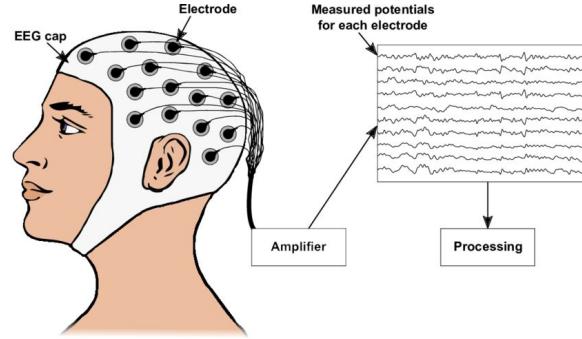
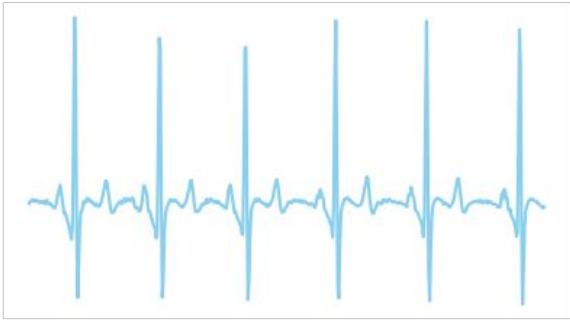


Programme du cours

- Introduction et contexte
- Réseaux de neurones et optimisation
- Deep Computer Vision
- Deep Sequence model
- Deep Generative model

Données séquentielles



AACCCGGGGATATTCTTATATCGTGCGGTGCGA
TCAGACGGCGTAAATTGGAGCTCATGCGAC
AAGGGACGGGTACTAGACTGCCGATGGCGGAGC
CAATGGTAGTCATAATTCGGCGCGCTGGCC
AGATGTTAACCCCAACCTTAACCTGGTTACCA
GGGGCACATAAGTCCCTCCGGAAATCCCCA
GACCTTGGGCGCTGGACGGTCTATGCTG
AAAGATTCTGGTGTTCACCTCCGAGATAGANTCT
TCCGCTAAATTGGCGGAGGGTCATGCTG
GCCGTGCCTTGACACGGGGCTGGCTCG
GGTAGCTTACACATACTTCGGNGACAGA
GGTAGCTTACACATACTTCGGNGACAGA
TGGCAGAGTCATCCCAACATATTG
GAGACCAATTATTGAGAGGATCTG
AFTTGTGTTCTGACATTATTG
GGTGCAAGACTCTGCTCTGG
GGTGCAAGACTCTGCTCTGG
GGTGCAATTAGTAGTAAATG
GGTGCAATTAGTAGTAAATG
GGTGCAATTAGTAGTAAATG
GGTGCAATTAGTAGTAAATG



Données textuelles

Analyse des sentiments



Description à partir d'une image

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

Génération de texte



Explain to me measure theory



Alright, buckle up! Measure theory is a branch of mathematics that deals with the study of measures, which are essentially a way of assigning a "size" to sets. It provides a more general and abstract framework for understanding concepts like length, area, and volume.

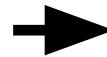
In measure theory, a measure is a function that assigns a non-negative real number to sets in a systematic way. The key properties of a measure include non-negativity (the measure of any set is non-negative), null empty set (the measure of the empty set is zero), and sigma-additivity (the measure of the union of countably many disjoint sets is the sum of their individual measures).

The set of all possible measurable sets is called a sigma-algebra, and the triple consisting of a set, a sigma-algebra, and a measure is called a measurable space.



Traduction de texte

I love Deep Learning



Rakastan
syväoppimista

Données Audio

Reconnaissance vocale



Génération audio

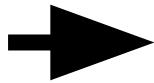


Reconnaissance des émotions à partir de la voix



Source image:
[nvidia](#)

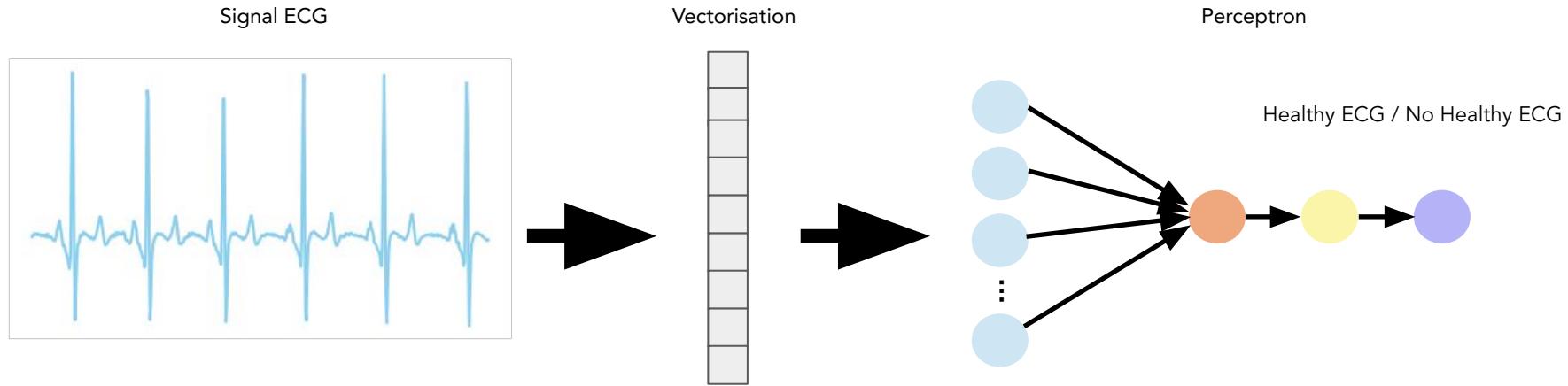
Classification de séries temporelles



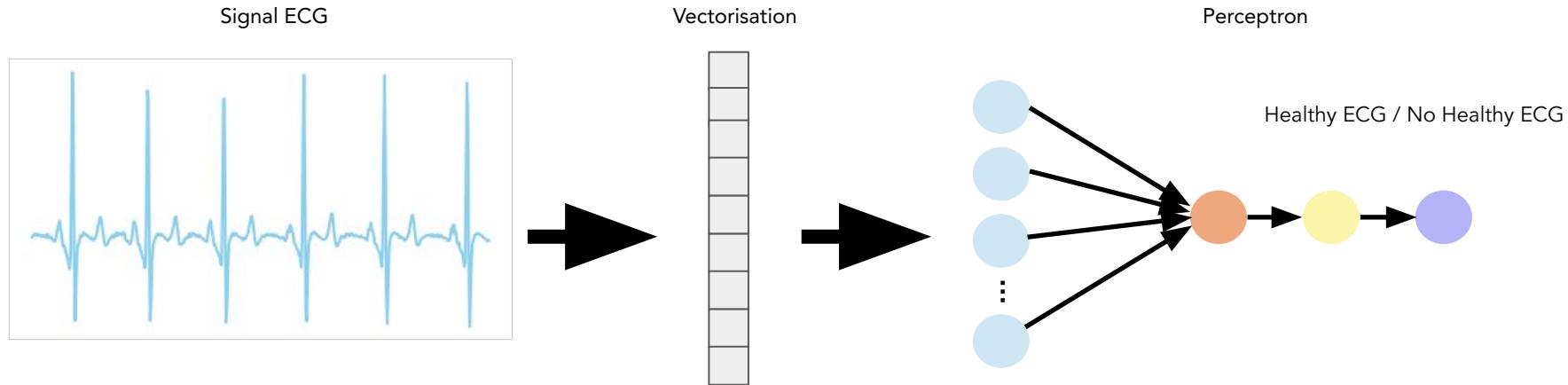
Classification binaire

Healthy ECG / No Healthy ECG

Classification de séries temporelles

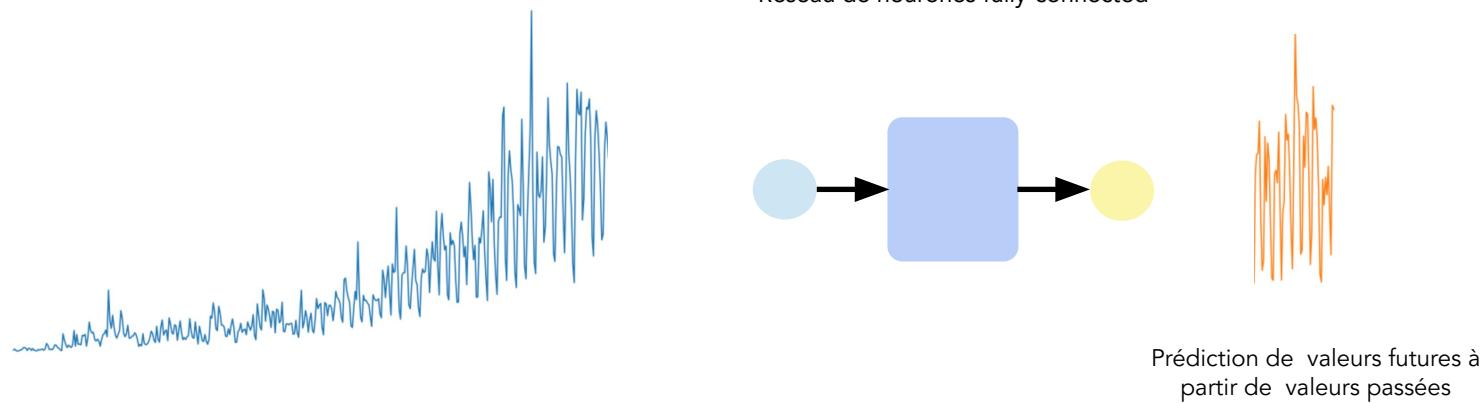


Classification de séries temporelles



- Perte de la connectivité temporelle
- Entrée de taille fixe
- Peut résulter en un nombre élevé de paramètres, ce qui peut en retour induire de l'overfitting
- Absence d'invariance à la translation. Le réseau ne comprend pas intrinsèquement que des motifs situés à des positions différentes dans la séquence peuvent avoir la même signification

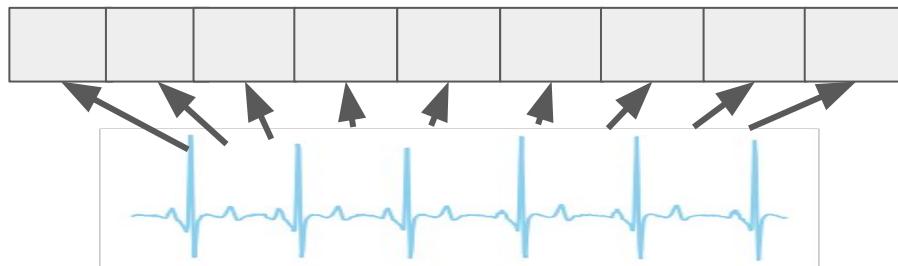
Time Series forecasting



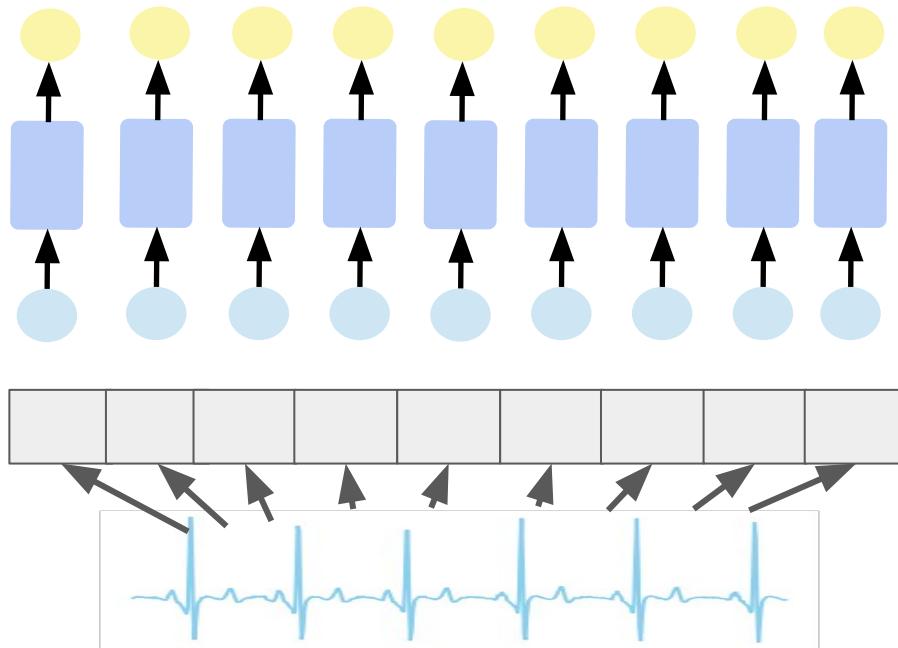
Peut-on faire mieux ?



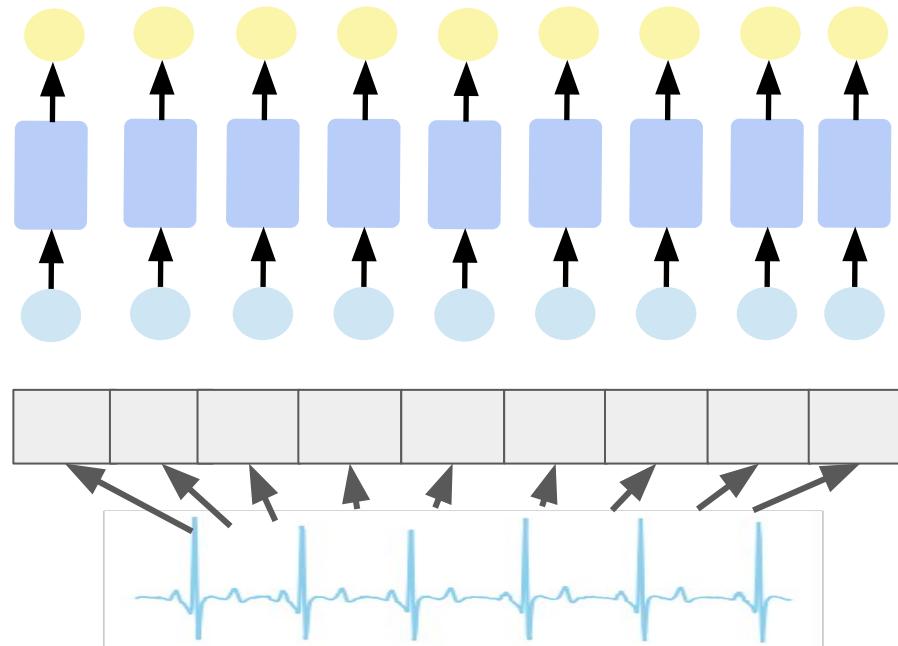
Peut-on faire mieux ?



Peut-on faire mieux ?

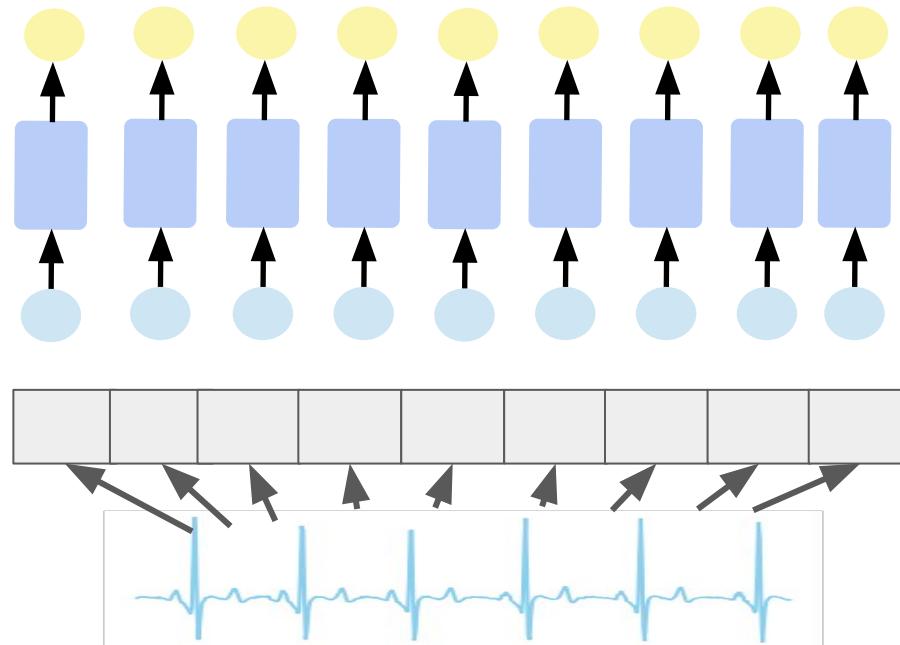


Peut-on faire mieux ?



Traitement indépendant de chaque pas de temps

Peut-on faire mieux ?



Traitements indépendants de chaque pas de temps → Comment prendre en compte la connectivité temporelle ?

Bases du NLP (Natural Language processing)

Bases du NLP (Natural Language processing)

- Sous-domaine de l'analyse de données séquentielles: focus sur le **langage**

Bases du NLP (Natural Language processing)

- Sous-domaine de l'analyse de données séquentielles: focus sur le **langage**
- Exemples d'application:
 - Classification de documents
 - Topic modelling
 - Traduction
 - Chatbot, assistant virtuels, ...
 - Résumé de textes
 - ...

Bases du NLP (Natural Language processing)

- Pourquoi c'est difficile ?

Bases du NLP (Natural Language processing)

- Pourquoi c'est difficile ?

- **Ambiguité:** plusieurs mots/phrases peuvent avoir différents sens sémantique et/ou syntactique
- **Contexte:** qu'il soit direct ou indirect
- **Nuances et subtilités liées à la forme:** émotion, sarcasme, ...
- **Données annotées**
- **Interdisciplinarité:** lié à d'autres domaines (linguistique, psychologie,...)

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Encodage des mots en **1-hot** → Simple à implémenter

The diagram illustrates the 1-hot encoding of words into vectors. It shows four words: Rome, Paris, Italy, and France, each mapped to a specific index in a vector space. The vectors are represented as brackets containing a sequence of binary digits (0 or 1). Arrows point from the words to their corresponding vector components. The first component of the vector is labeled 'word V'.

