blue}{t}_2) P(t_4 | \textcolor{red}{X} \textcolor{blue}{t}_2 \textcolor{blue}{t}_3) \dots P(t_n | \textcolor{red}{X} \cdot \textcolor{red}{X} \cdot \textcolor{blue}{t}_{n-2} t_{n-1})$$

- Acceptation plus restreinte de la tâche de modèle de langue ou « language modeling » :

- **Le problème est ramené à estimer des probabilités** $P(t_i | t_{i-k} \dots t_{i-1})$
- par ex. distribution de probabilités $P(\cdot | \textit{petit chat boit du})$

Modèle de langue neuronal

Bengio et al. 2003 : 3 idées fondamentales

1. Estimer $P(\cdot | k \text{ mots de contexte})$ via un réseau neuronal

- classifieur
- dont les « classes » sont les mots du vocabulaire
- entrée = k mots de contexte

Modèle de langue neuronal

Bengio et al. 2003 : 3 idées fondamentales

2. Obtenir trivialement des exemples d'apprentissage à partir de textes segmentés

- on parle d'apprentissage **auto-supervisé** (self-supervised)

... On a bien le droit d'échouer. De tenter seulement de faire un peu de lumière et des ombres, comme la lampe dans l'escalier, et de s'éteindre ensuite sans bruit...



$$\begin{aligned}x^{(1)} &= (*d^*, *d^*, *d^*) \\x^{(2)} &= (*d^*, *d^*, \text{On}) \\x^{(3)} &= (*d^*, \text{On}, a) \\x^{(4)} &= (\text{On}, a, \text{bien}) \\x^{(5)} &= (a, \text{bien}, \text{le}) \\x^{(6)} &= \dots\end{aligned}$$

$$\begin{aligned}y^{(1)} &= \text{On} \\y^{(2)} &= a \\y^{(3)} &= \text{bien} \\y^{(4)} &= \text{le} \\y^{(5)} &= \text{droit}\end{aligned}$$

Modèle de langue neuronal

Bengio et al. 2003 : 3 idées fondamentales

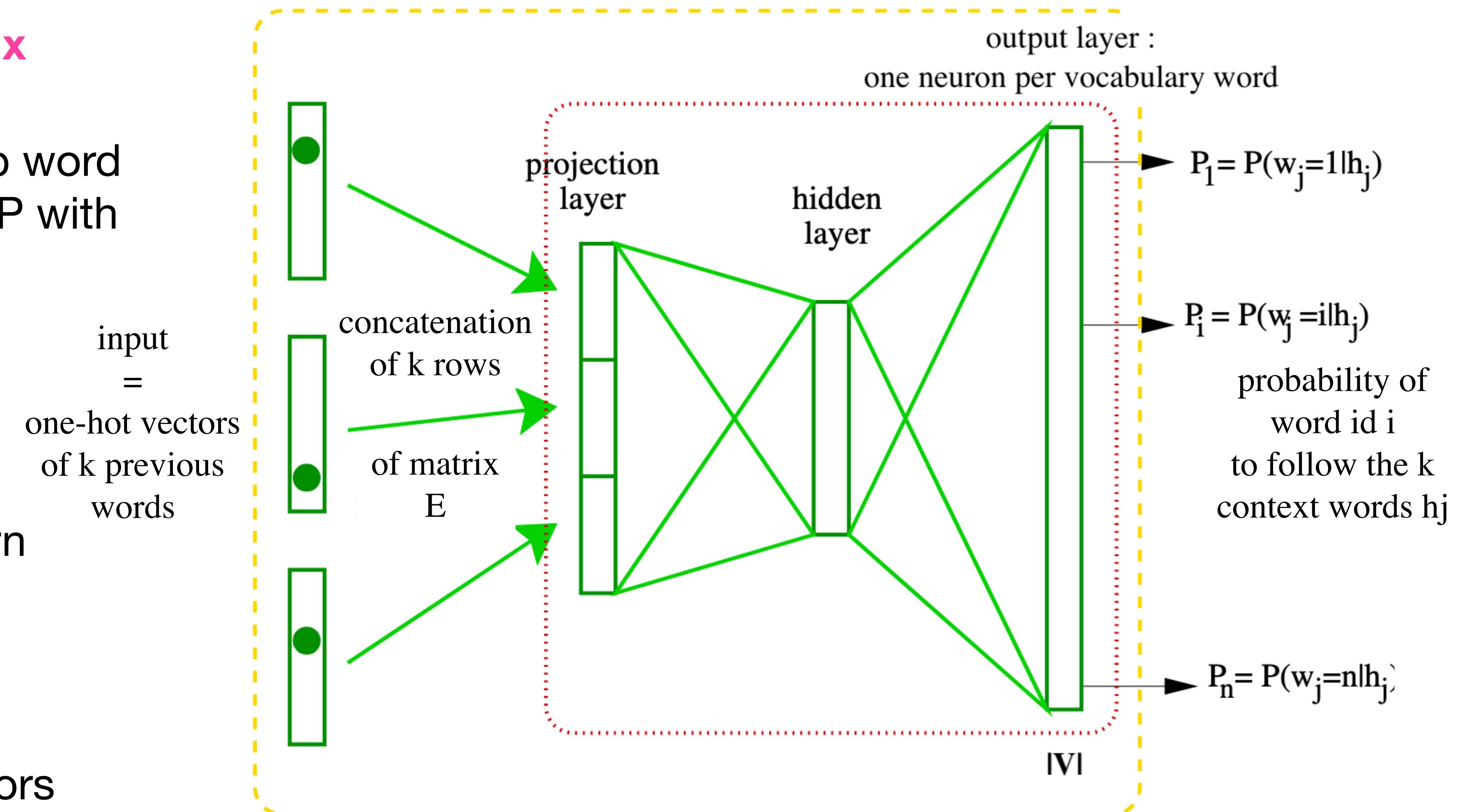
3. Transformer chaque mot de contexte en un vecteur dense

- ces vecteurs étant appris comme paramètres du réseau
- l'apprentissage du réseau donne comme **produit dérivé une matrice de vecteurs de mots**
 - denses, et courts (typiquement taille ~100 / 1000)

Modèle de langue neuronal

Bengio et al. 2003 - architecture simplifiée

- **Embedding matrix**
- $E \in \mathbb{R}^{|V| \times e}$
- one row per vocab word
- Then classical MLP with input of size $k \times e$
- parameters to learn
 - $E \in \mathbb{R}^{|V| \times e}$
 - $W^{(1)} \in \mathbb{R}^{h \times ke}$
 - $W^{(2)} \in \mathbb{R}^{|V| \times h}$
 - and 2 bias vectors



Word2vec

Mikolov et al. 2013

- très célèbre
- simplification (réseau sans non-linéarité)
- avec focus sur le produit dérivé (matrice d'embeddings) et pas sur la tâche apprise
- d'où 2 tâches bizarres
 - **CBOW**: prédire un mot sachant son **contexte gauche et droit**
 - cf. pas d'exigence de calculer la probabilité complète d'une séquence de mots
 - **SKIP-GRAM** : prédire un mot de contexte sachant un mot cible

