Transformer

Mohamed Abbas KONATE

Plan

Pourquoi c’est une révolution ?

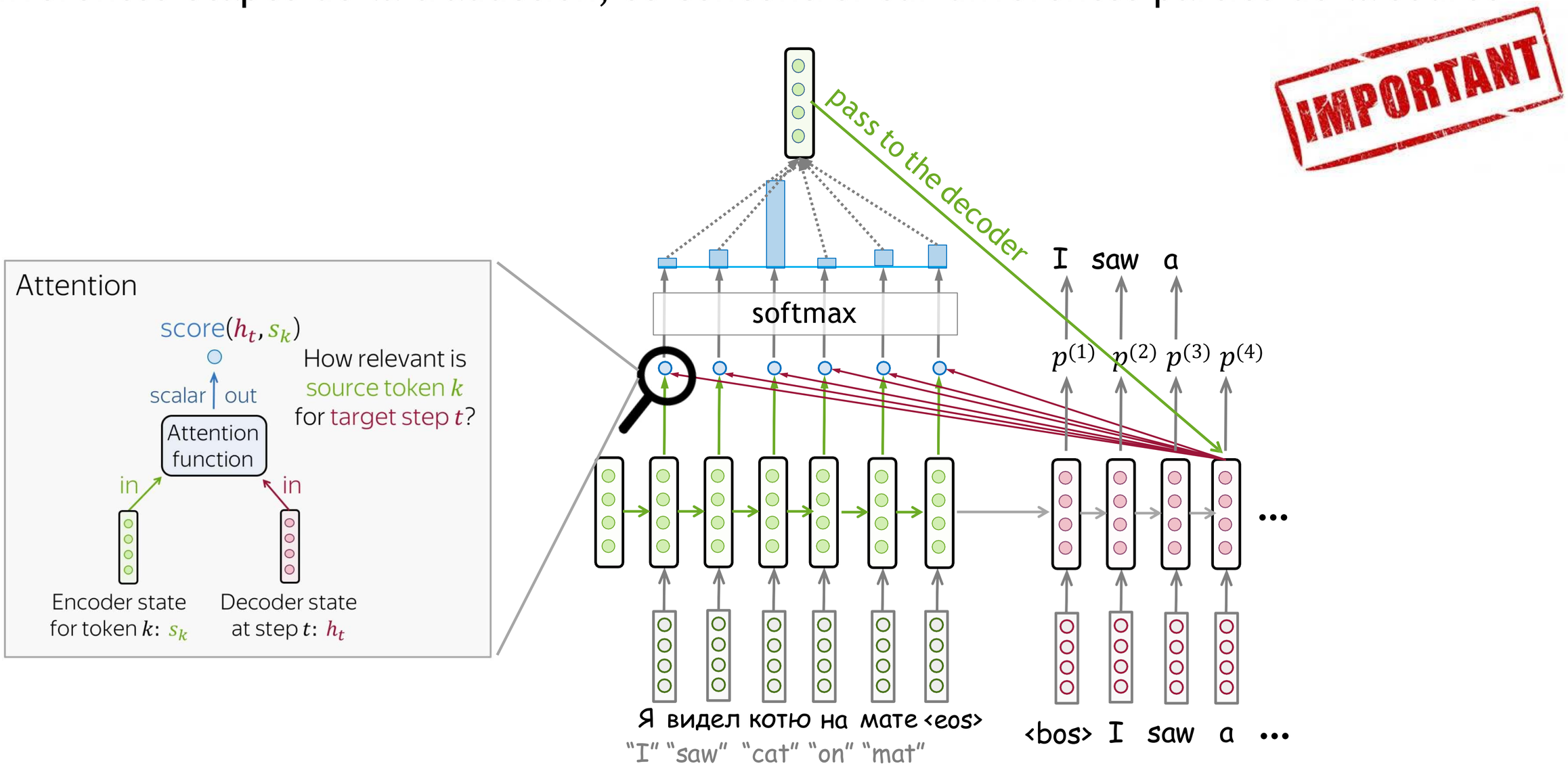
Self Attention
Position Embedding
Skip Connection
Layer Normalisation
Masked Attention

But du chapitre : Découvrir les transformers

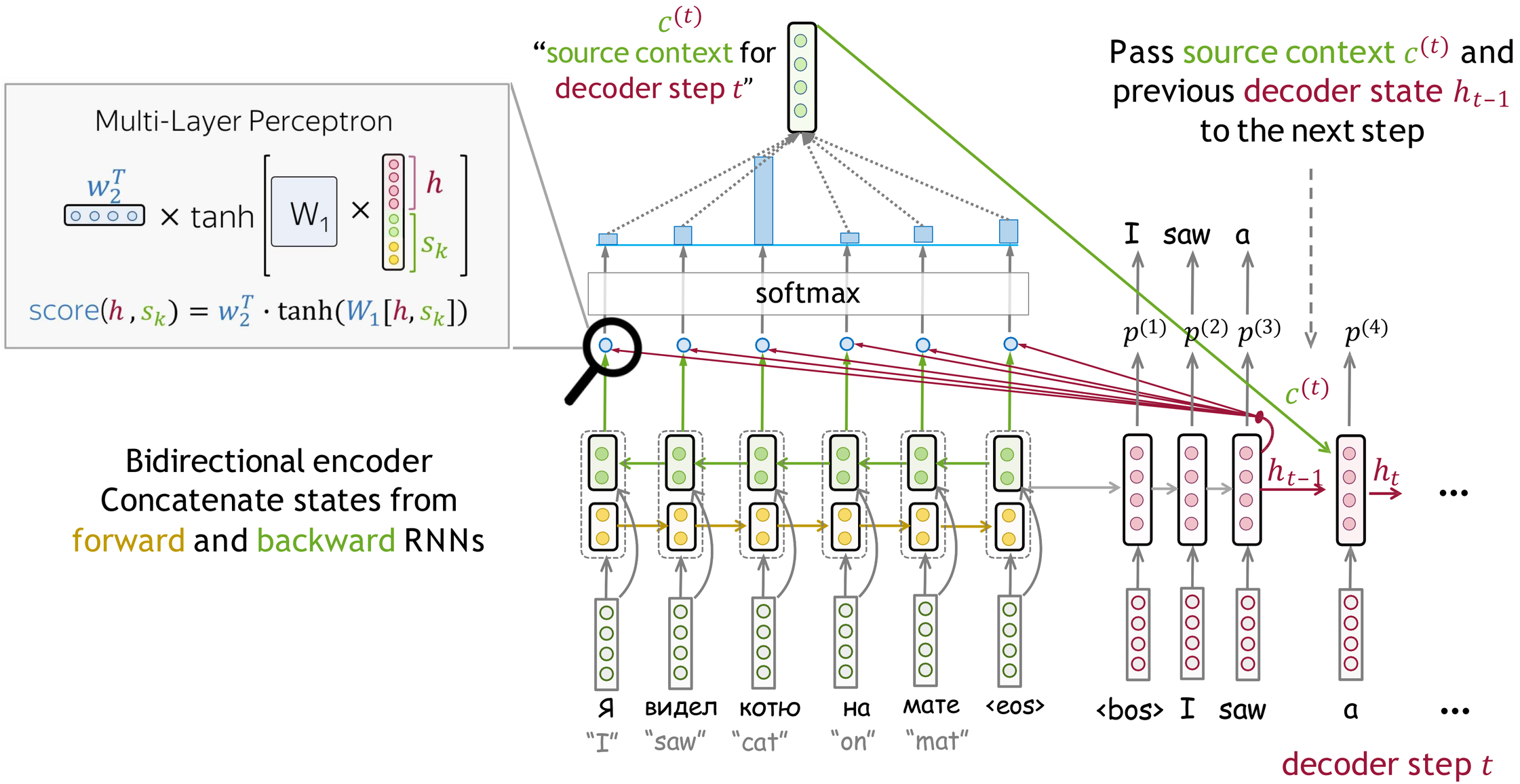
Rappel

Attention dans les RNN avec Encodeur-Décodeur

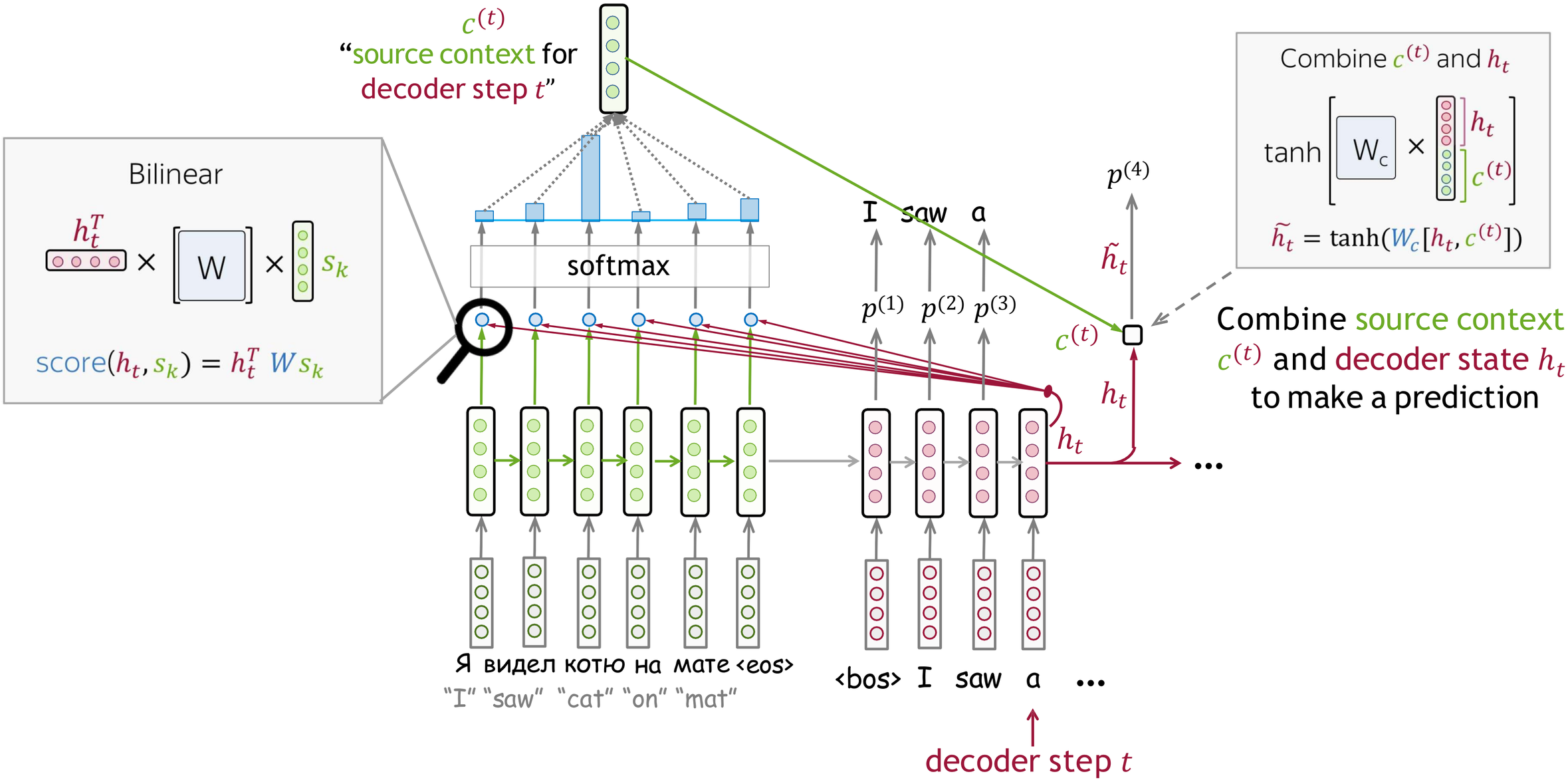
A différentes étapes de la traduction, se concentrer sur différentes parties de la source



Bahdanau Model (the original attention model)



Luong Model


$c^{(t)}$
"source context for decoder step t"
Bilinear
h_t^T × W × s_k
score$(h_t, s_k) = h_t^T W s_k$
softmax
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
$\tilde{h}_t$
$c^{(t)}$
h_t
Combine $c^{(t)}$ and h_t
tanh [W_c × h_t ; $c^{(t)}$]
$\tilde{h}_t = \tanh(W_c[h_t, c^{(t)}])$
Combine source context $c^{(t)}$ and decoder state h_t to make a prediction
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a ...
decoder step t

