Attention mechanism and Transformers

Clément Romac (Hugging Face & Inria)

clement.romac@inria.fr
https://clementromac.github.io/teaching/

Petit sondage 🔍

Que vous évoque le mot

“Transformer” ?

Contenu 📗

- Rappels RNNs et Seq2Seq
- Attention mechanism
- Self-Attention
  - Multi-hop
  - Multi-head
- Transformer architecture

A retenir 🥋

- Limites des RNNs
- Compréhension du principe d'attention + self-attention
- Compréhension globale de l'architecture Transformer
- Intuition des très nombreux tricks *(Positional Encoding, Layer Norm, Residual connections…)*

Ressources

Cours:

- Waterloo university: https://www.youtube.com/watch?v=OyFJWRnt_AY

Talks:

- Arthur Szlam: https://www.youtube.com/watch?v=M-HCvbdQ8wA

Lectures:

- https://jalammar.github.io/illustrated-transformer/
- https://lilianweng.github.io/posts/2018-06-24-attention/
- https://lilianweng.github.io/posts/2020-04-07-the-transformer-family/
- https://transformersbook.com/

Rappels sur les RNNs et Seq2Seq

RNNs pour les séquences

Comment gérer les dépendances temporelles ?

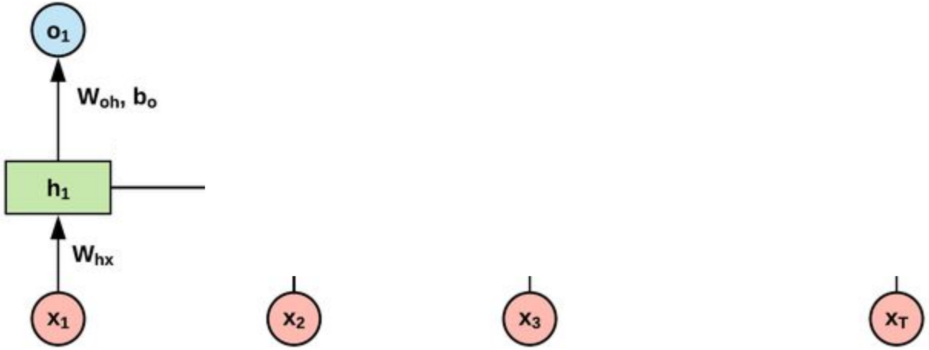
Exemple 1 : J' aime le ... Learning

Exemple 2 : $\underbrace{\text{[02, 10, 2015, 1500]}}_{\text{Jour, mois, année, valeur}}$ [03, 10, 2015, 1205] [04, 10, 2015, 1820] ... [05, 10, 2015, 1900]

RNNs pour les séquences

Pour un élément:

1) Représentation (hidden state)
2) Sortie

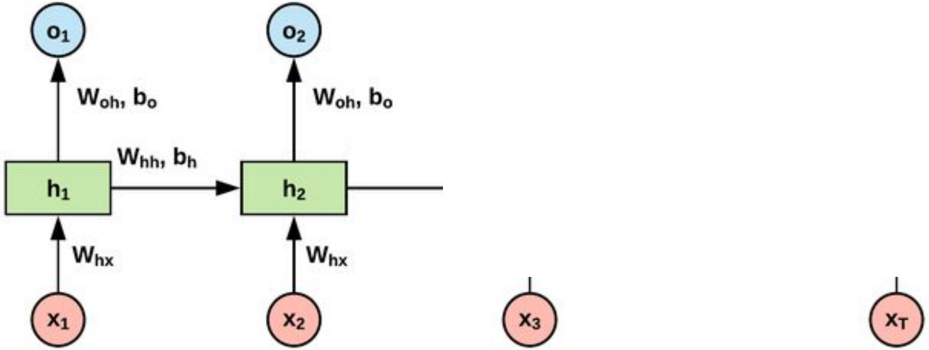


- **hidden state** $$h_1 = \sigma_h(W_{hx}x_1)$$
- **sortie** $$o_1 = \sigma_o(W_{oh}h_1 + b_o)$$

RNNs pour les séquences

Pour un élément:

1) Représentation (hidden state)
2) Sortie

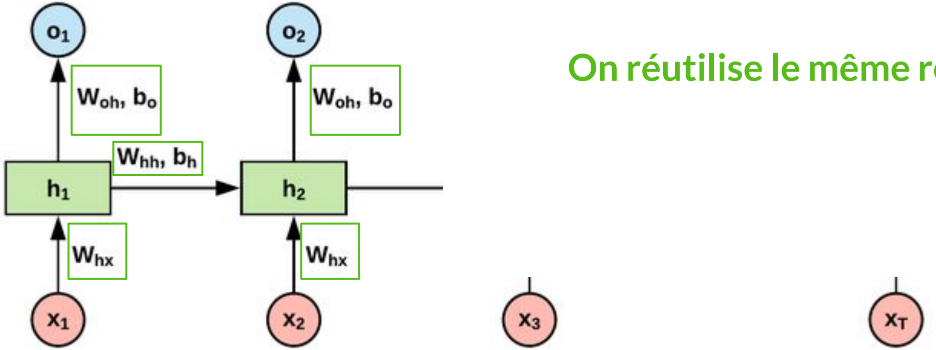


- **hidden state** $h_1 = \sigma_h(W_{hx}x_1)$ | $h_2 = \sigma_h(W_{hx}x_2 + W_{hh}h_1 + b_h)$
- **sortie** $o_1 = \sigma_o(W_{oh}h_1 + b_o)$ | $o_1 = \sigma_o(W_{oh}h_1 + b_o)$

RNNs pour les séquences

Pour un élément:

1) Représentation (hidden state)
2) Sortie



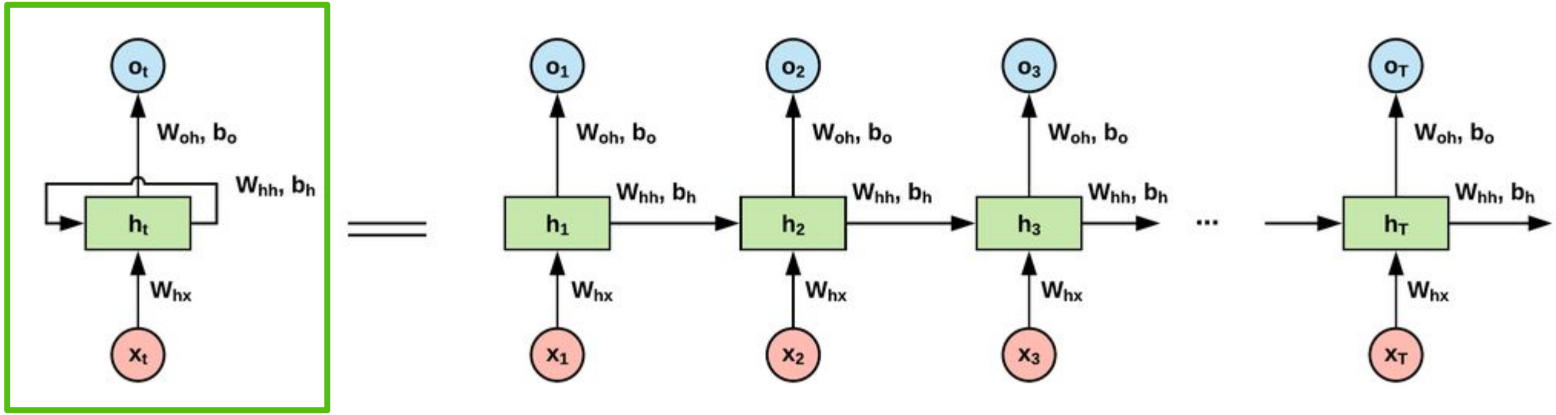
On réutilise le même réseau !

- **hidden state** $h_1 = \sigma_h(\boxed{W_{hx}} x_1)$
- **sortie** $o_1 = \sigma_o(\boxed{W_{oh}} h_1 \boxed{+ b_o})$

$$h_2 = \sigma_h(\boxed{W_{hx}} x_2 + W_{hh} h_1 + b_h)$$

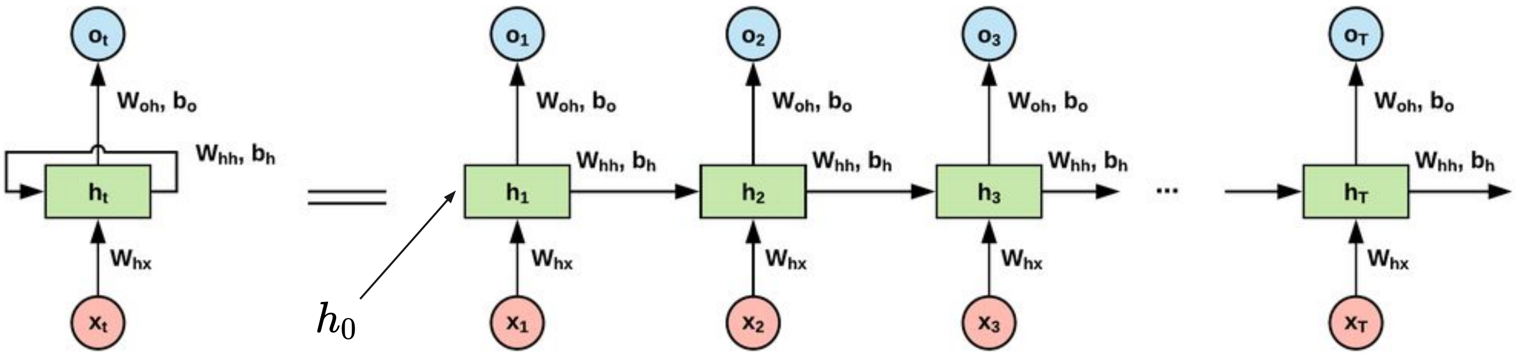
$$o_1 = \sigma_o(\boxed{W_{oh}} h_1 \boxed{+ b_o})$$

RNNs pour les séquences



- **hidden state** $h_1 = \sigma_h(\boxed{W_{hx}} x_1)$ | $h_2 = \sigma_h(\boxed{W_{hx}} x_2 + W_{hh} h_1 + b_h)$
- **sortie** $o_1 = \sigma_o(\boxed{W_{oh}} h_1 \boxed{+ b_o})$ | $o_1 = \sigma_o(\boxed{W_{oh}} h_1 \boxed{+ b_o})$

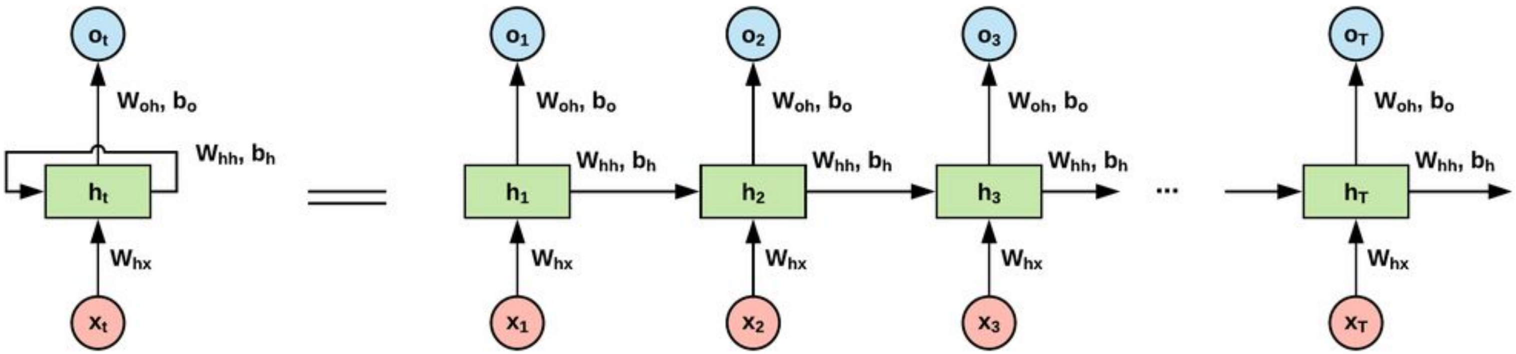
RNNs pour les séquences



- **hidden state** $h_1 = \sigma_h(W_{hx}x_1 + W_{hh}h_0 + b_h)$ | $h_2 = \sigma_h(W_{hx}x_2 + W_{hh}h_1 + b_h)$
- **sortie** $o_1 = \sigma_o(W_{oh}h_1 + b_o)$ | $o_1 = \sigma_o(W_{oh}h_1 + b_o)$

RNNs pour les séquences

Prochain mot   Ventes



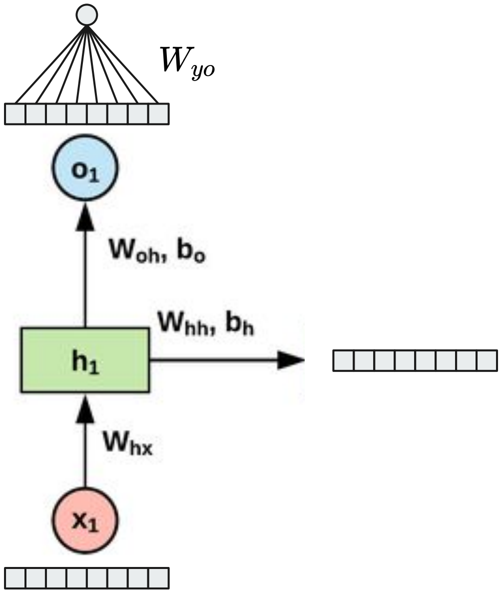
Exemple 1 : J' aime le ... Learning

Exemple 2 : [02, 10, 2015, 1500] [03, 10, 2015, 1205] [04, 10, 2015, 1820] ... [05, 10, 2015, 1900]