Word2vec : modèle CBOW

Exemples d'apprentissage

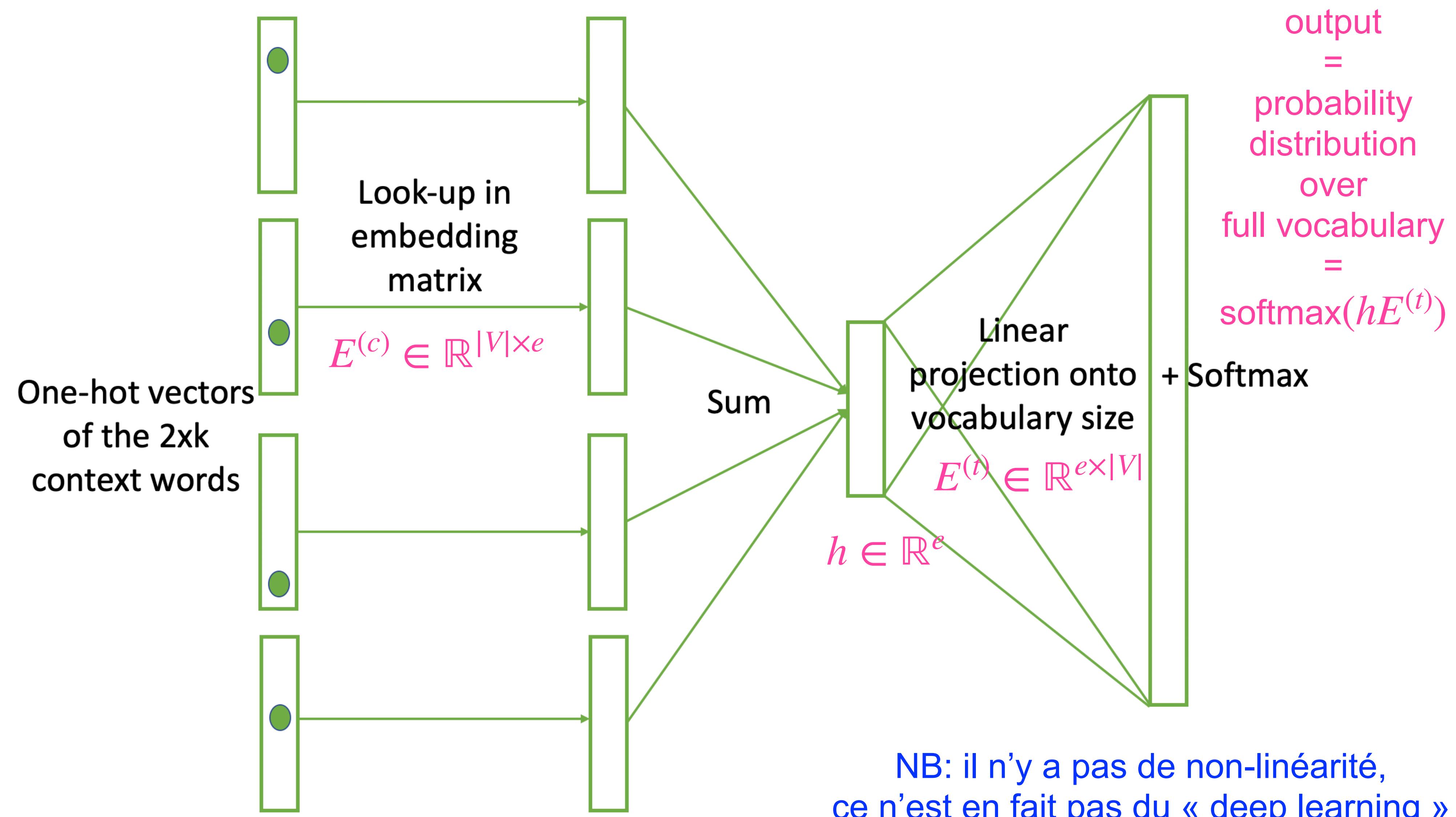
- Apprentissage auto-supervisé : création triviale des exemples
 - pour chaque occurrence y , $x = k$ mots précédents et k mots suivants

... On a **bien** le **droit d'échouer**. De tenter seulement de faire un peu de lumière et des ombres, comme la lampe dans l'escalier, et de s'éteindre ensuite sans bruit...



$$\begin{array}{ll} x^{(1)} = (*d^*, *d^*, a, \text{bien}) & y^{(1)} = \text{On} \\ x^{(2)} = (*d^*, \text{On}, \text{bien}, \text{le}) & y^{(2)} = \text{a} \\ x^{(3)} = (\text{On}, a, \text{le}, \text{droit}) & y^{(3)} = \text{bien} \\ x^{(4)} = (a, \text{bien}, \text{droit}, d') & y^{(4)} = \text{le} \\ x^{(5)} = (\text{bien}, \text{le}, d', \text{échouer}) & y^{(5)} = \text{droit} \\ x^{(6)} = \dots & \end{array}$$

Word2vec : architecture CBOW



Word2vec : modèle Skip-Gram

Exemples d'apprentissage

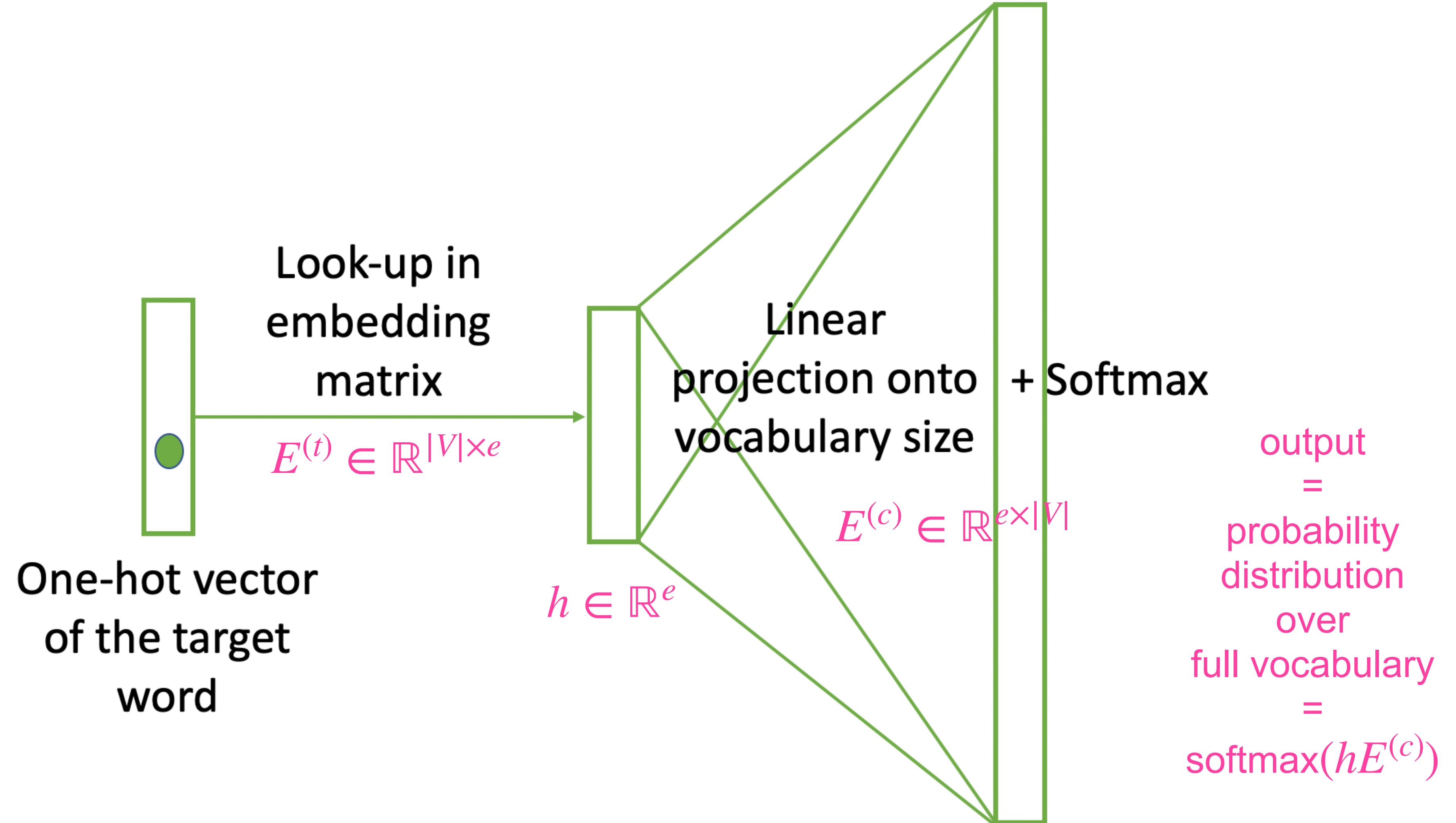
- Apprentissage auto-supervisé : création triviale des exemples
 - pour chaque occurrence x ,
 - y vaut un mot dans la fenêtre des k mots précédents et k mots suivants

... On a bien le **droit** d'**échouer**. De tenter seulement de faire un peu de lumière et des ombres, comme la lampe dans l'escalier, et de s'éteindre ensuite sans bruit...