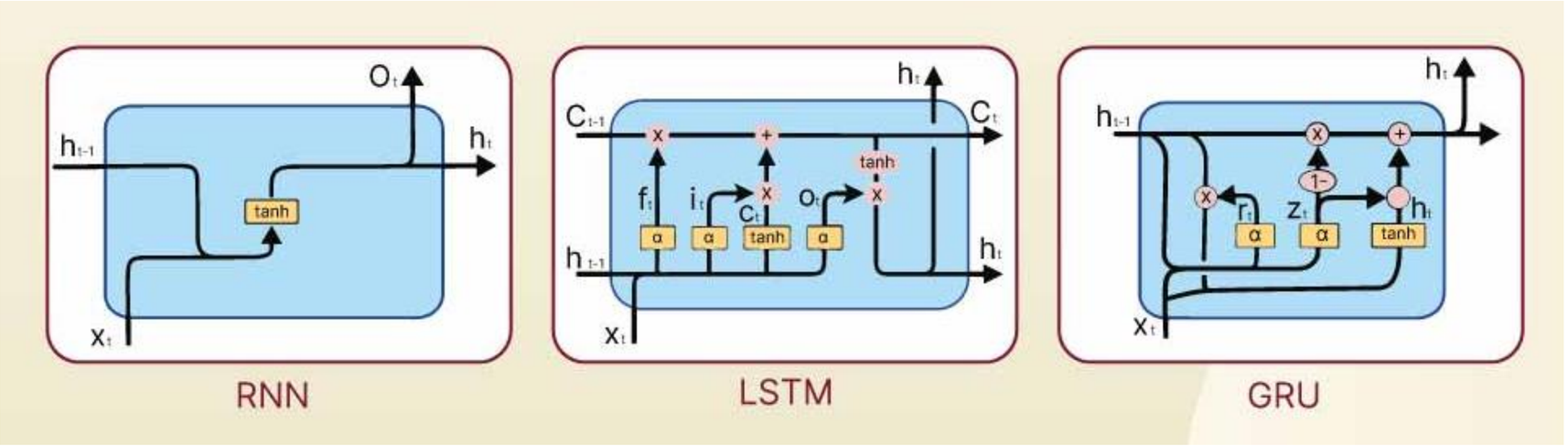
Entrainer un modèle Sequence à Séquence

Ingérer des séquences de tailles différentes : **Réseaux de neurones récurrents**

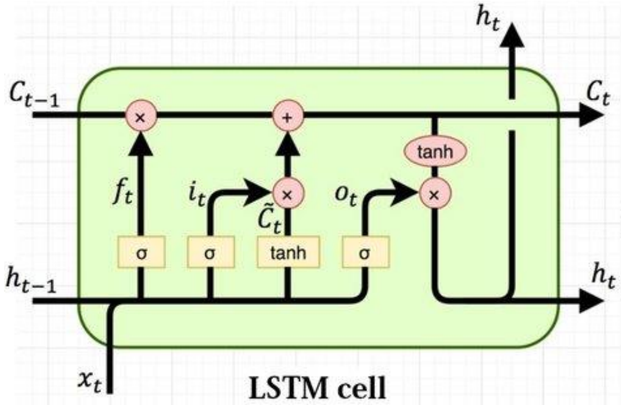
Entrainer un modèle Séquence à Séquence

Ingérer des séquences de tailles différentes : **Réseaux de neurones récurrents**

Evanescence ou explosion du gradient : LSTM ou GRU au lieu Vanilla RNN

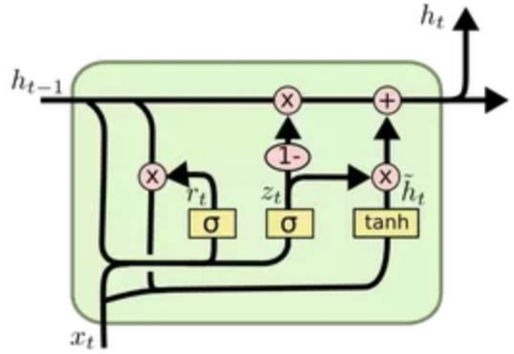


Entrainer un modèle Séquence à Séquence



LSTM cell

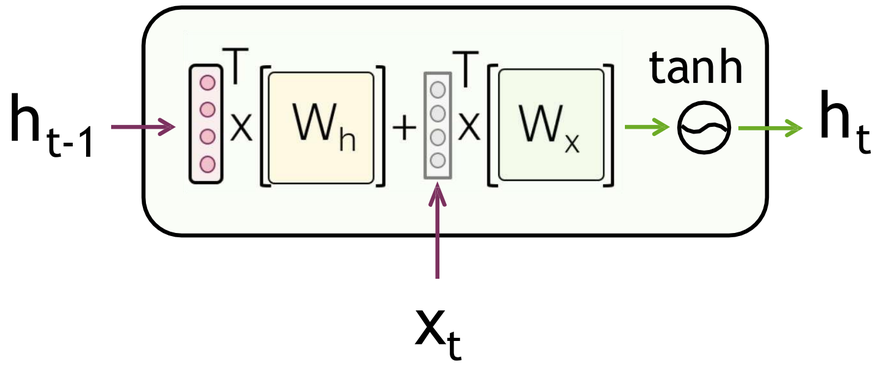
$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$



GRU cell

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

$$h_t = \tanh(h_{t-1} W_h + x_t W_x)$$



Vanilla RNN

Entrainer un modèle Séquence à Séquence

Ingérer des séquences de tailles différentes : **Réseaux de neurones récurrents**

Evanescence ou explosion du gradient : LSTM ou GRU au lieu Vanilla LSTM

Améliorer le contexte du décodeur : **Attention**

Entrainer un modèle Séquence à Séquence

Des problèmes persistent :

Dépendances à long terme

Entrainer un modèle Séquence à Séquence

Des problèmes persistent :

Dépendances à long terme

Exécution séquentielle

Entrainer un modèle Séquence à Séquence

Des problèmes persistent :

Dépendances à long terme

Exécution séquentielle

Architecture complexe

Notre lettre au père Nöel

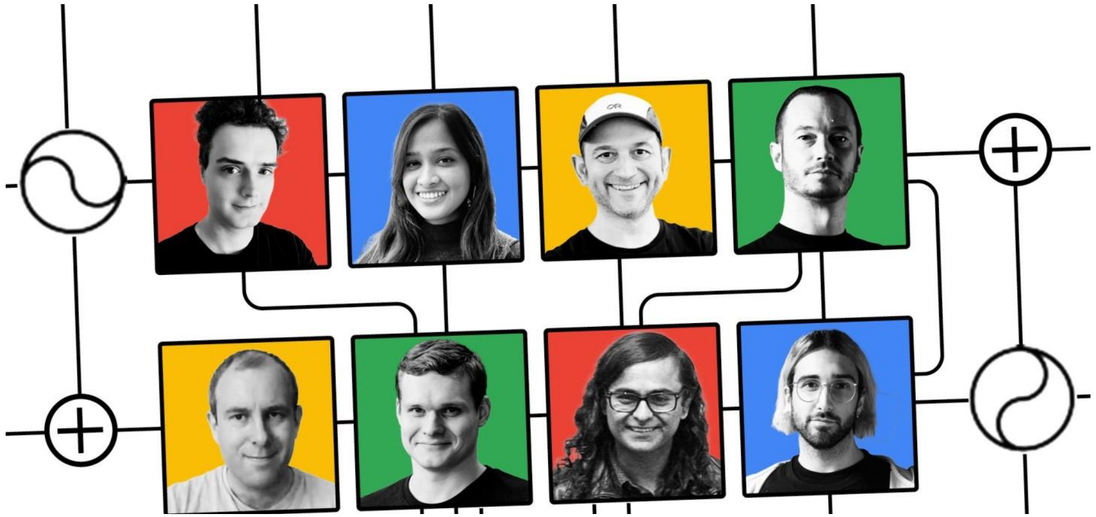
Des problèmes persistent :

Dépendances à long terme -> Gestion de contexte très grand

Exécution séquentielle -> Exécution parallèle

Architecture complexe -> Se passer de la récursivité

Nos Pères et Mères Nöel



Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Google Brain

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Idée: Attention is All You Need

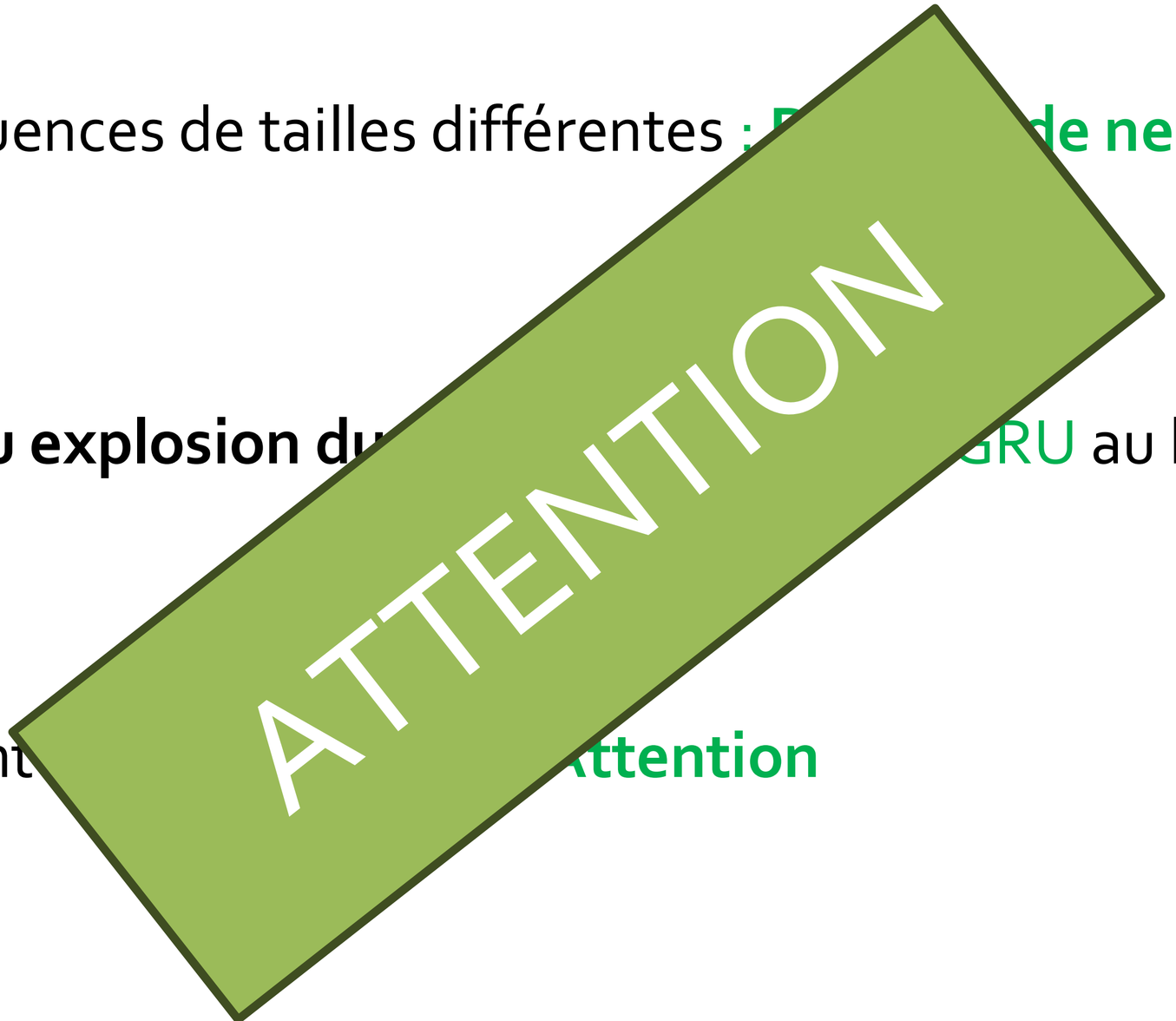
	Seq2seq without attention	Seq2seq with attention	Transformer
processing within encoder	RNN/CNN	RNN/CNN	attention
processing within decoder	RNN/CNN	RNN/CNN	attention
decoder-encoder interaction	static fixed-sized vector	attention	attention