$\underbrace{[02, 10, 2015, 1500]}_{\text{Jour, mois, année, valeur}}$

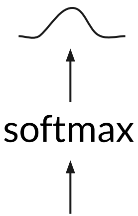
RNNs pour les séquences

$$y_1 = W_{yo} o_1$$



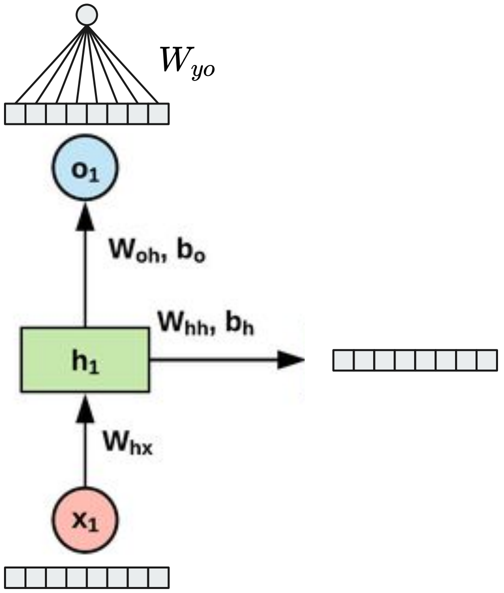
$$y1 \in \mathbb{R}$$

RNNs pour les séquences



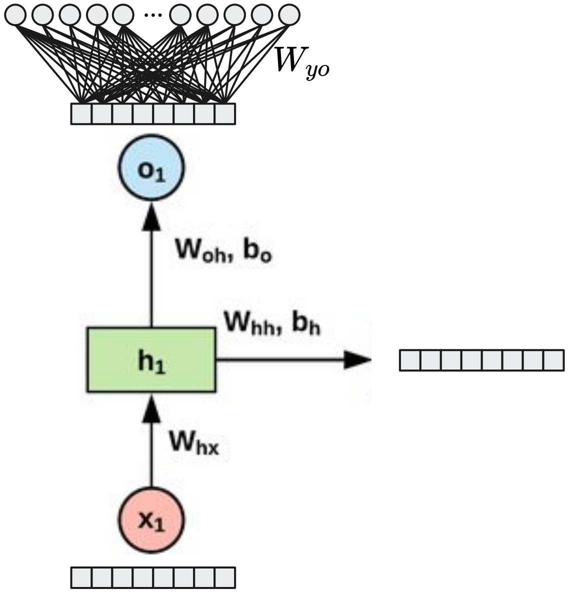
$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

$$y_1 = W_{yo} o_1$$



$$y1 \in \mathbb{R}$$

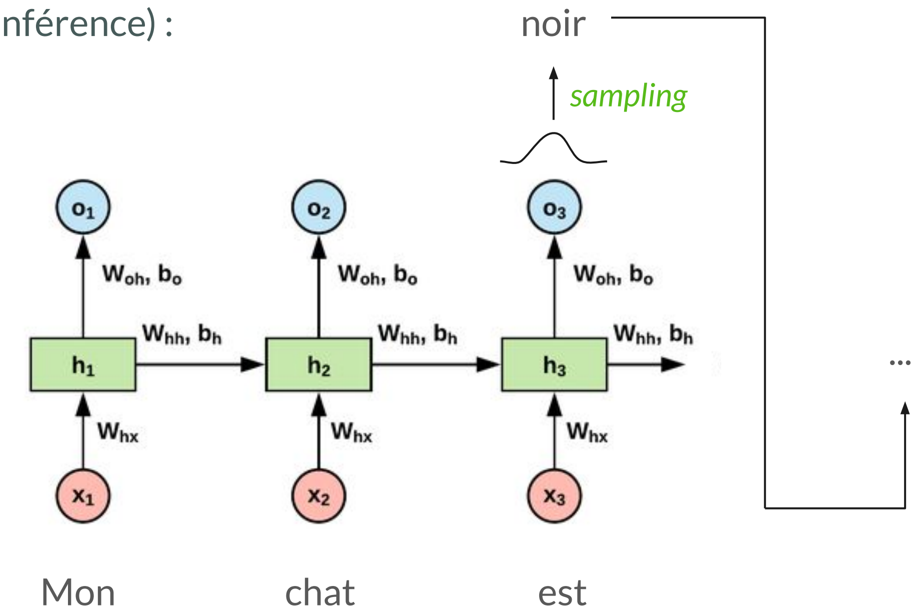
$$y_1 = W_{yo} o_1$$



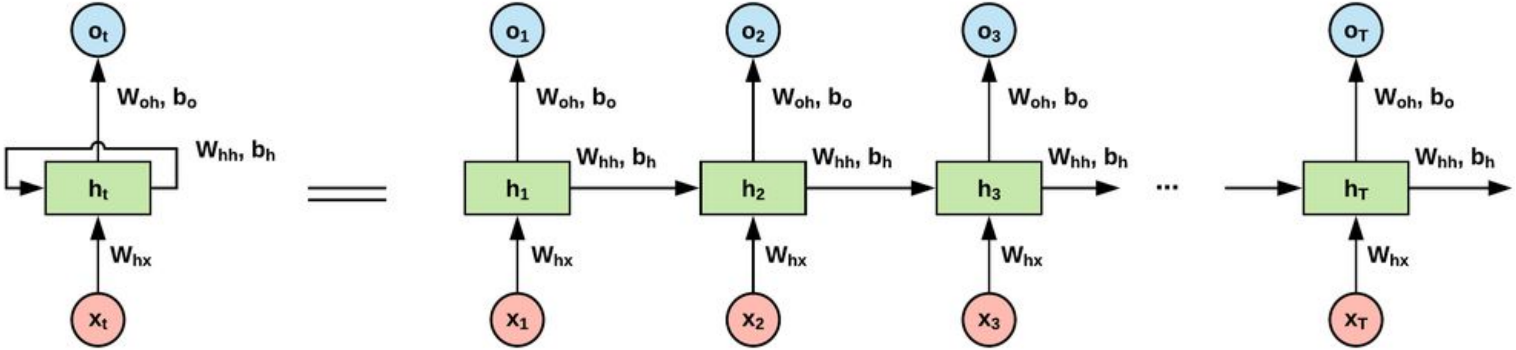
$$y1 \in \mathcal{Y} = \{u_0, u_1, \ldots, u_N\}$$

RNNs pour les séquences

Exemple de decoding (inférence) :



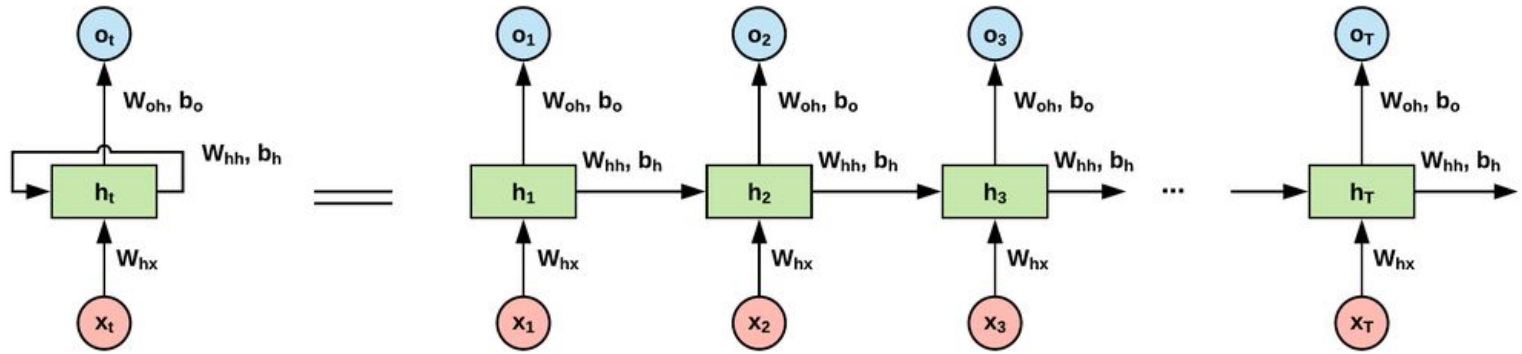
RNNs pour les séquences



Inconvénients:

- Dépendances long terme
- Gradient vanishing
- Impossible de paralléliser

RNNs pour les séquences



Inconvénients:

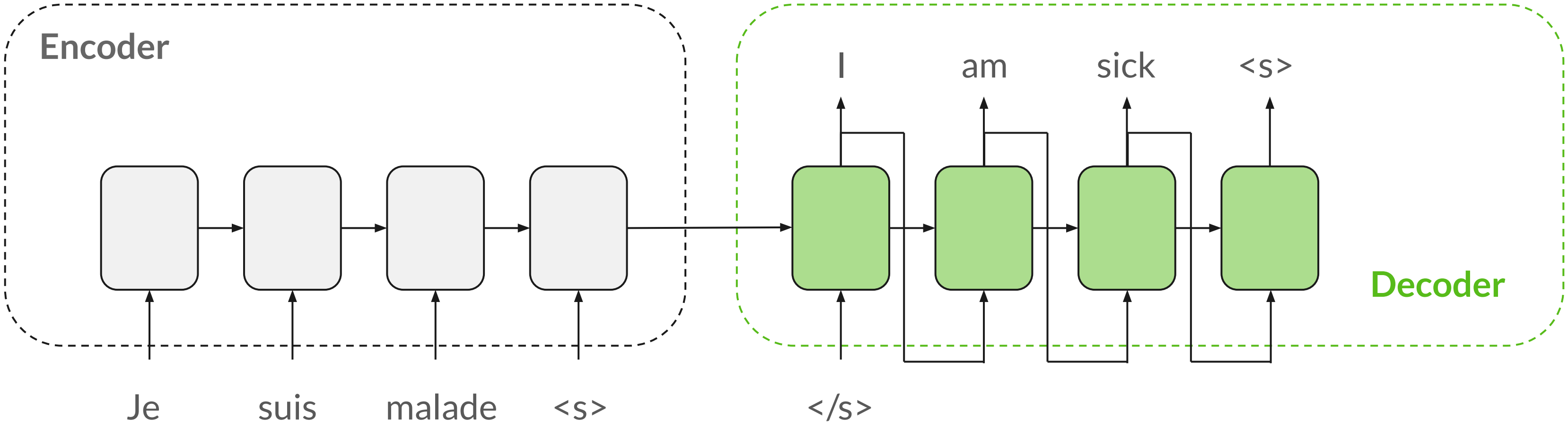
- Dépendances long terme
- Gradient vanishing
- Impossible de paralléliser

→ Long Short Term Memory (LSTM) *(Hochreiter & Schmidhuber, 1997)*

Le problème Seq2Seq

Une séquence en entrée => une séquence en sortie

Exemple: la traduction

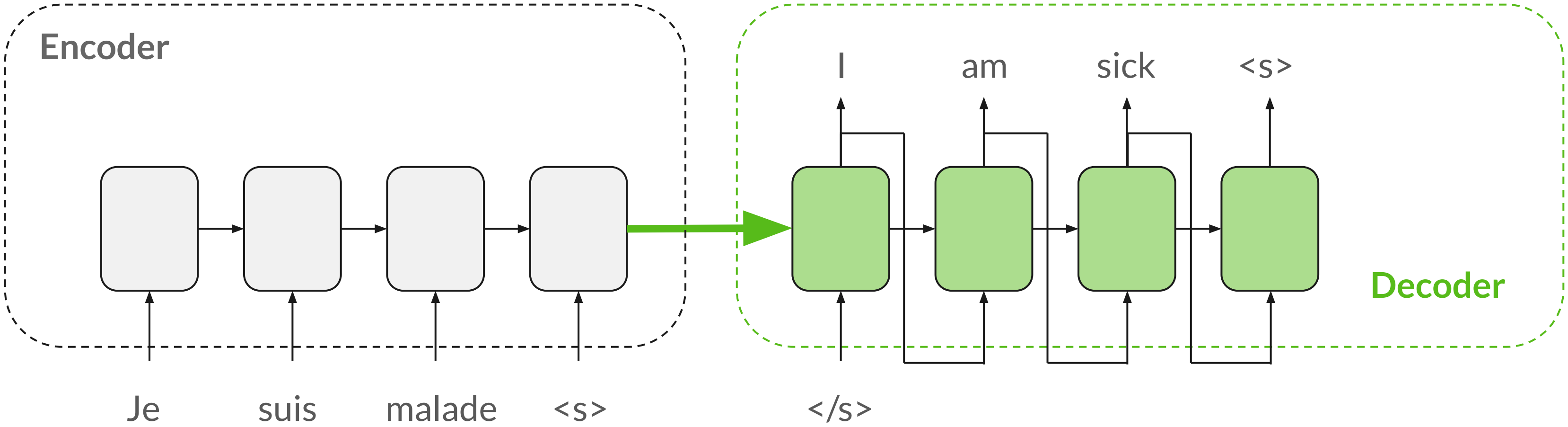


Le problème Seq2Seq

Une séquence en entrée => une séquence en sortie

Exemple: la traduction

Le dernier hidden state doit représenter toute la séquence d'entrée /!



Attention mechanism

Attention mechanism

Imitation d’un principe de retrieval

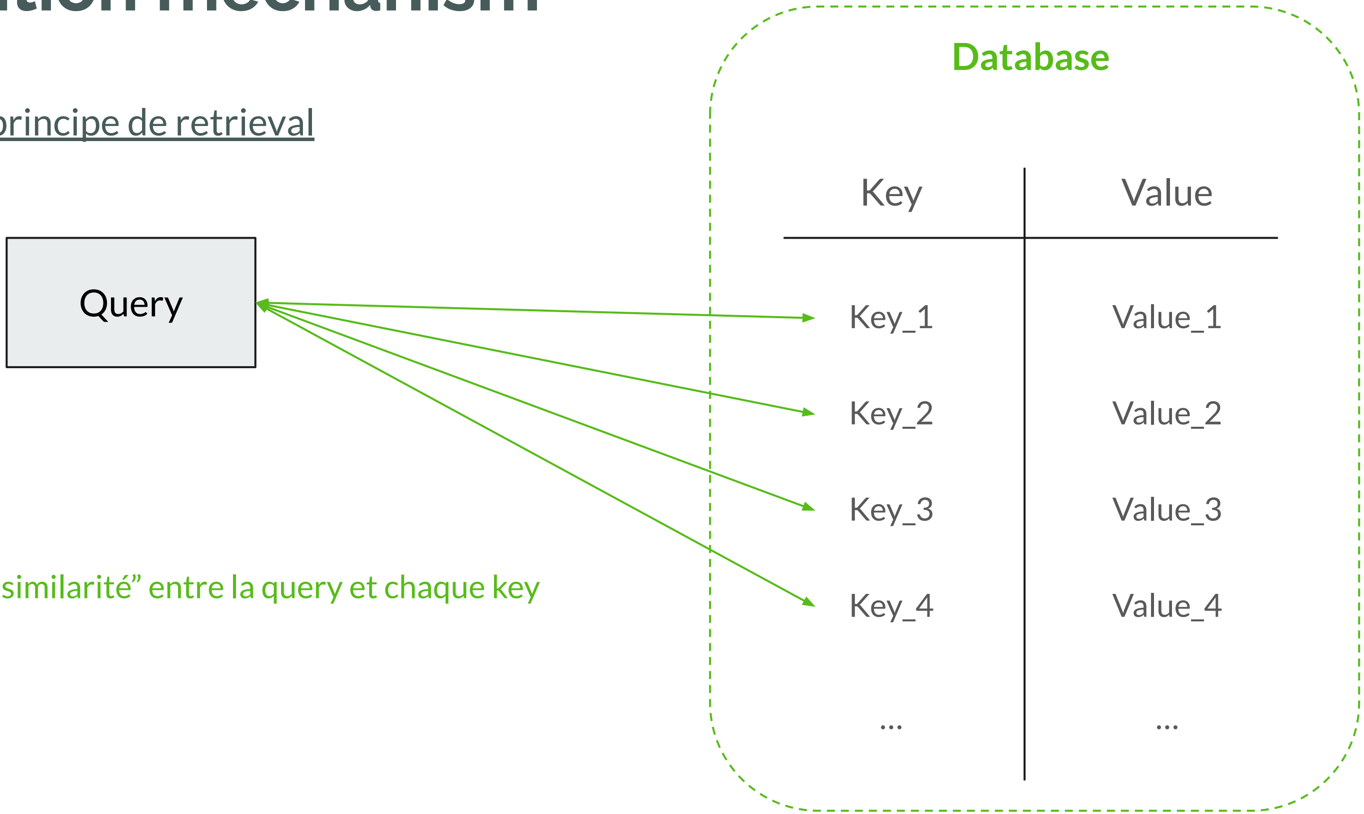
Query

Database

Key	Value
Key_1	Value_1
Key_2	Value_2
Key_3	Value_3
Key_4	Value_4
...	...