Rome	=	[1, 0, 0, 0, 0, 0, ..., 0]
Paris	=	[0, 1, 0, 0, 0, 0, ..., 0]
Italy	=	[0, 0, 1, 0, 0, 0, ..., 0]
France	=	[0, 0, 0, 1, 0, 0, ..., 0]

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Encodage des mots en 1-hot → Simple à implémenter

→ **Tous les mots ont la même distance**

The diagram illustrates the 1-hot encoding of words into vectors. Four words are shown: Rome, Paris, Italy, and France. Arrows point from each word to its corresponding vector representation. The vectors are represented as lists of binary values (0s and 1s) followed by ellipses and a final 0. The word 'word V' is also labeled with an arrow pointing to the final zero in the vector for Rome.

Rome	=	[1, 0, 0, 0, 0, 0, ..., 0]
Paris	=	[0, 1, 0, 0, 0, 0, ..., 0]
Italy	=	[0, 0, 1, 0, 0, 0, ..., 0]
France	=	[0, 0, 0, 1, 0, 0, ..., 0]

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Utilisation d'**embedding** des mots = projection des mots dans un espace vectoriel abstrait “qui a du sens”

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Utilisation d'**embedding** des mots = projection des mots dans un espace vectoriel abstrait “qui a du sens”

$$E : \text{Vocabulaire} \rightarrow \mathbb{R}^d$$

Exemple: $E(\text{"roi"}) - E(\text{"homme"}) + E(\text{"femme"}) \approx E(\text{"reine"})$

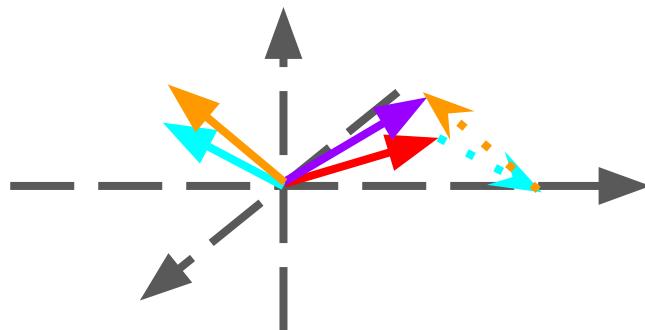
Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Utilisation d'**embedding** des mots = projection des mots dans un espace vectoriel abstrait "qui a du sens"

$$E : \text{Vocabulaire} \rightarrow \mathbb{R}^d$$

Exemple: $E(\text{"roi"}) - E(\text{"homme"}) + E(\text{"femme"}) \approx E(\text{"reine"})$



Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Utilisation d'**embedding** des mots = projection des mots dans un espace vectoriel abstrait "qui a du sens"

$$E : \text{Vocabulaire} \rightarrow \mathbb{R}^d$$

Exemple: $E("roi") - E("homme") + E("femme") \approx E("reine")$

Typiquement d est de l'ordre de la centaine ($d = 100, 200, 500, \dots$) donc bien moins que la taille du dictionnaire

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Utilisation d'**embedding** des mots = projection des mots dans un espace vectoriel abstrait “qui a du sens”

$$E : \text{Vocabulaire} \rightarrow \mathbb{R}^d$$

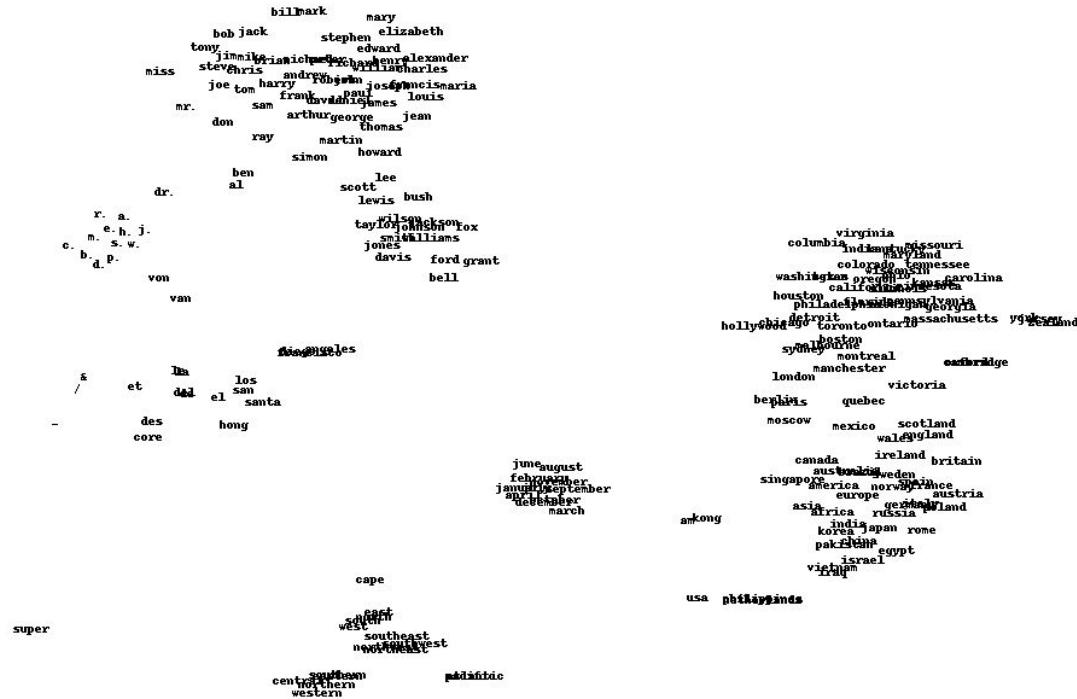
Exemple: $E("roi") - E("homme") + E("femme") \approx E("reine")$

Typiquement d est de l'ordre de la centaine ($d = 100, 200, 500, \dots$) donc bien moins que la taille du dictionnaire

→ On peut apprendre E avec un réseaux de neurones !

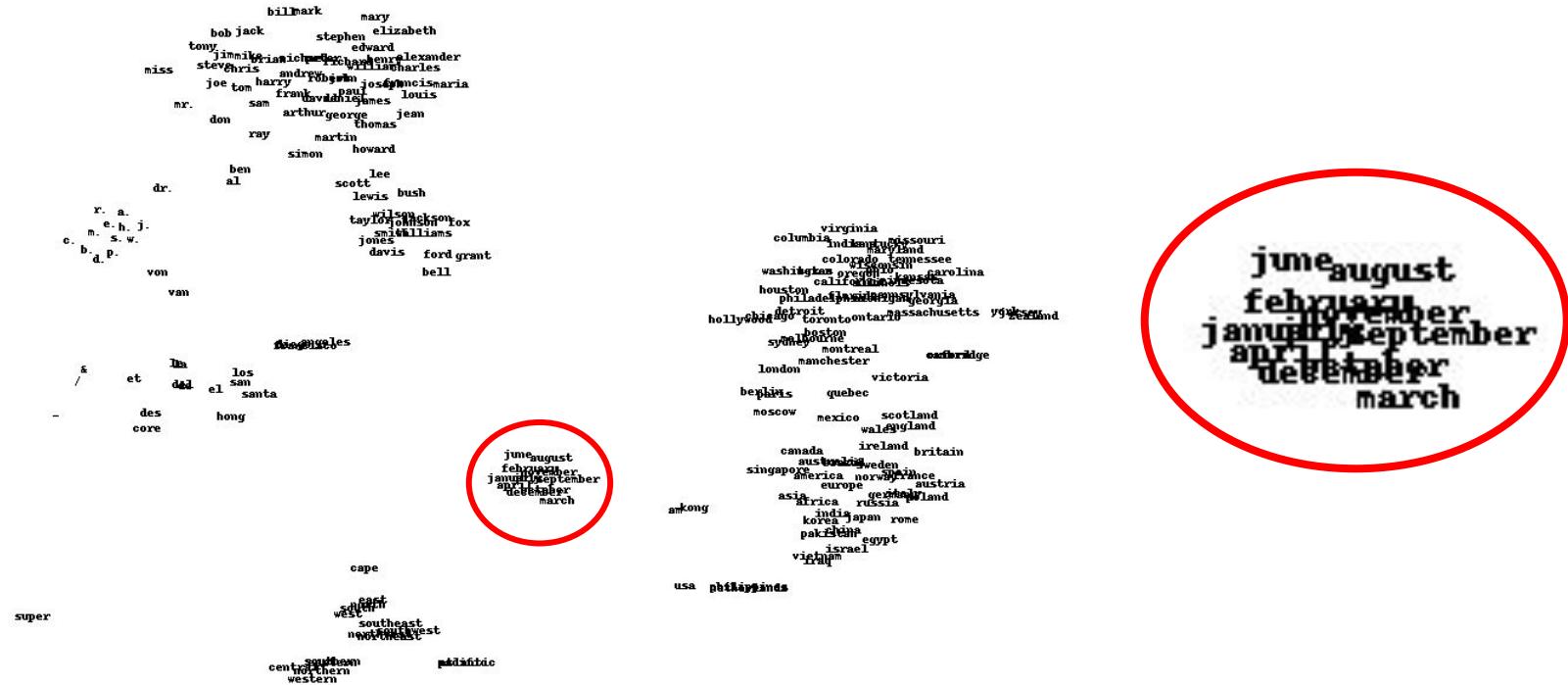
Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation



Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation



Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

→ Cette représentation utilise des **Bag-of-Words (BoW)**

Bases du NLP (Natural Language processing)

Représentation des mots et vectorisation

- Cette représentation utilise des **Bag-of-Words (BoW)**
- Problème: on perd la notion de séquence (i.e. l'ordre des mots)

Bases du NLP (Natural Language processing)

Modèle de langage

→ Idée = donner une probabilité à une certaine séquence de mots

Bases du NLP (Natural Language processing)

Modèle de langage

→ Idée = donner une probabilité à une certaine séquence de mots

Ex: Prob("La voiture **est** bleue") > Prob("La voiture **mange** bleue")

Bases du NLP (Natural Language processing)

Modèle de langage

→ Idée = donner une probabilité à une certaine séquence de mots

Ex: Prob("La voiture **est** bleue") > Prob("La voiture **mange** bleue")

Solution possible: modèle **auto-régressif**

$$p_{\theta}(w_0) \cdot p_{\theta}(w_1 | w_0) \dots p_{\theta}(w_n | w_{n-1}, w_{n-2}, \dots, w_0)$$

Bases du NLP (Natural Language processing)

Modèle de langage

→ Idée = donner une probabilité à une certaine séquence de mots

Ex: Prob("La voiture **est** bleue") > Prob("La voiture **mange** bleue")

Solution possible: modèle **auto-régressif**

$$p_{\theta}(w_0) \cdot p_{\theta}(w_1 | w_0) \dots p_{\theta}(w_n | w_{n-1}, w_{n-2}, \dots, w_0)$$

→ p_{θ} est paramétré par un réseaux de neurones !

Bases du NLP (Natural Language processing)

Modèle de langage conditionnés - exemples

Traduction: $p_{\theta}(T_{target}|S_{source})$

- Source = “I like cats”
- Target = “J'aime les chats”

$p_{\theta}(w_0, S_{source}).p_{\theta}(w_1|w_0, S_{source})...p_{\theta}(w_n|w_{n-1}, w_{n-2}, ..., w_0, S_{source})$

Bases du NLP (Natural Language processing)

Modèle de langage conditionnés - exemples

Question / Réponses: $p_{\theta}(Reponse|Question, Contexte)$

Bases du NLP (Natural Language processing)

Modèle de langage conditionnés - exemples

Question / Réponses: $p_{\theta}(Reponse|Question, Contexte)$

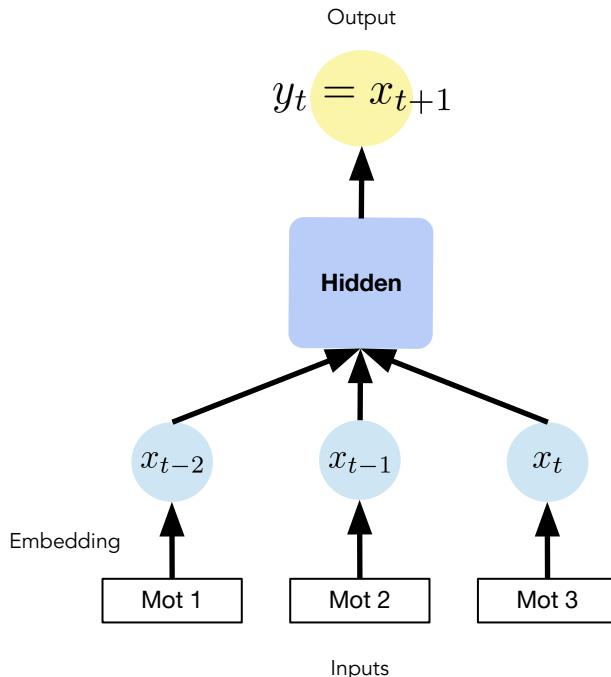
- Contexte = {*Jean met 2 verres sur la table
 - *Alice en ajoute 2 autres
 - *Jean part faire des courses}
- Question = Combien de verres sont sur la table ?
- Réponse = Il y a 4 verres sur la table

Bases du NLP (Natural Language processing)

Modèle de langage simple - prédiction du mot suivant

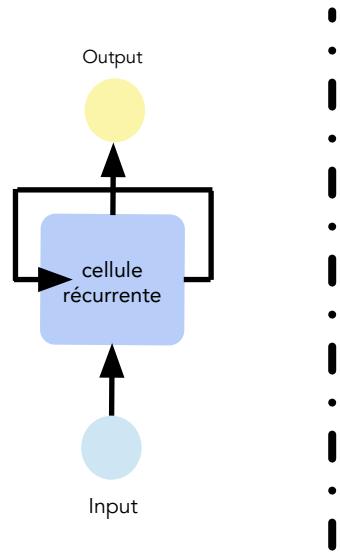
Bases du NLP (Natural Language processing)

Modèle de langage simple - prédiction du mot suivant

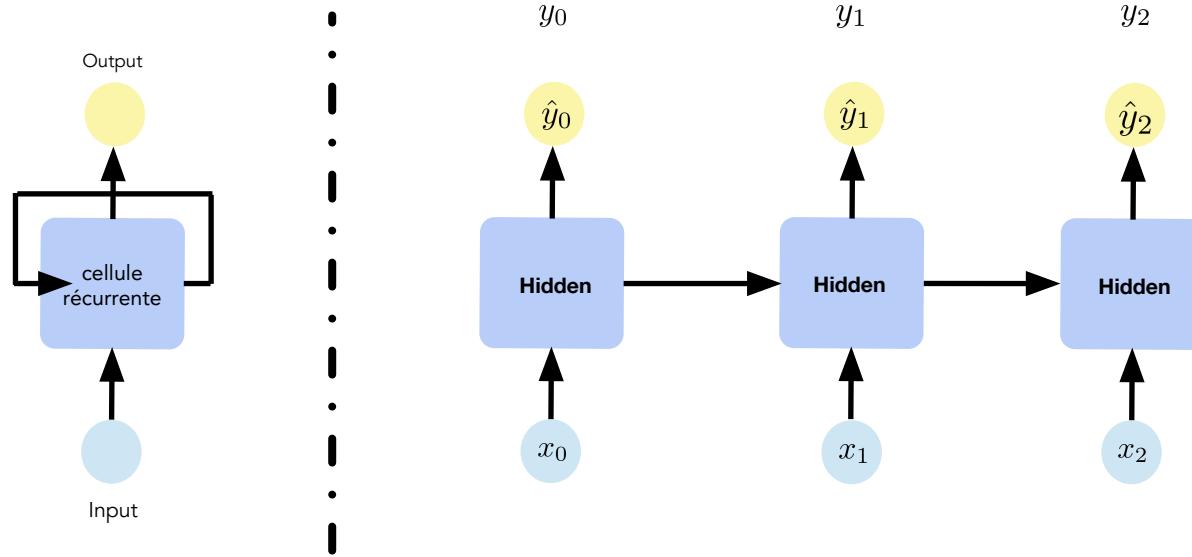


- **Aggrégation des embedding:** perte de l'information séquentielles
- **Concaténation des embedding:** peut devenir très grand
- **Convolution 1D:** bonne idée ! Mais difficile si la taille varie

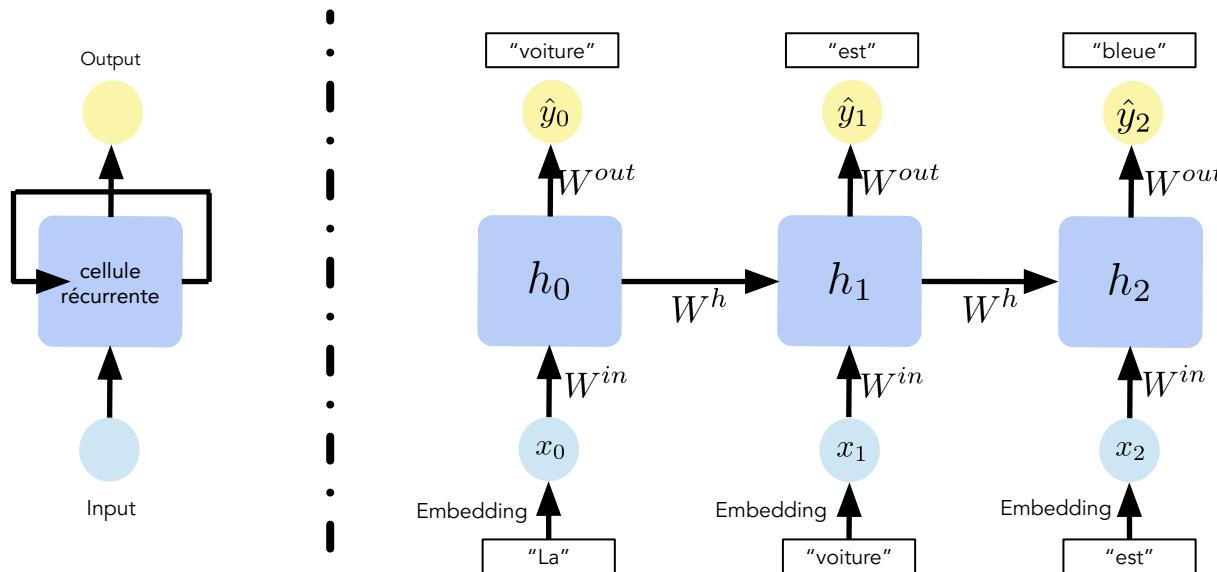
Comment pourrait-on prendre en compte la connectivité temporelle ?