Entrainer un modèle Séquence à Séquence

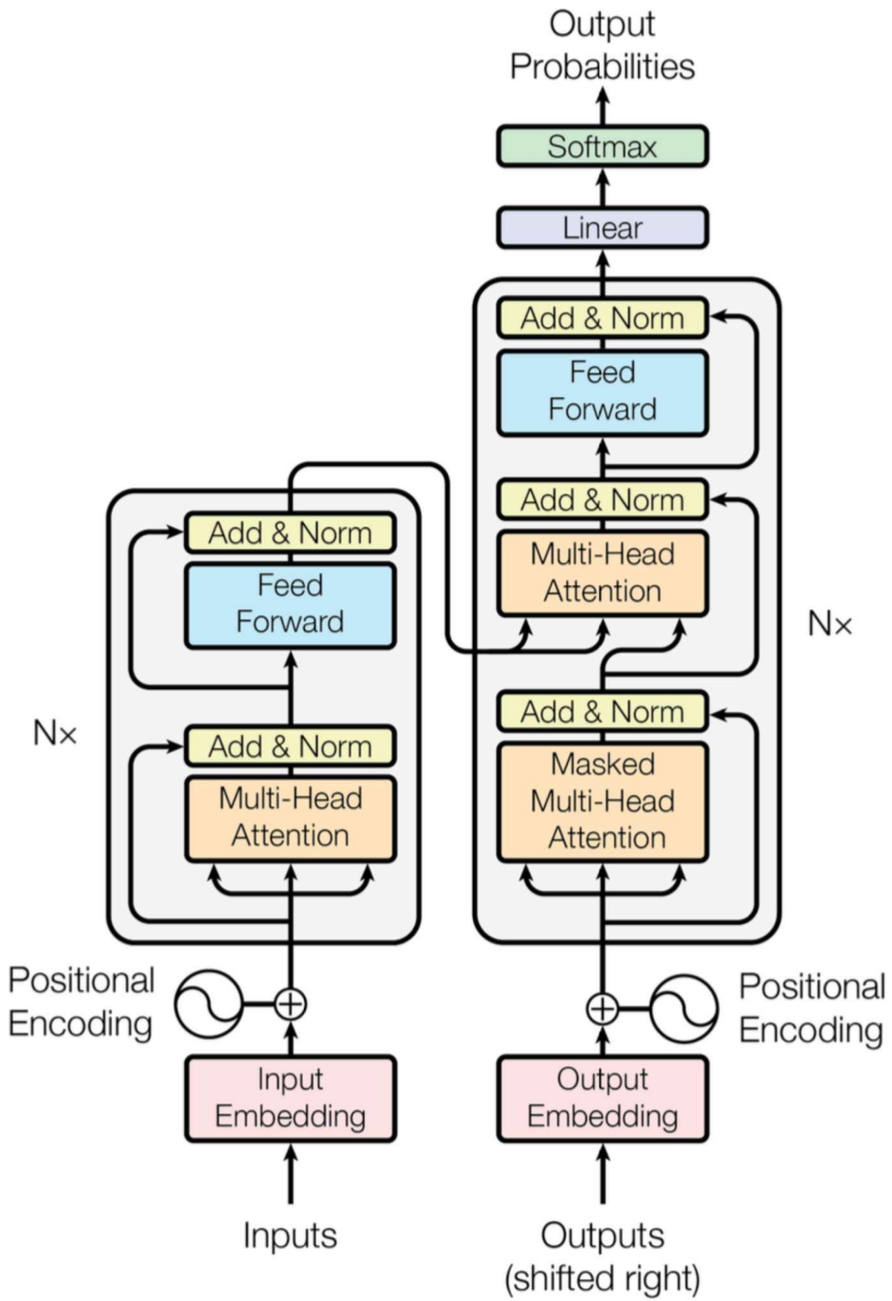
Ingérer des séquences de tailles différentes : **R de neurones récurrents**