$x^{(1)} = \text{droit}$	$y^{(1)} = \text{bien}$
$x^{(2)} = \text{droit}$	$y^{(2)} = \text{le}$
$x^{(3)} = \text{droit}$	$y^{(3)} = \text{d'}$
$x^{(4)} = \text{droit}$	$y^{(4)} = \text{échouer}$
$x^{(5)} = \text{d'}$	$y^{(5)} = \text{le}$
$x^{(6)} = \dots$	

Word2vec : architecture Skip-Gram



Word2vec

Apprentissage

- Apprentissage avec perte NLL pour les 2 architectures
- Quelle est la perte en CBOW pour l'exemple $x = (\text{On}, \text{a}, \text{le}, \text{droit})$, $y = \text{bien}$?
 - si les ids de mots sont
 - $\text{on}=>17$, $\text{a}=>1$, $\text{le}=>10$, $\text{droit}=>28$, $\text{bien}=>3$

Word2vec

Sortie

- On récupère les matrices de vecteurs de mots $E^{(t)}$ et $E^{(c)}$
 - la tâche ayant servi à l'apprentissage n'est plus utilisée ensuite
 - mais a donné le signal d'apprentissage :
 - CBOW : représentation des mots cible et de contexte telle que les mots cibles plausibles dans un contexte auront des forts scores
 - Skip-gram : vice-versa
- NB: les vecteurs de mots sont dits « **statiques** »
 - un seul vecteur par graphie
 - mélange des éventuels différents sens (éventuelle homonymie ou polysémie)
 - *fraise, direction*

Word2vec

Astuces à la création d'exemples

- Window size sampling
 - pour chaque occurrence, tirer au sort (uniformément) la taille de la demi-fenêtre, entre 1 et k
- Subsampling of frequent words
 - if $\text{freq}(w) > t$, choose to ignore word with probability $1 - \sqrt{\frac{t}{\text{freq}(w)}}$

Utilisation des vecteurs de mots

- figés
- fine-tunés
- Exemple : l'étiquetage morpho-syntaxique (POS tagging)
- TODO

Apprentissage par transfert

- paramètres appris pour tâche A
- réutilisés (figés ou pas) dans réseau appris pour tâches aval B1, B2, ...

Vecteurs de mots contextualisés

- Historique:
 - RNN
 - modèle seq2seq pour la TA
 - attention
 - transformers
- le modèle BERT

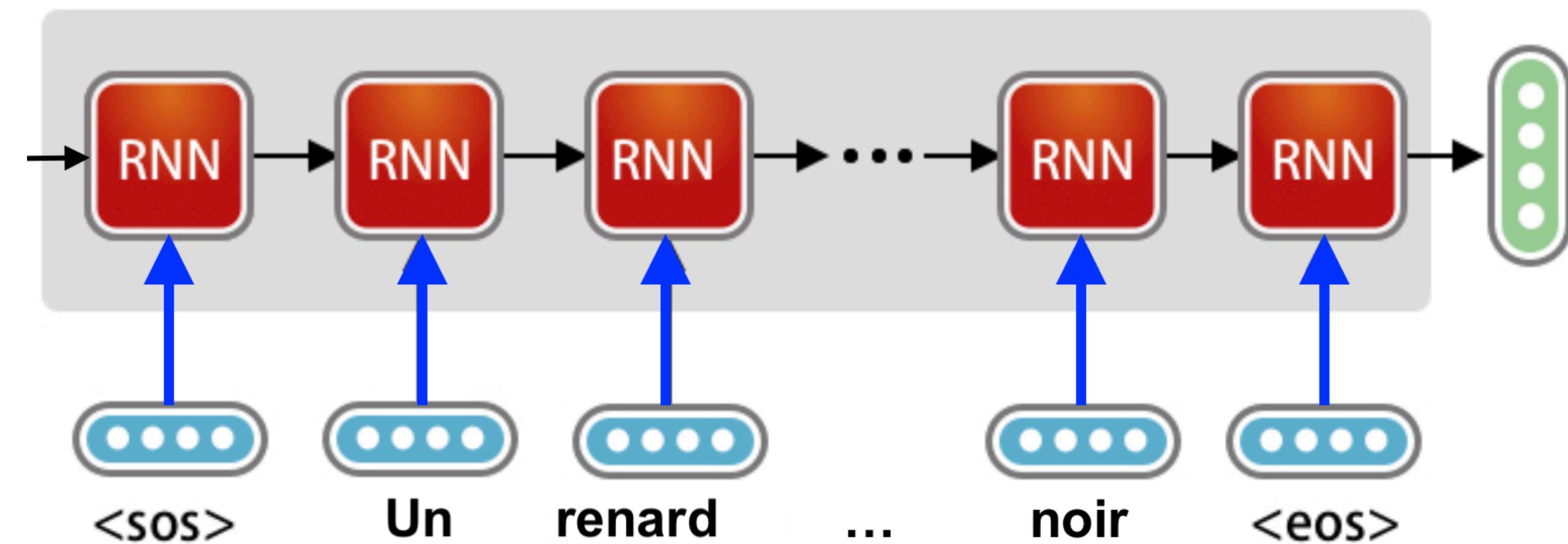


iconspng

Historique: encodage via réseau récurrent

Recurrent Neural Network (RNN)

- **Encodage** = transformer une séquence en une représentation vectorielle
- **Référence** = même réseau (mêmes paramètres) prenant en entrée :
 - → un vecteur **statique** représentant le prochain symbole de la séquence
 - → le vecteur de sortie à la position précédente : représente l'historique

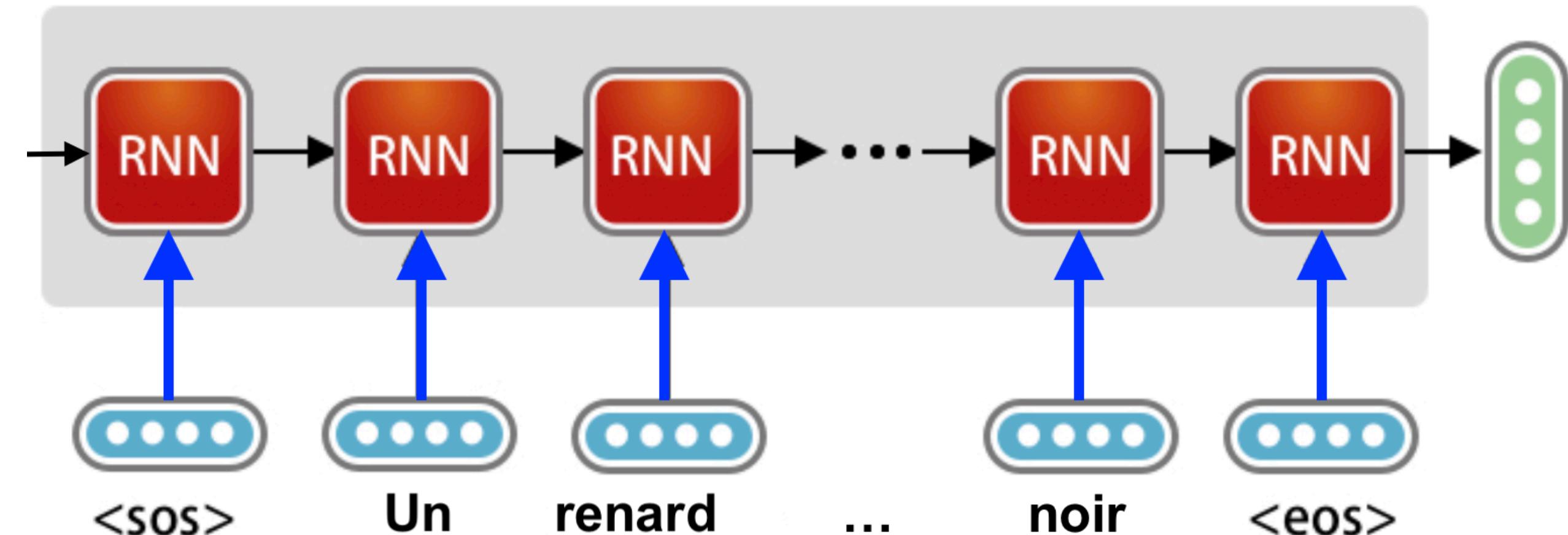


-

Historique: encodage via réseau récurrent

Recurrent Neural Network (RNN)