Comment pourrait-on prendre en compte la connectivité temporelle ?



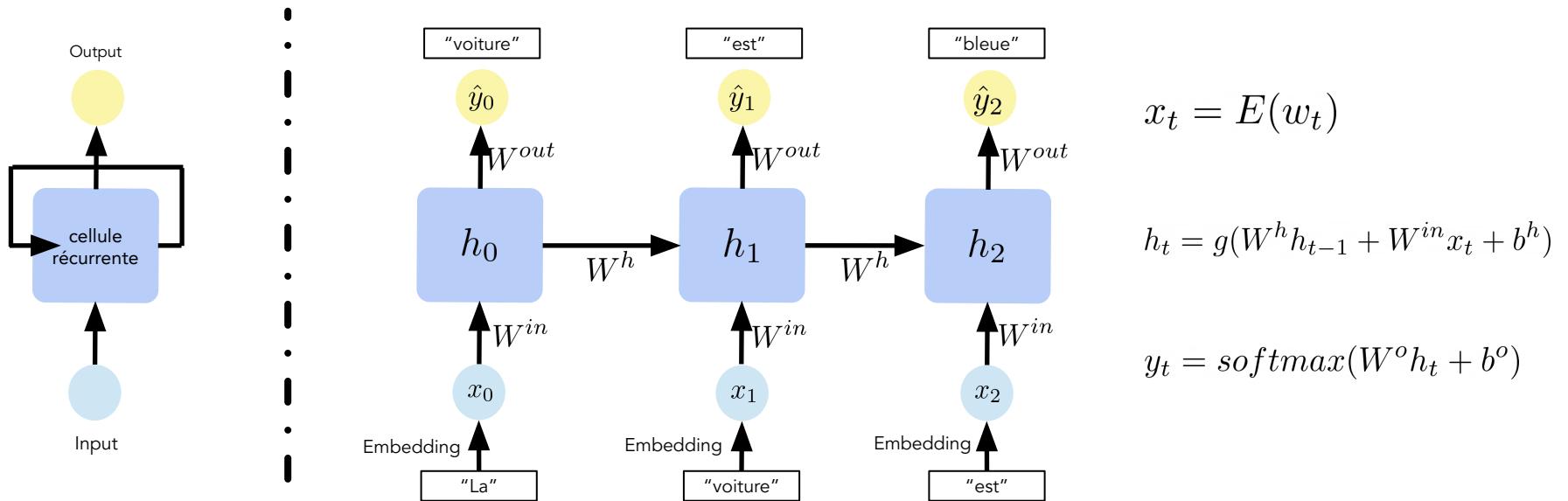
Recurrent Neural Network (RNN)



Input: séquence de mots (w_0, \dots, w_t) en 1-hot

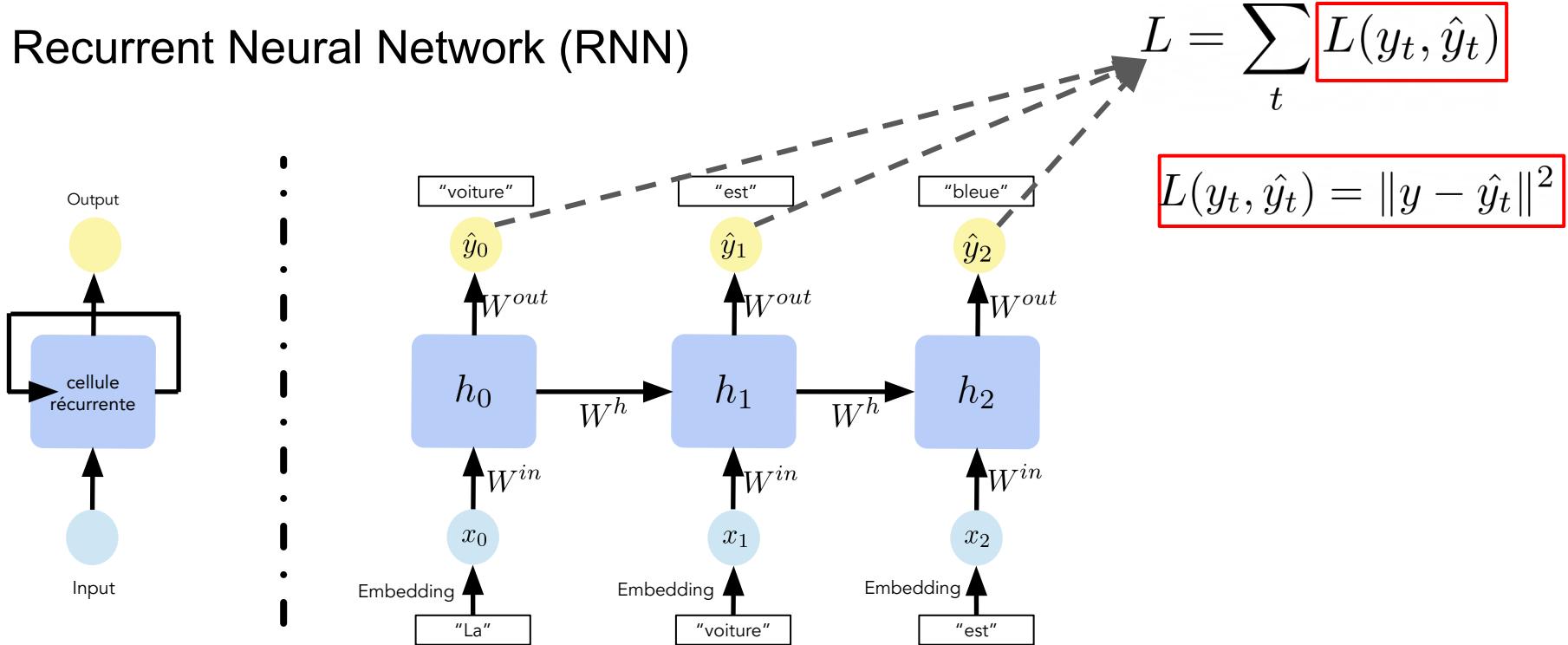
Output: séquence de mots "**shifté**" (w_1, \dots, w_{t+1}) en 1-hot

Recurrent Neural Network (RNN)



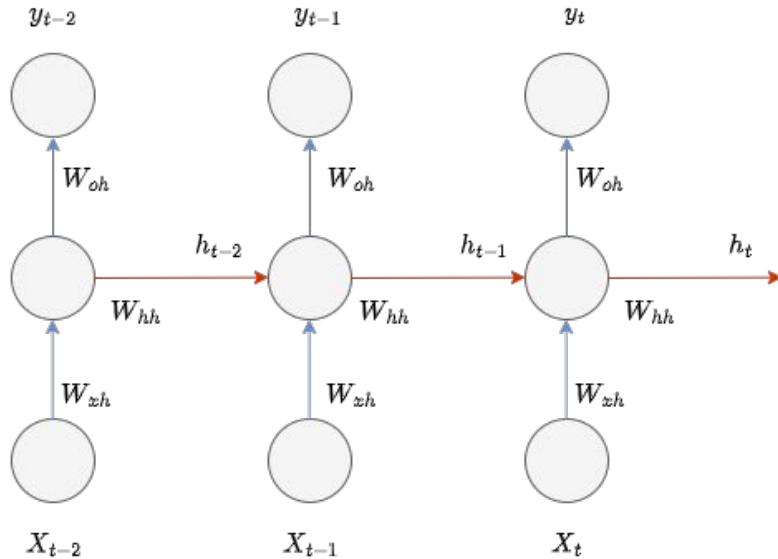
La fonction de coût J est calculé à partir de la somme du coût associé à chaque pas de temps.

Recurrent Neural Network (RNN)



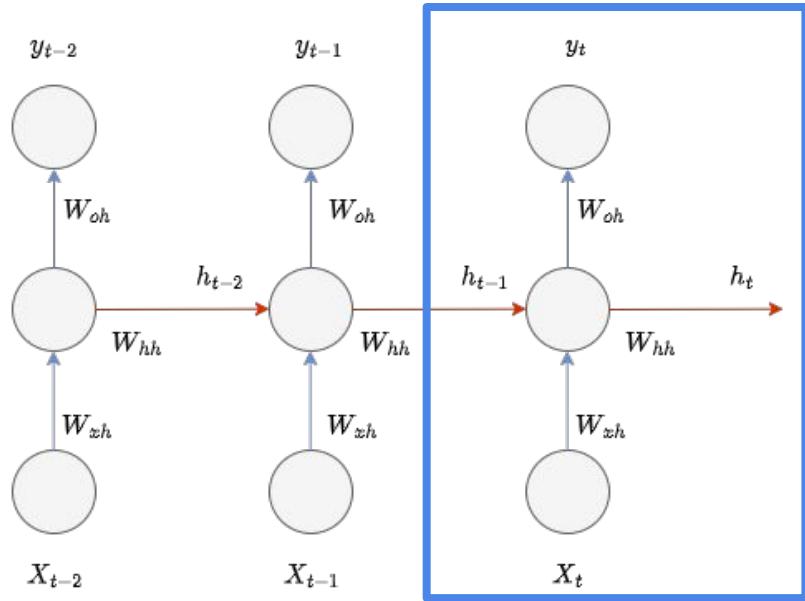
La fonction de coût L est calculé à partir de la somme du coût associé à chaque pas de temps.

Backpropagation through time (BPTT)



Source
image: [pycodemates](#)

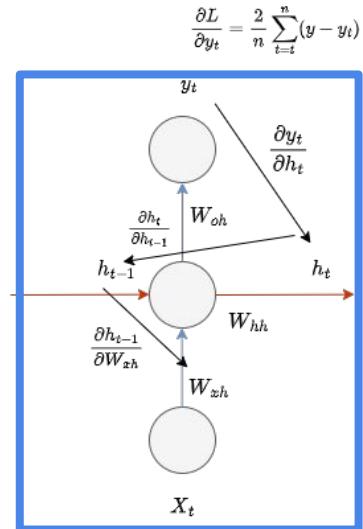
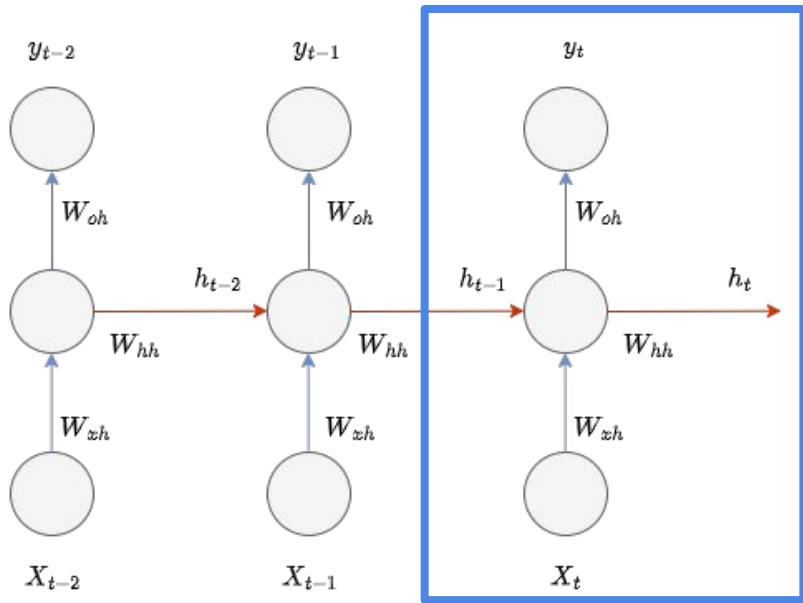
Backpropagation through time (BPTT)



$$\frac{\partial L}{\partial W_{xh}} =$$

Gradient de la fonction de coût par rapport aux paramètres

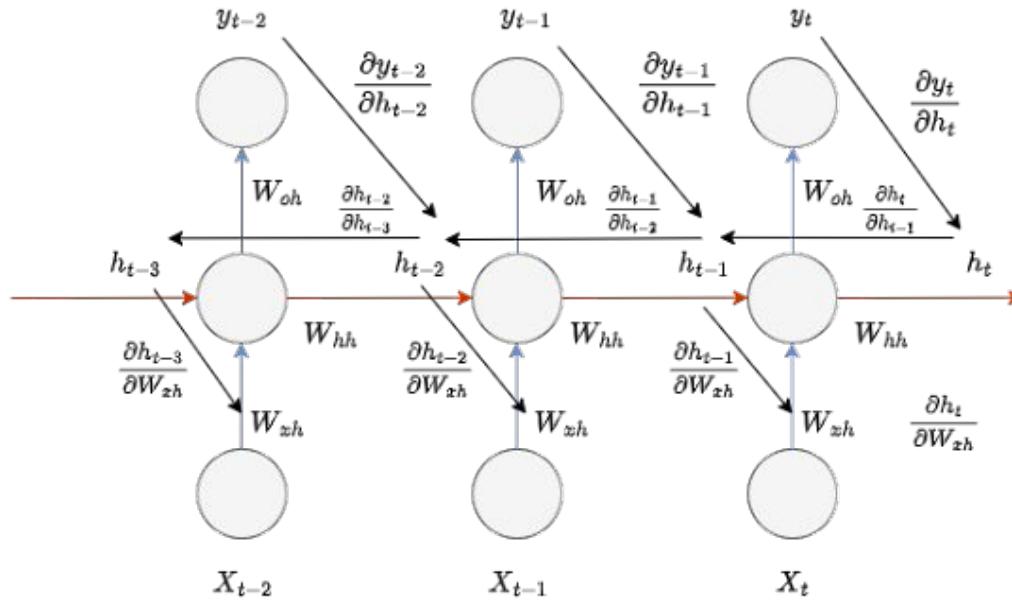
Backpropagation through time (BPTT)



$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}}$$

Gradient de la fonction de coût par rapport aux paramètres

Backpropagation through time (BPTT)

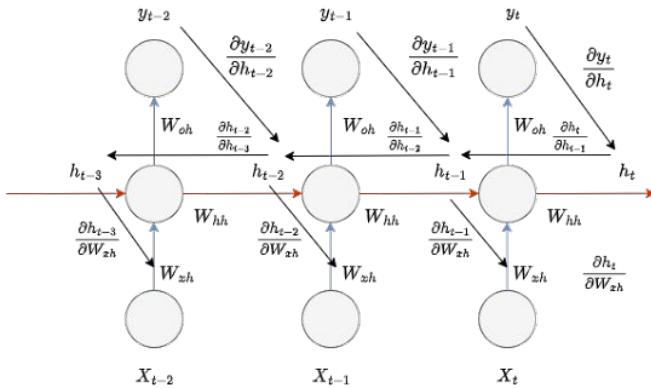


Source
image: [pycodemates](#)

Backpropagation through time (BPTT)

Pour un seul pas de temps, nous avons :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}}$$



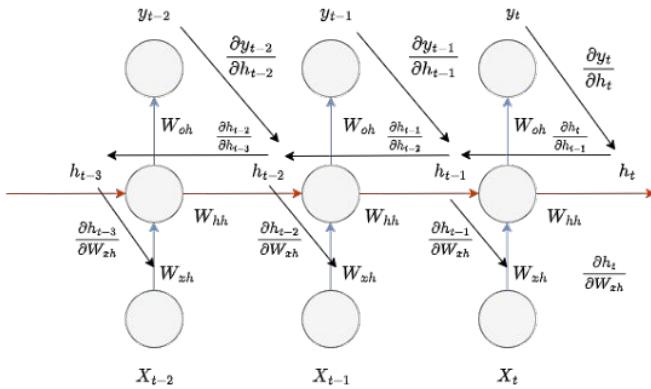
Backpropagation through time (BPTT)

Pour un seul pas de temps, nous avons :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}}$$

Pour deux pas de temps :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-1}} \frac{\partial y_{t-1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{xh}}$$



Backpropagation through time (BPTT)

Pour un seul pas de temps, nous avons :

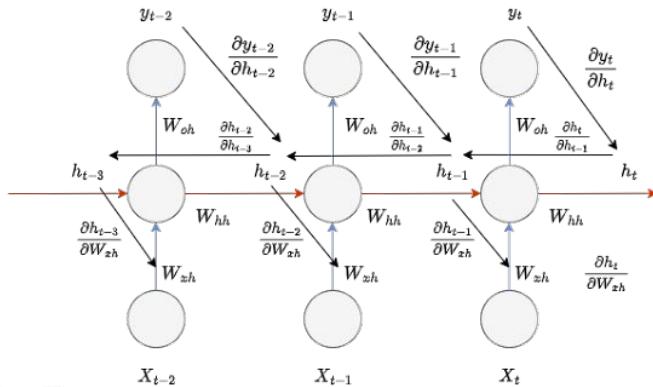
$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}}$$

Pour deux pas de temps :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-1}} \frac{\partial y_{t-1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{xh}}$$

Pour trois pas de temps :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-1}} \frac{\partial y_{t-1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-2}} \frac{\partial y_{t-2}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \frac{\partial h_{t-3}}{\partial W_{xh}}$$



Backpropagation through time (BPTT)

Pour un seul pas de temps, nous avons :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}}$$

Pour deux pas de temps :

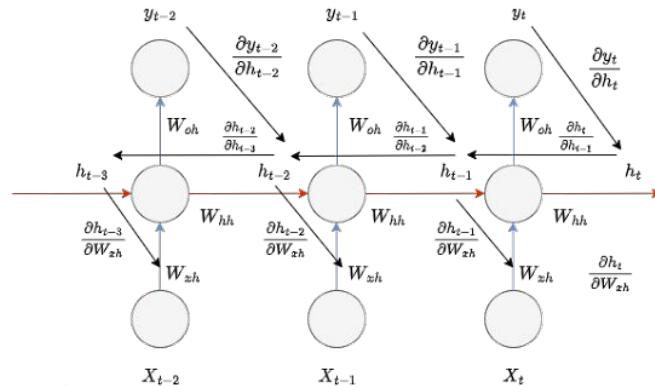
$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-1}} \frac{\partial y_{t-1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{xh}}$$

Pour trois pas de temps :

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-1}} \frac{\partial y_{t-1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{xh}} + \frac{\partial L}{\partial y_{t-2}} \frac{\partial y_{t-2}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \frac{\partial h_{t-3}}{\partial W_{xh}}$$

$$\frac{\partial L}{\partial W_{xh}} = \sum_{i=0}^t \frac{\partial L}{\partial y_{t-i}} \frac{\partial y_{t-i}}{\partial h_{t-i}} \left(\prod_{j=t-i+1}^t \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-i-1}}{\partial W_{xh}}$$