Evanescence ou explosion du GRU au lieu Vanilla LSTM

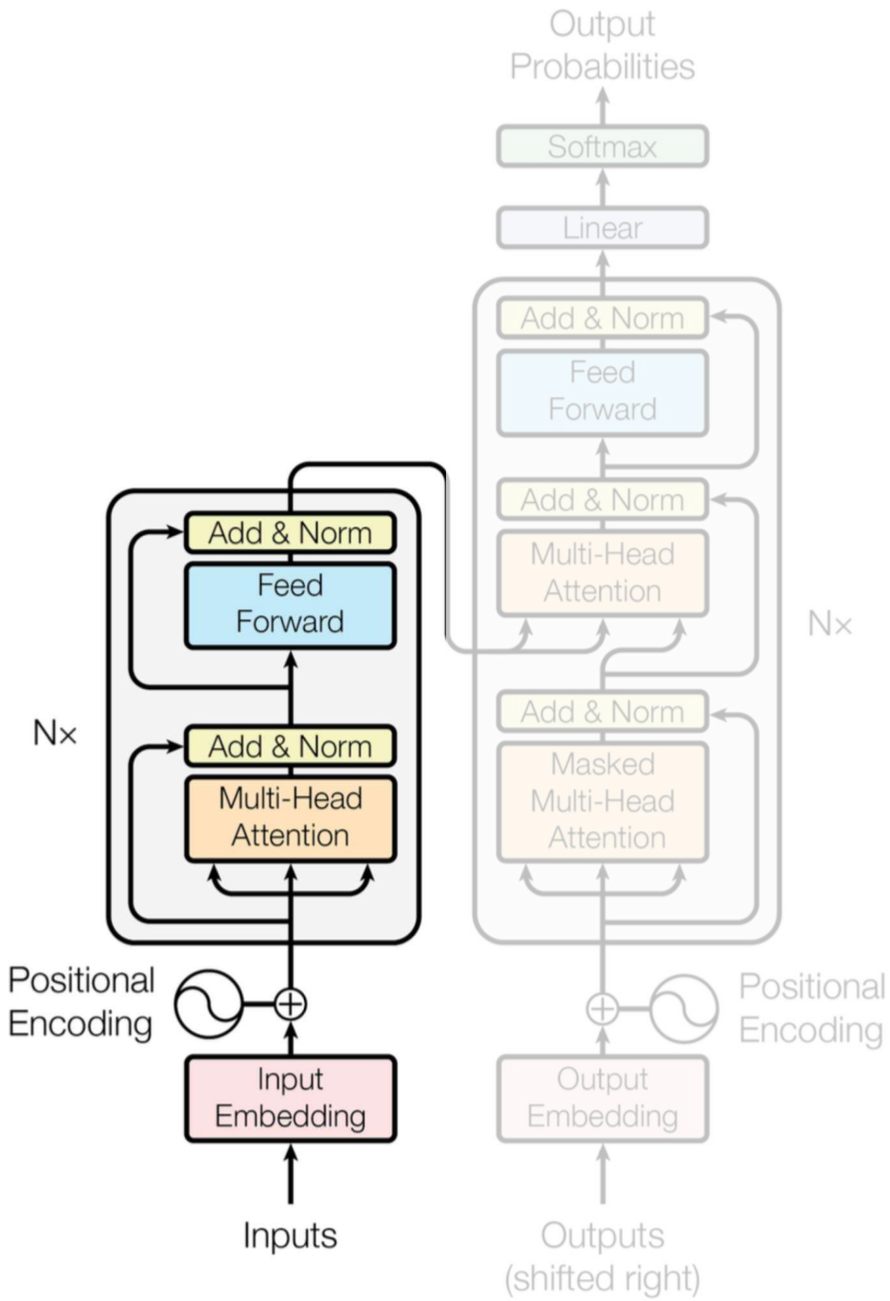
Améliorer le cont **ttention**



Transformer

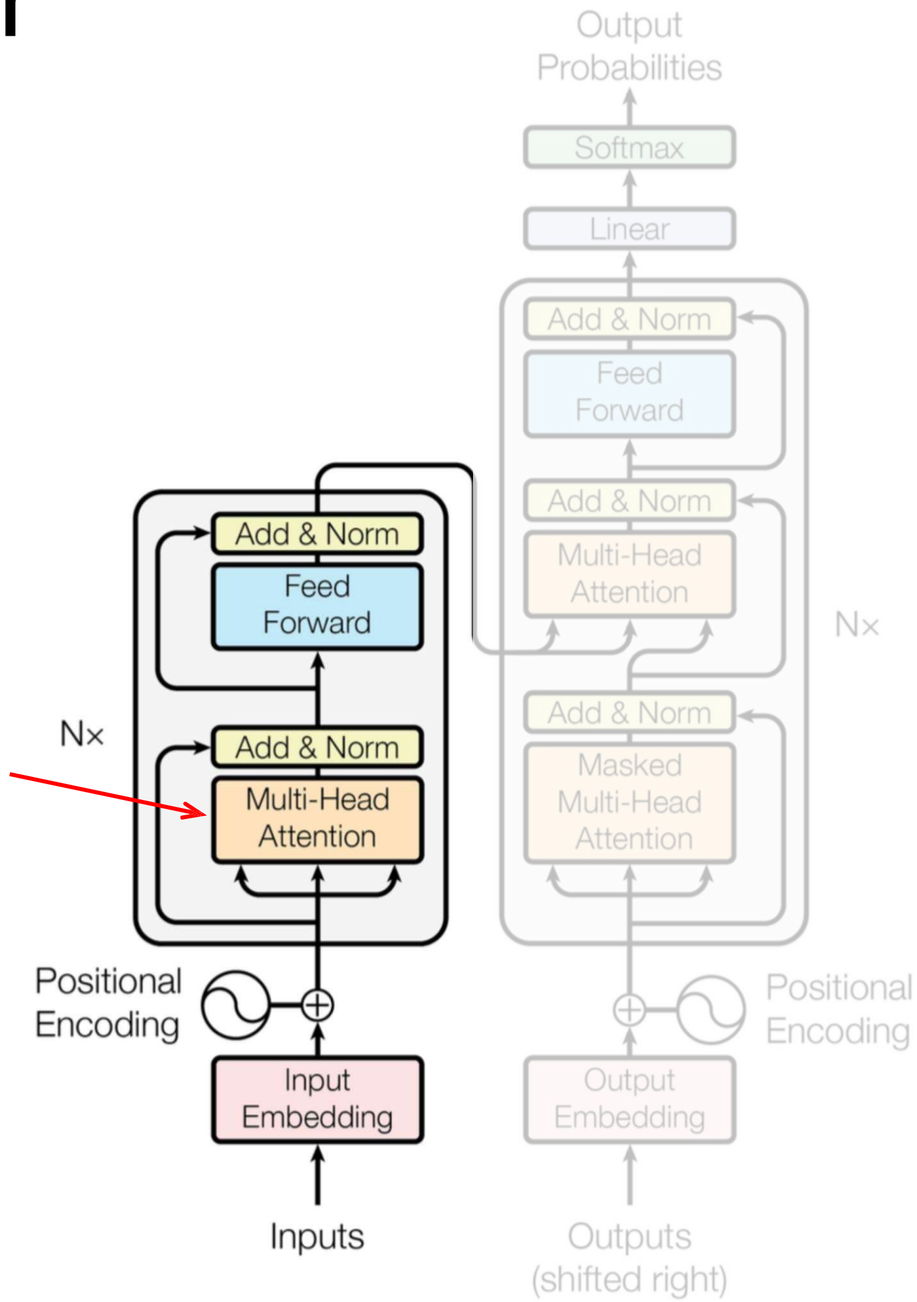


Transformer



Transformer

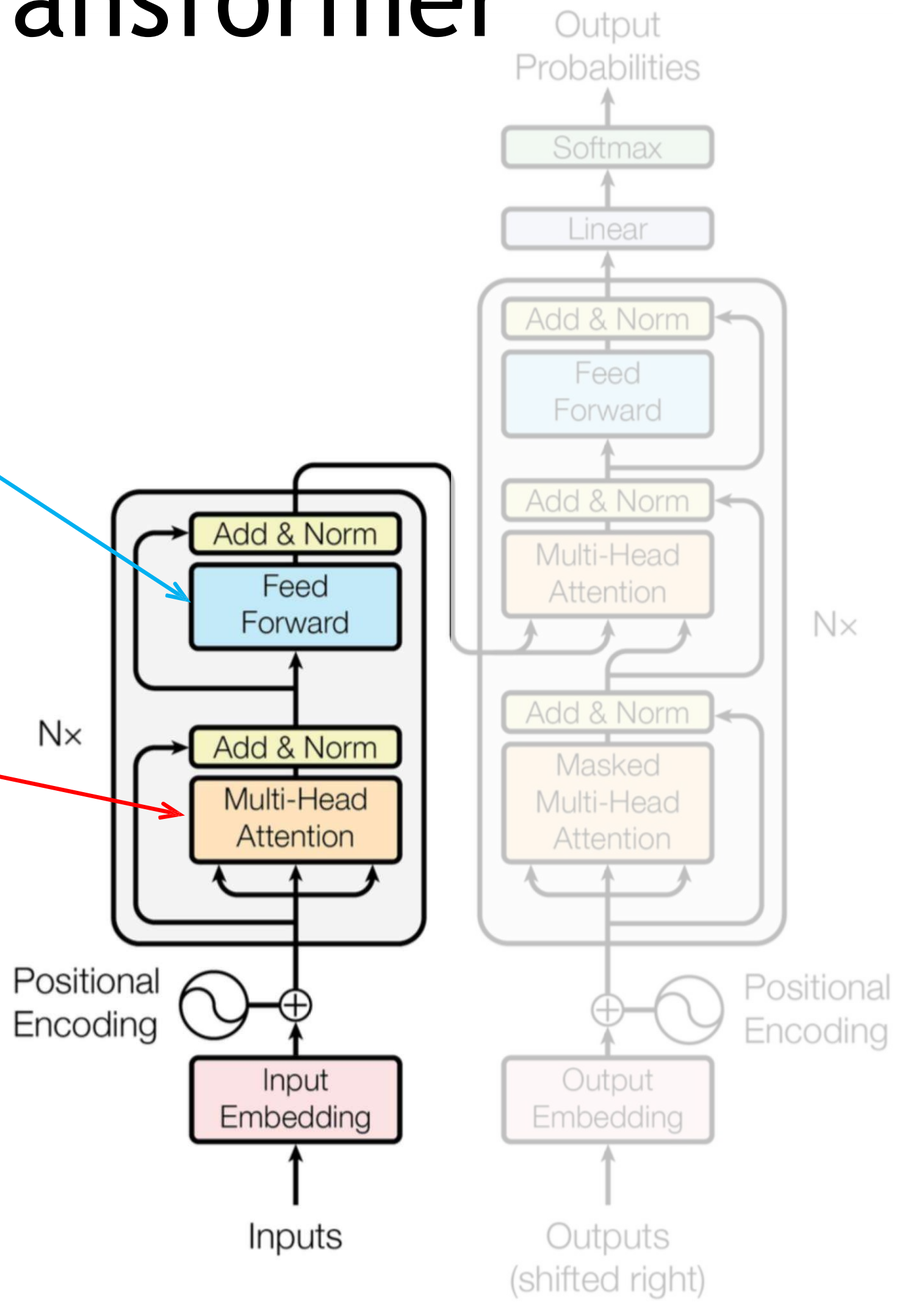
Encoder self-attention: tokens



Encodeur Transformer

Feed-forward network:

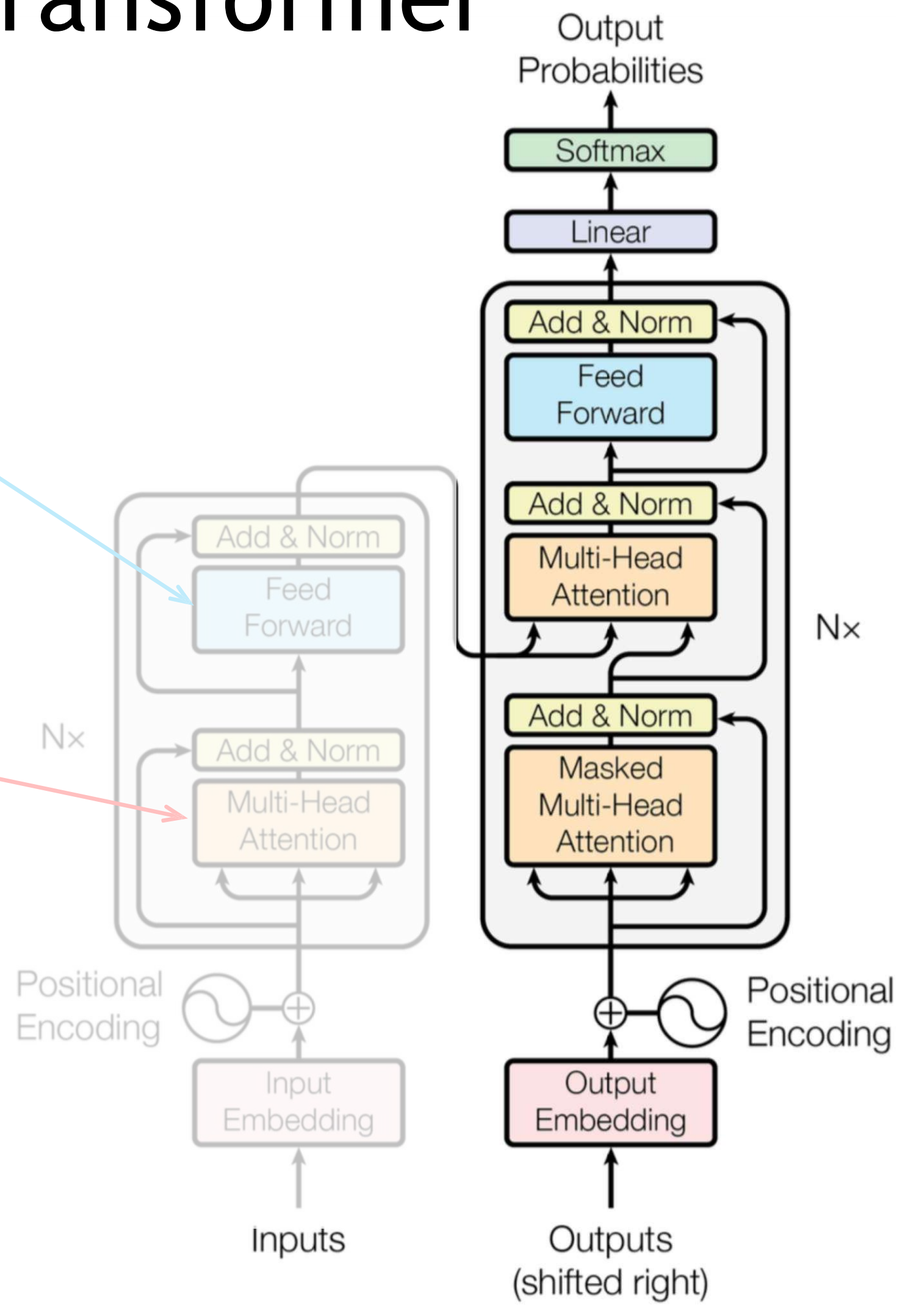
Encoder self-attention:
tokens look at each other



Décodeur : Transformer

Feed-forward network:

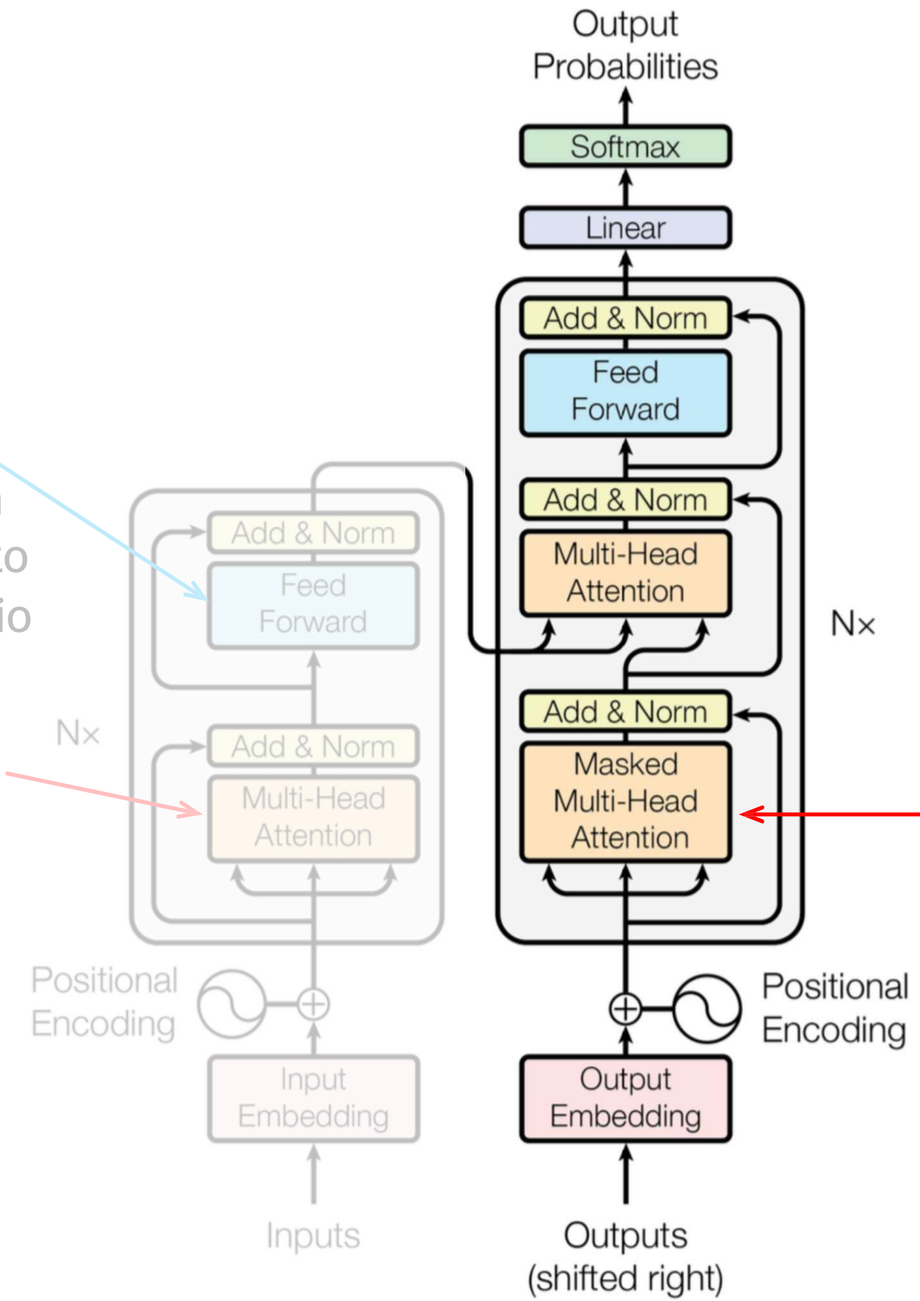
Encoder self-attention:
tokens



Feed-forward network:
after taking information from other tokens, take a moment to think and process this informatio