- **Encodage** = transformer une séquence en une représentation vectorielle
- **Récurrence** = même réseau (mêmes paramètres) prenant en entrée:
 - → un vecteur **statique** représentant le prochain symbole de la séquence
 - → le vecteur de sortie à la position précédente : représente l'historique

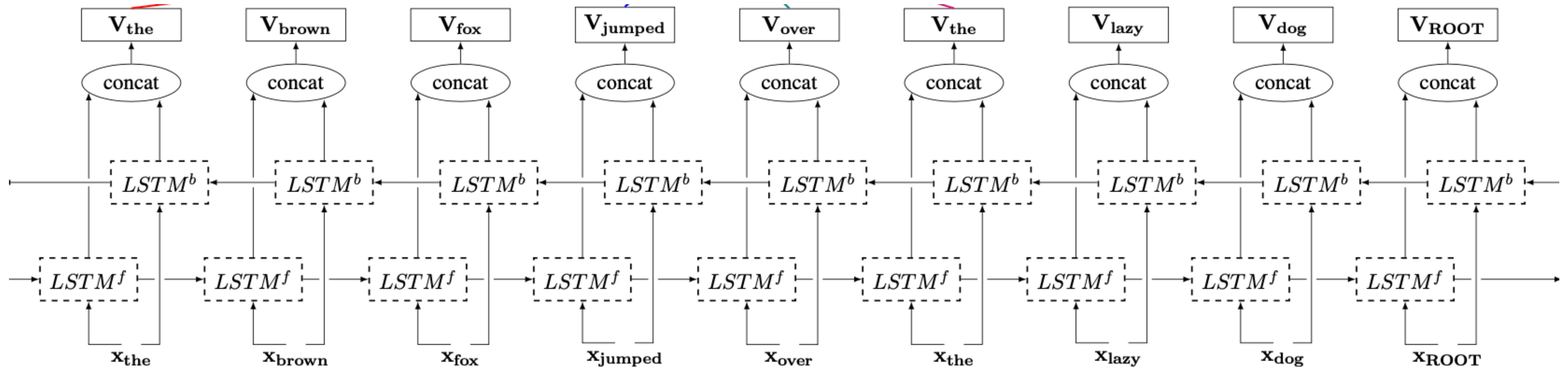


- la sortie à une position donnée encode le mot courant et tous les précédents
- ⇒ vecteur de mot **contextualisé**

Historique: encodage via réseau récurrent

RNN bidirectionnel

- Obtention de vecteurs contextualisés gauche et droite
 - par concaténation des sorties de 2 réseaux → et ←



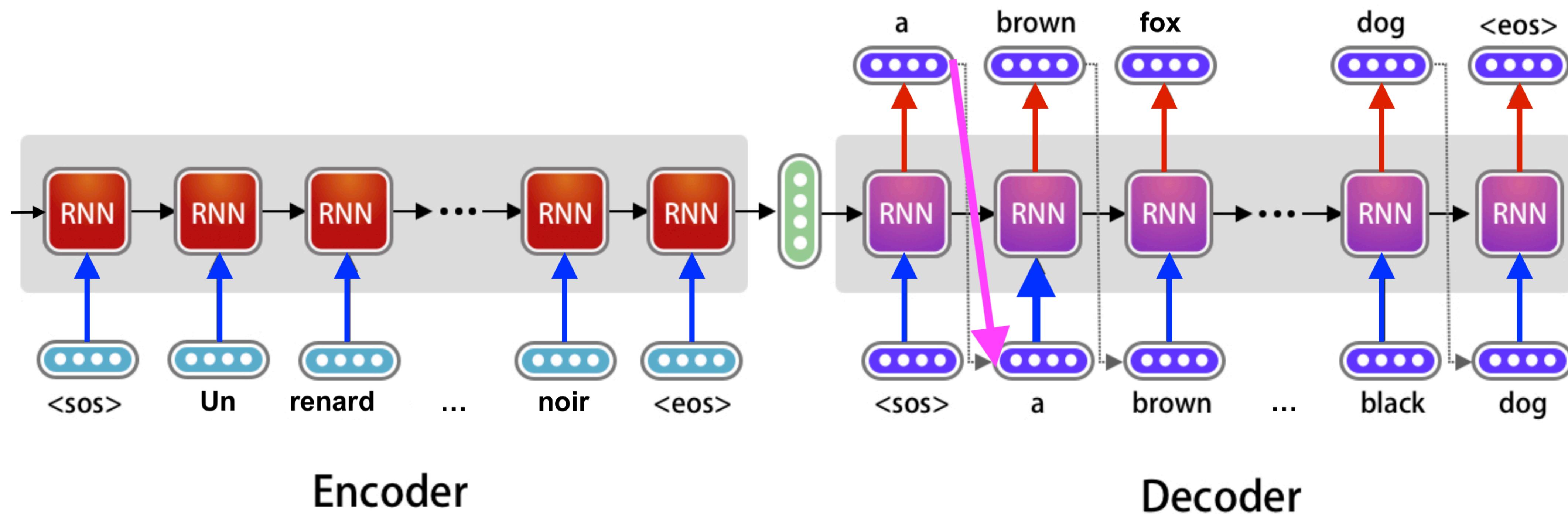
- NB: les paramètres des 2 RNN sont appris en même temps que d'autres
 - dans un réseau pour une tâche avale

Kiperwasser & Goldberg 2016 (<https://aclanthology.org/Q16-1023/>)

Historique : le modèle seq2seq pour la TA

« sequence to sequence » Sutskever et al. 2014

- Traduction automatique
 - Exemples = couples de phrases traduction l'une de l'autre
 - Encodage bi-RNN de la phrase source