Attention mechanism

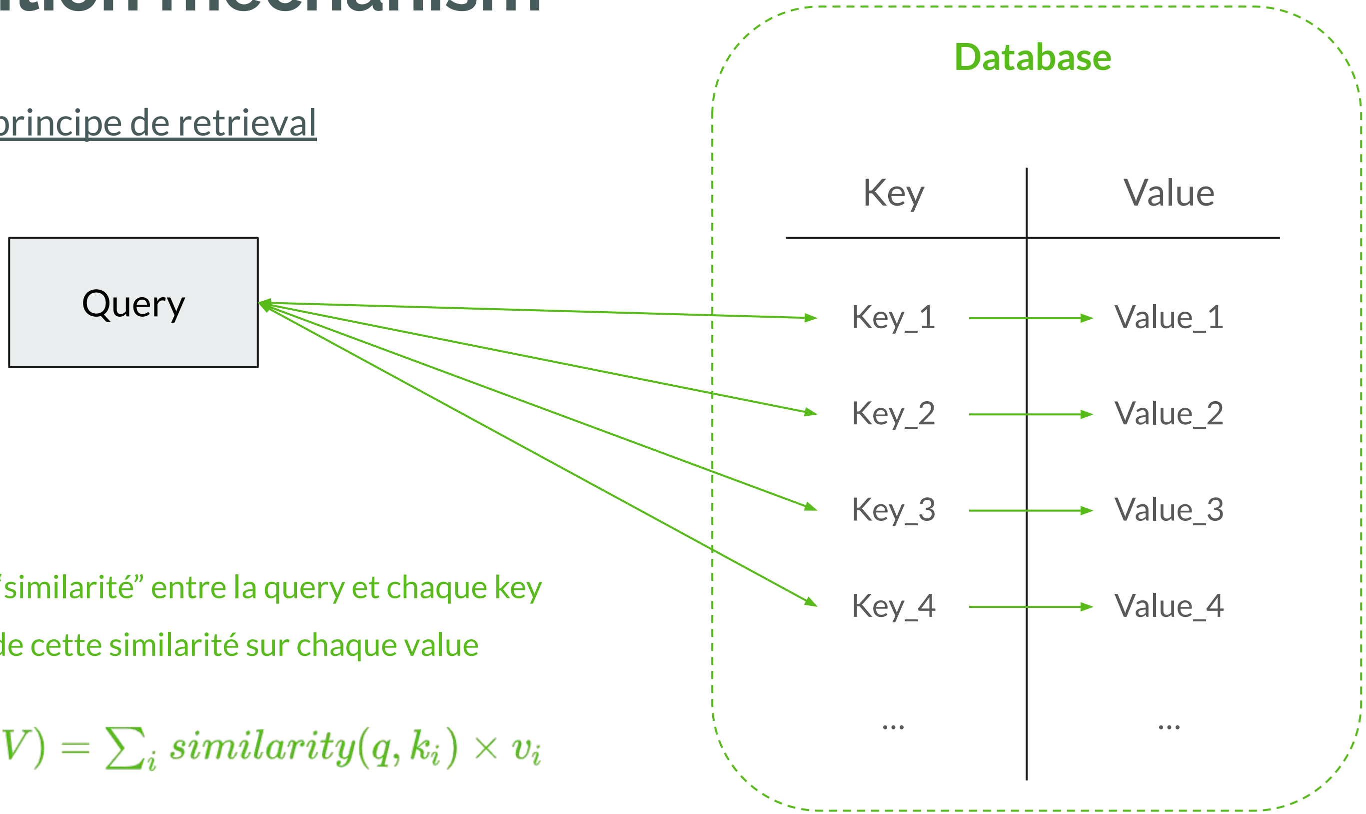
Imitation d'un principe de retrieval



1. Calcul d'une "similarité" entre la query et chaque key

Attention mechanism

Imitation d’un principe de retrieval



1. Calcul d’une “similarité” entre la query et chaque key
2. “Utilisation” de cette similarité sur chaque value

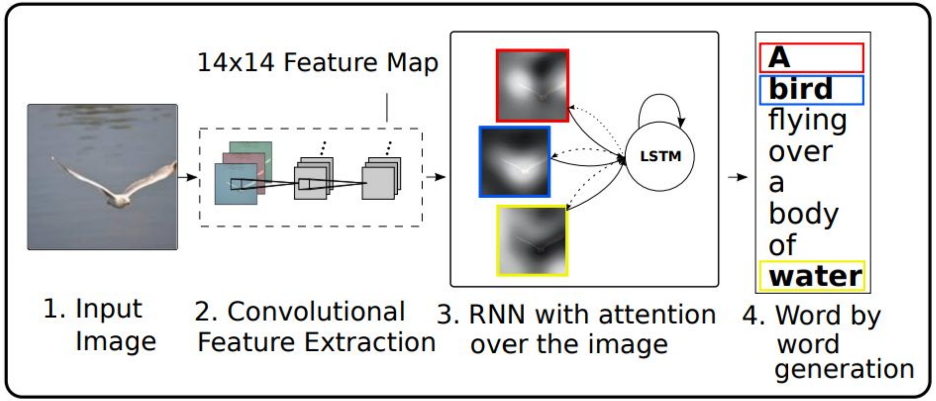
$$attention(q, K, V) = \sum_i similarity(q, k_i) \times v_i$$

Attention mechanism

Bref historique

Vision:

- Mnih et al., 2014 => Sequence of focusing (RL)
- Xu et al., 2015 => Captioning with attention on feature maps



Attention mechanism

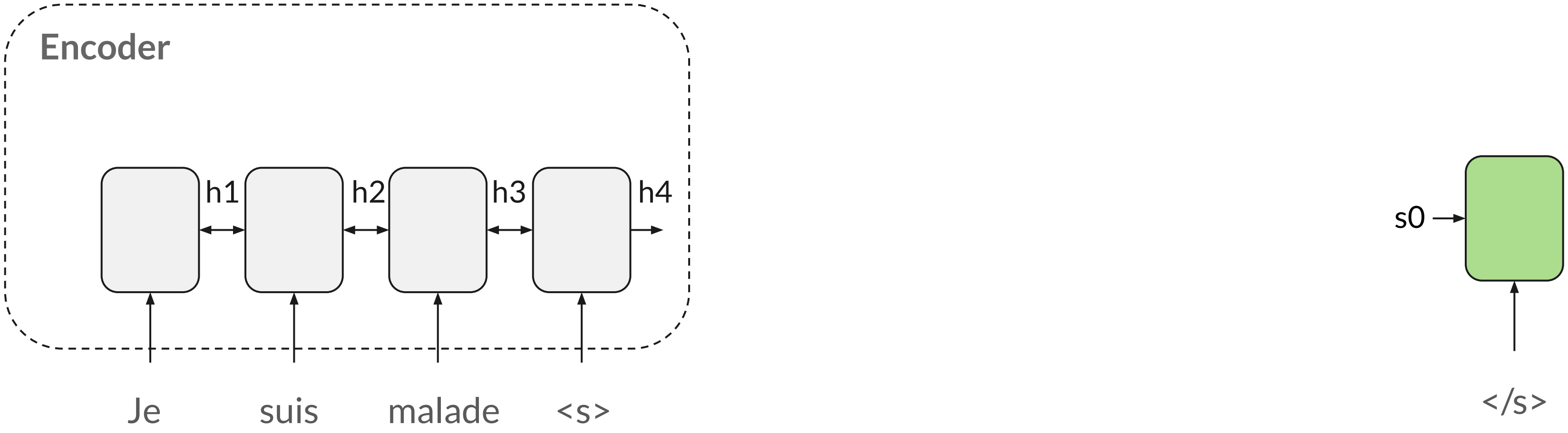
Bref historique

Vision:
- Mnih et al., 2014 => Sequence of focusing (RL)
- Xu et al., 2015 => Captioning with attention on feature maps

NLP:
- Brown et al., 1993 => Alignment in translation (Hard attention)
- **Bahdanau et al., 2015 => Attention as focusing (SOTA in NMT /!\)**

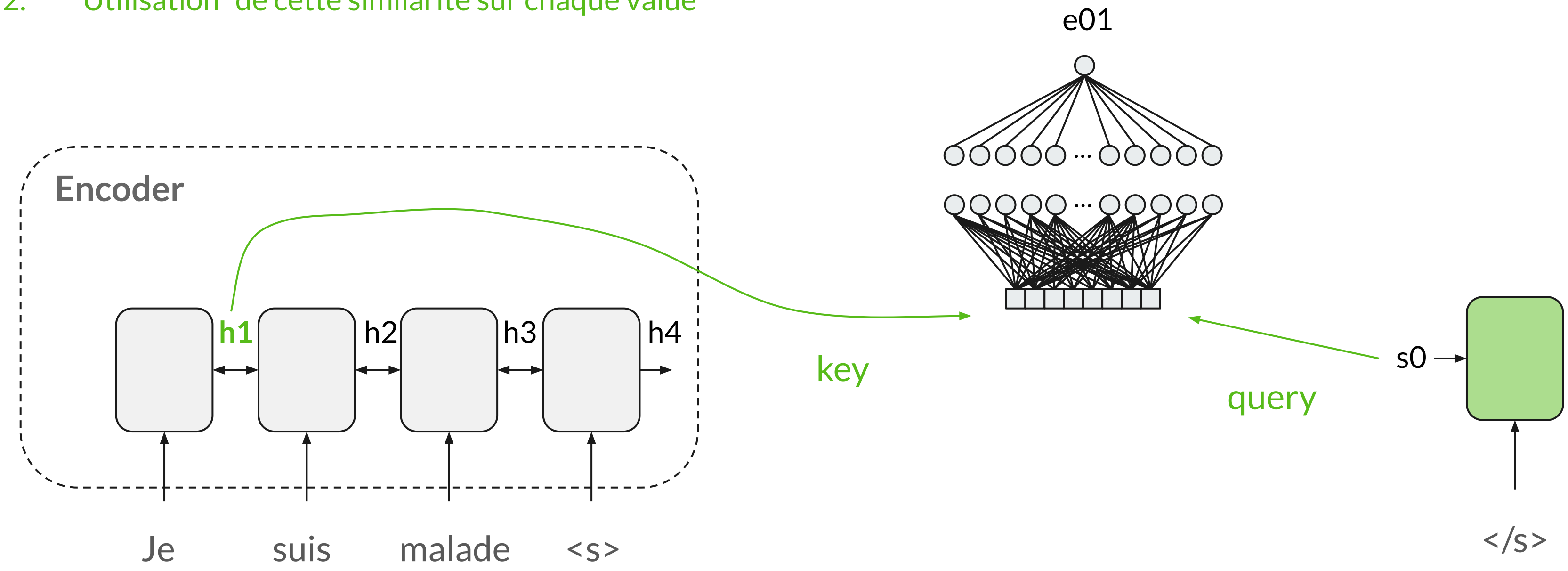
Additive Attention *(Bahdanau et al., 2015)*

1. Calcul d’une “similarité” entre la query et chaque key
2. “Utilisation” de cette similarité sur chaque value



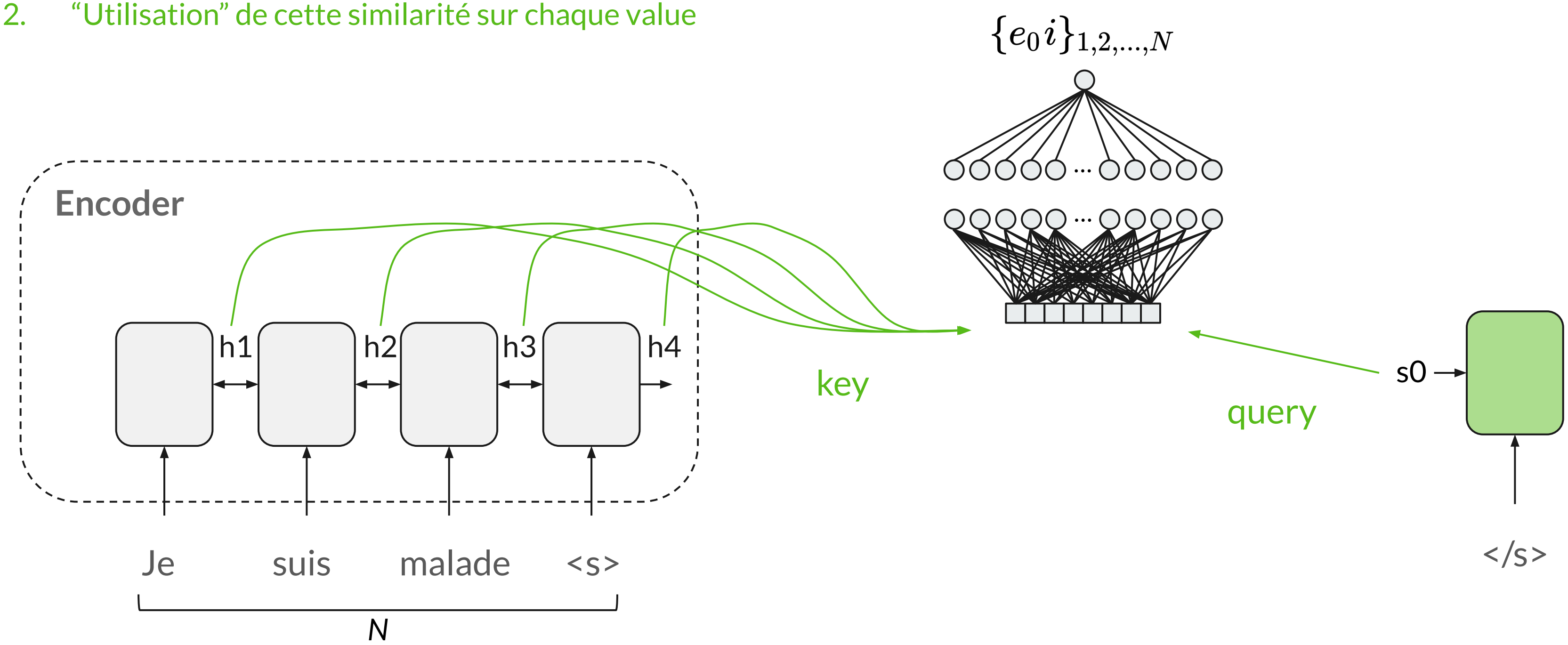
Additive Attention *(Bahdanau et al., 2015)*

1. Calcul d’une “similarité” entre la query et chaque key
2. “Utilisation” de cette similarité sur chaque value



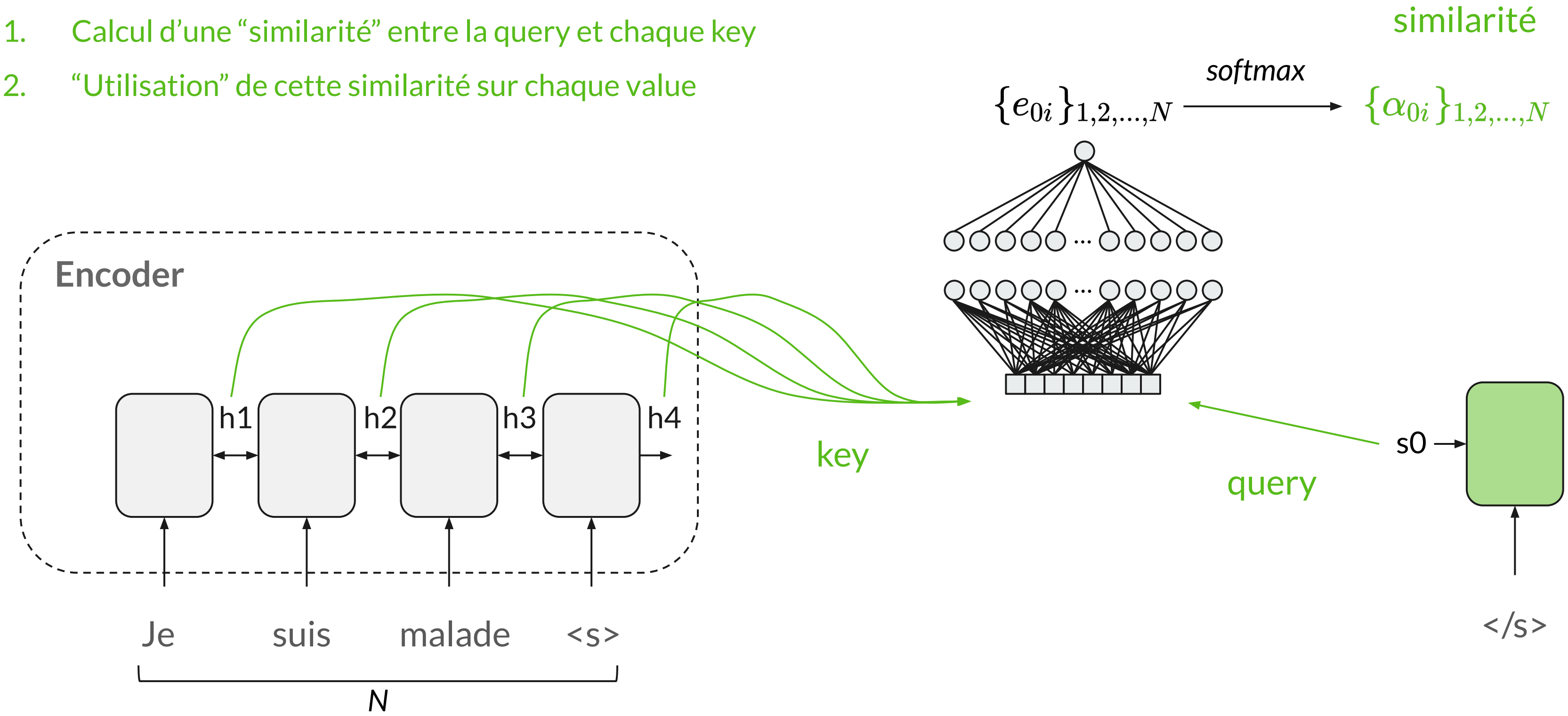
Additive Attention *(Bahdanau et al., 2015)*