Encoder self-attention:
tokens look at each other

queries, keys, values are computed from encoder states



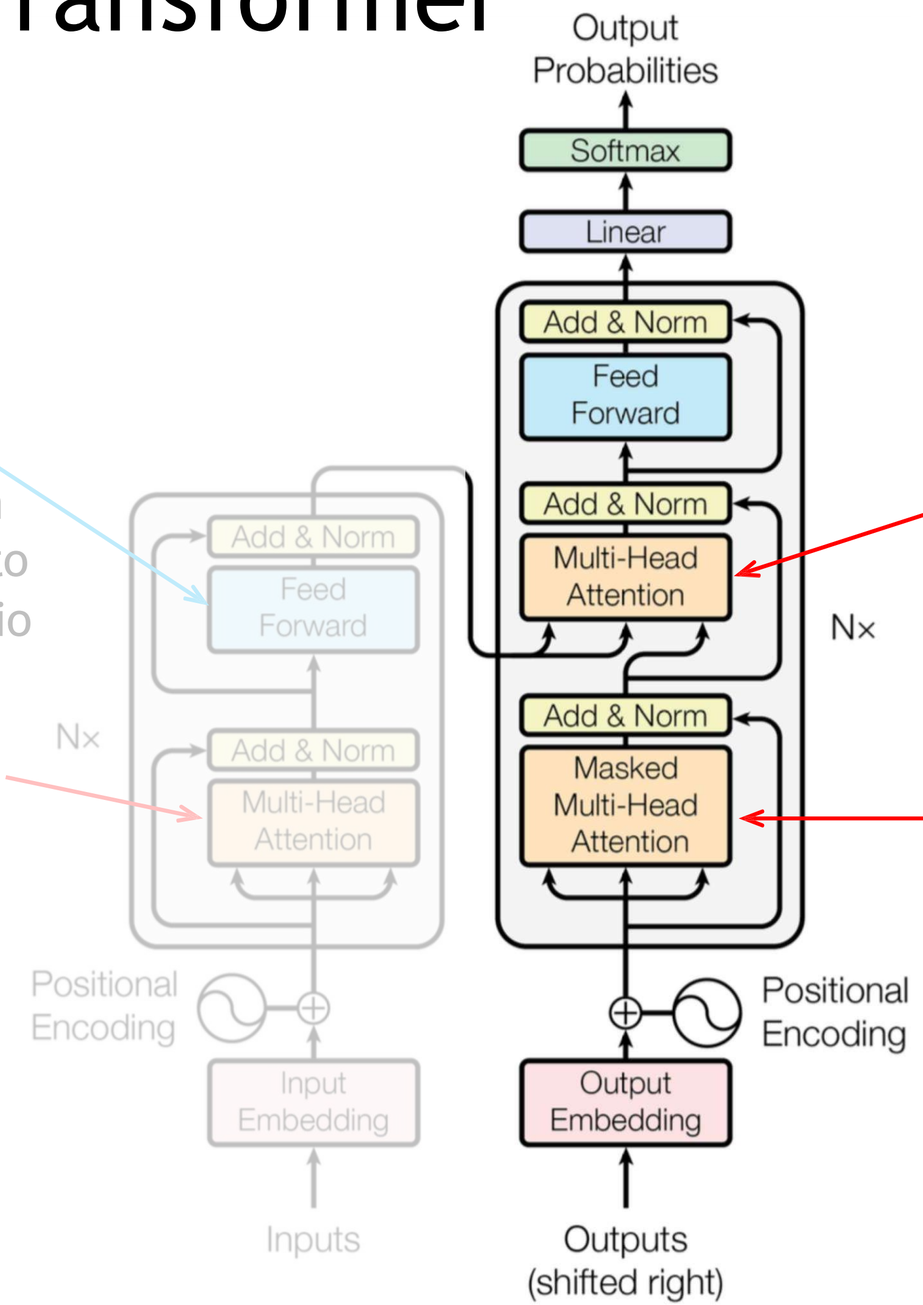
Decoder self-attention (masqué):

Décodeur : Transformer

Feed-forward network:
after taking information from other tokens, take a moment to think and process this informatio

Encoder self-attention:
tokens look at each other

queries, keys, values are computed from encoder states



Decoder-encoder attention:

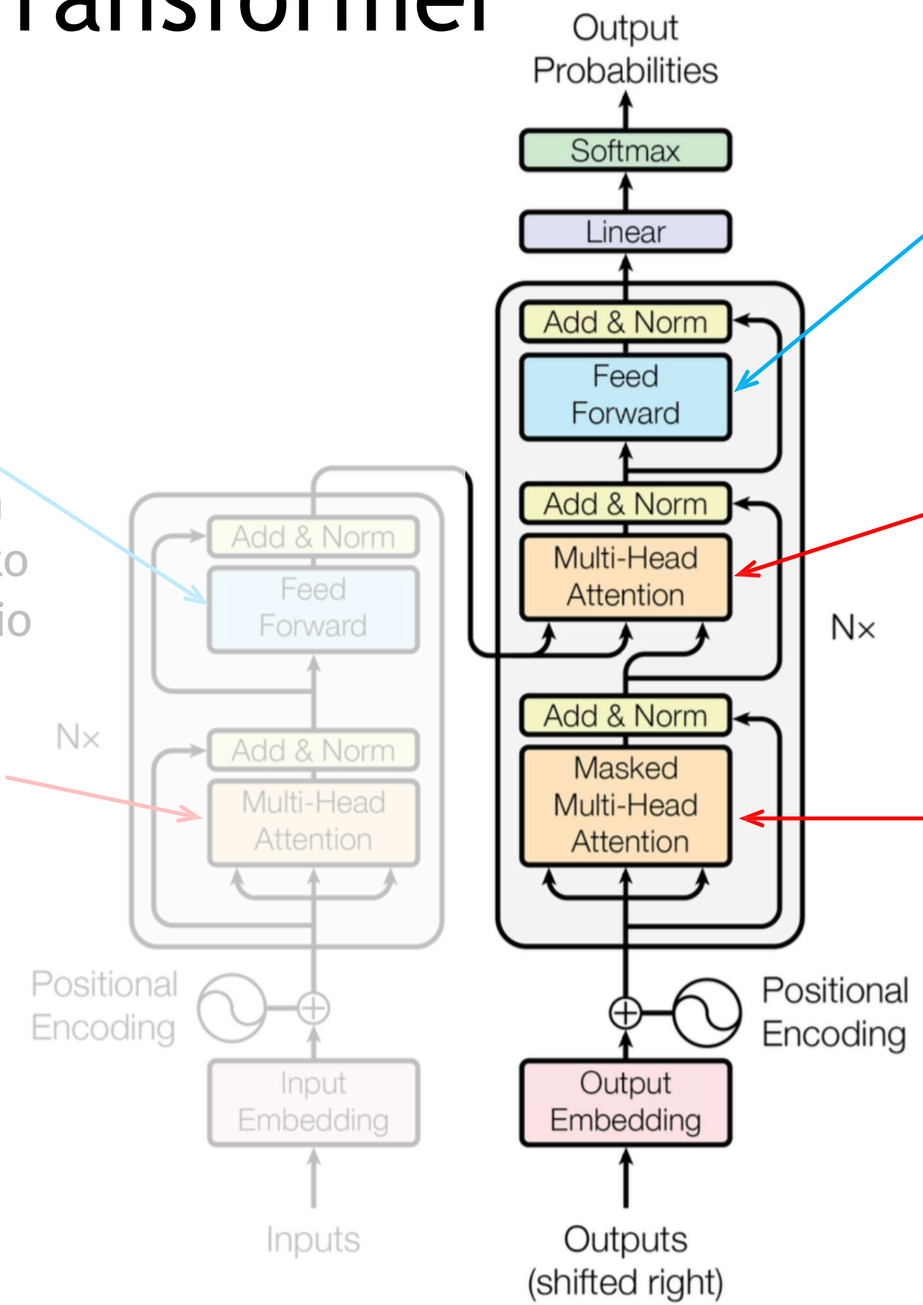
Decoder self-attention

Décodeur : Transformer

Feed-forward network: after taking information from other tokens, take a moment to think and process this informatio