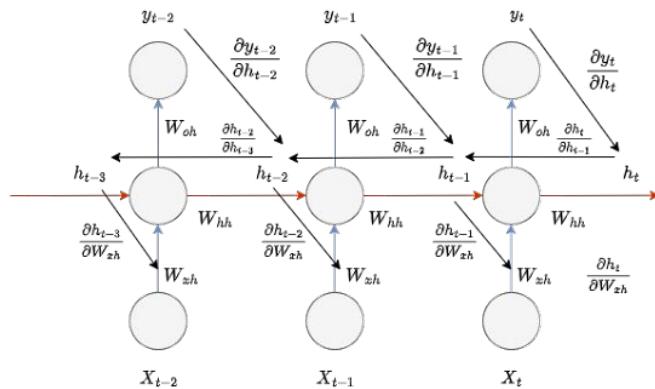
Généralisation à t pas de temps



Backpropagation through time (BPTT)

Pour la dérivée de la fonction de coût par rapport au poids dans les unités cachées, nous avons :

$$\frac{\partial L}{\partial W_{hh}} = \sum_{i=0}^t \frac{\partial L}{\partial y_{t-i}} \frac{\partial y_{t-i}}{\partial h_{t-i}} \left(\prod_{j=t-i+1}^t \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-i-1}}{\partial W_{hh}}$$



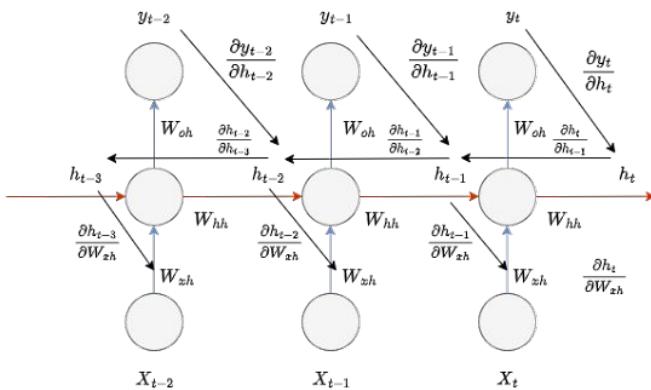
Backpropagation through time (BPTT)

Pour la dérivée de la fonction de coût par rapport au poids dans les unités cachées, nous avons :

$$\frac{\partial L}{\partial W_{hh}} = \sum_{i=0}^t \frac{\partial L}{\partial y_{t-i}} \frac{\partial y_{t-i}}{\partial h_{t-i}} \left(\prod_{j=t-i+1}^t \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-i-1}}{\partial W_{hh}}$$

Concernant la dérivée de la fonction de coût par rapport au poids de sortie, nous avons :

$$\frac{\partial L}{\partial W_{oh}} = \sum_{i=0}^t \frac{\partial L}{\partial y_{t-i}} \frac{\partial y_{t-i}}{\partial W_{oh}}$$



Backpropagation through time (BPTT)

Updating W_{oh} ,

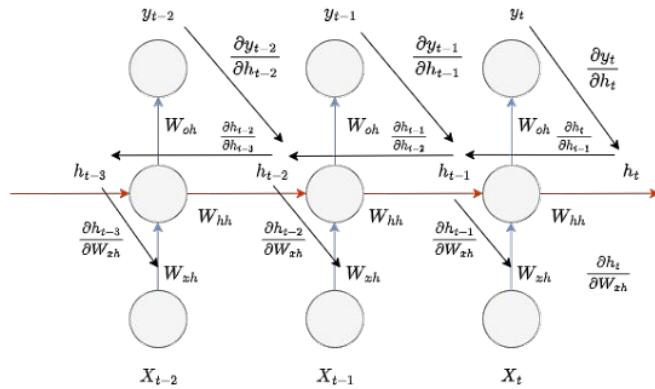
$$W_{oh} \leftarrow W_{oh} - \alpha \cdot \frac{\partial L}{\partial W_{oh}}$$

Updating W_{xh} ,

$$W_{xh} \leftarrow W_{xh} - \alpha \cdot \frac{\partial L}{\partial W_{xh}}$$

Updating W_{hh} ,

$$W_{hh} \leftarrow W_{hh} - \alpha \cdot \frac{\partial L}{\partial W_{hh}}$$



RNN

Avantages

- ◎ Traitement de séquence de taille différente
- ◎ Le nombre de paramètre ne croit pas en fonction de la taille de la séquence
- ◎ Les paramètres sont partagé à travers le temps
- ◎ L'apprentissage tient compte des informations passées

•
•
•
•
•
•
•
•
•

RNN

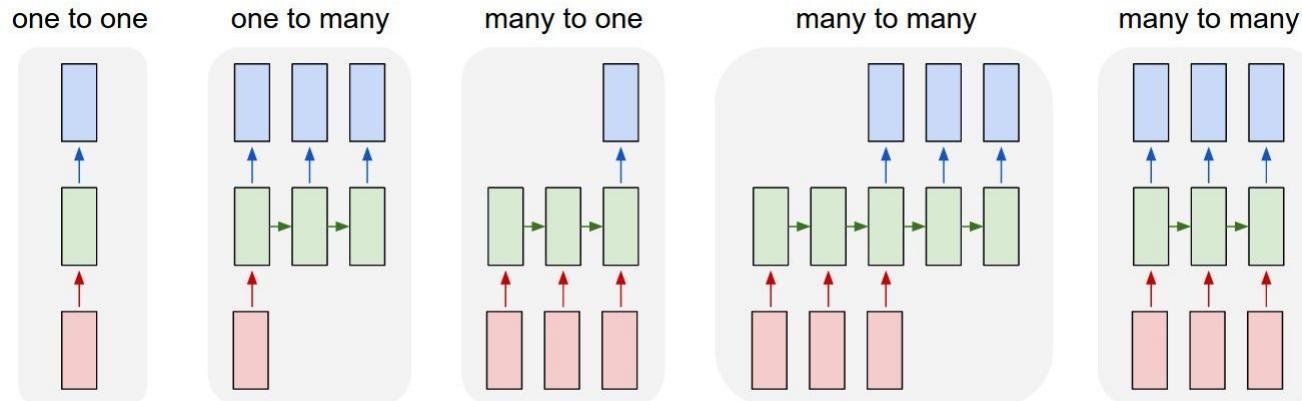
Avantages

- ◎ Traitement de séquence de taille différente
- ◎ Le nombre de paramètre ne croît pas en fonction de la taille de la séquence
- ◎ Les paramètres sont partagé à travers le temps
- ◎ L'apprentissage tient compte des informations passées

Inconvénients

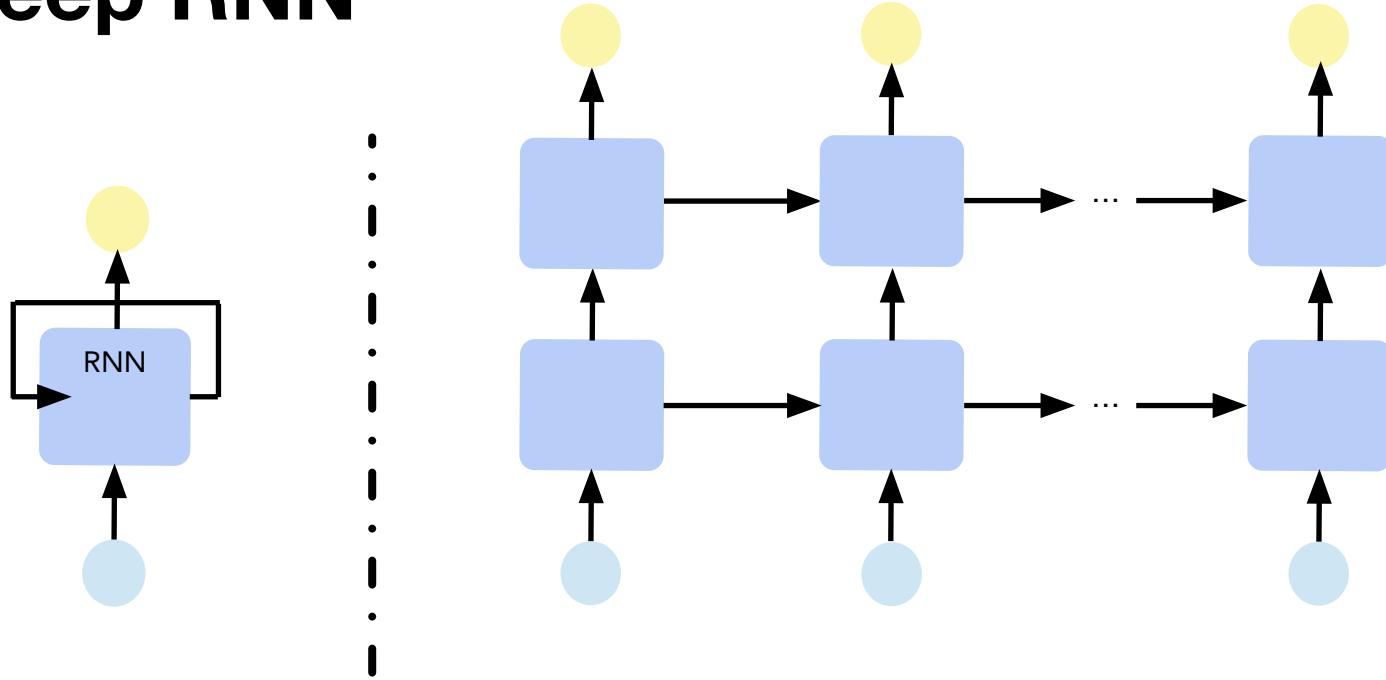
- -
 -
 -
 -
 -
 -
 -
 -
 -
- ◎ Entraînement lent
 - ◎ Difficulté de prendre en compte une information lointaine
 - ◎ Optimisation délicate (vanishing/exploding gradient)

RNN pour la modélisation de séquences



Source image: [Andrei
Karpthy](#)

Deep RNN



RNN avec Pytorch

```
class RNNModel(nn.Module): 1usage new *
    def __init__(self, input_size, hidden_size, output_size, num_layers): new *
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # RNN layer
        self.rnn = nn.RNN(*args: input_size, hidden_size, num_layers, batch_first=True)

        # Output layer (fully connected layer)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x): new *
        # Init hidden states à zéro
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Passe par le RNN
        out, _ = self.rnn(x, h0)

        # fully connected layer
        out = self.fc(out[:, -1, :]) # On ne conserve que le dernier temps
        return out
```

Initialise les **hidden states**

RNN

Fully connected

1er hidden initialisé à zéro

Vanishing gradient

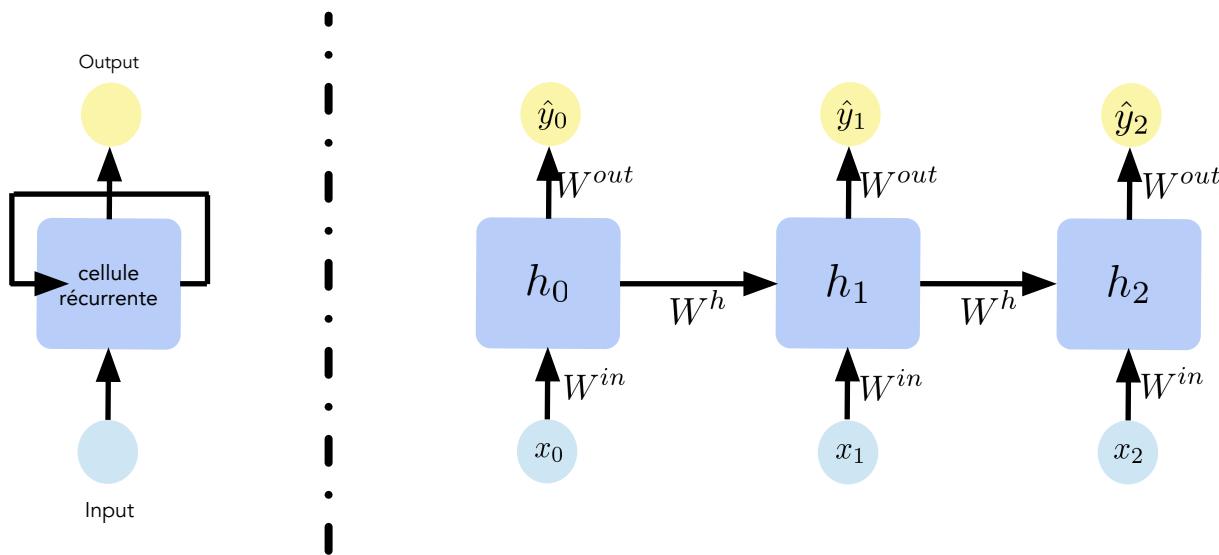
- Les problèmes de vanishing gradient sont souvent rencontré lors de d'entraînement des réseaux de neurones récurrents puisqu'il y a beaucoup de steps pour rétro propager.
- La raison est qu'il est difficile pour ce type d'architecture de capturer des dépendances à long terme du au fait de la multiplicativité du gradient qui peut exponentiellement augmenter/diminuer en fonction du nombre de couche.

Vanishing gradient

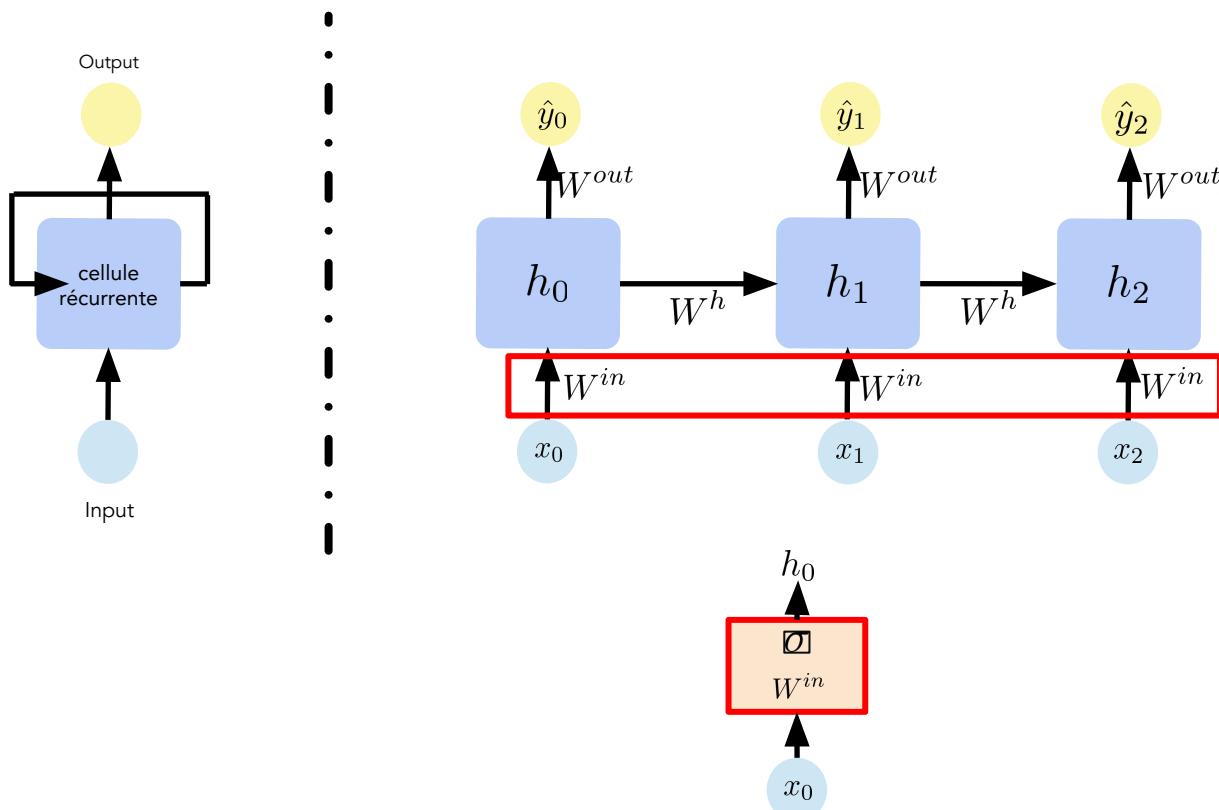
- Les problèmes de vanishing gradient sont souvent rencontré lors de d'entraînement des réseaux de neurones récurrents puisqu'il y a beaucoup de steps pour rétro propager.
- La raison est qu'il est difficile pour ce type d'architecture de capturer des dépendances à long terme dû au fait de la multiplicativité du gradient qui peut exponentiellement augmenter/diminuer en fonction du nombre de couche.

→ Solution: “**Gating**” pour contrôler le flot d'information.