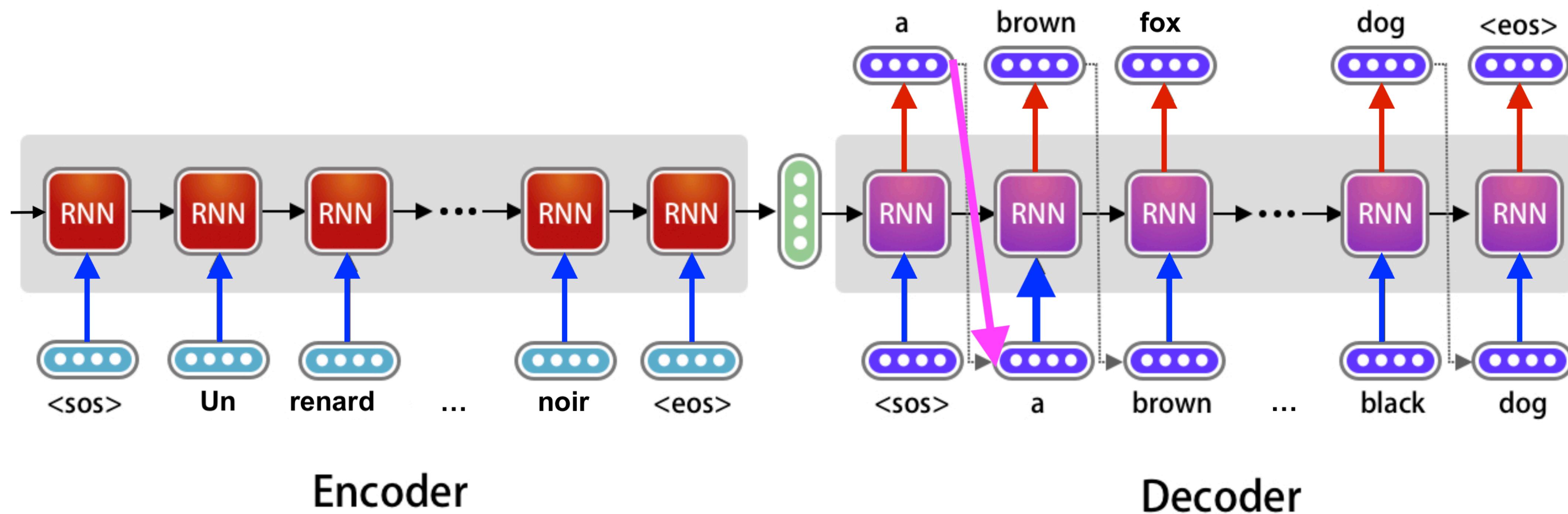


Encoder

Decoder

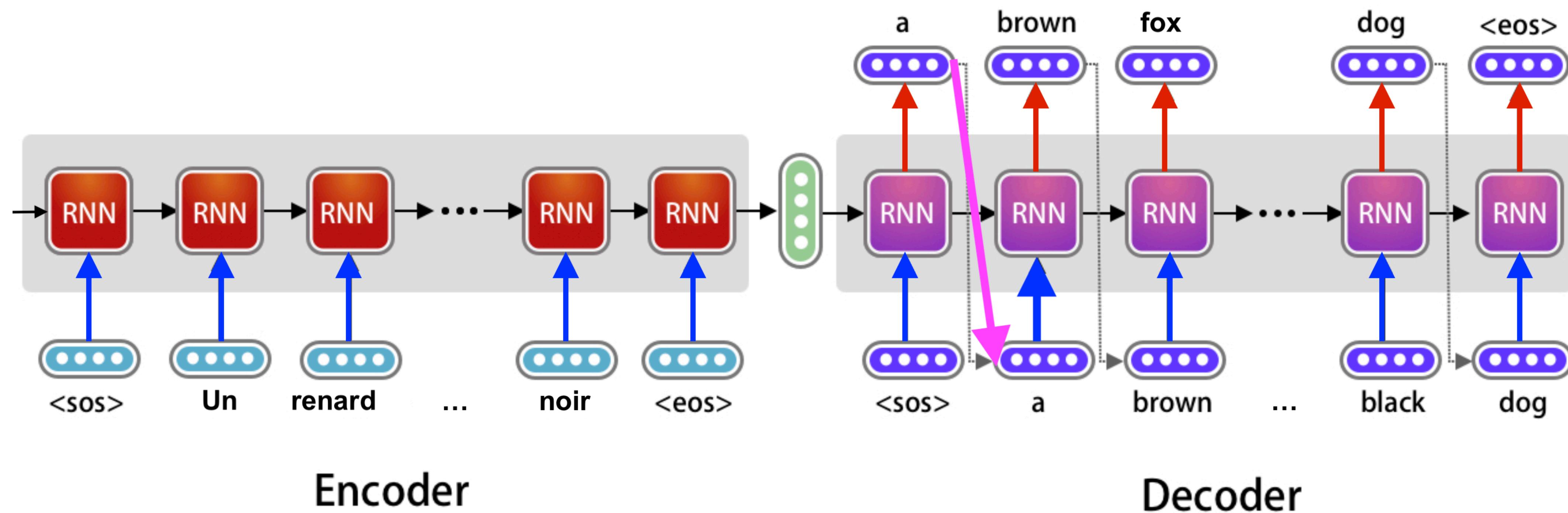
Historique : le modèle seq2seq pour la TA

- Décodeur : RNN + « tête de language modeling »
 - tête de LM = MLP
 - entrée = représentation à cette position,
 - sortie = probas sur mots du vocabulaire



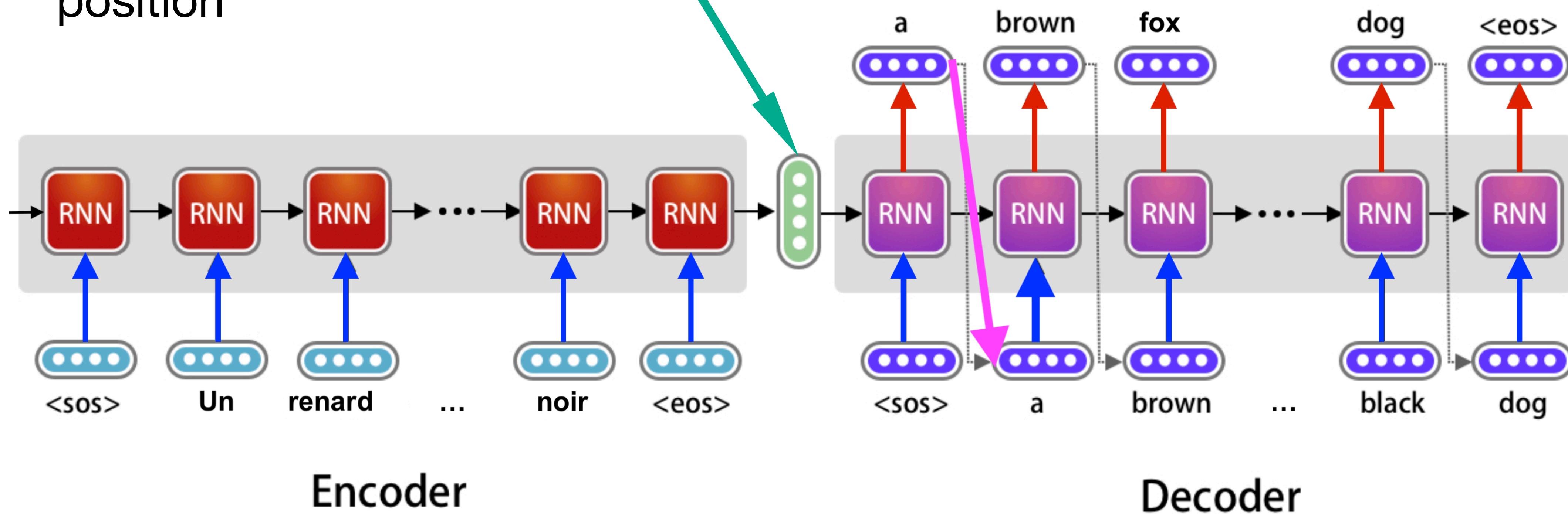
Historique : le modèle seq2seq pour la TA

- Décodeur : RNN + « tête de language modeling »
 - le RNN reçoit:
 - à l'apprentissage : le vecteur statique du mot cible gold
 - à l'inférence : le vecteur statique du mot généré à la position précédente



Historique : le modèle seq2seq pour la TA

- Problème :
 - la **représentation de la phrase source** est unique
 - ne capture pas liens spécifiques entre mots source et mots cible
 - => d'où l'idée d'adapter la représentation du contexte source à chq position