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

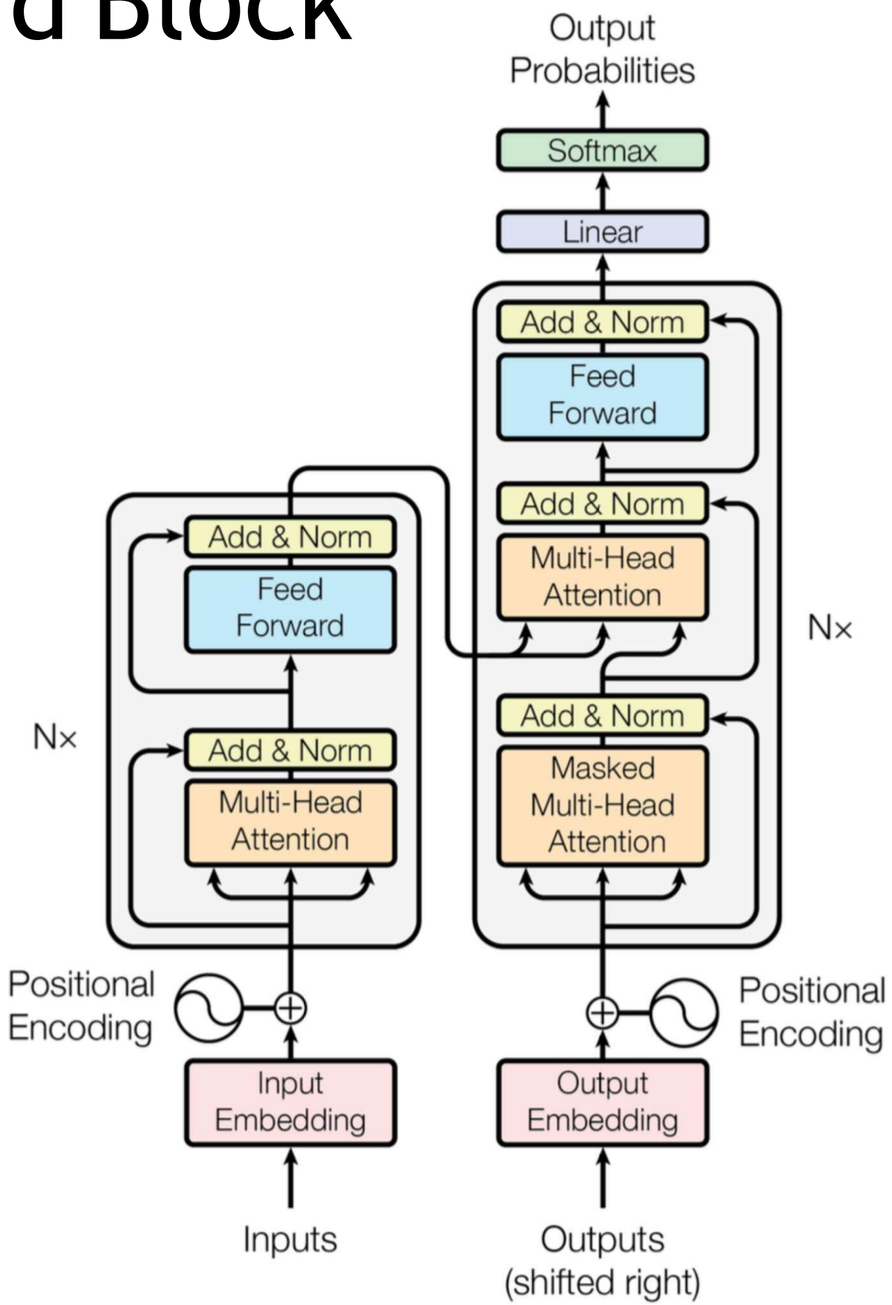


Feed-forward network

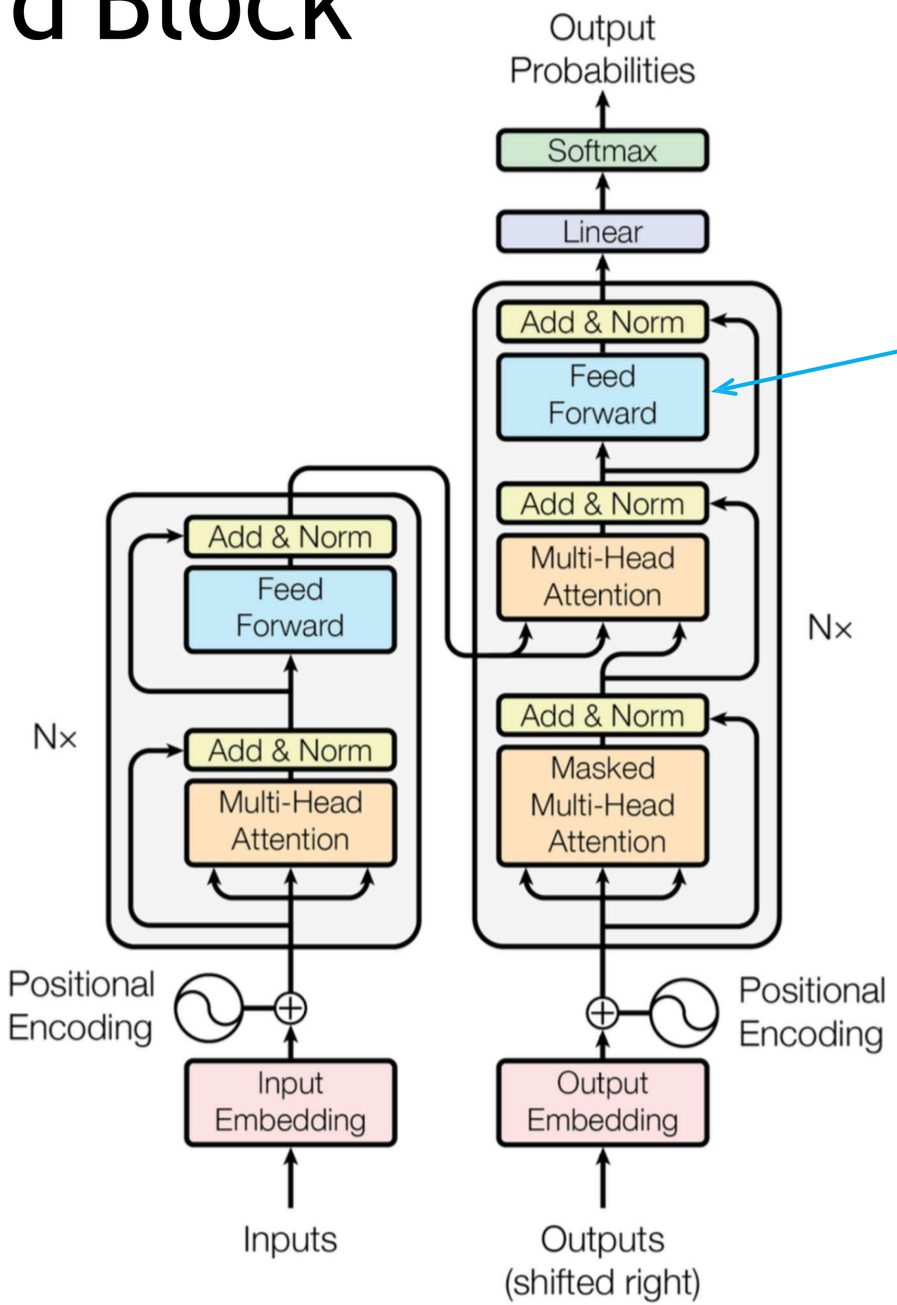
Decoder-encoder attention

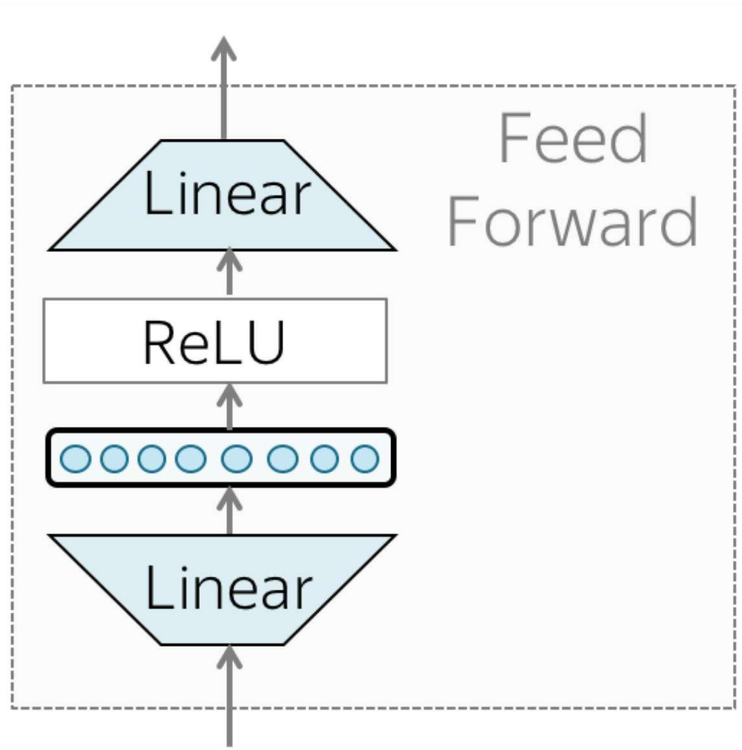
Decoder self-attention (masqué)

Feed Forward Block



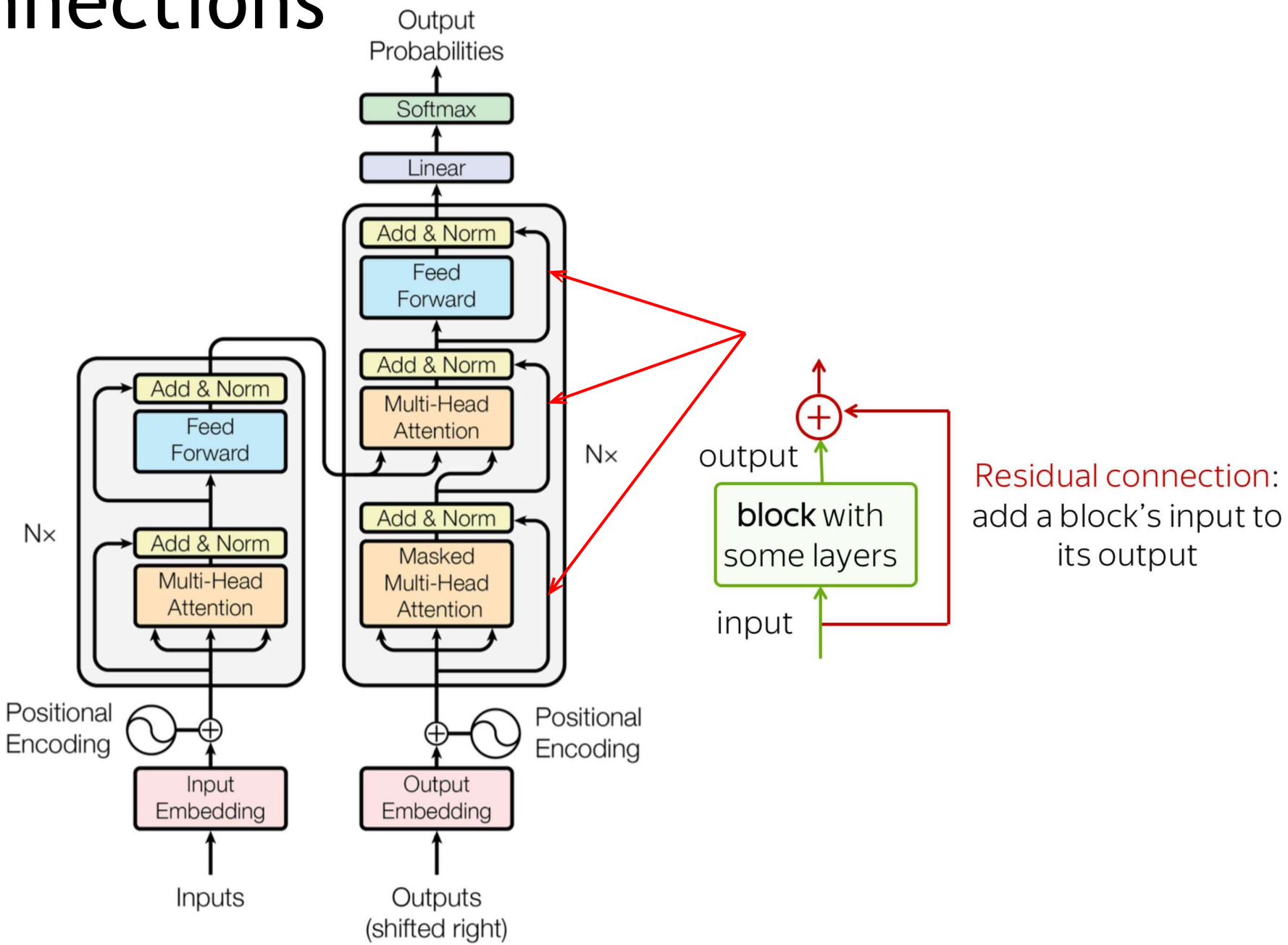
Feed Forward Block



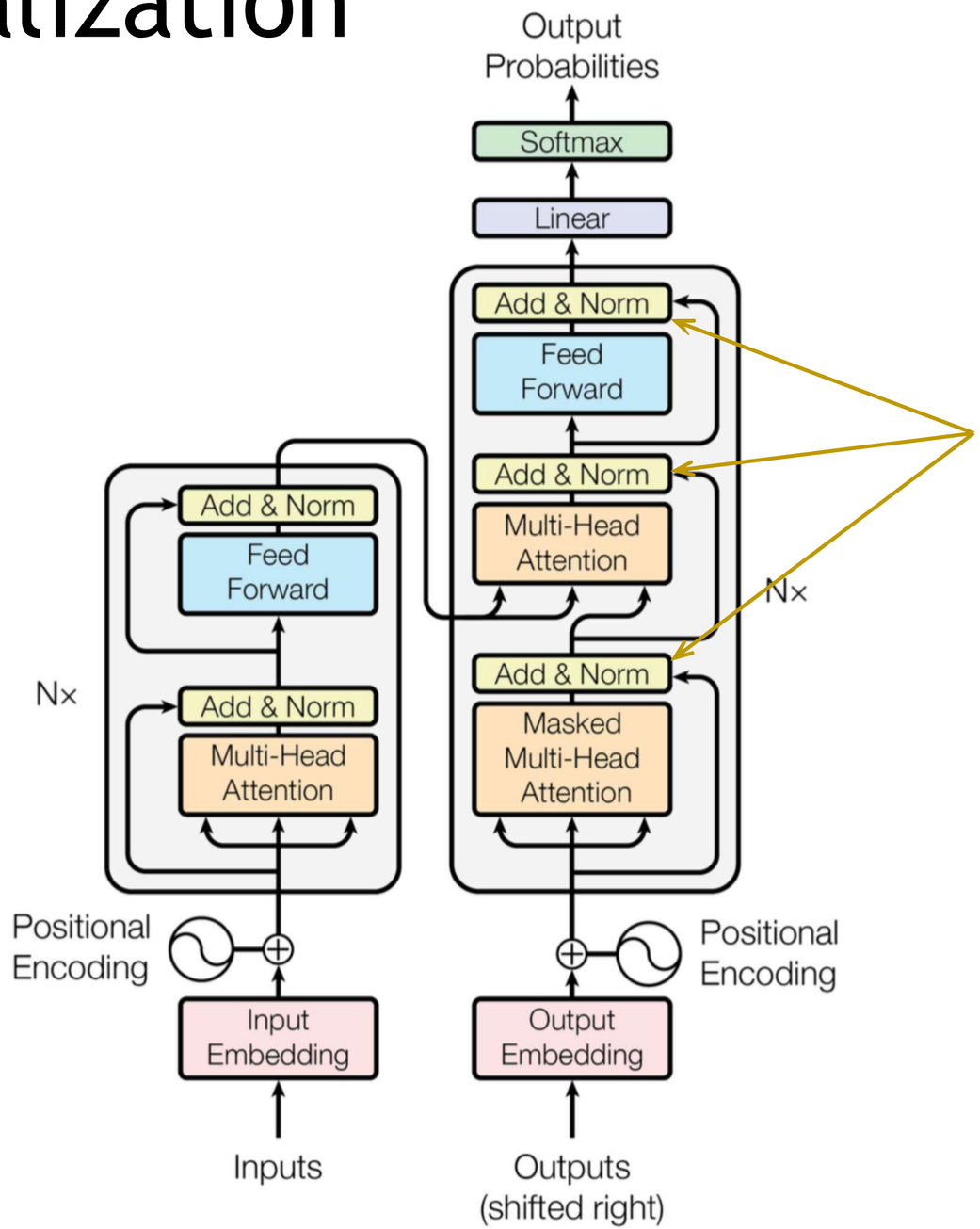


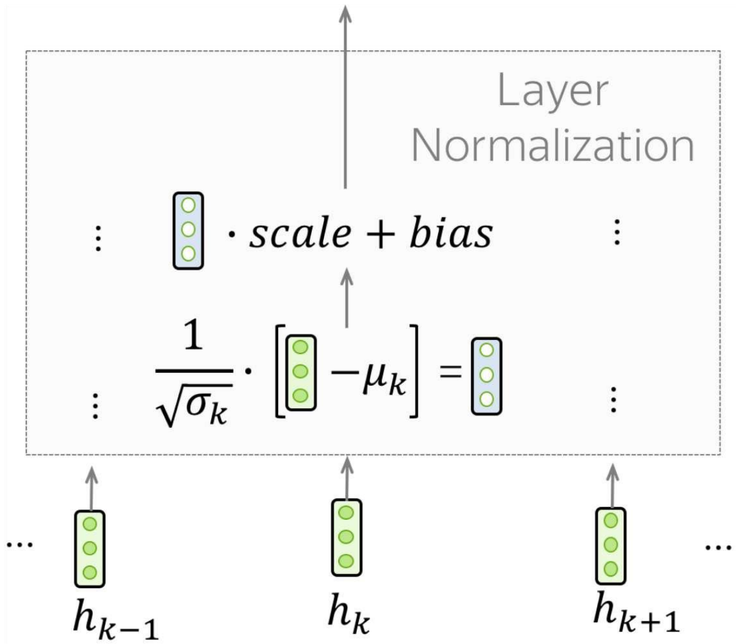
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Residual Connections