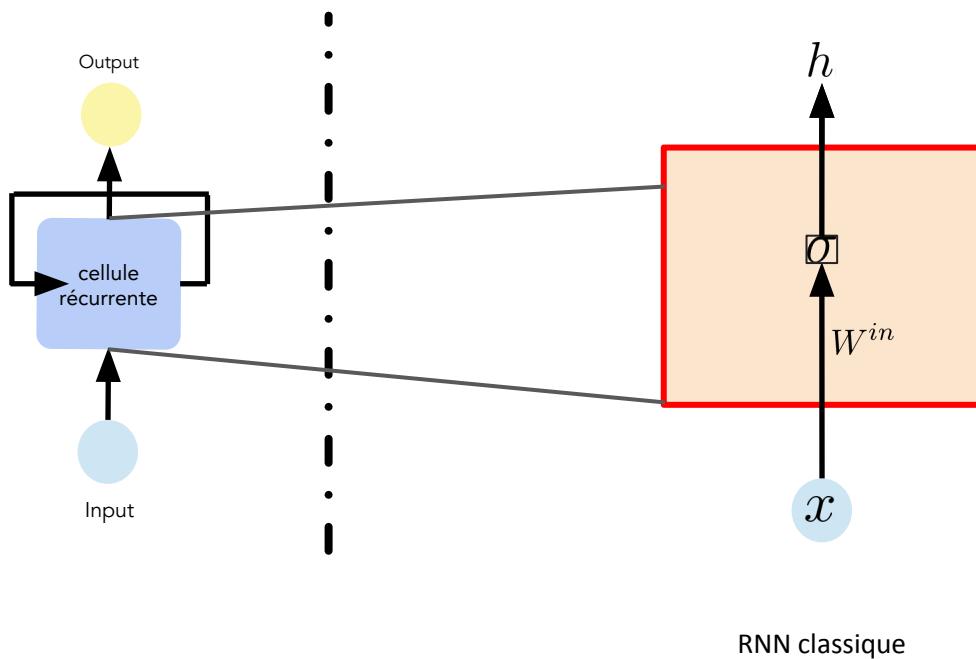
Long-short term memory (LSTM)



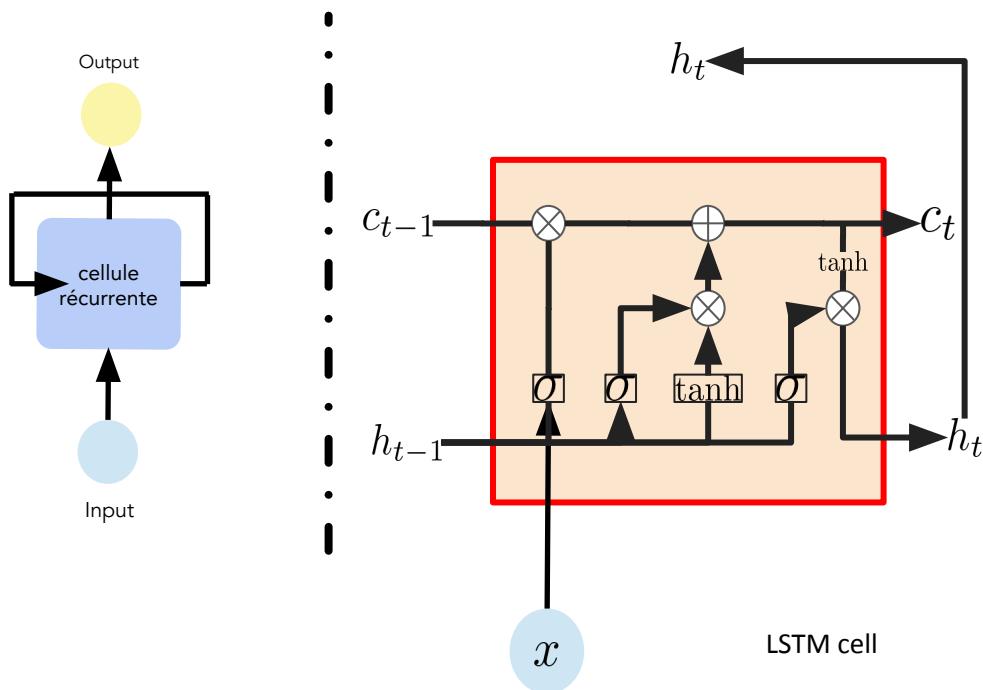
Long-short term memory (LSTM)



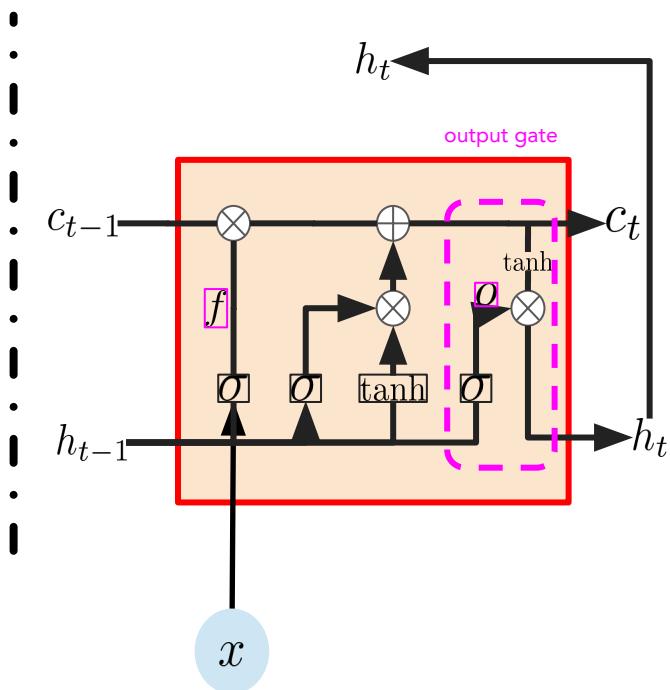
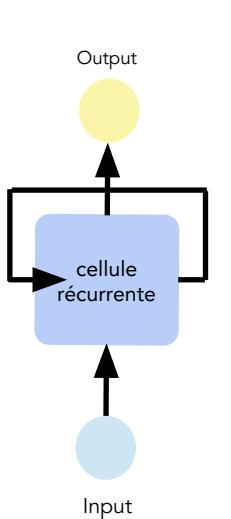
Long-short term memory (LSTM)



Long-short term memory (LSTM)

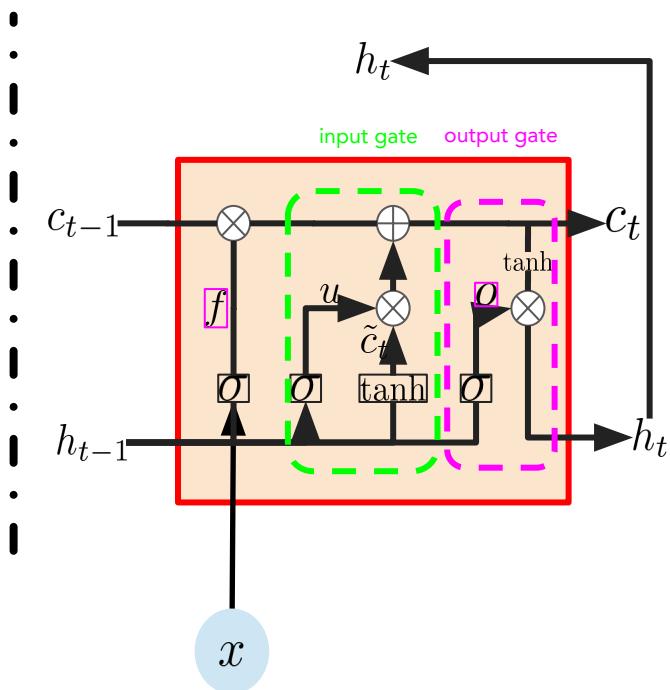
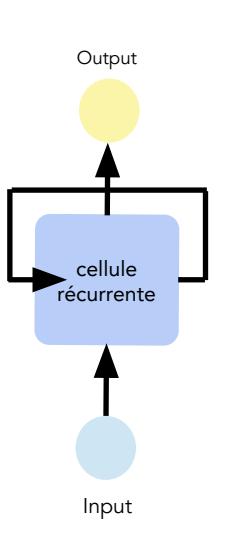


Long-short term memory (LSTM)



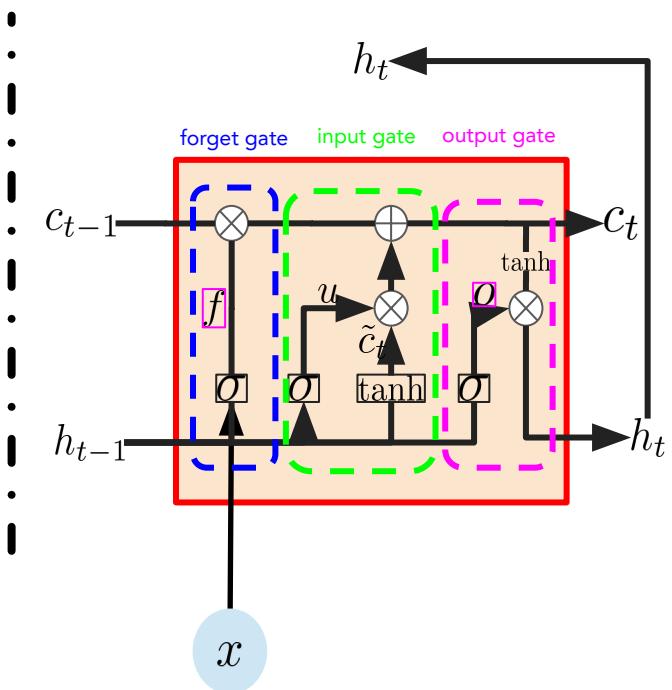
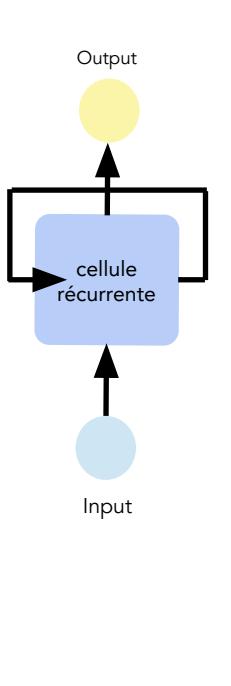
$$\begin{aligned} c_t &= f \odot c_{t-1} + u \odot \tilde{c}_t \\ o &= \sigma(W^o h_{t-1} + I^o x_t + b^o) \\ h_t &= o \odot \tanh(c_t) \end{aligned}$$

Long-short term memory (LSTM)



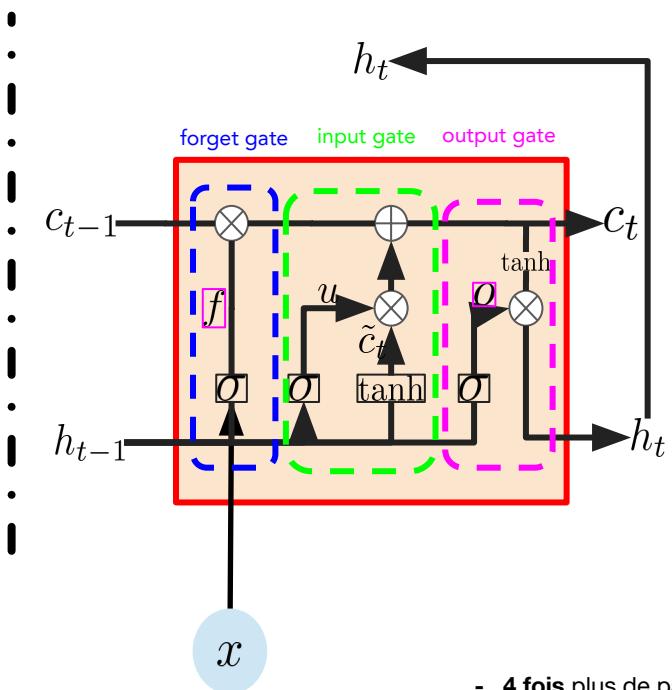
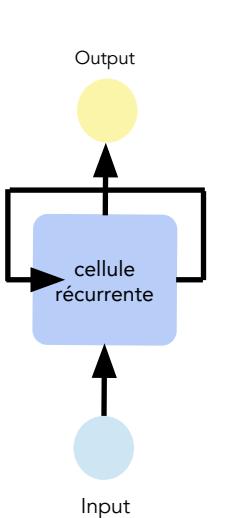
$$\begin{cases} u = \sigma(W^u h_{t-1} + I^u x_t + b^u) \\ \tilde{c}_t = \tanh(W^c h_{t-1} + I^c x_t + b^c) \\ c_t = f \odot c_{t-1} + u \odot \tilde{c}_t \\ o = \sigma(W^o h_{t-1} + I^o x_t + b^o) \\ h_t = o \odot \tanh(c_t) \end{cases}$$

Long-short term memory (LSTM)



$$\boxed{f = \sigma(W^f h_{t-1} + I^f x_t + b^f)}$$
$$\boxed{u = \sigma(W^u h_{t-1} + I^u x_t + b^u)}$$
$$\boxed{\tilde{c}_t = \tanh(W^c h_{t-1} + I^c x_t + b^c)}$$
$$\boxed{c_t = f \odot c_{t-1} + u \odot \tilde{c}_t}$$
$$\boxed{o = \sigma(W^o h_{t-1} + I^o x_t + b^o)}$$
$$\boxed{h_t = o \odot \tanh(c_t)}$$

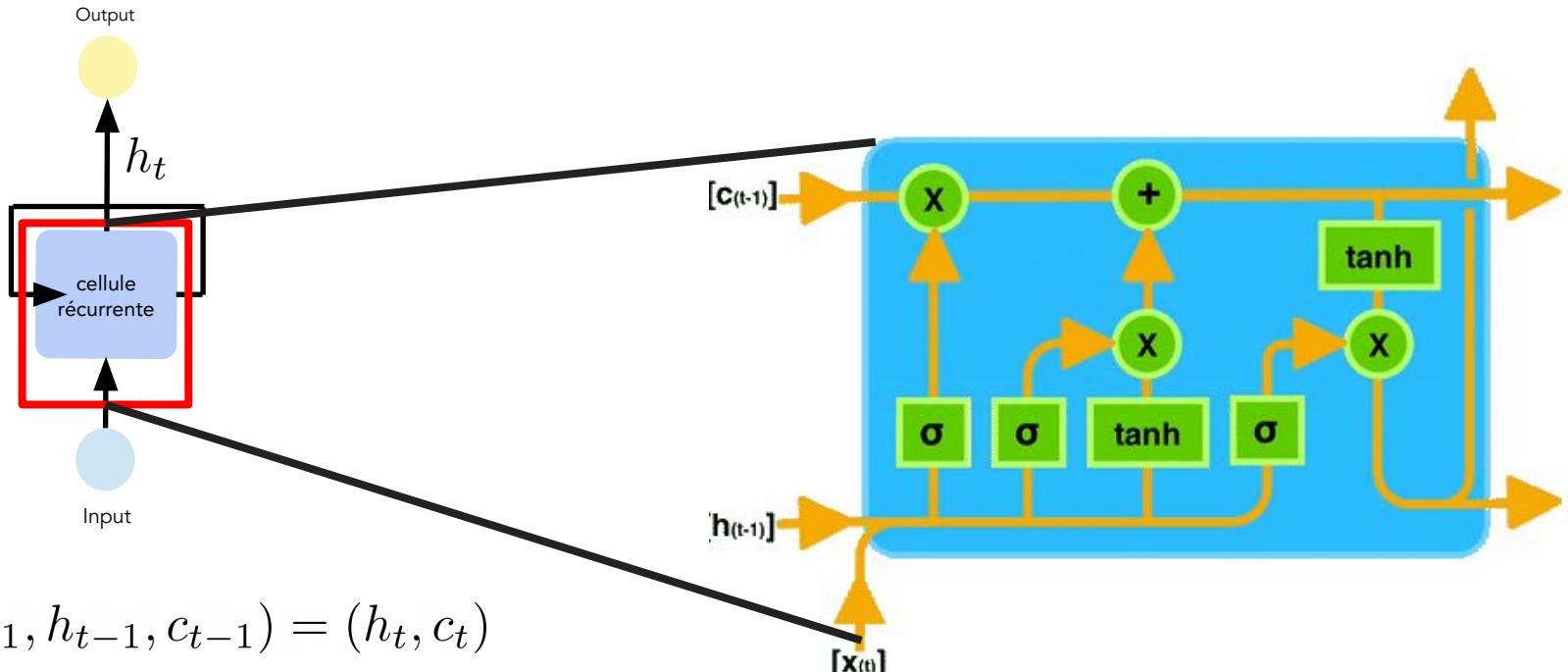
Long-short term memory (LSTM)



$$\begin{aligned}
 f &= \sigma(W^f h_{t-1} + I^f x_t + b^f) \\
 u &= \sigma(W^u h_{t-1} + I^u x_t + b^u) \\
 \tilde{c}_t &= \tanh(W^c h_{t-1} + I^c x_t + b^c) \\
 c_t &= f \odot c_{t-1} + u \odot \tilde{c}_t \\
 o &= \sigma(W^o h_{t-1} + I^o x_t + b^o) \\
 h_t &= o \odot \tanh(c_t)
 \end{aligned}$$

- 4 fois plus de paramètres qu'un RNN classique
- Gère le flot d'info par **gating**
- Encore état de l'art pour certains problèmes

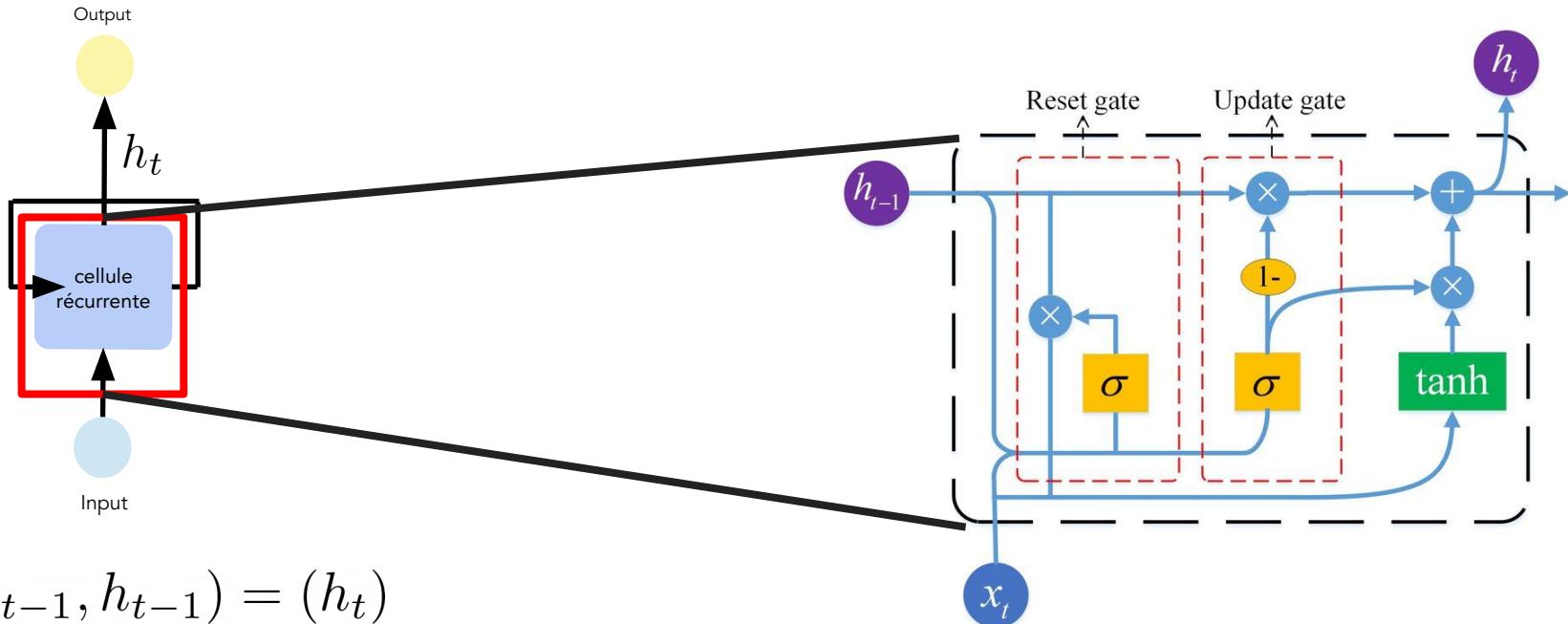
Long-short term memory (LSTM)



$$lstm(x_{t-1}, h_{t-1}, c_{t-1}) = (h_t, c_t)$$

```
# LSTM layer
self.lstm = nn.LSTM(*args: input_size, hidden_size, num_layers, batch_first=True)
```