1. Calcul d’une “similarité” entre la query et chaque key
2. “Utilisation” de cette similarité sur chaque value



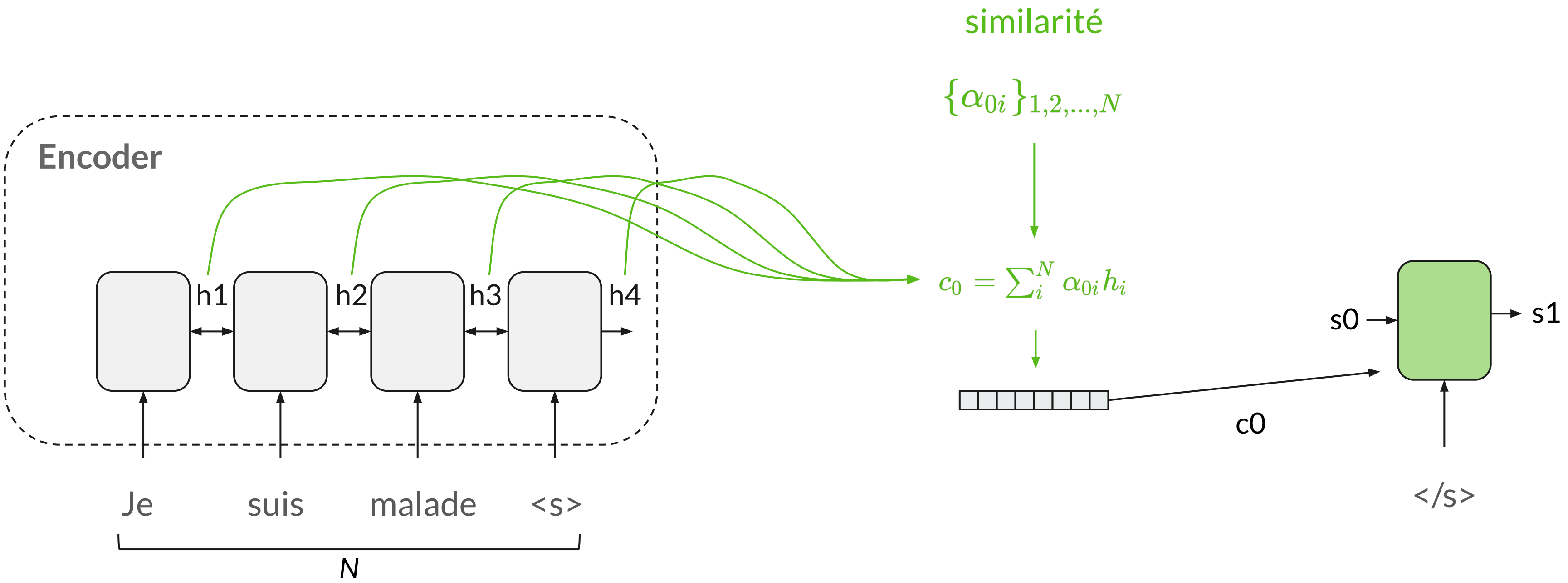
Additive Attention *(Bahdanau et al., 2015)*

1. Calcul d'une "similarité" entre la query et chaque key
2. "Utilisation" de cette similarité sur chaque value



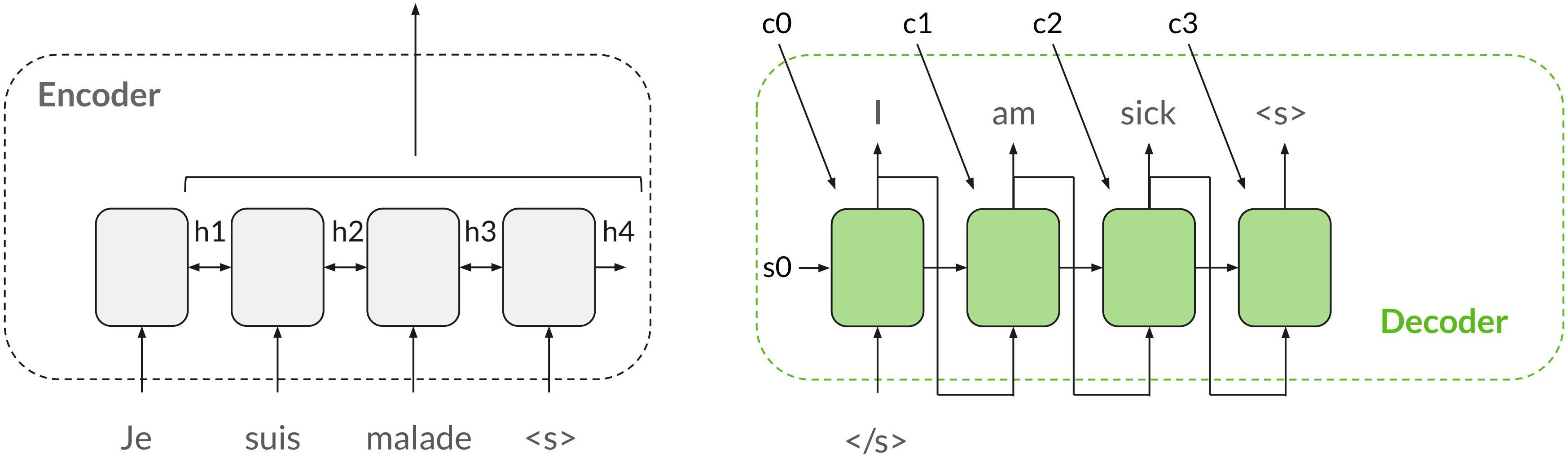
Additive Attention *(Bahdanau et al., 2015)*

1. Calcul d'une "similarité" entre la query et chaque key
2. "Utilisation" de cette similarité sur chaque value



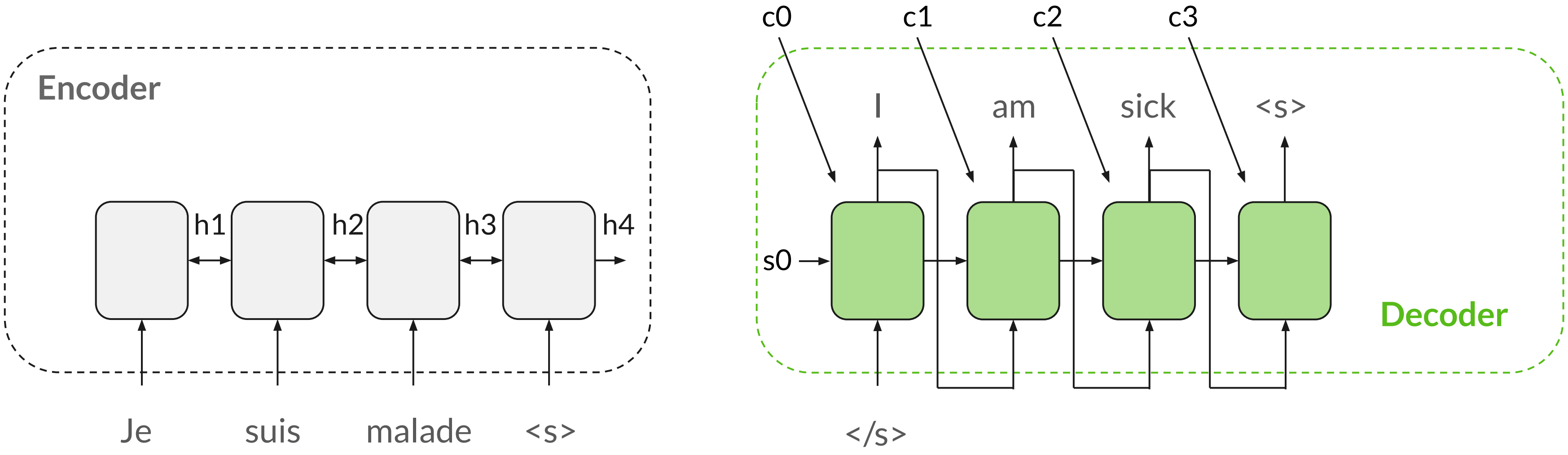
Additive Attention *(Bahdanau et al., 2015)*

$$\mathrm{h} = \{h_1, h2, \ldots, h_N\}$$

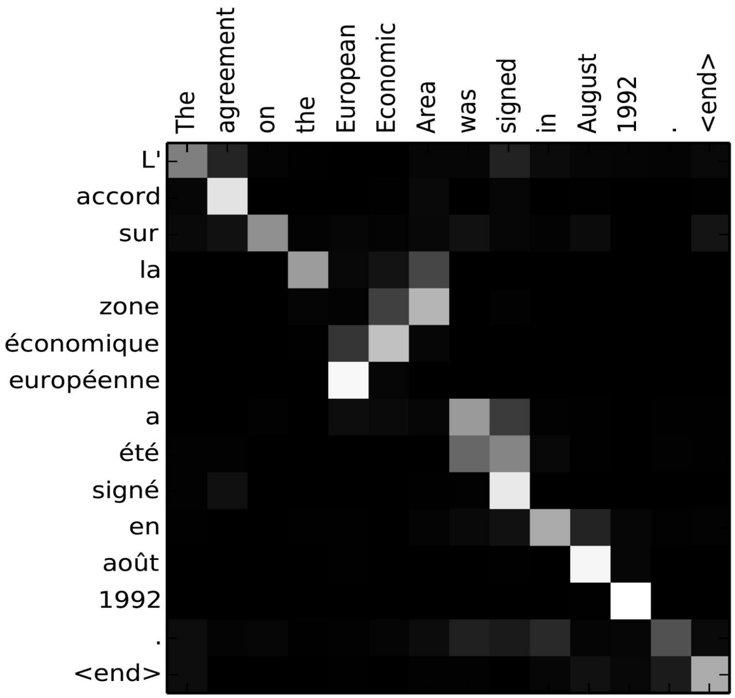


Additive Attention *(Bahdanau et al., 2015)*

- Il y a désormais un contexte résumant la phrase d’entrée pour chaque mot du Decoder
- Chaque mot décodé a son propre contexte (query = s-1)
- Les hidden states de l’Encoder sont à la fois les clés et valeurs, ceux du Decoder les queries



Additive Attention *(Bahdanau et al., 2015)*



L'attention permet à chaque mot décodé de se "concentrer" sur certains mots d'entrée

Attention mechanismS (https://lilianweng.github.io/posts/2018-06-24-attention/)

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

Attention mechanismS (https://lilianweng.github.io/posts/2018-06-24-attention/)

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

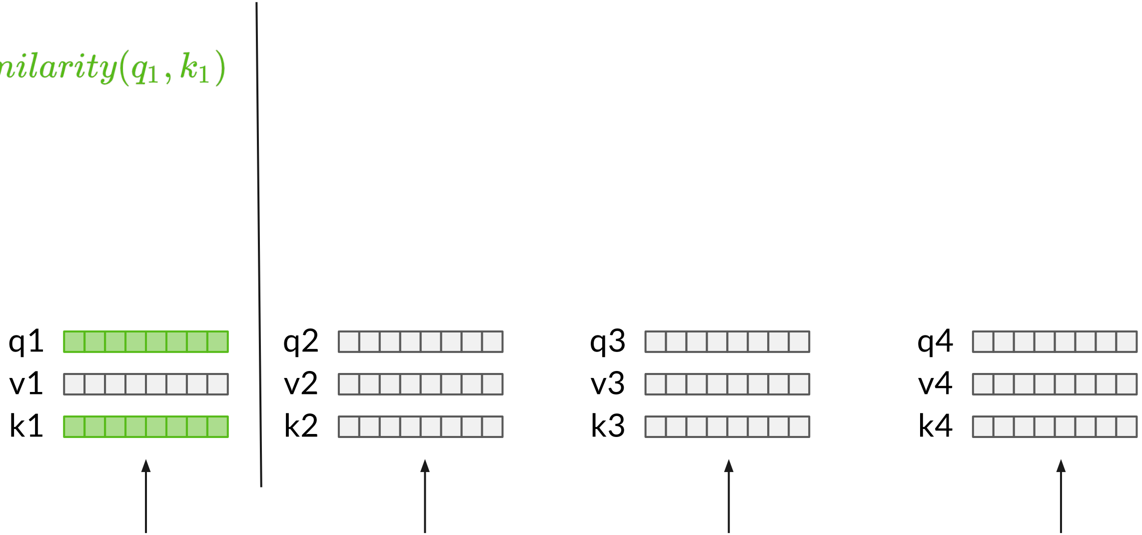
(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