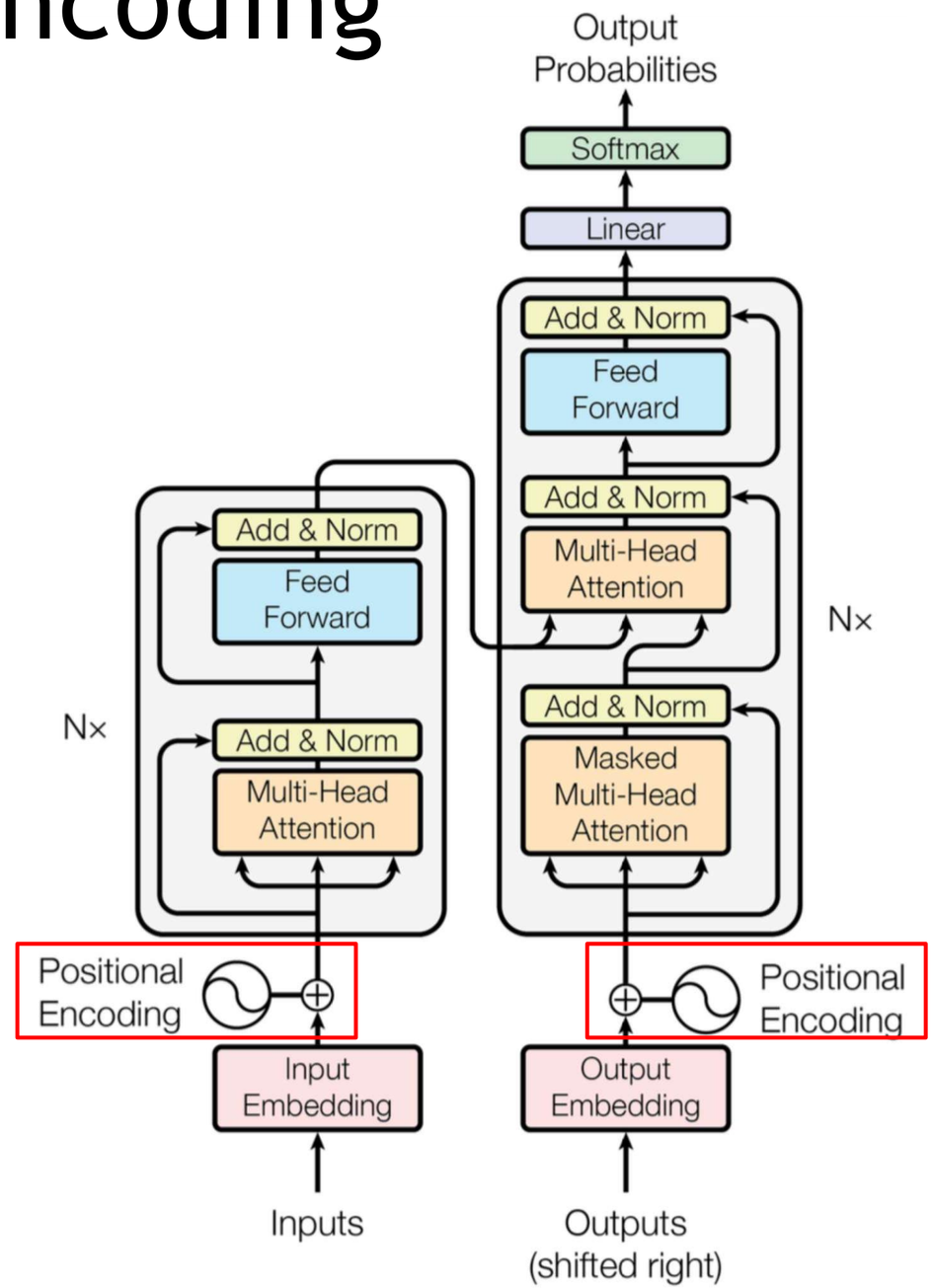


Layer Normalization

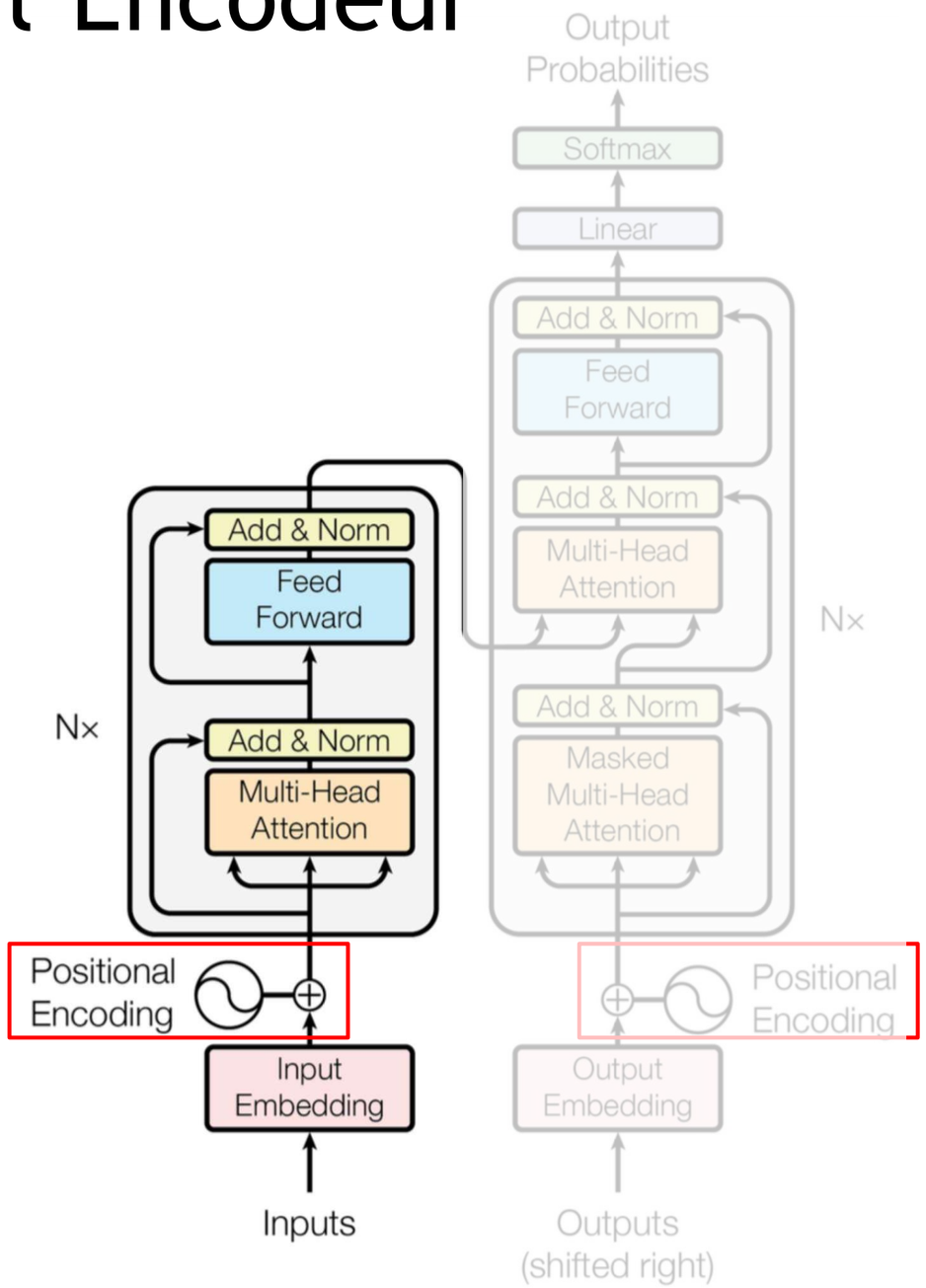




Positional Encoding

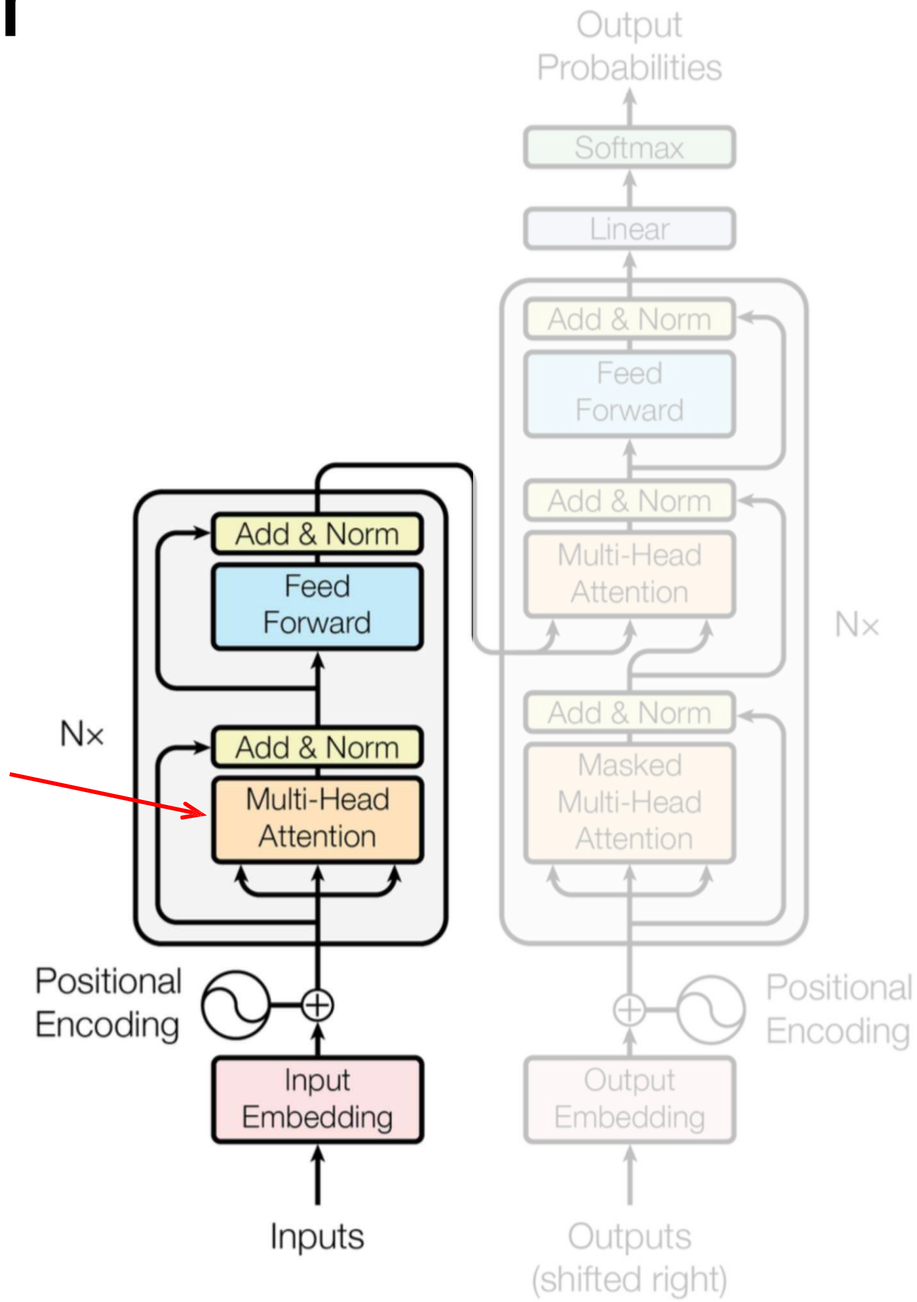


Comprendre l’Encodeur



Transformer

Encoder self-attention: tokens



Self-Attention: Pourquoi “Self”?