Gated Recurrent Unit (GRU)



$$GRU(x_{t-1}, h_{t-1}) = (h_t)$$

```
# Define the GRU layer
self.gru = nn.GRU(*args: input_size, hidden_size, num_layers, batch_first=True)
```

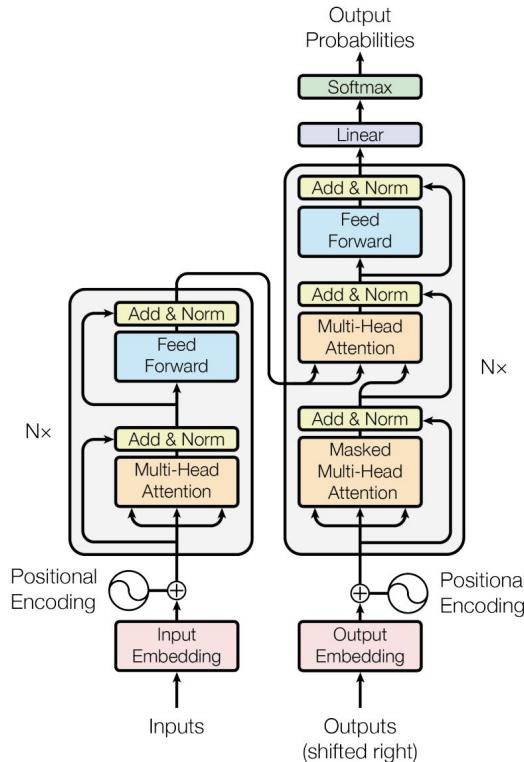
LSTM and GRU

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

Characterization	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>} , x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>} , x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

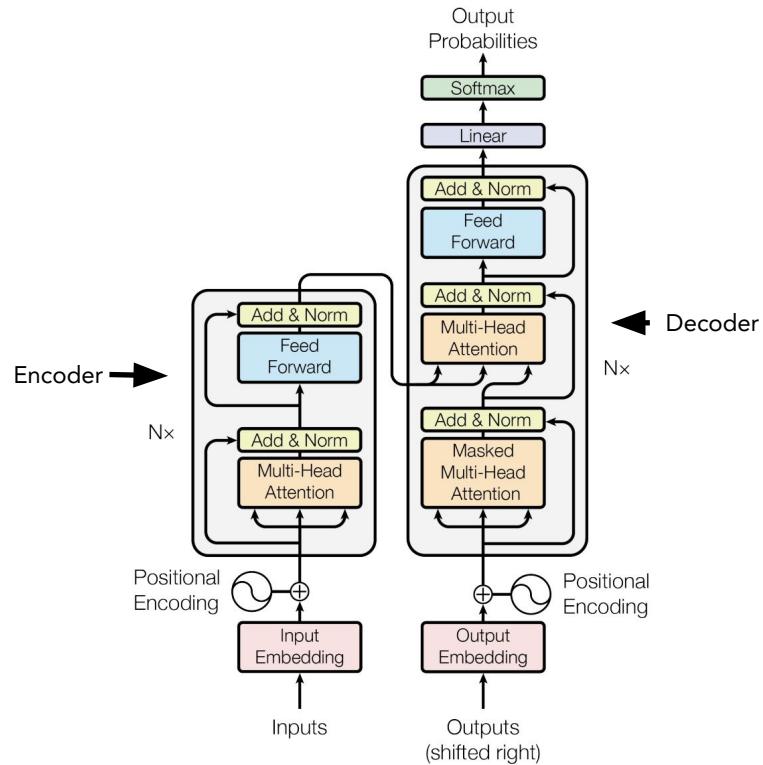
Source image: [Stanford
edu](https://stanford.edu)

L'architecture Transformer



Vaswani et al. Attention Is All You Need, 2017.
NeurIPS.

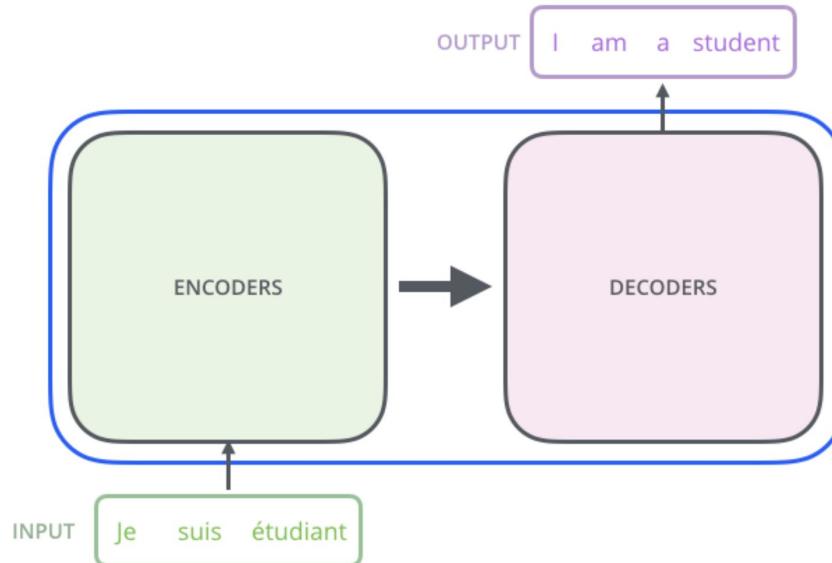
L'architecture Transformer



Vaswani et al. Attention Is All You Need, 2017.
NeurIPS.

Le Transformer

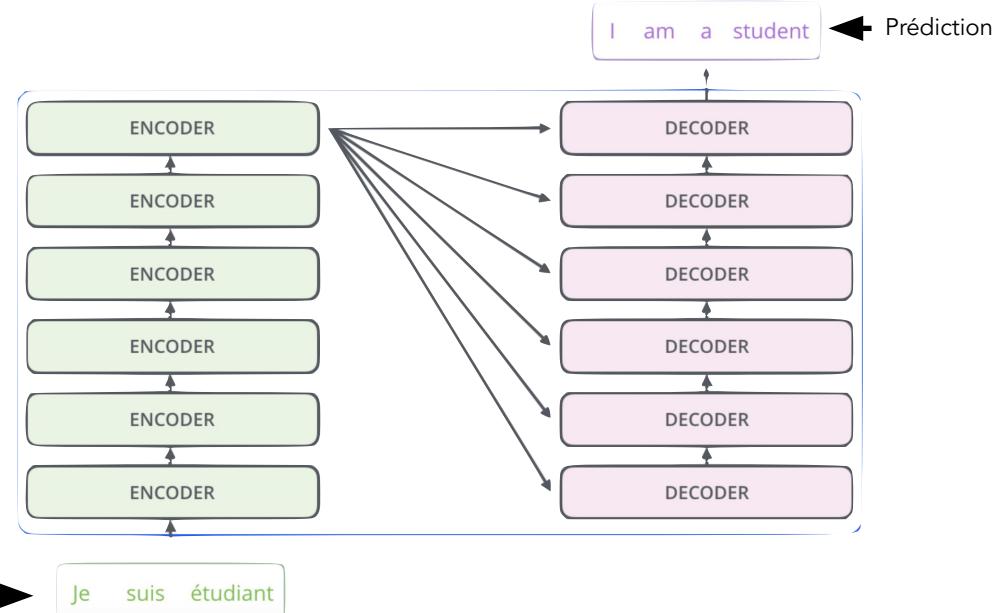
- Le Transformer a initialement été conçu pour la tâche de traduction automatique
- Dans l'exemple ci-dessous, le modèle va apprendre à traduire un texte du français vers l'anglais



Source image: [Jalammar tutorial](#)

Le Transformer

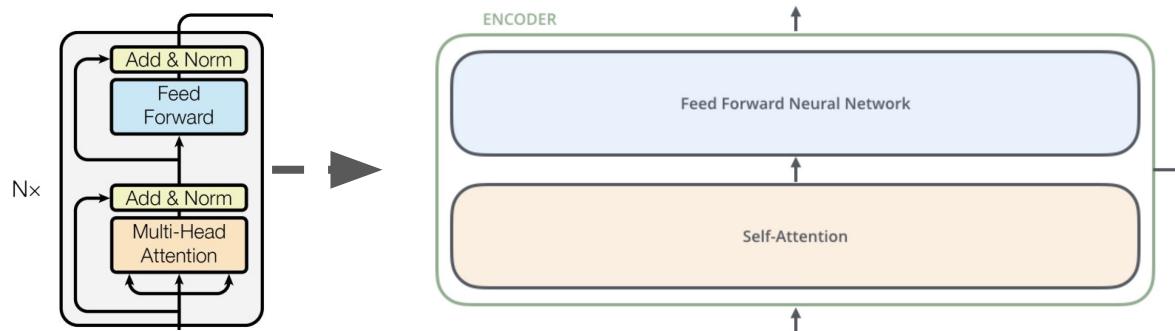
La partie responsable de l'encodage est composée d'un ensemble d'encodeurs, tandis que la partie responsable du décodage est composée d'un ensemble de décodeurs.



Source image: [Jalammar tutorial](#)

Zoom au niveau de l'Encoder

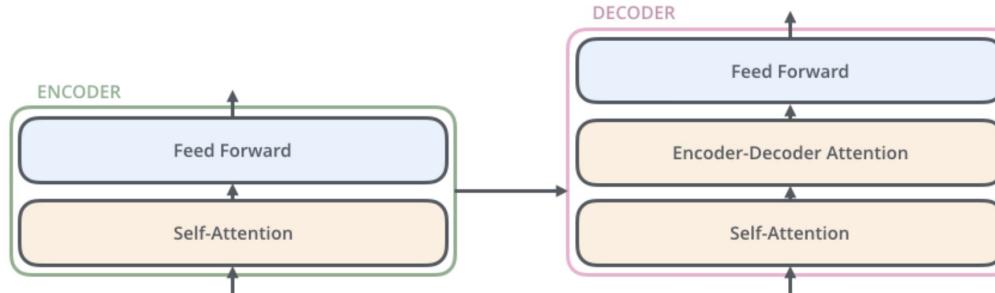
- Tous les encodeurs sont identique en terme de structure. Chacun étant composé d'une couche de Self-Attention et d'une couche de Feed Forward Neural Network
- Les différents encoders ne partagent pas les mêmes paramètres
- Chaque encodeur prend en entrée la sortie de l'encodeur précédent
- La sortie de la couche de Self-Attention est envoyé à la couche de Feed Forward. Le dernier est appliqué à chaque position indépendamment.



Source image: [Jalammar tutorial](#)

Zoom au niveau du Decoder

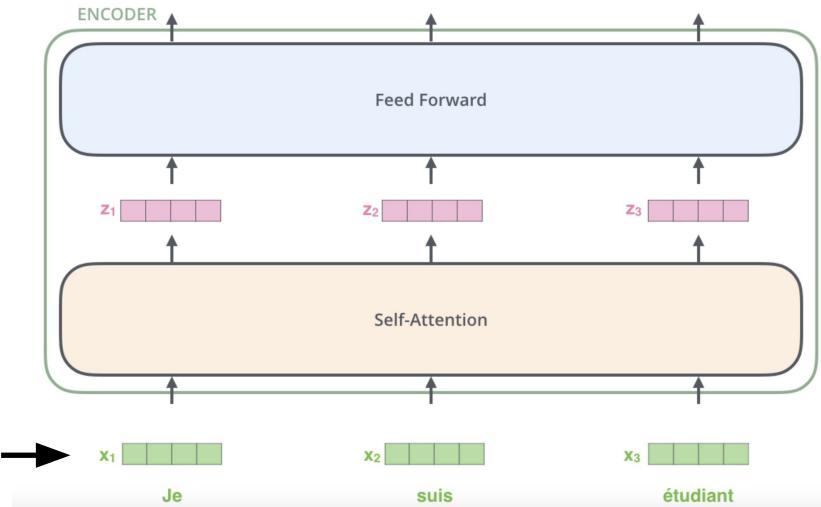
- Le decoder quand à lui est composé comme pour l'encoder d'une couche de Self-Attention et d'une couche feed forward avec entre les deux une couche de cross-attention encoder-decoder.
- La couche de cross-attention encoder-decoder permet au decoder de se concentrer sur les parties importantes de la séquence de données en entrée.



Source image: [Jalammar tutorial](#)

Self-Attention

- La couche de Self-Attention va transformer la représentation numérique de chaque mot en une nouvelle représentation qui prendra en compte la relation du mot avec les autres mots de la séquence



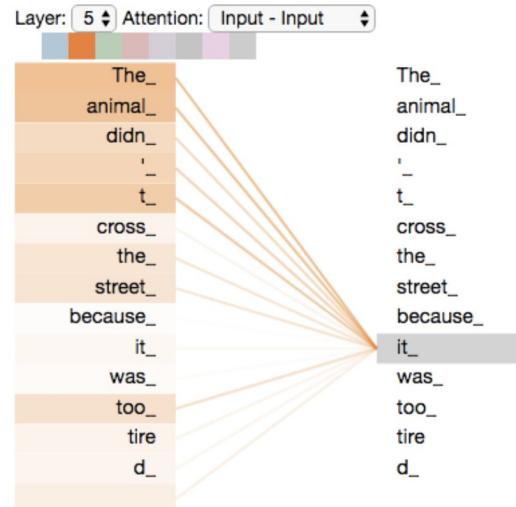
Transformation numérique des mots en
utilisant un algorithme d'embedding

Source image: [Jalammar tutorial](#)

Self-Attention intuition

- Le modèle traite chaque mot, c'est-à-dire chaque position dans la séquence d'entrée. La couche de Self-Attention lui permet d'examiner d'autres positions au sein de cette séquence, afin de rechercher des indices susceptibles d'améliorer l'encodage précis du mot en question.

"The animal didn't cross the street because **it** was too tired"



Source image: [Jalammar tutorial](#)

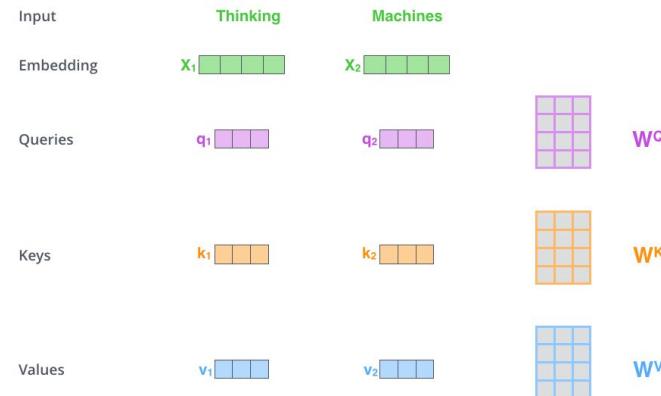
Self-Attention intuition

- Le modèle traite chaque mot, c'est-à-dire chaque position dans la séquence d'entrée. La couche de Self-Attention lui permet d'examiner d'autres positions au sein de cette séquence, afin de rechercher des indices susceptibles d'améliorer l'encodage précis du mot en question.

<https://www.youtube.com/watch?v=eMlx5fFNoYc&t=5s>

Calcul du Self-Attention

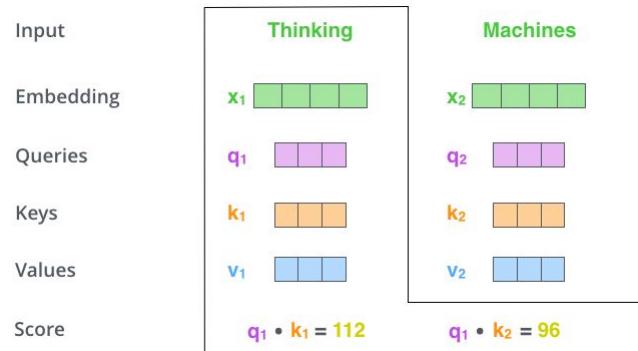
- La première étape consiste à partir du vecteur d'encodage représentant un mot donné pour le transformer en trois vecteurs Q, K et V
- Ces trois vecteurs sont créés en multipliant notre vecteur d'encodage par trois matrices de paramètres. Ces matrices appelées sont optimisé durant l'entraînement
- Généralement ces vecteurs sont de dimension inférieur à celle du vecteur d'encodage.



Source image: [Jalammar tutorial](#)

Calcul du Self-Attention

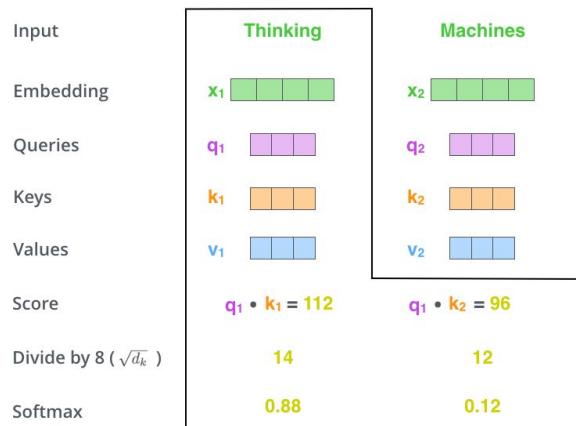
- Le vecteur Q représente le mot sur lequel l'attention est portée et aide le modèle à comprendre sur quoi il devrait se concentrer dans d'autres parties de la séquence
- Le vecteur K aide à établir des relations entre le vecteur Q et d'autres mots de la séquence. Il fournit des informations sur la manière dont chaque mot est lié au vecteur Q.
- Le vecteur V contient les informations relatives au contenu des mots dans la séquence. Une fois que les scores d'attention (calculés à l'aide de Q et K) ont été obtenus, ils sont utilisés pour pondérer les vecteurs V obtenu pour chaque mot, afin de déterminer l'importance du contenu de chaque mot par rapport au vecteur Q.