Self-Attention

Self-attention basics

Q, K, V viennent du même **ensemble**

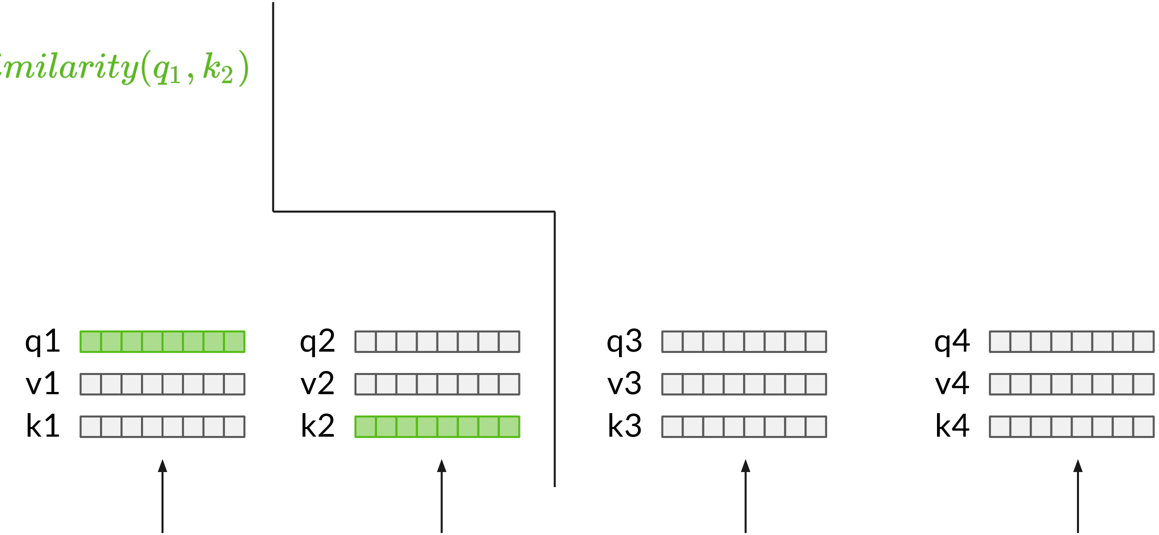
$$a_{11} = similarity(q_1, k_1)$$



Self-attention basics

Q, K, V viennent du même **ensemble**

$$a_{12} = similarity(q_1, k_2)$$

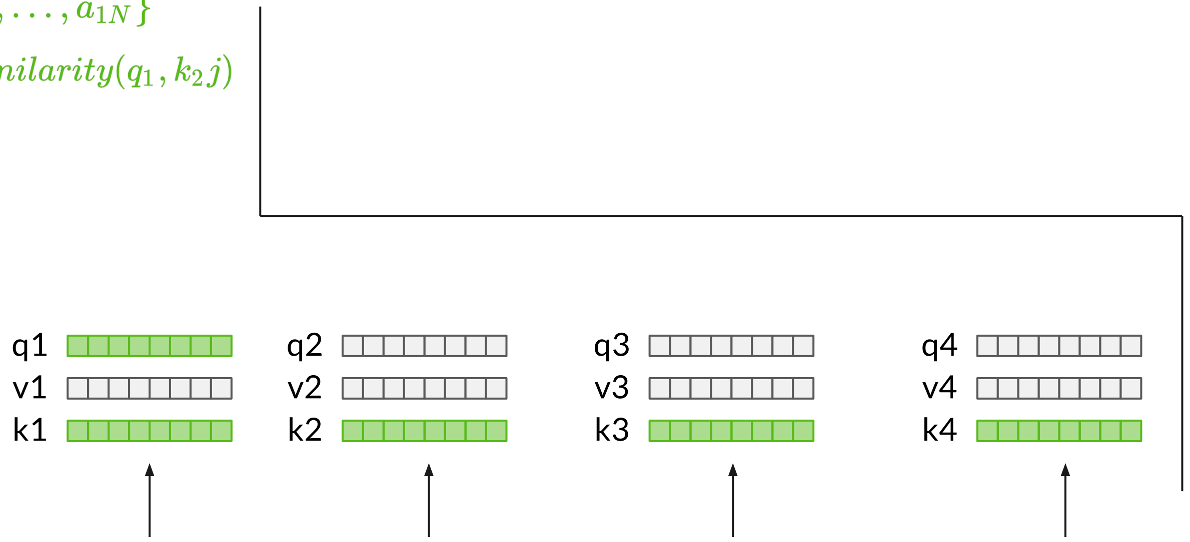


Self-attention basics

Q, K, V viennent du même **ensemble**

$\{a_{11}, a_{12}, \ldots, a_{1N}\}$

$a_{1j} = similarity(q_1, k_2 j)$



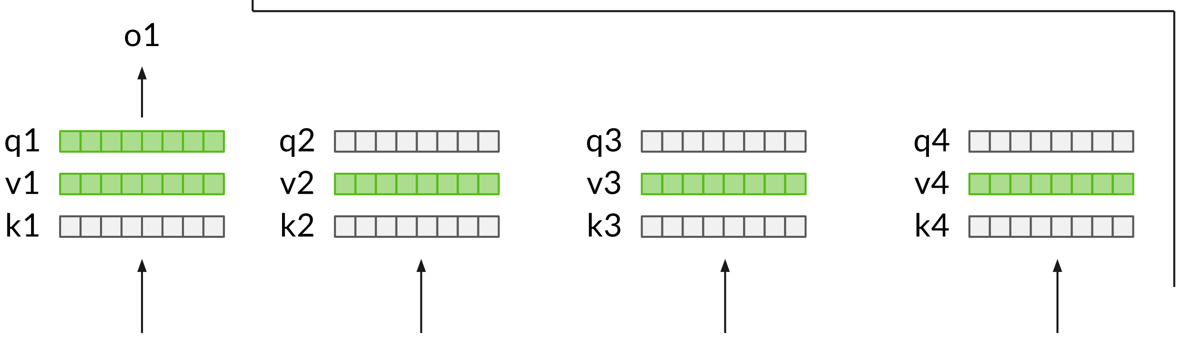
Self-attention basics

Q, K, V viennent du même **ensemble**

$$\{a_{11}, a_{12}, \ldots, a_{1N}\}$$

$$a_{1j} = similarity(q_1, k_2 j)$$

$$o_1 = \sum_i^N a_{1i} v_i$$



Self-attention basics

Q, K, V viennent du même **ensemble**

$$\{a_{11}, a_{12}, \ldots, a_{1N}\}$$

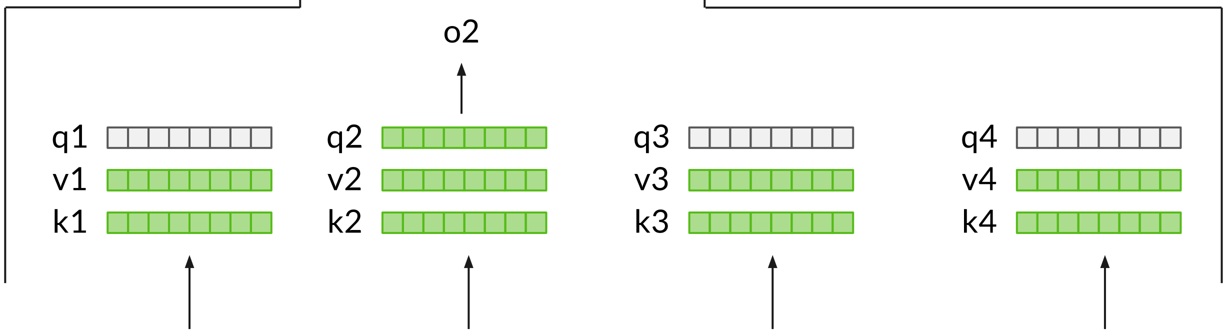
$$a_{1j} = similarity(q_1, k_j)$$

$$o_1 = \sum_i^N a_{1i} v_i$$

$$\{a_{21}, a_{22}, \ldots, a_{2N}\}$$

$$a_{2j} = similarity(q_2, k_j)$$

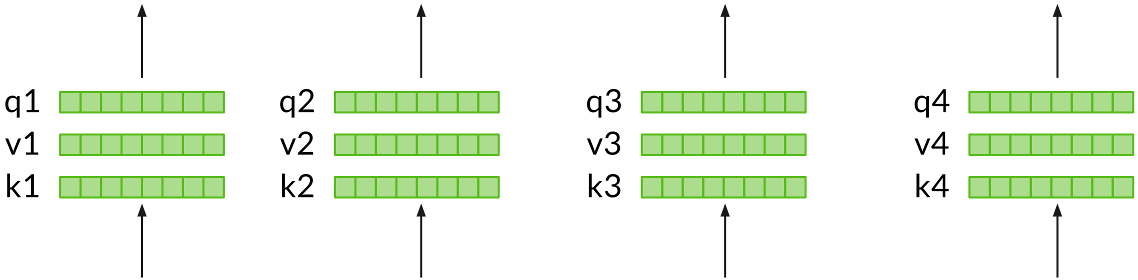
$$o_2 = \sum_i^N a_{2i} v_i$$



Self-attention basics

<u>Q, K, V viennent du même **ensemble**</u>

$$o_1 = \sum_i^N a_{1i} v_i \qquad o_2 = \sum_i^N a_{2i} v_i \qquad o_3 = \sum_i^N a_{3i} v_i \qquad o_4 = \sum_i^N a_{4i} v_i$$



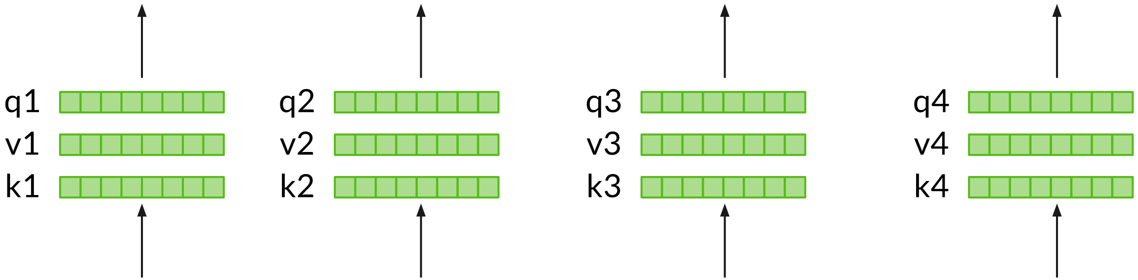
Self-attention basics

Q, K, V viennent du même **ensemble**

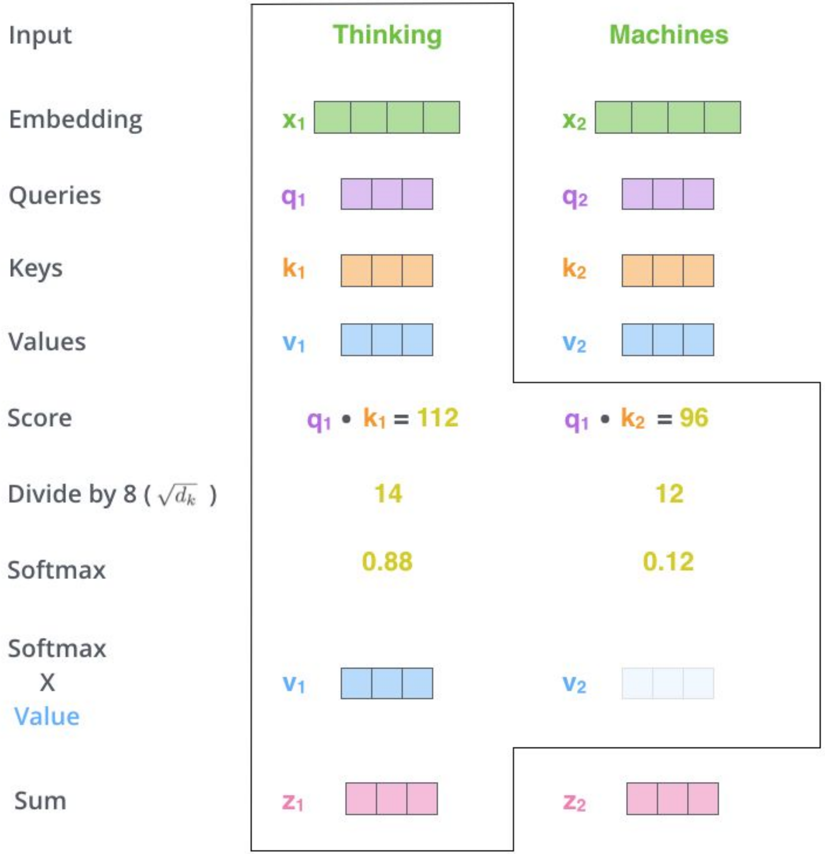
Scaled Dot-Product Attention

$$a_{ij} = \text{softmax}\left(\frac{q_i k_j^\top}{\sqrt{d_k}}\right)$$

$$o_1 = \sum_i^N a_{1i} v_i \qquad o_2 = \sum_i^N a_{2i} v_i \qquad o_3 = \sum_i^N a_{3i} v_i \qquad o_4 = \sum_i^N a_{4i} v_i$$



Self-attention basics



(https://jalammar.github.io/illustrated-transformer/)

Transformer architecture

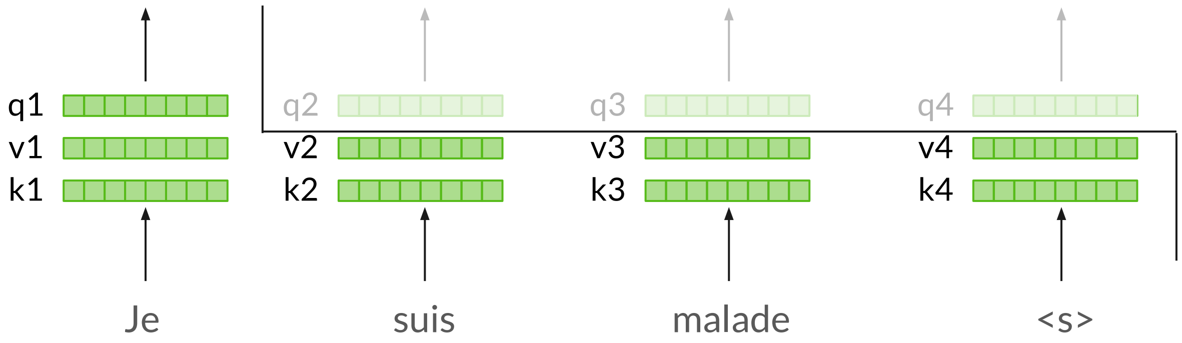
(Vaswani et al., 2019)

Attention is all you need ! *(Vaswani et al., 2019)*

On enlève les RNNs !

Eléments clés:

- Self-attention (multi hidden-state propagation)



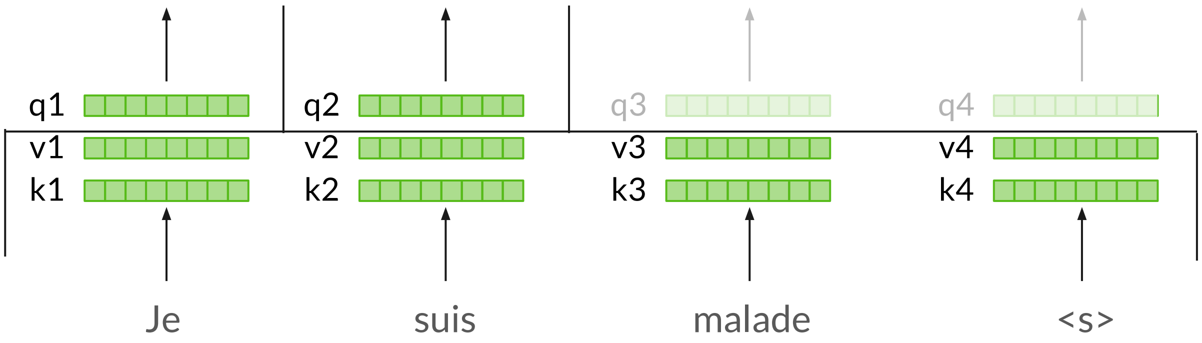
Attention is all you need ! *(Vaswani et al., 2019)*

On enlève les RNNs !

Eléments clés:

- Self-attention (multi hidden-state propagation)

Tous les Oi sont produits en même temps (un peu comme dans un RNN bi-directionnel)



Attention is all you need ! *(Vaswani et al., 2019)*

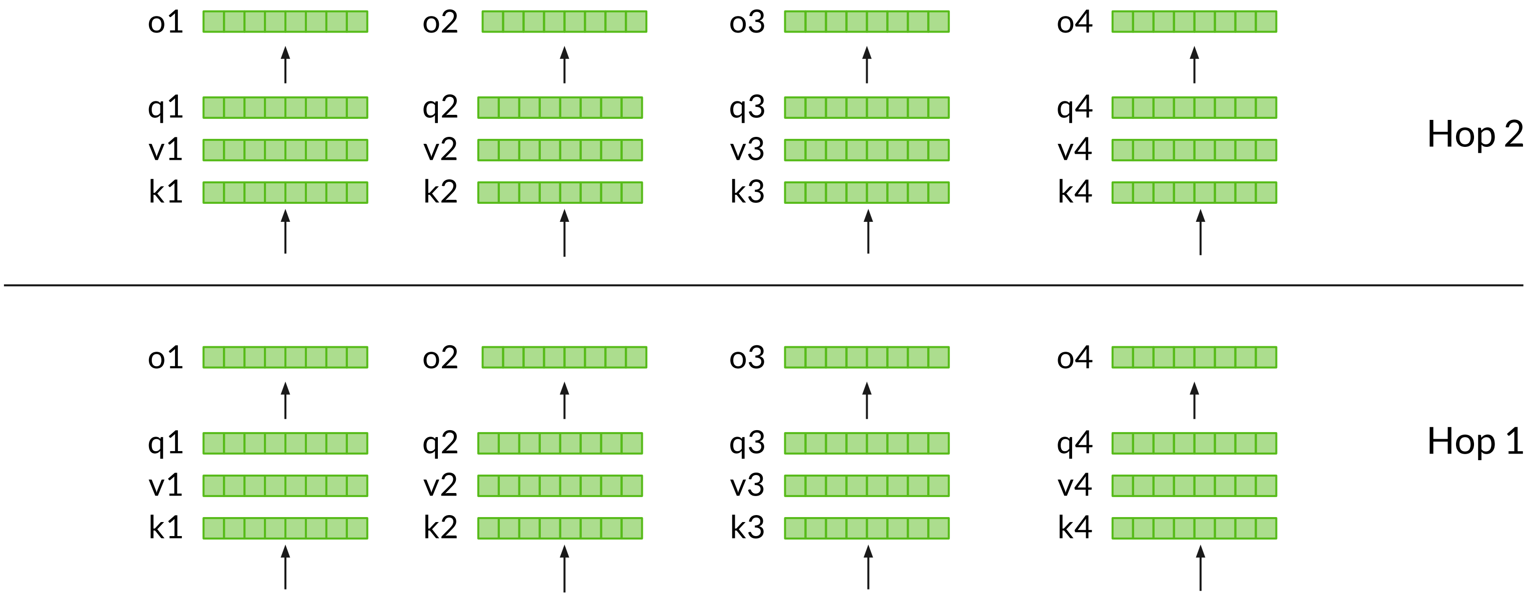
On enlève les RNNs !