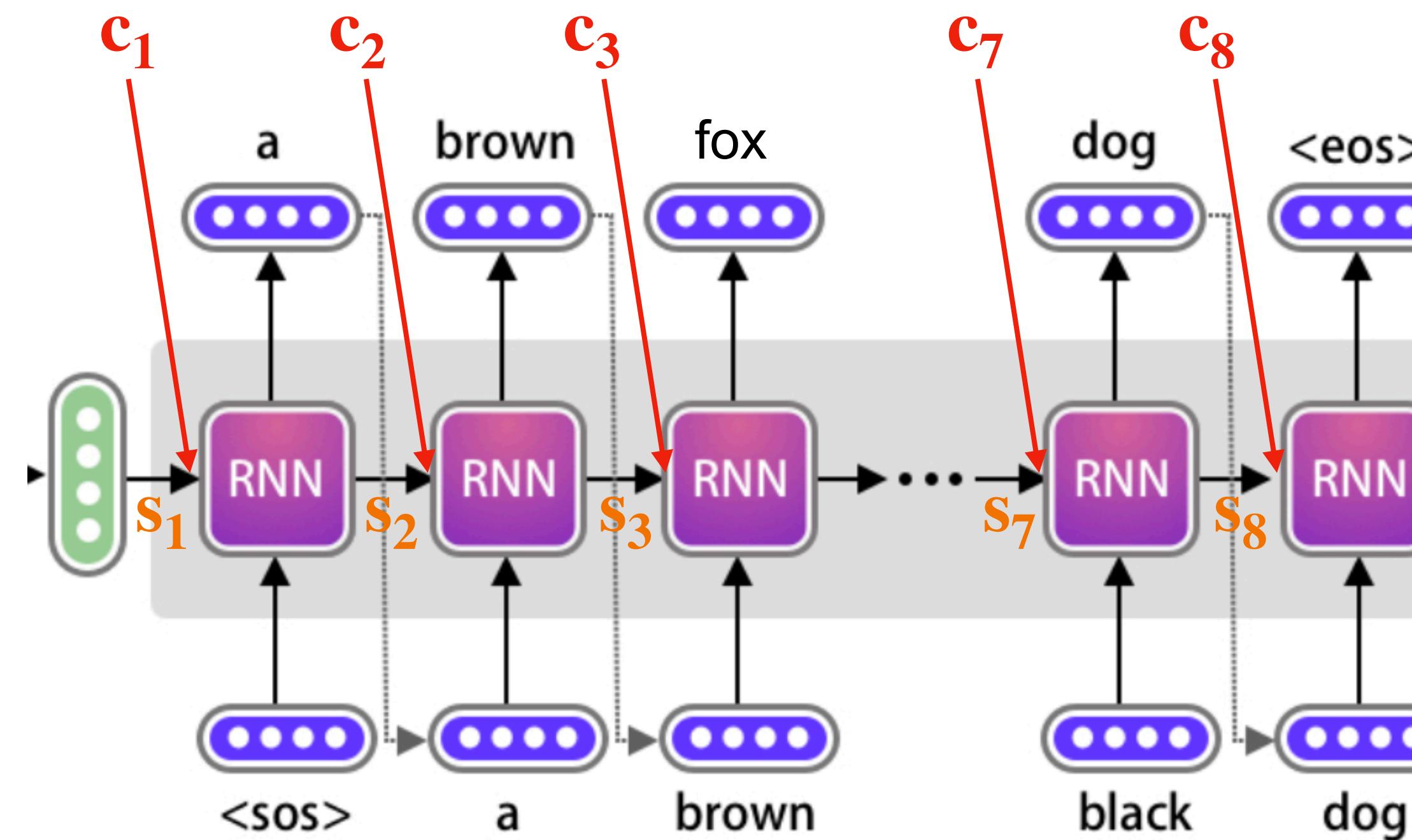
Encoder

Decoder

Historique : modèle seq2seq avec attention

Bahdanau et al. 2014

- Le décodeur prend une entrée supplémentaire c_i variant à chaque position



Decoder

Historique : modèle seq2seq avec attention

Bahdanau et al. 2014

- chaque c_i est une somme pondérée des vecteurs h_j encodant chq mot source

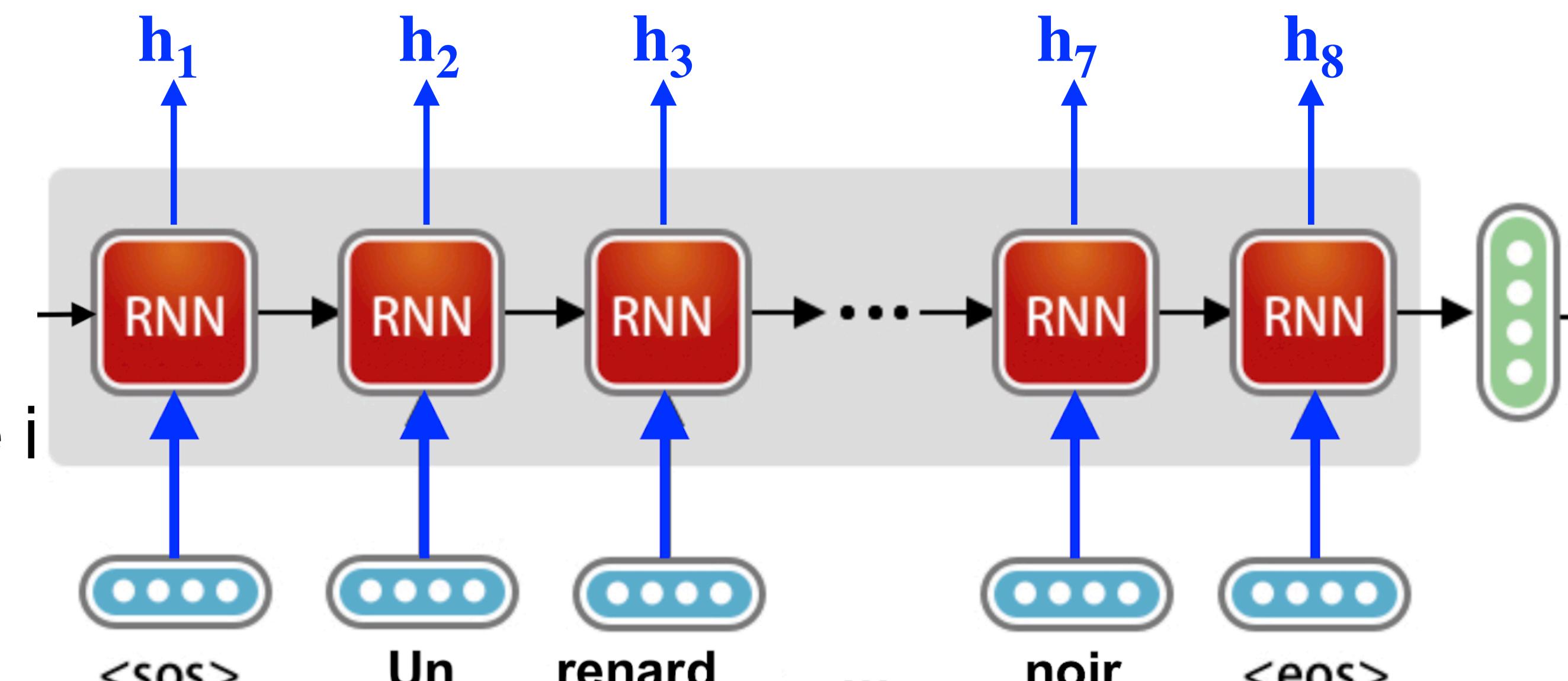
$$c_i = \sum_j \alpha_{ij} h_j$$

- α_{ij} est le poids d'**attention**

- portée au token source j
- lors de la génération du token cible i
- valeur softmaxisée d'une fonction d'attention $a(s_{i-1}, h_j)$