Source image: [Jalammar tutorial](#)

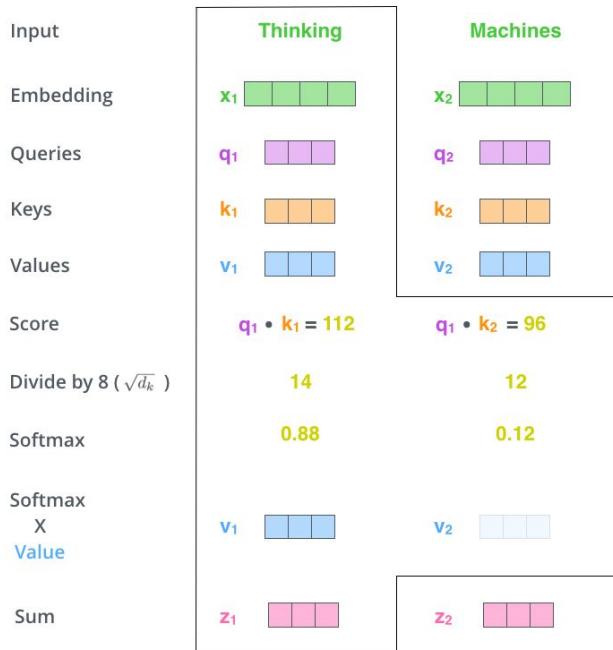
Calcul du Self-Attention

- La fonction Softmax permet de déterminer, en termes de probabilité, la contribution de chaque mot par rapport à un mot donné. Comme on peut le voir ci-dessous, généralement, le mot lui-même aura la probabilité la plus élevée. Néanmoins, il est toujours utile d'examiner les autres mots pertinents pour le mot en question



Source image: [Jalammar tutorial](#)

Calcul du Self-Attention



Source image: [Jalammar tutorial](#)

Calcul matriciel du Self-Attention

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W}^Q \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{Q} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

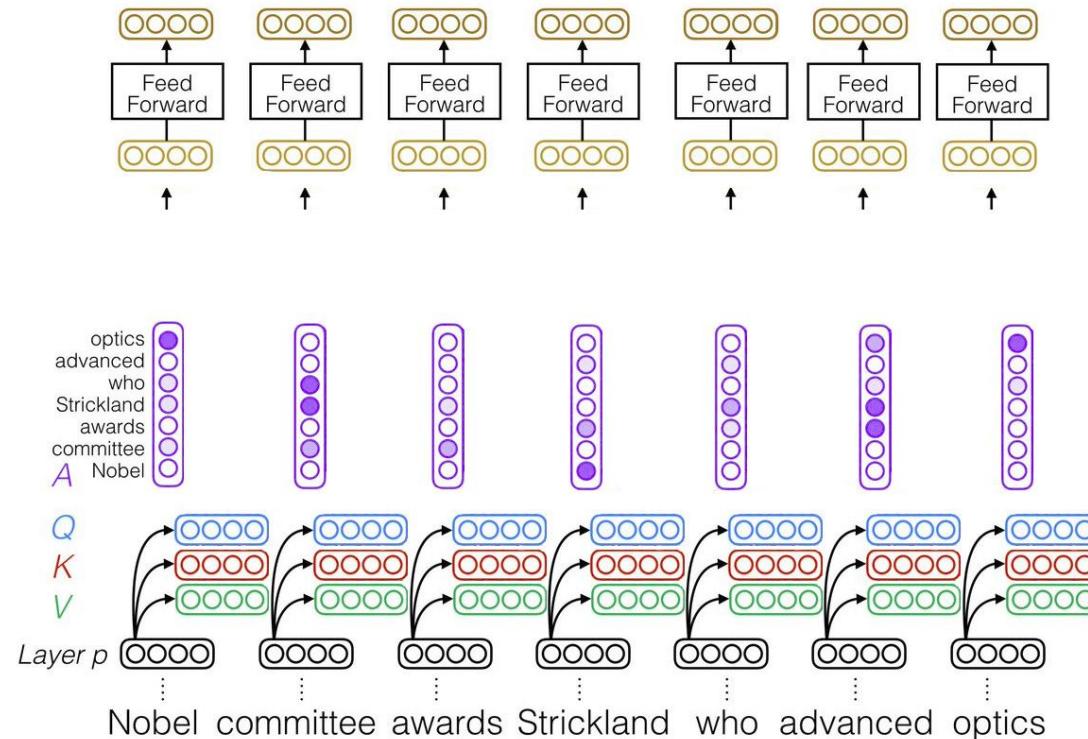
$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W}^K \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{K} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W}^V \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{V} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

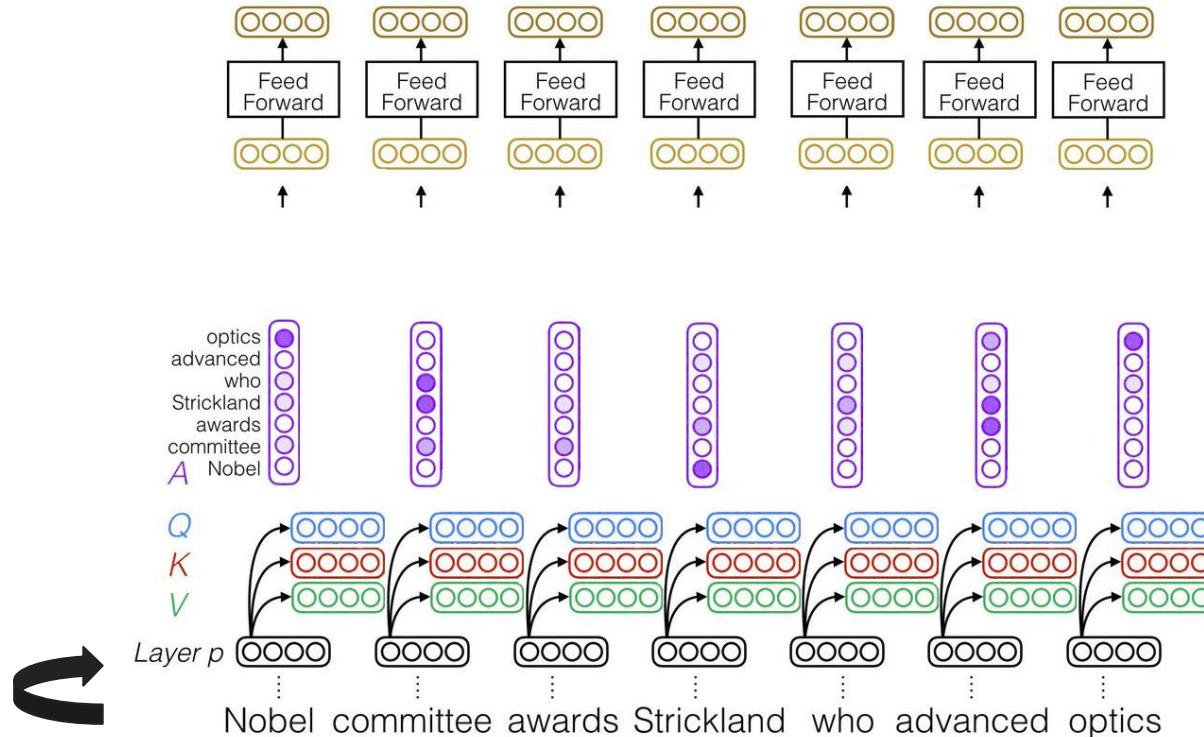
$$\text{softmax} \left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) = \mathbf{Z}$$
$$\mathbf{Z} \times \mathbf{V}$$

Source image: [Jalammar tutorial](#)