Eléments clés:

- Self-attention (multi hidden-state propagation)
- Multi-hop (layers)

Multi-hop

Plusieurs itérations

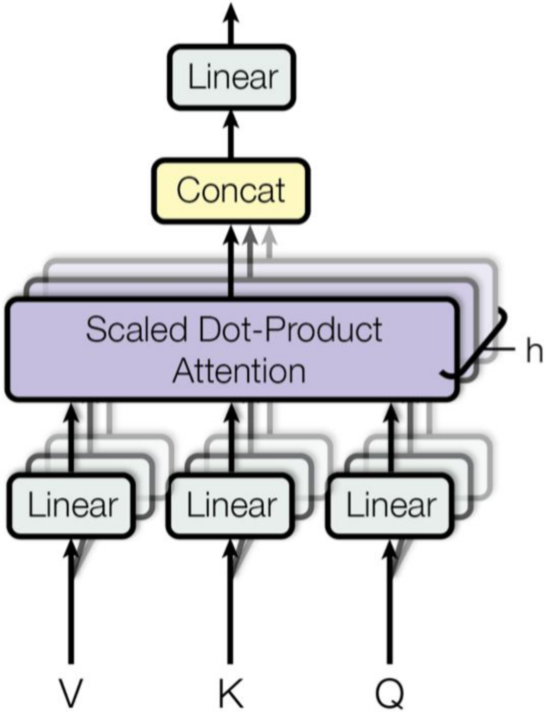


Attention is all you need ! *(Vaswani et al., 2019)*

On enlève les RNNs !

Eléments clés:

- Self-attention (multi hidden-state propagation)
- Multi-hop (layers)
- Multi-head



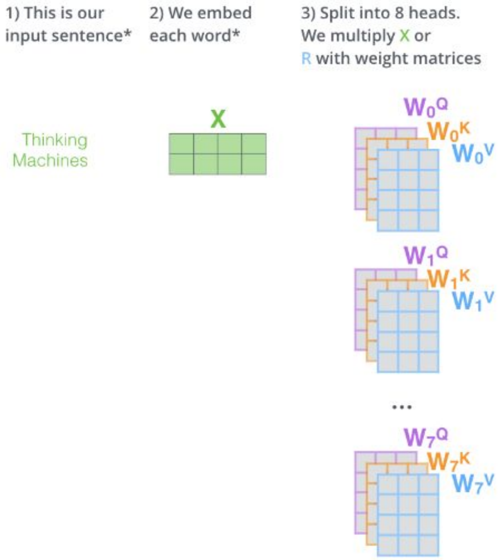
Attention is all you need ! *(Vaswani et al., 2019)*

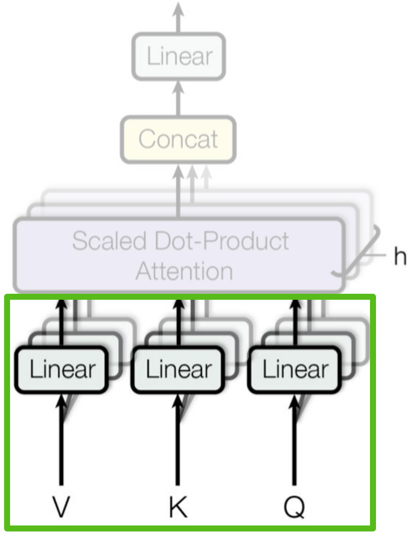
- 1 matrice de poids par tête pour Q, K, V

- Projection avec une transformation linéaire

- Dimensions de Q, K, V définies par celle des poids

- On peut “batcher” les opérations





(https://jalammar.github.io/illustrated-transformer/)

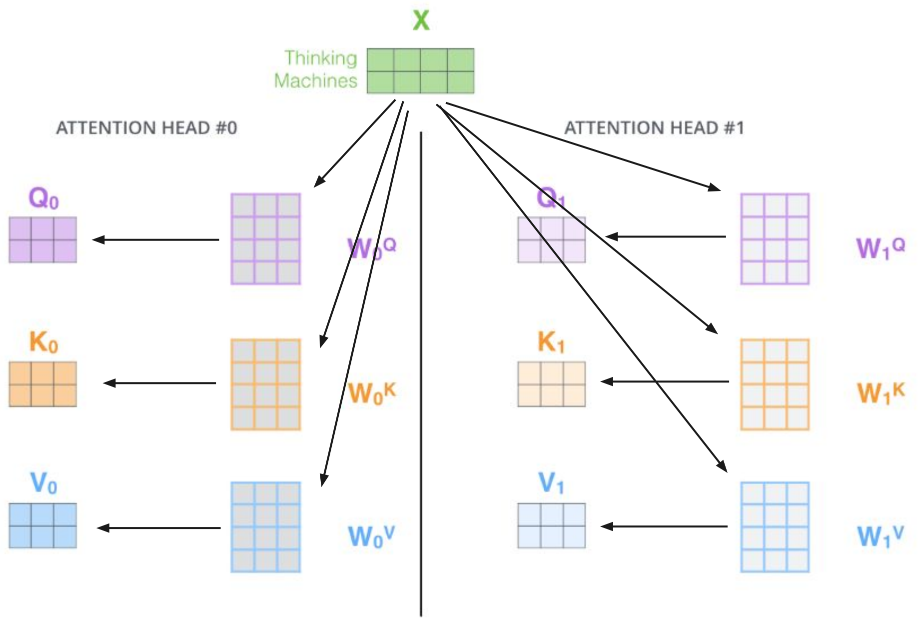
Attention is all you need ! *(Vaswani et al., 2019)*

- 1 matrice de poids par tête pour Q, K, V

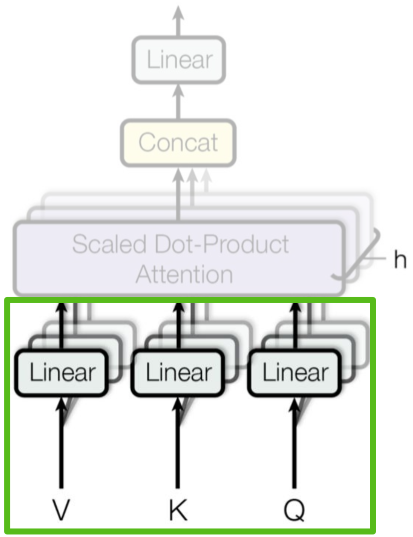
- Projection avec une transformation linéaire

- Dimensions de Q, K, V définies par celle des poids

- On peut “batcher” les opérations



With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.



(https://jalammar.github.io/illustrated-transformer/)

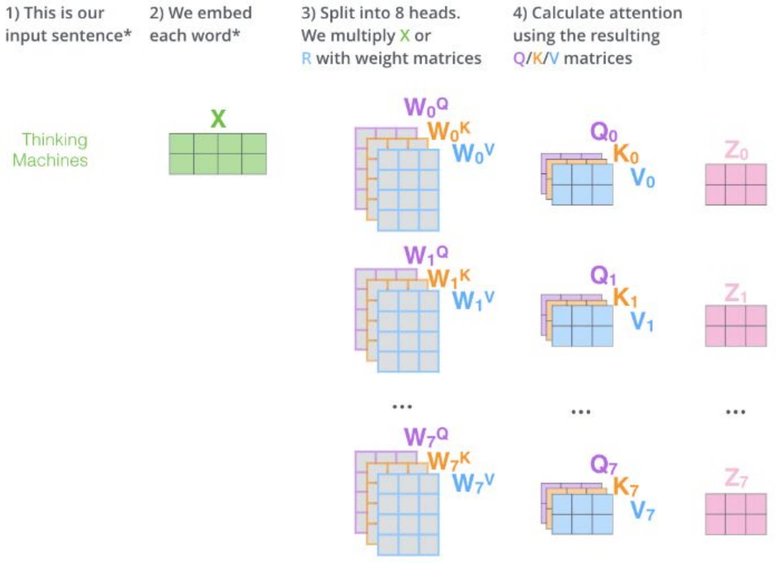
Attention is all you need ! *(Vaswani et al., 2019)*

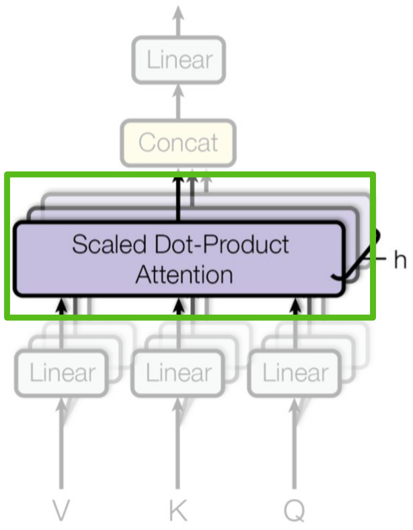
- 1 matrice de poids par tête pour Q, K, V

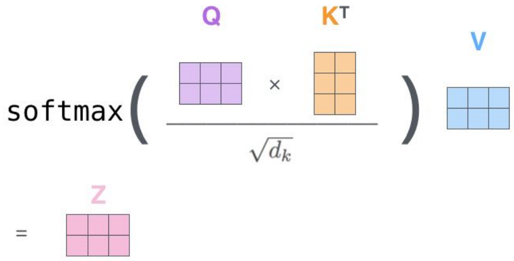
- Projection avec une transformation linéaire

- Dimensions de Q, K, V définies par celle des poids

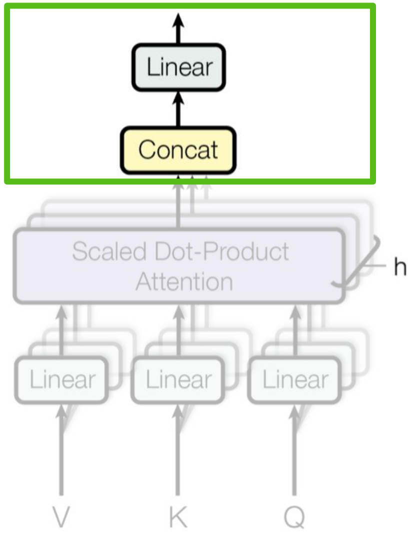
- On peut “batcher” les opérations





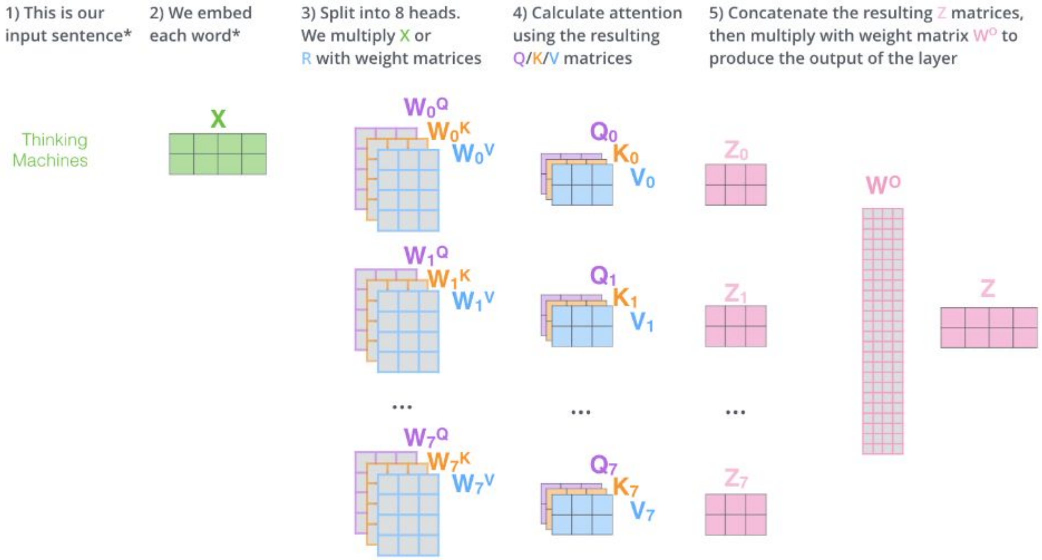


Attention is all you need ! *(Vaswani et al., 2019)*



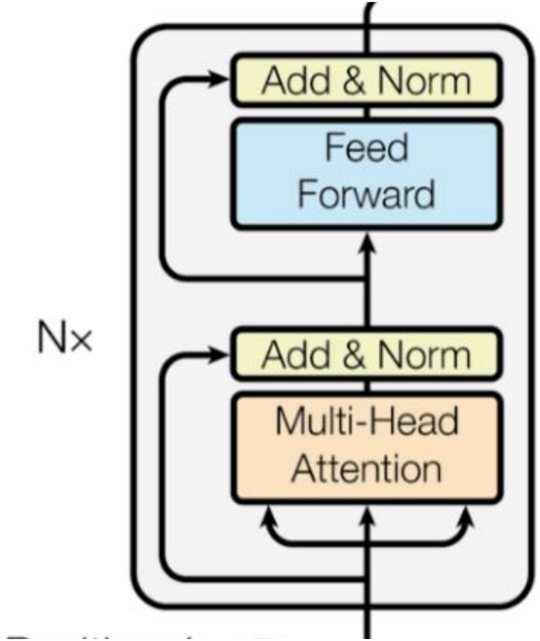
- On concatène les vecteurs produits par chaque tête (= un grand vecteur par entrée / mot)

- On utilise une transformation linéaire pour revenir à la taille de l'embedding



(https://jalammar.github.io/illustrated-transformer/)