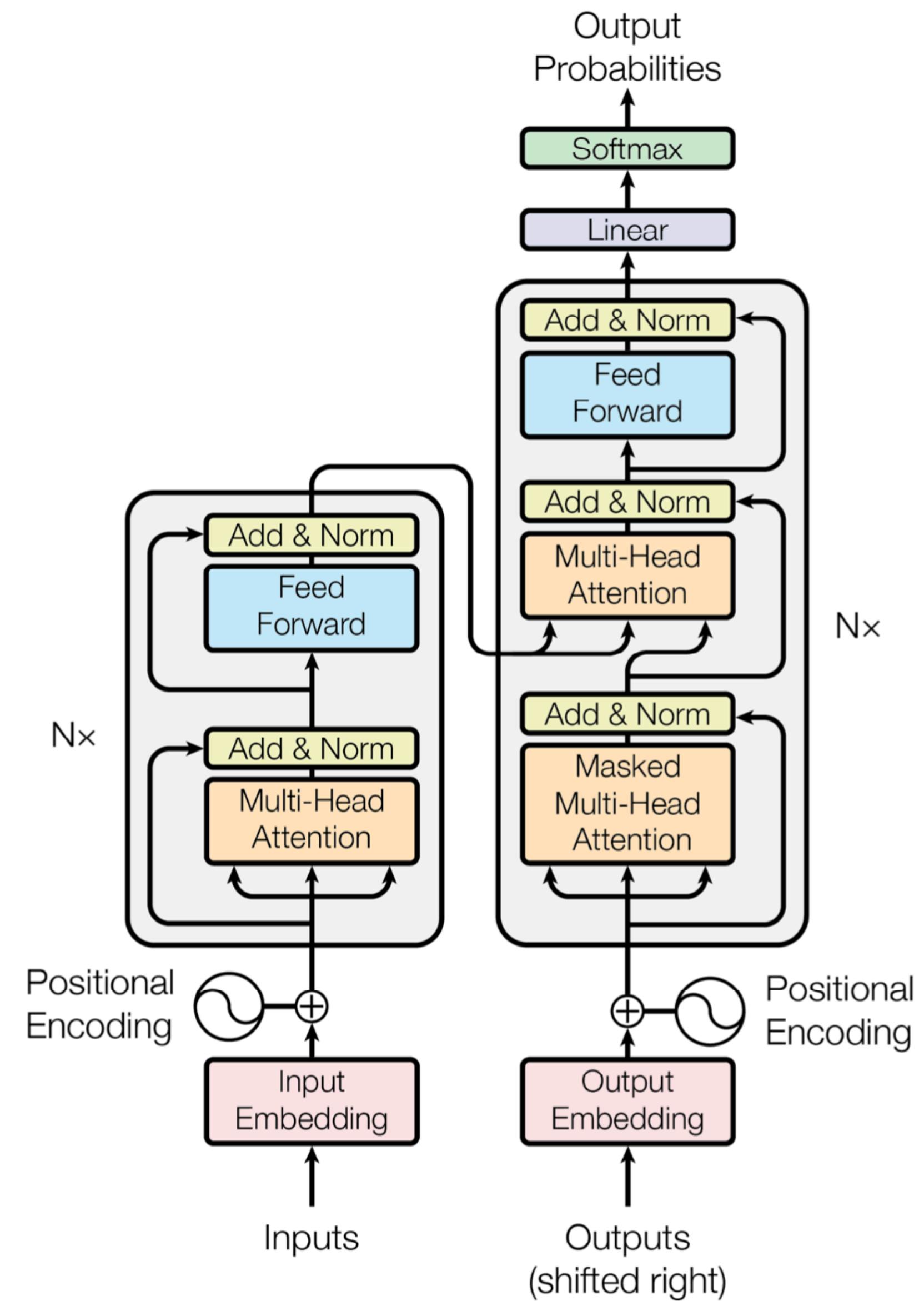
- chez Bahdanau et al.:

$$a(s, h) = MLP(s, h)$$



Encoder

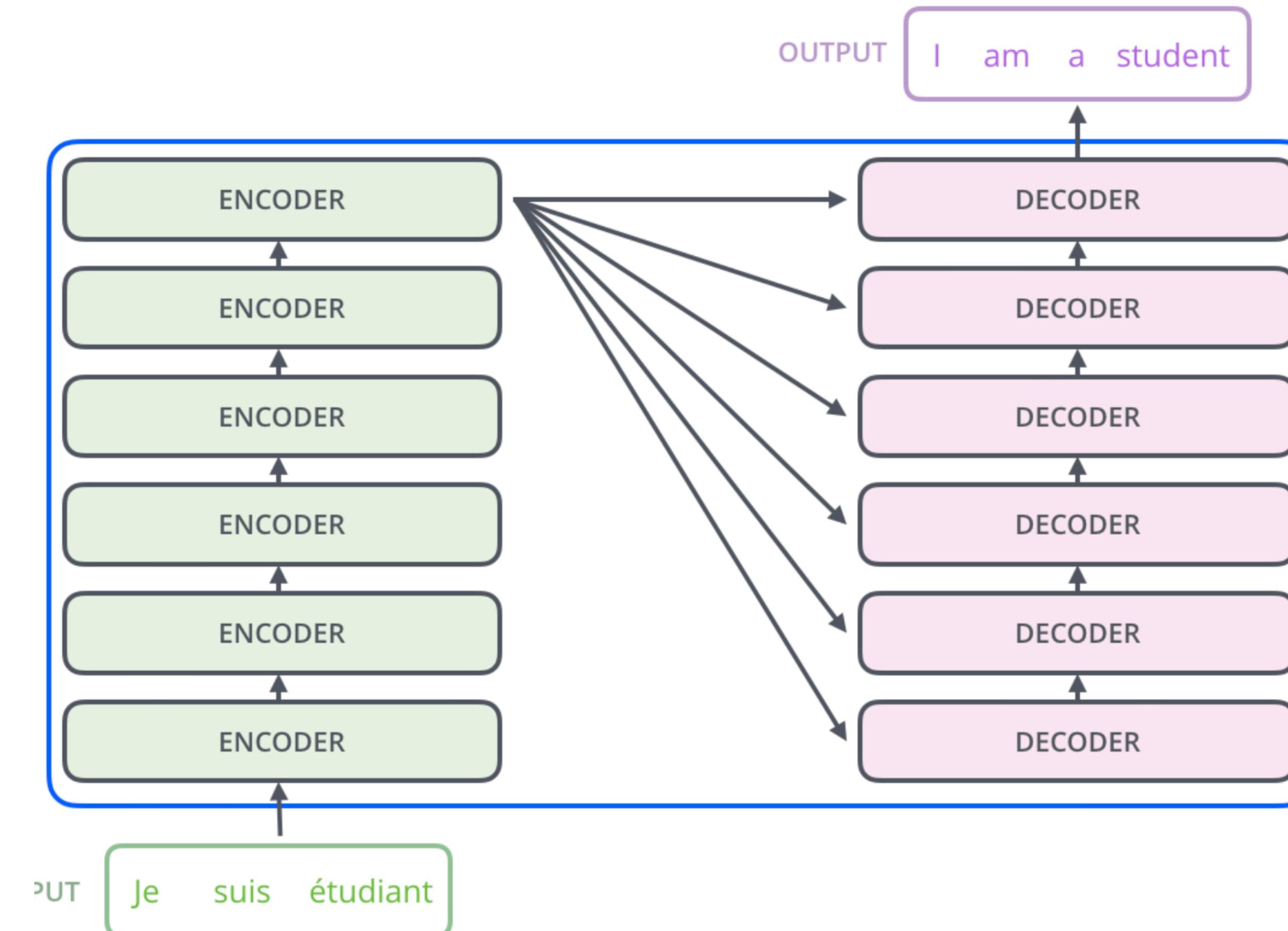
Prêts pour les transformers



Transformers

Attention is all you need (Vaswani et al. 2017)