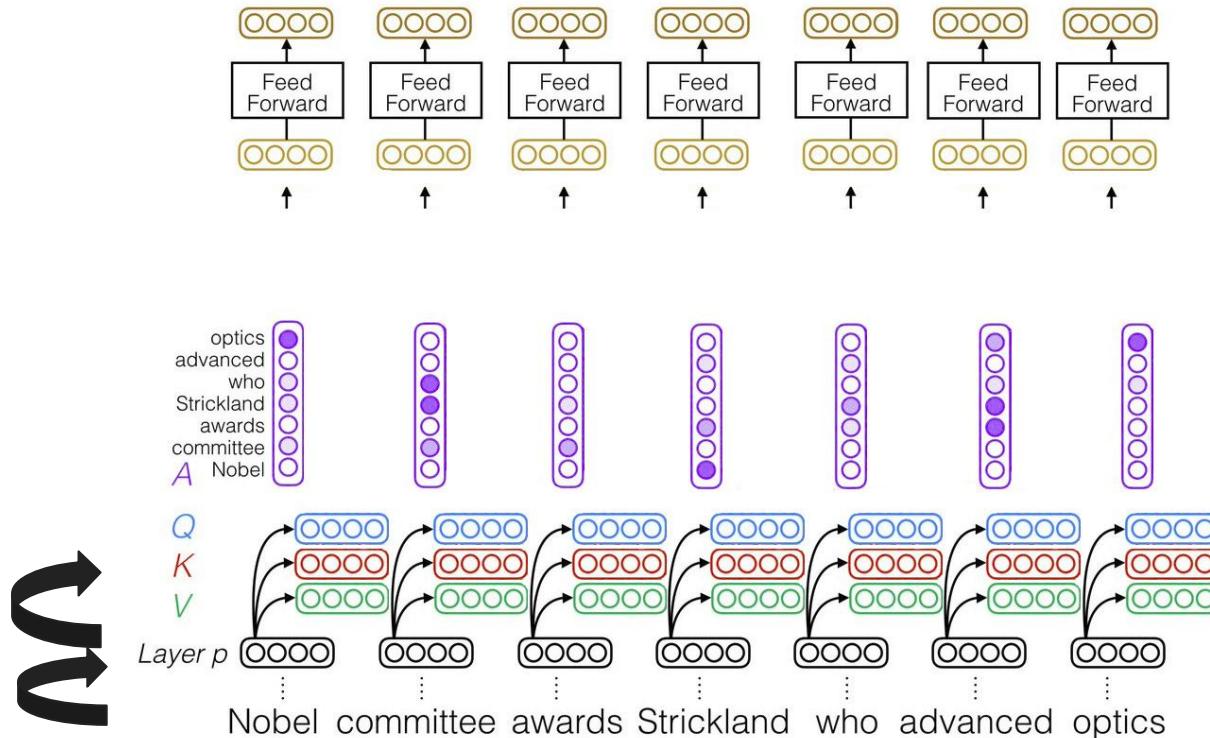
Single-head Self-Attention



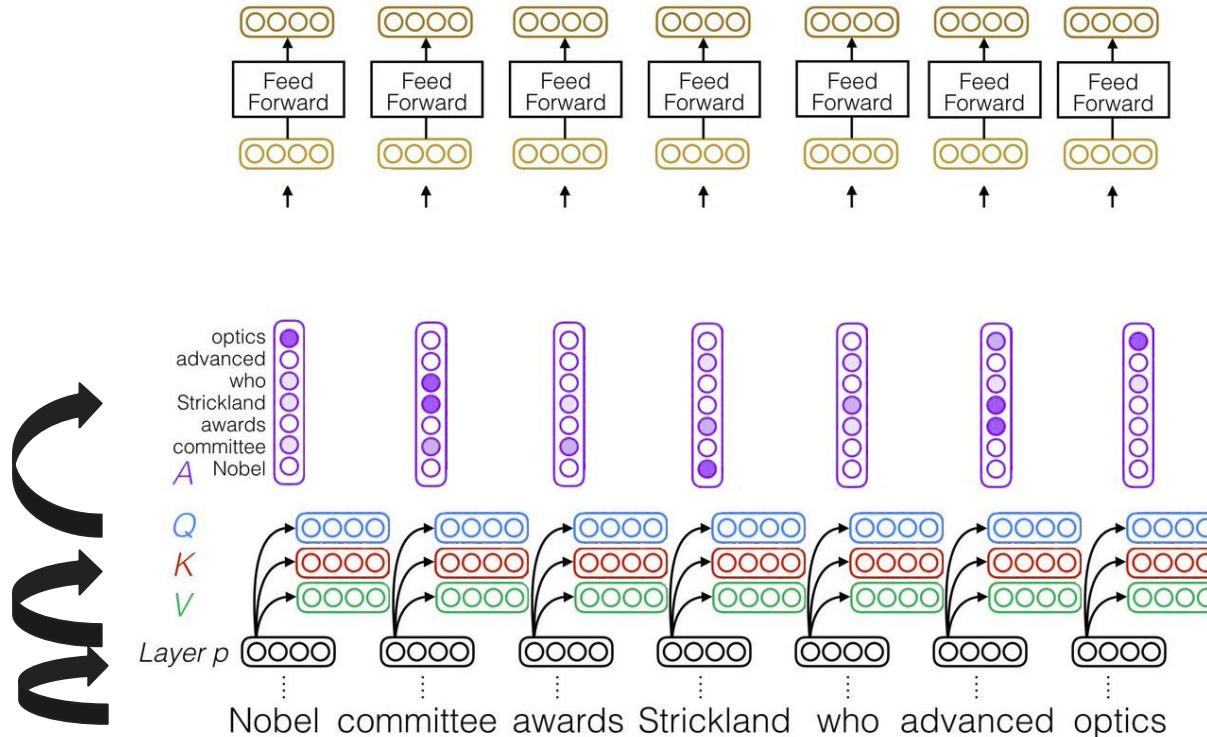
Single-head Self-Attention



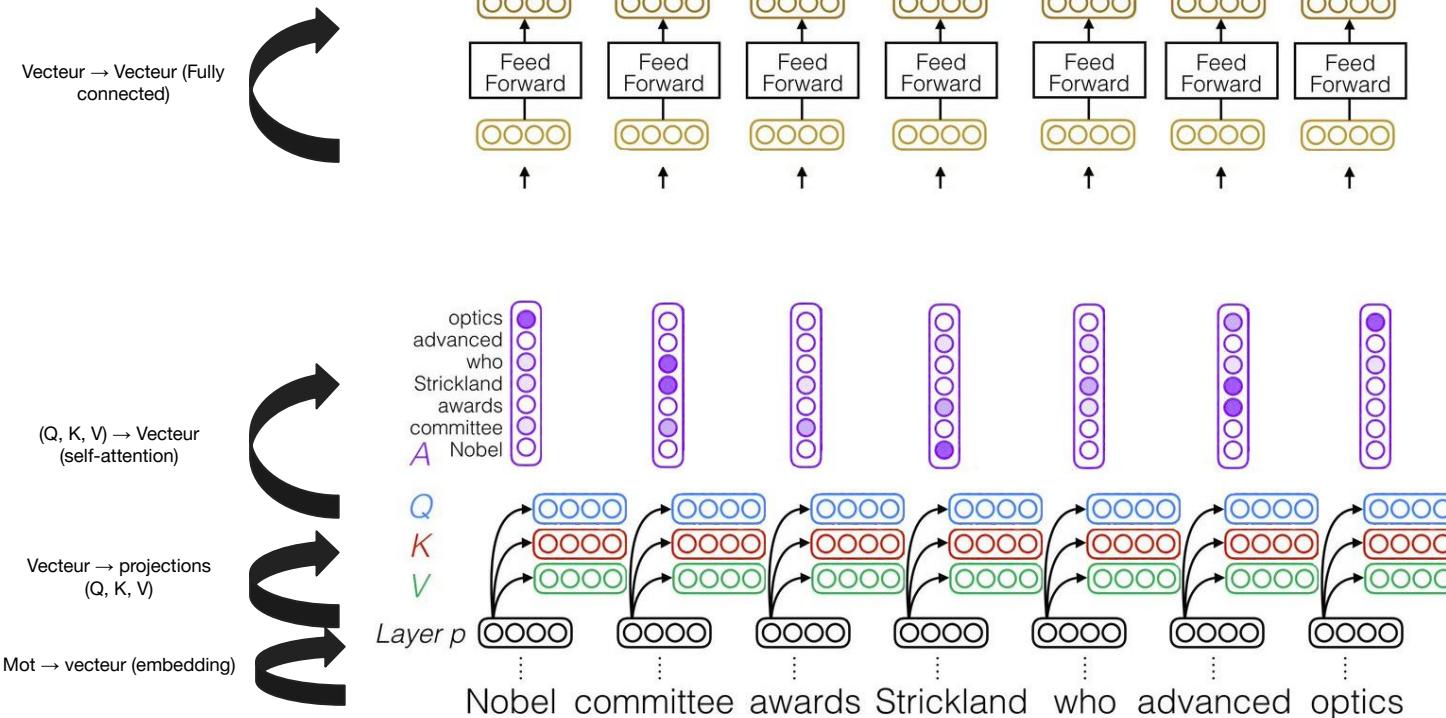
Single-head Self-Attention



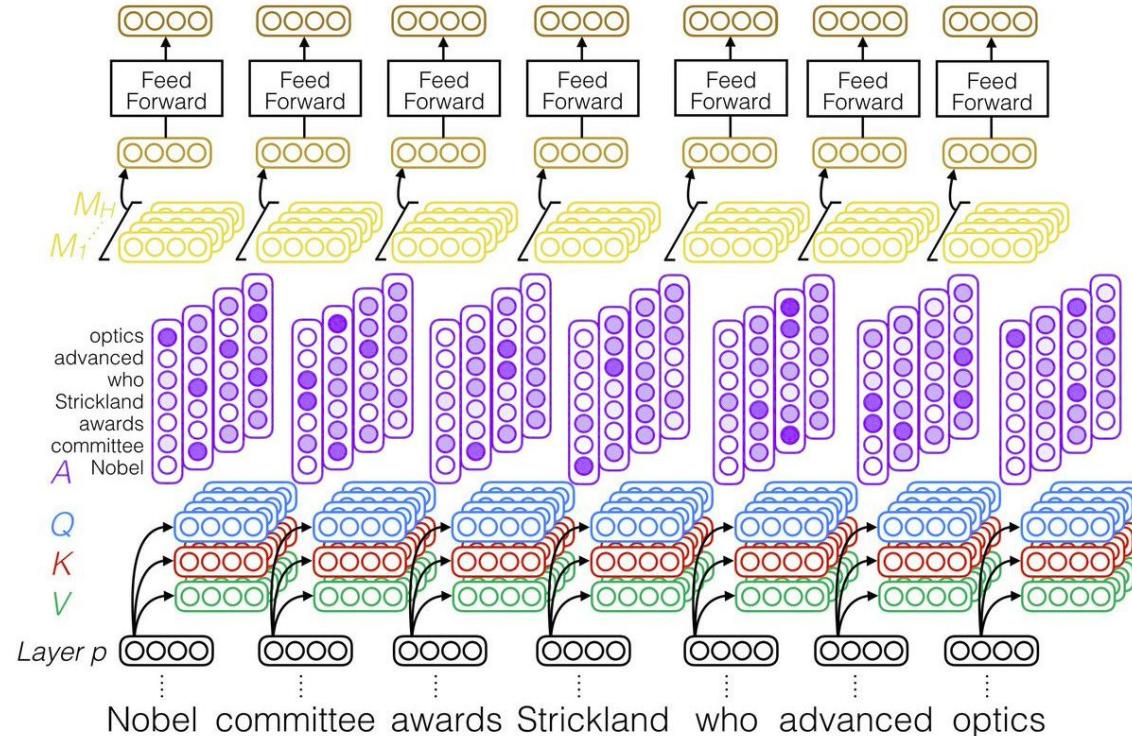
Single-head Self-Attention



Single-head Self-Attention

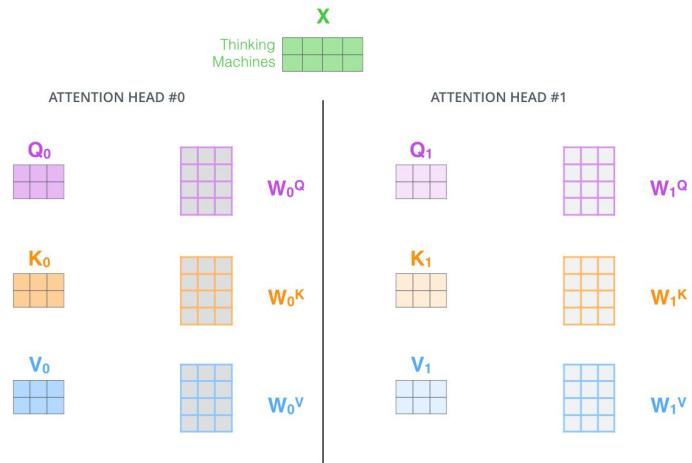


Multi-head Self-Attention



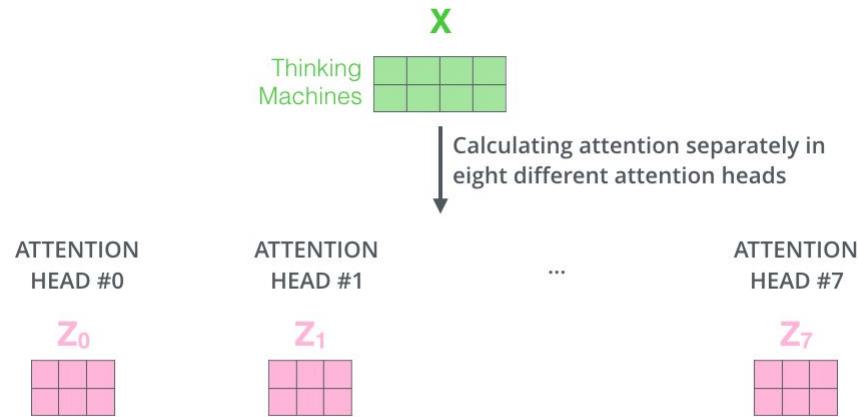
Multi-Head Attention

- Ce mécanisme permet d'élargir la capacité du modèle à se concentrer sur différentes positions
- L'idée est d'effectuer le calcul de l'attention plusieurs fois en parallèle, en utilisant à chaque fois différentes projections linéaires apprises (têtes)
- Chaque tête calcule l'attention de manière indépendante, ce qui permet au modèle de se concentrer sur différentes parties de l'entrée de différentes manières.



Source image: [Jalammar tutorial](#)

Multi-Head Attention



Source image: [Jalammar tutorial](#)

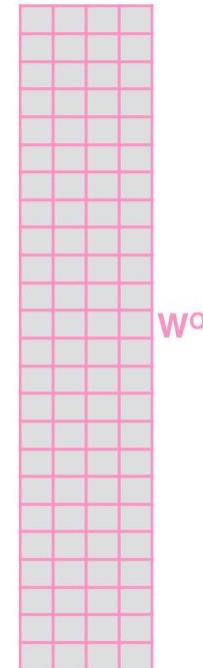
Multi-Head Attention

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

$$= \begin{matrix} Z \\ \hline \end{matrix}$$

Source image: [medium tutorial](#)

Multi-Head Attention

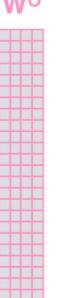
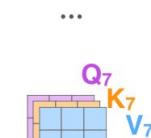
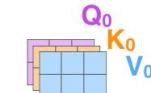
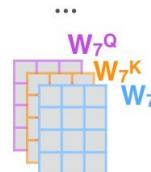
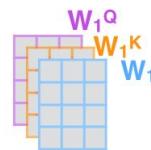
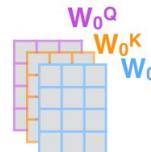
1) This is our
input sentence*

2) We embed
each word*

3) Split into 8 heads.
We multiply X or
 R with weight matrices

4) Calculate attention
using the resulting
 $Q/K/V$ matrices

5) Concatenate the resulting Z matrices,
then multiply with weight matrix W^O to
produce the output of the layer

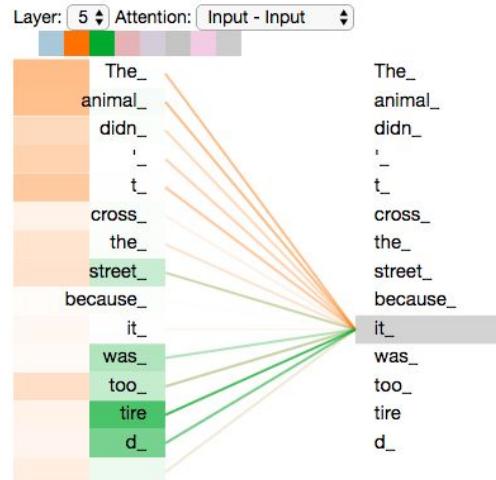


* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one



Multi-Head Attention

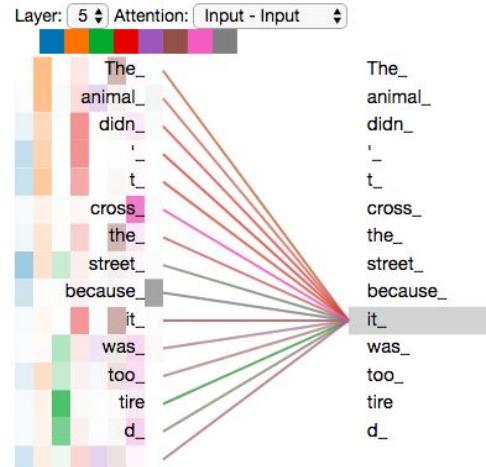
Exemple pour 2 têtes



Source image: [Jalammar tutorial](#)

Multi-Head Attention

Exemple avec les 8 têtes



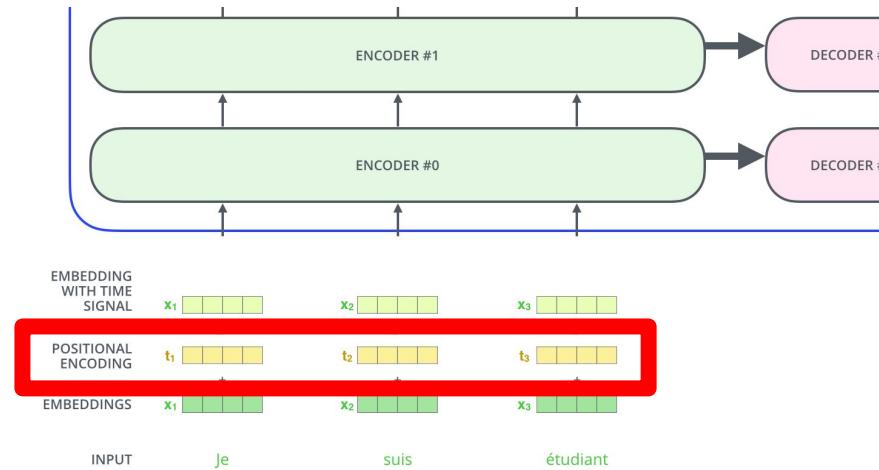
Source image: [Jalammar tutorial](#)

Positional Encoding

- Le seul problème avec le calcul du Self-Attention est que contrairement aux réseaux de neurones récurrents, il n'y a pas de notion d'ordre entre les mots de la séquence

Positional Encoding

- Le seul problème avec le calcul du Self-Attention est que contrairement aux réseaux de neurones récurrents, il n'y a pas de notion d'ordre entre les mots de la séquence
- Pour pallier à ce problème, il serait envisageable d'ajouter à chaque embedding un vecteur afin de prendre en compte cette notion d'ordre



Source image: [Jalammar tutorial](#)

Positional Encoding



INPUT

Je

suis

étudiant

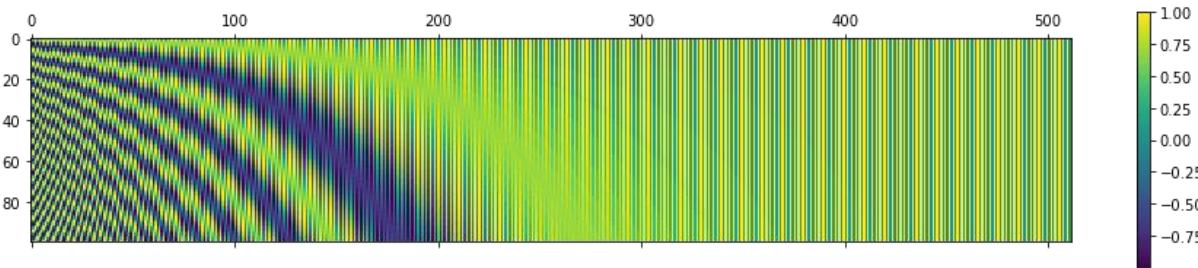
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Positional Encoding

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

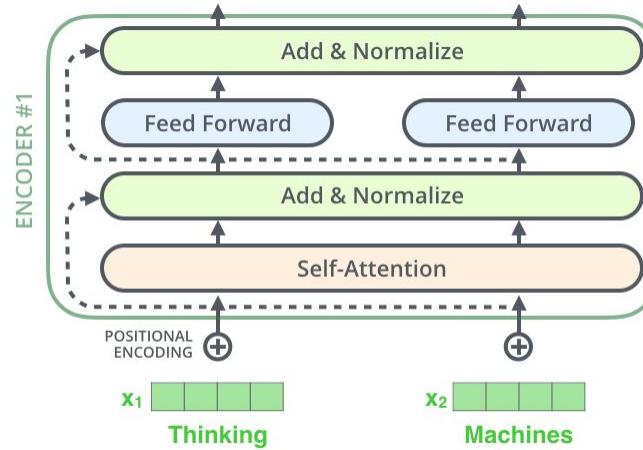
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$



Intuition: sinus et cosinus donnent une notion de fréquence en fonction de la position relative

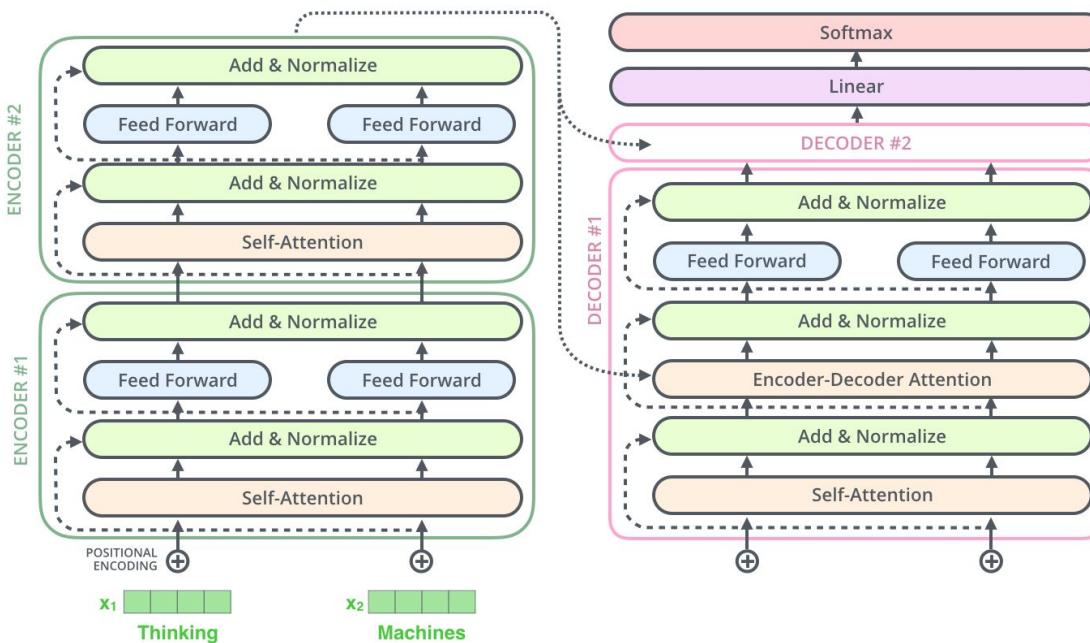
- Mot proche du début = basse fréquence
- Mot loin dans la séquence = haute fréquence

Autre détails importants



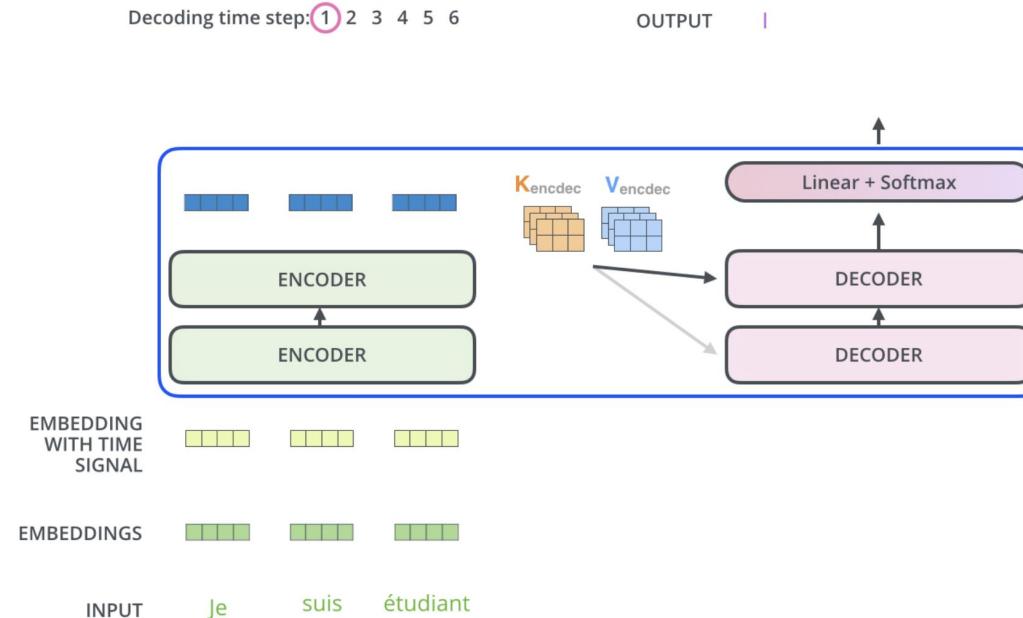
Source image: [Jalammar tutorial](#)

Encoder-Decoder



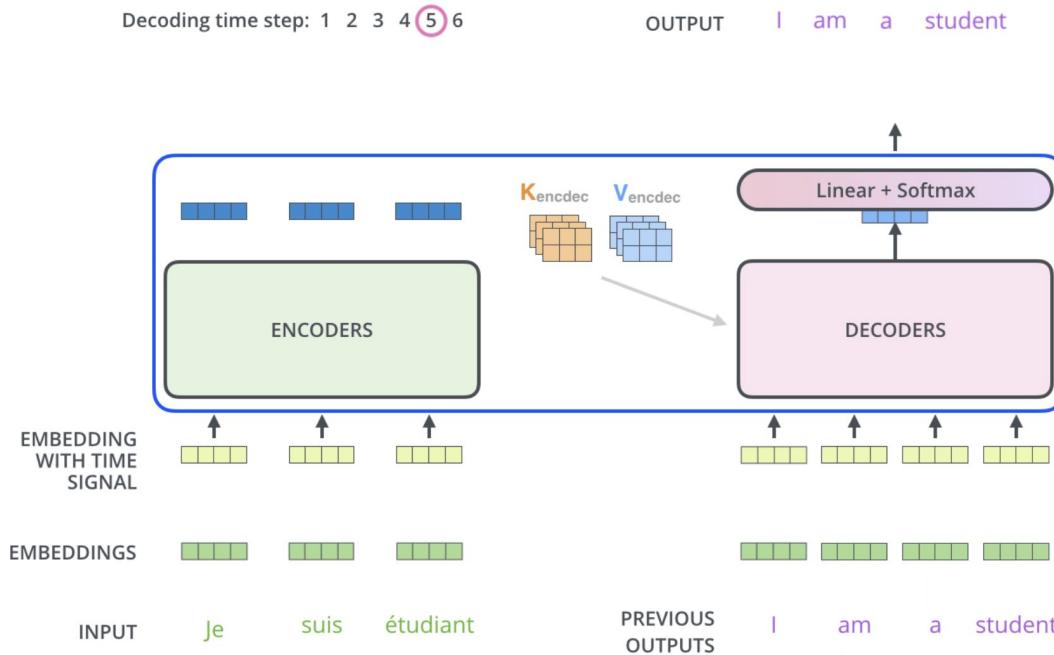
Source image: [Jalammar tutorial](#)

Decoder



Source image: [Jalammar tutorial](#)

Decoder



Source image: [Jalammar tutorial](#)

Couche de sortie

Which word in our vocabulary
is associated with this index?

am

Get the index of the cell
with the highest value
(`argmax`)

5

5

5

Decoder stack output

`log_probs`



Softmax

`logits`



Linear

<https://www.youtube.com/watch?v=eMlx5fFNoYc&t=5s>

TP 3: Données séquentielles