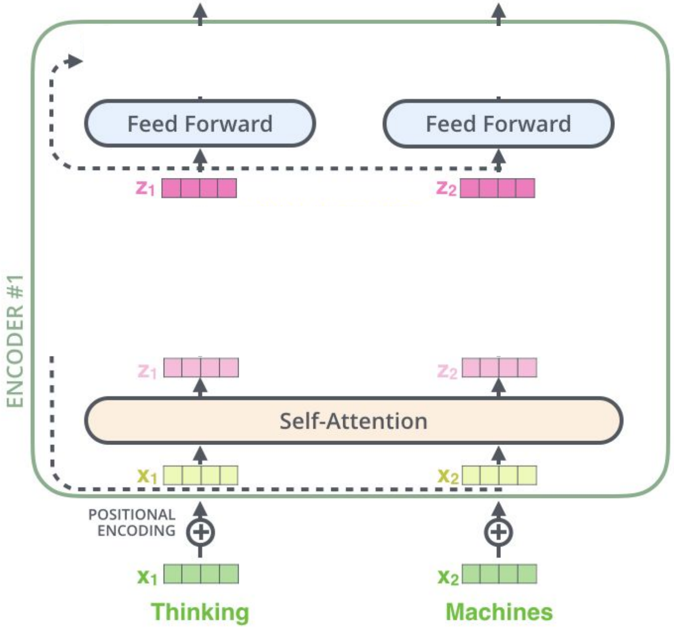
Transformer block

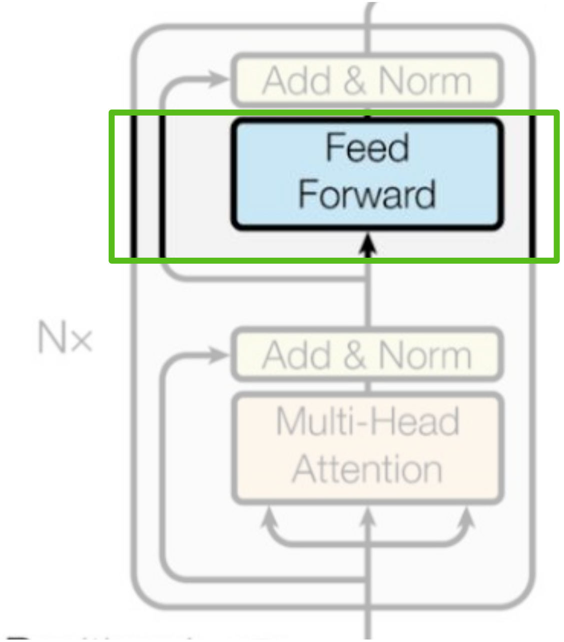


Transformer block

Feedforward

- Introduction de non-linéarités grâce à un feedforward





(https://jalammar.github.io/illustrated-transformer/)

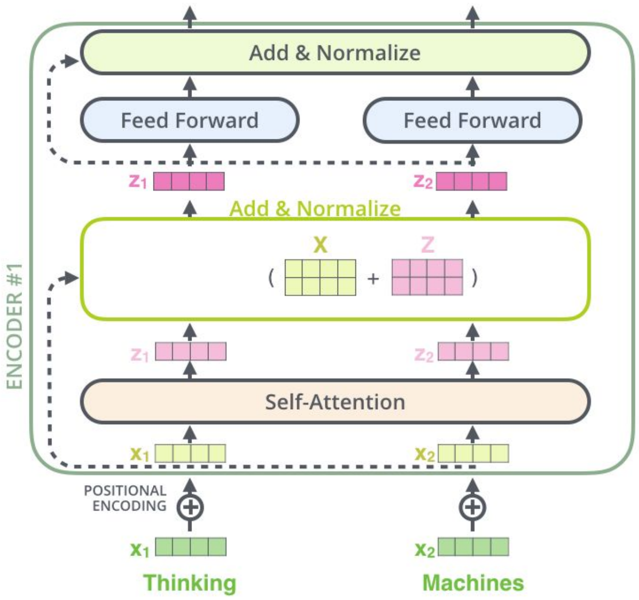
Transformer block

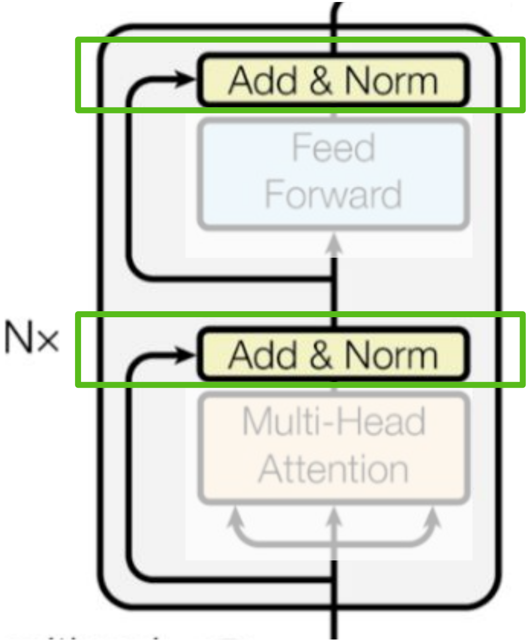
Residual connections

Intuition => garder de l'information sur l'embedding avant l'attention

En pratique: juste une somme

Note: ResNet (He et al., 2015)





(https://jalammar.github.io/illustrated-transformer/)

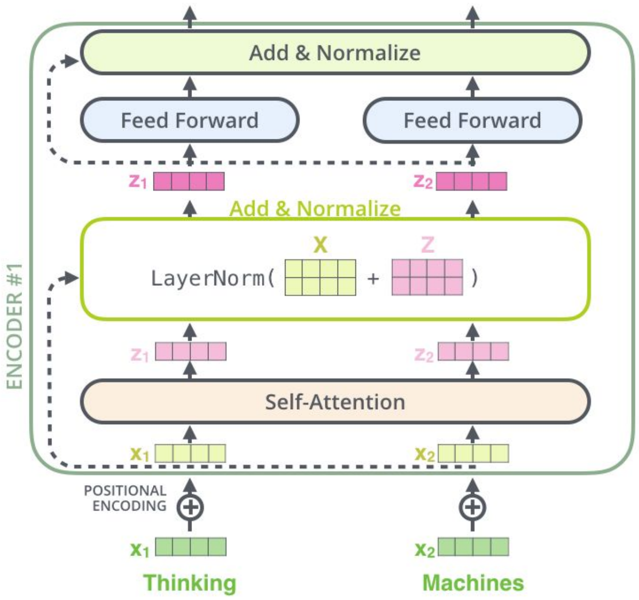
Transformer block

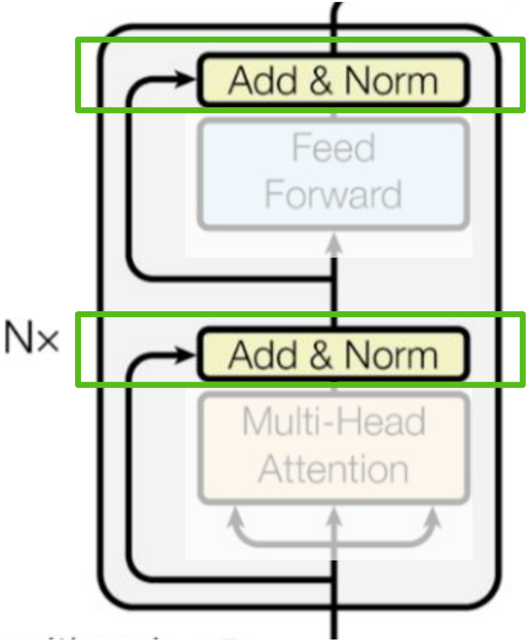
Layer normalization

Intuition => stabiliser l'apprentissage et éviter l'explosion de gradients

En pratique: on normalize (moyenne = 0, variance = 1) chaque vecteur de sortie

Note: fonctionne avec les petits batchs à la différence d'une BatchNorm





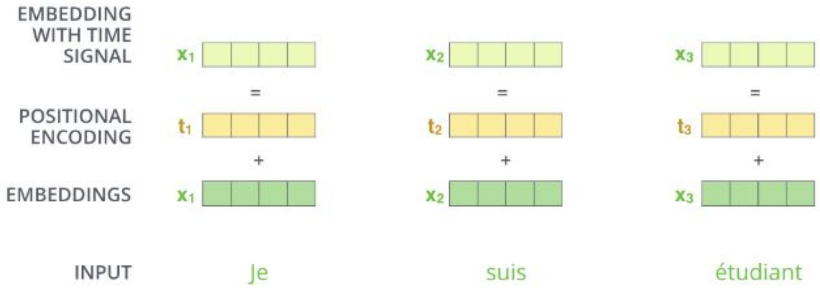
(https://jalammar.github.io/illustrated-transformer/)

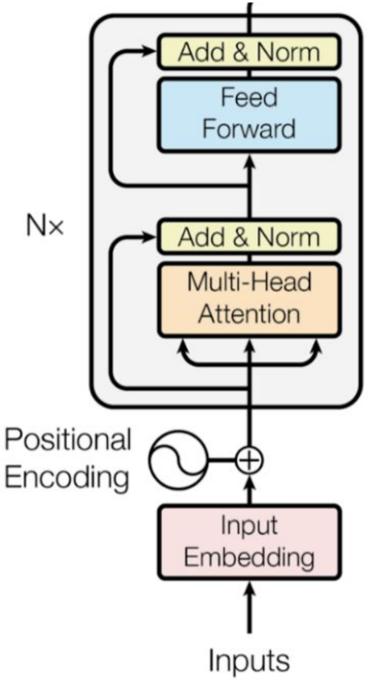
Positional Encoding

Rappel: La (Self-)Attention opère sur des ensembles != RNNs

=> L'opération n'est pas impactée par l'ordre des mots

Solution: Ajouter une information à l'embedding dépendant de la position





Note: Sinusoidal PE dans le papier d'origine

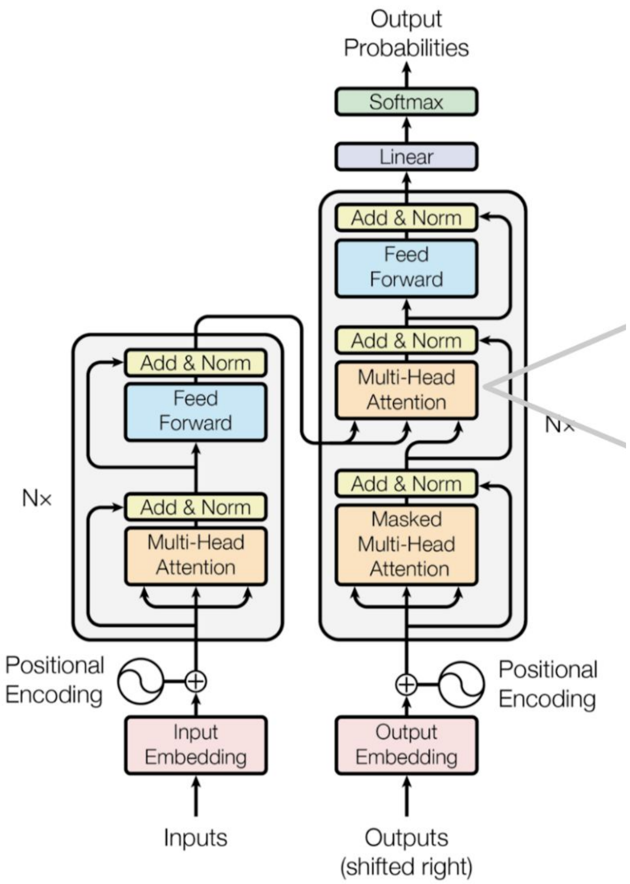
(https://jalammar.github.io/illustrated-transformer/)

Encoder-Decoder

L'architecture complète !

Encoder: Produire une représentation de l'input *(un vecteur / input => concaténés dans une seule matrice de dimension N)*

Decoder: Générer à partir de cette représentation



Encoder-Decoder

Causal Masking

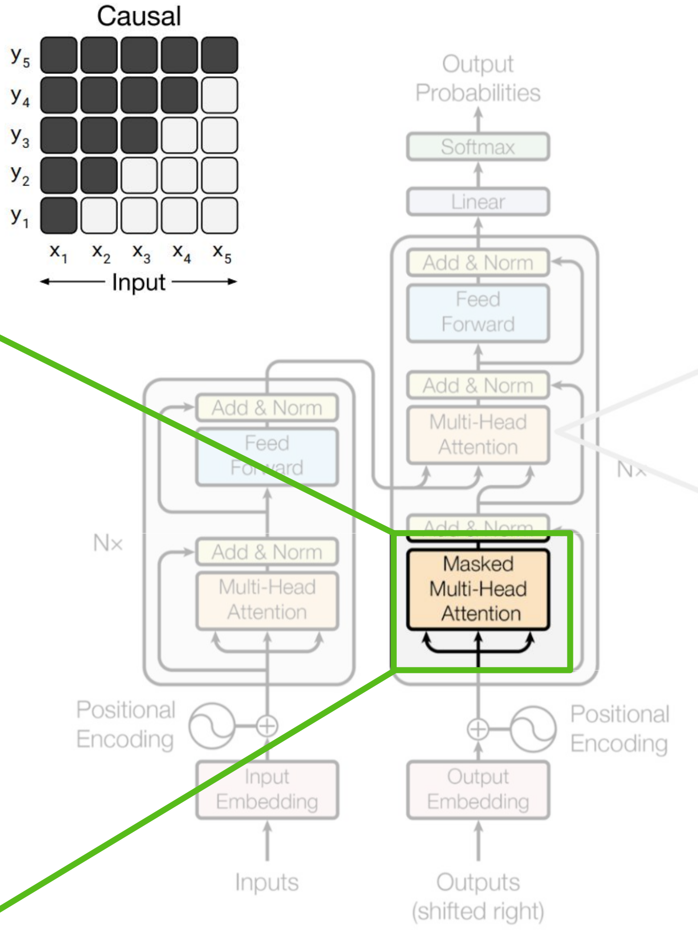
- Quand on génère du texte, on ne connaît pas le mot d'après (on y a pourtant accès à l'entraînement)

- **Causal Attention**: L'attention ne peut utiliser que les K, V d'avant le Q

- Solution: Appliquer un **masque** sur la sortie de l'attention avant la softmax

$$a_{ij} = \text{softmax}\left(\frac{q_i k_j^\top}{\sqrt{d_k}}\right) \Rightarrow a_{ij} = \text{softmax}\left(\frac{q_i k_j^\top + m_j}{\sqrt{d_k}}\right)$$

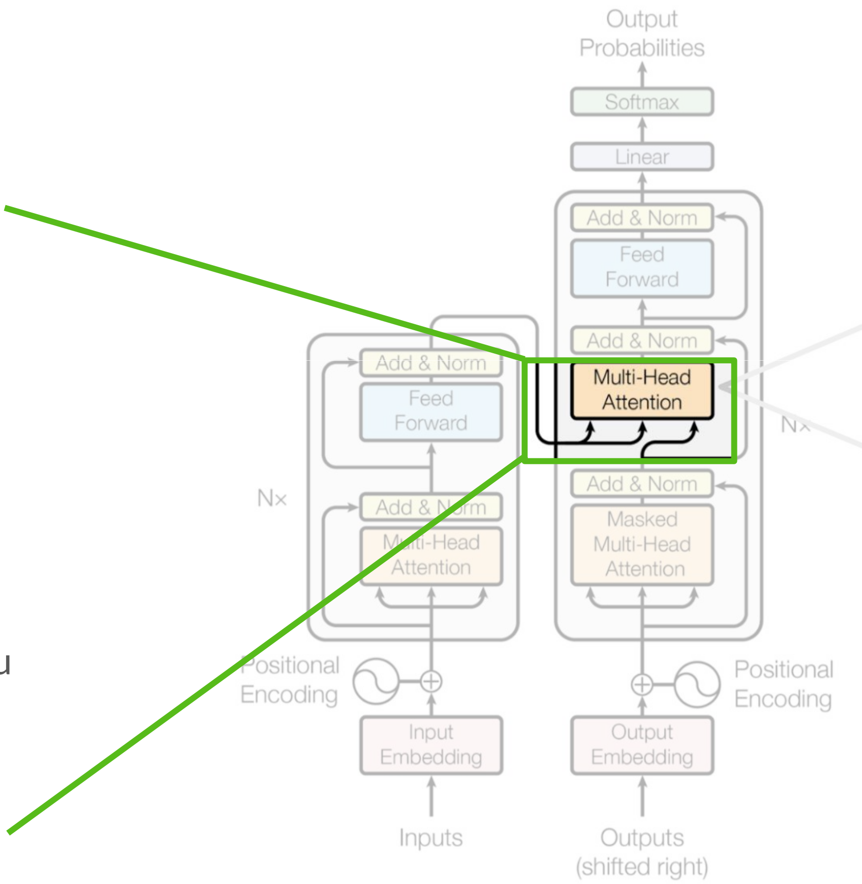
$$m_j = \begin{cases} 0 & \text{si } j \leq i \\ -\inf & \text{si } j > i \end{cases}$$