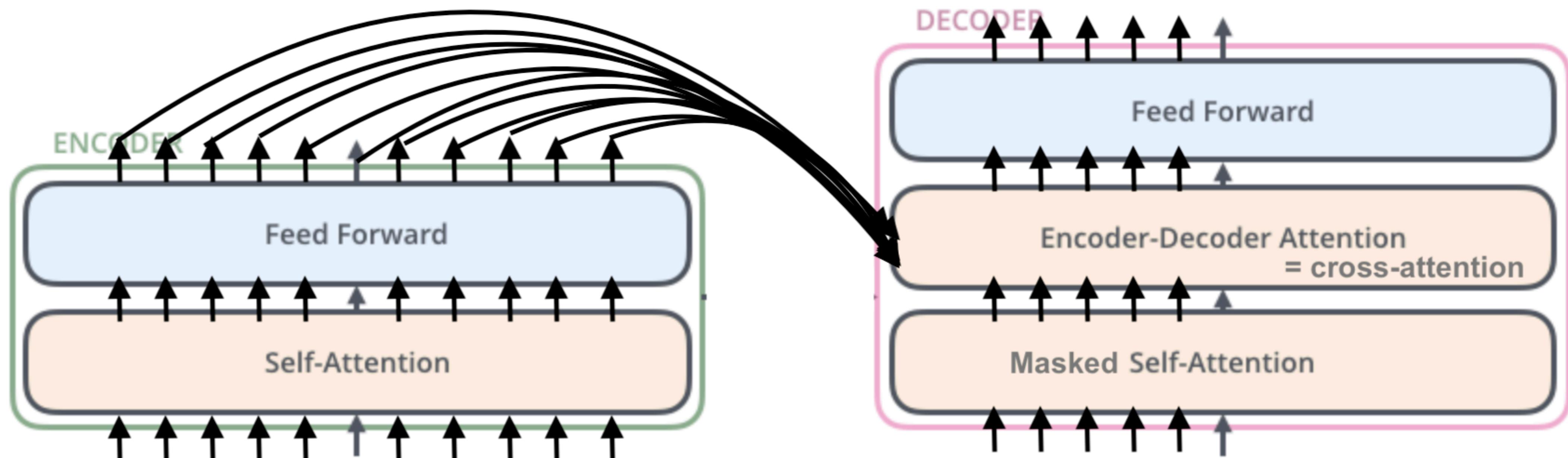
- Suppression de la partie récurrente
- Encodeurs empilés
 - self-attention
- Décodeurs empilés
 - cross-attention
 - masked self-attention
- Massivement parallélisable



Transformers

Attention is all you need (Vaswani et al. 2017)

- Zoom en sur une couche d'encoder et de décodeur



modifié de J. Alammar's blog <https://jalammar.github.io/illustrated-transformer/>

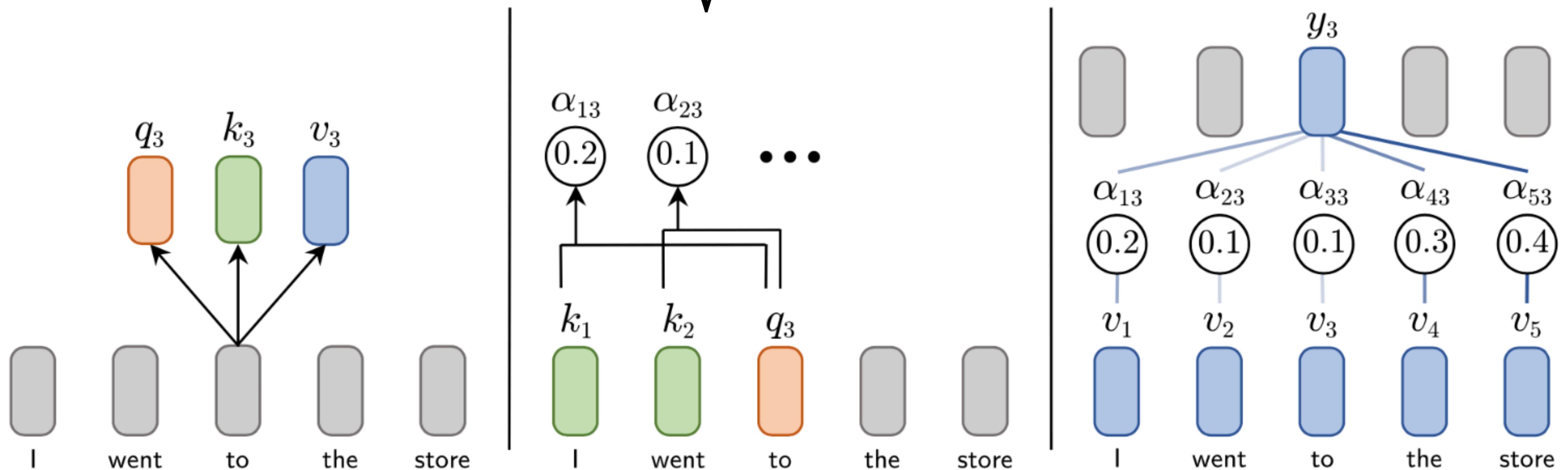
- + connexions résiduelles et normalisation

Transformers

Self-attention : attention sur les tokens de la même séquence

- Une couche encodeur l reçoit des vecteurs d'entrée h_i^{l-1} pour chaque position i
- 3 matrices de params transforment les h_i^{l-1} en **keys** k_i , **queries** q_i , **values** v_i
- self-attention à la couche l : somme pondérée des values $y_i^l = \sum_j \alpha_{ji} v_j^l$
- α_{ji} = poids d'attention vers token j , pour le token i
- = version softmaxisée de : $q_i \cdot k_j / \sqrt{\text{len}(k_j)}$

- 3 matrices de params transforment les h_i^{l-1} en **keys** k_i , **queries** q_i , **values** v_i
- self-attention à la couche l : somme pondérée des values $y_i^l = \sum_j \alpha_{ji} v_j^l$
- α_{ji} = poids d'attention vers token j, pour le token i
- = version softmaxisée de : $q_i \cdot k_j / \sqrt{\text{len}(k_j)}$



Each input vector is linearly transformed into query, key, and value vectors

Attention weights are normalized inner products of query and key vectors

Outputs are weighted sums of value vectors

Transformers

Masked self-attention : attention sur les tokens gauche

- Dans le décodeur
- Idem mais attention seulement vers les tokens de gauche de la phrase générée

Transformers

Cross-attention : attention sur les tokens de la phrase source

- Après la masked self-attention, une couche décodeur l applique l'attention vers la phrase source
- Même Principe avec
 - **query** = position courante dans le décodeur
 - **keys** et **values** : transformation des vecteurs de sortie de l'encodeur
- NB: l'attention de type « scaled dot » ($q_i \cdot k_j / \sqrt{\text{len}(k_j)}$) est computationnellement simple et parallélisable

Encodage de la position

- Avec la scaled dot attention : toutes les opérations d'attention sont insensibles à l'ordre des tokens!
- D'où l'ajout en entrée de l'encodeur / décodeur de vecteurs positionnés
 - Input à la position $i = \text{vecteur statique pour token } i + \text{embedding positionnel}$

BERT

Bidirectional encoder representations from transformers

- = Modèle de langue pré-entraîné (PLM)
 - Février 2018 : ELMo (article arXiv) : vecteurs biLSTM pré-entraînés
 - Juin 2018 : GPT (rapport technique OpenAI)
 - Octobre 2018 : BERT (papier arXiv)
- Relève de l'**apprentissage par transfert**
- Objectif de BERT = obtenir des vecteurs de mots conceptualisés
 - À utiliser pour tâches aval, avec 2 modes
 - « **feature-based** » : les vecteurs de mots contextualisés sont des features d'entrée du réseau aval (paramètres de BERT figés)
 - « **fine-tuning** » : réseau BERT comme première partie du réseau, ajustement sur tâche aval des paramètres pré-entraînés
 - 340M paramètres pré-entraînés versus quelques milliers pour chaque tâche aval

BERT

Bidirectional encoder representations from transformers

- Simplification des transformers : blocs **encodeurs** uniquement
 - Utilisation de la self-attention : accès à toute la phrase d'entrée
 - D'où le terme « bidirectional »
- Tâches utilisées à l'apprentissage :
 - Prédiction d'un mot à une position masquée (*masked language modeling*)
 - Classification binaire : pour une séquence de 2 phrases : « la 2ème phrase est-elle la suite de la première » ?
 - Toutes deux correspondent à de l'**auto-supervision** (apprentissage supervisé avec exemples créés trivialement)

BERT : pré-entraînement

- Entrée =
 - Une paire de phrases (séquences)
 - Tokenisées (voir TP)
 - [CLS] ... seq1... [SEP] ...seq2...[SEP]
 - 15% tokens remplacés par
 - [MASK]
 - Ou le token lui-même
 - Ou un token aléatoire