Encoder-Decoder

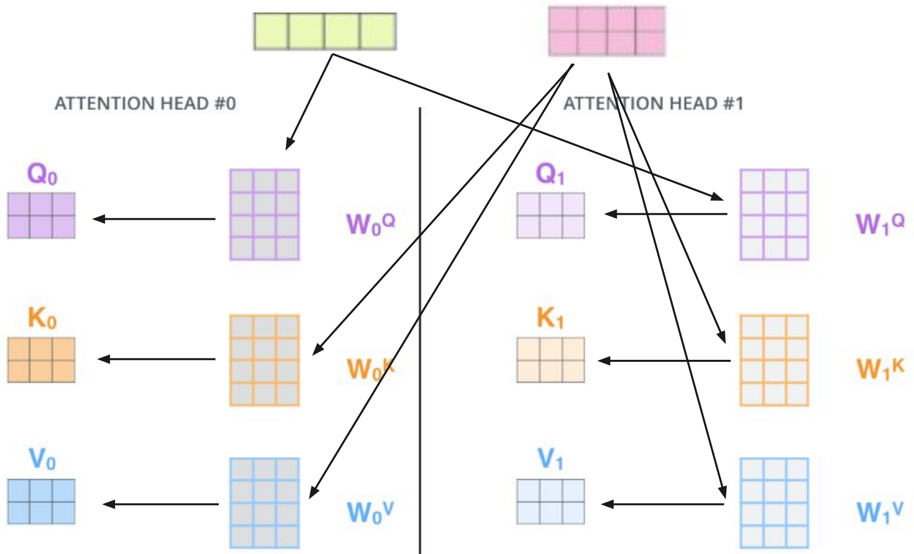
Attention over encoder's outputs

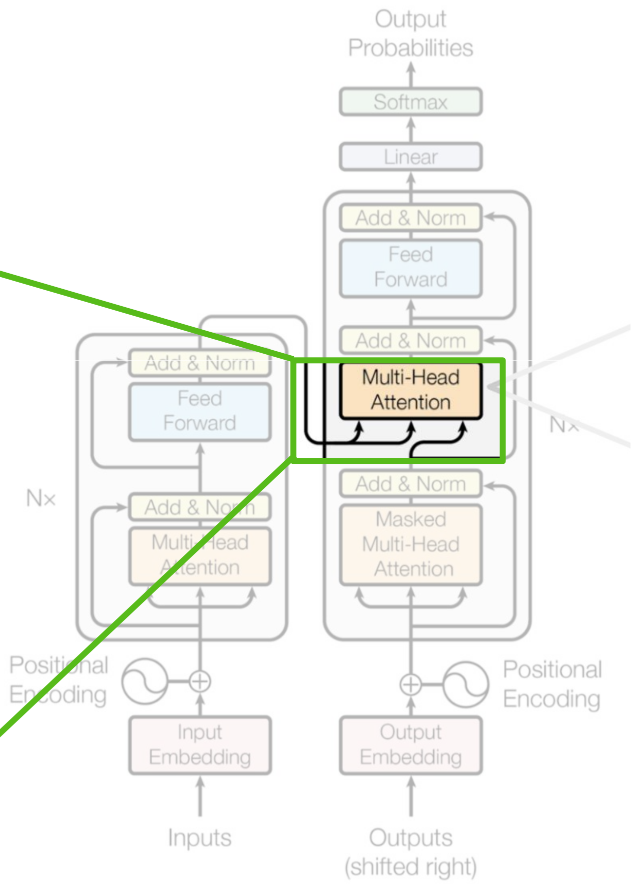
- Après un premier block de Self-Attention (+ residual & Layer Norm)
- **Attention sur l'encoding**: K, V viennent des vecteurs encodés par l'encoder, Q vient du Decoder
- La sortie devient une combinaison des encodings
- Le residual permet de garder aussi la représentation du Decoder



Encoder-Decoder

Attention over Encoder's outputs





(https://jalammar.github.io/illustrated-transformer/)

Encoder-Decoder

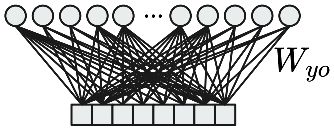
Decoding

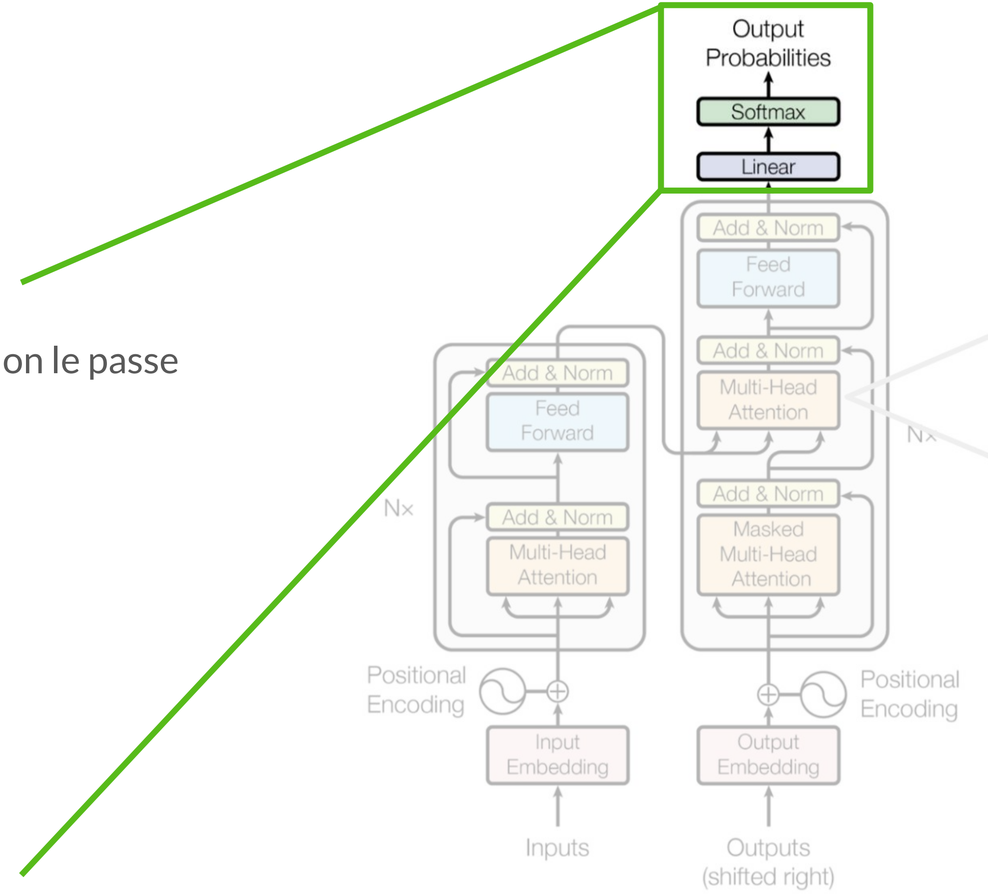
Comme avec les RNNs

=> on prend le vecteur produit par le Decoder et on le passe dans une couche linéaire + softmax

softmax $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$

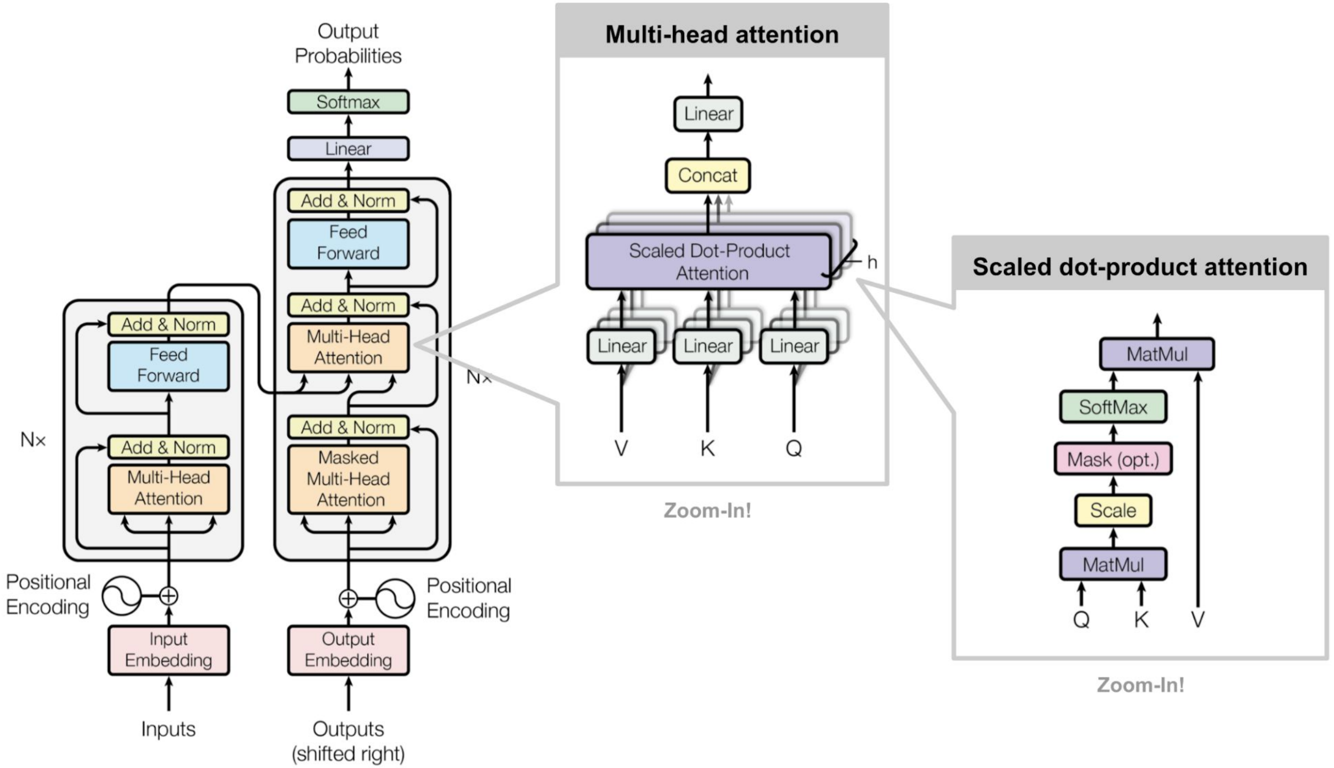
$$y_1 = W_{yo} o_1$$





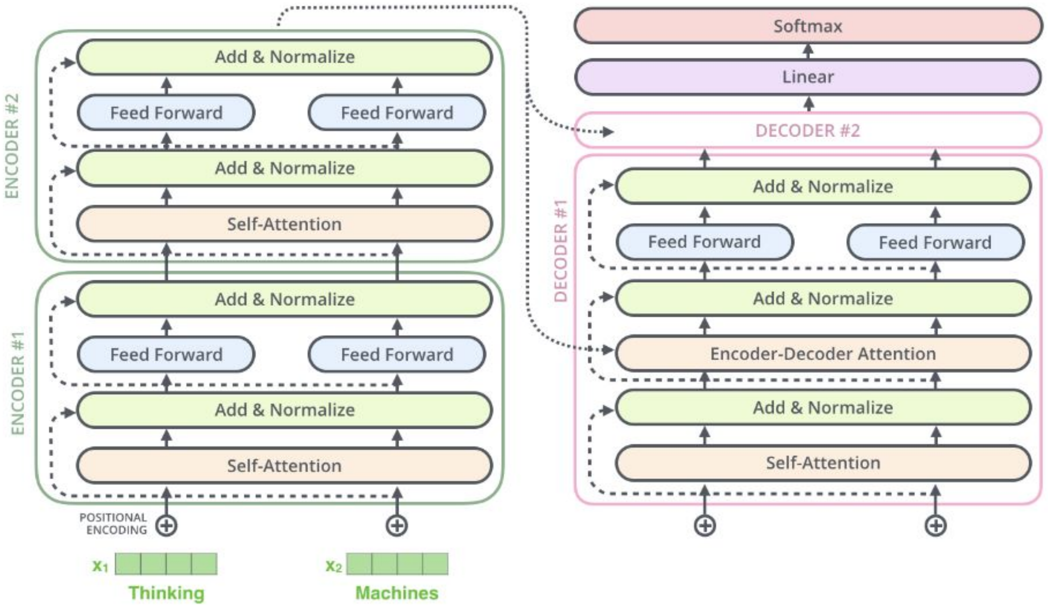
Encoder-Decoder

The full picture



Encoder-Decoder

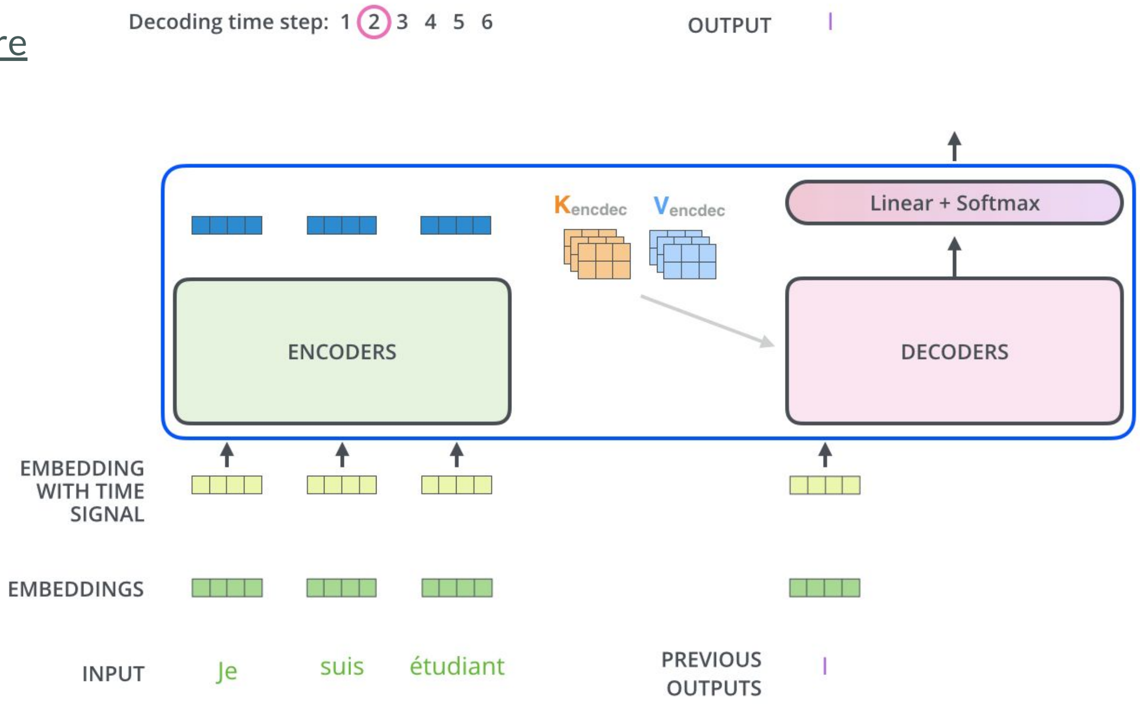
The full picture



(https://jalammar.github.io/illustrated-transformer/)

Encoder-Decoder

The full picture



(https://jalammar.github.io/illustrated-transformer/)

TP

Transformer for translation

https://colab.research.google.com/drive/11bJ6x0uoz_ApD-DdomTe-0fHArDB3ZmH?usp=sharing