Decoder-encoder attention
recherche

- de: un état actuel du décodeur
- vers: Tous les états de l’encodeur

Self-Attention: Pourquoi “Self”?

Decoder-encoder attention recherche

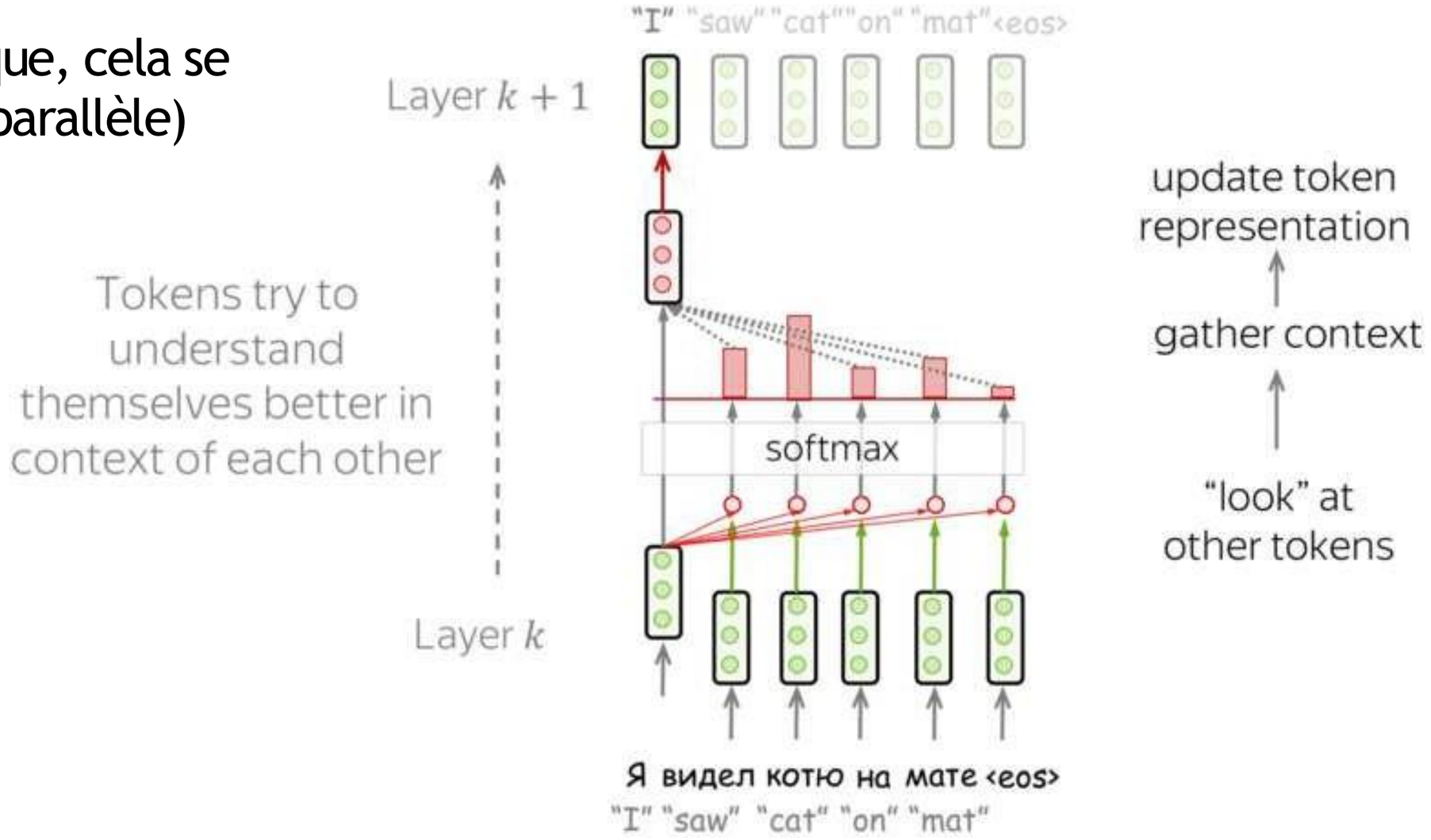
- de: un état actuel du décodeur
- vers: Tous les états du codeur

Self-attention recherche

- de: **chaque état** de chaque ensemble d’état
- vers: Tous les états du **même** ensemble d’état

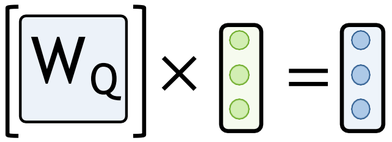
Self-Attention: “Look at Each Other”

(en pratique, cela se passe en parallèle)



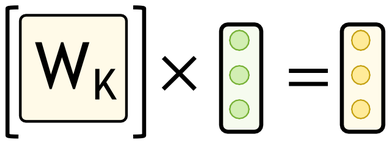
Query, Key, Value

Chaque vecteur reçoit trois représentations (« rôles »)



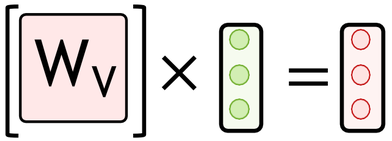
Query: vecteur à partir duquel l'attention regarde

« Salut, avez-vous cette information ? »



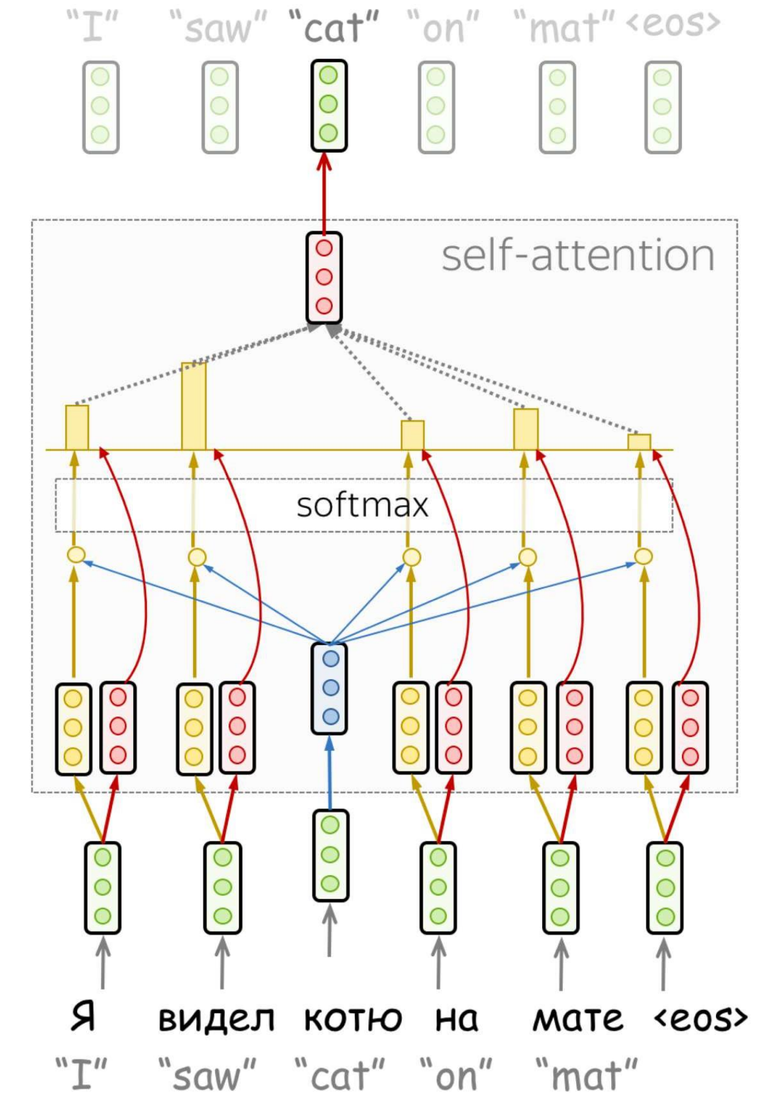
Key: vecteur sur lequel la requête cherche à calculer les poids

« Salut, j'ai cette information - donnez-moi un gros poids ! »



Value: Leur somme pondérée est la sortie d'attention

« Voici les informations que j'ai ! »

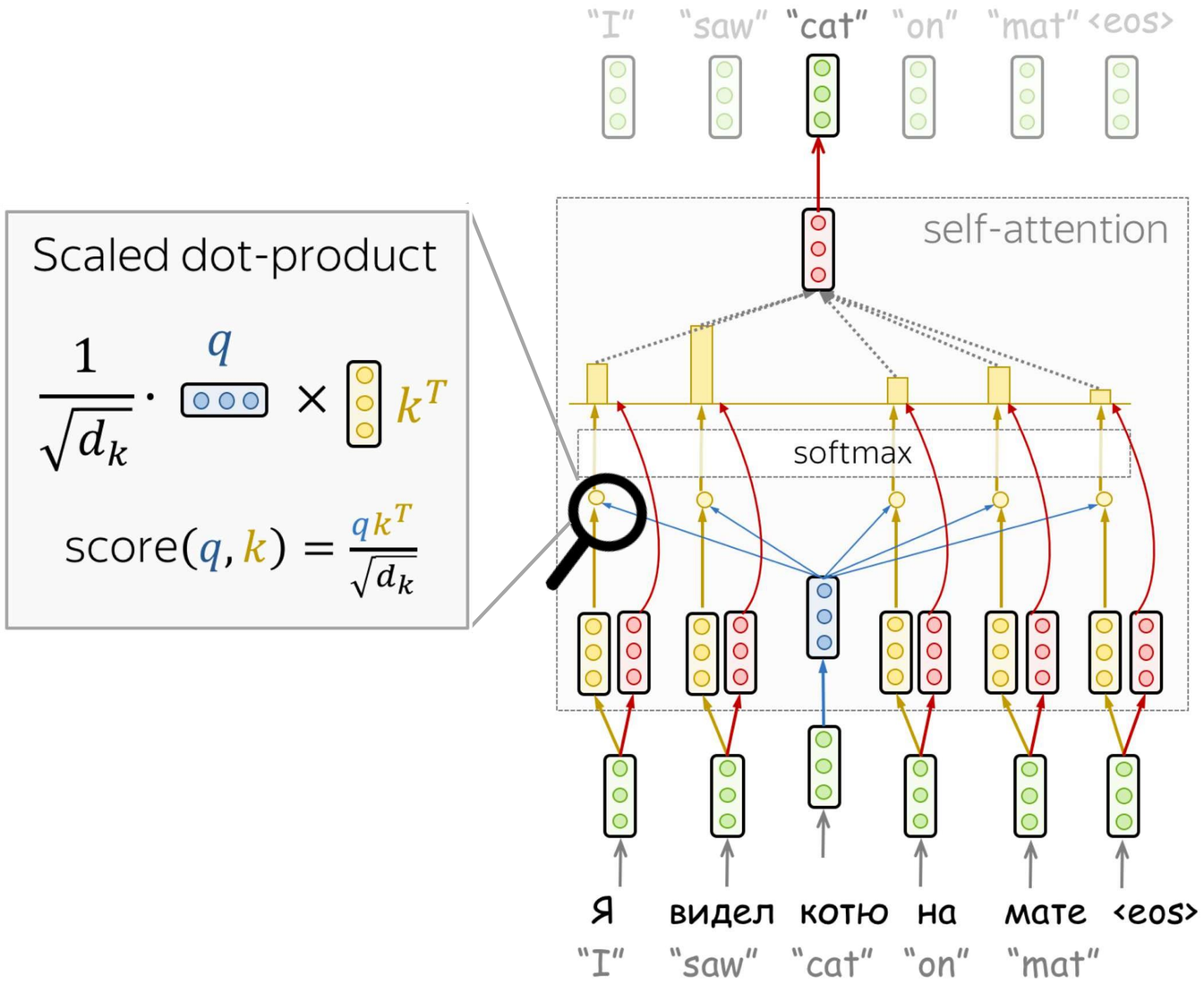


Query, Key, Value

Scaled dot-product

$$\frac{1}{\sqrt{d_k}} \cdot q \times k^T$$

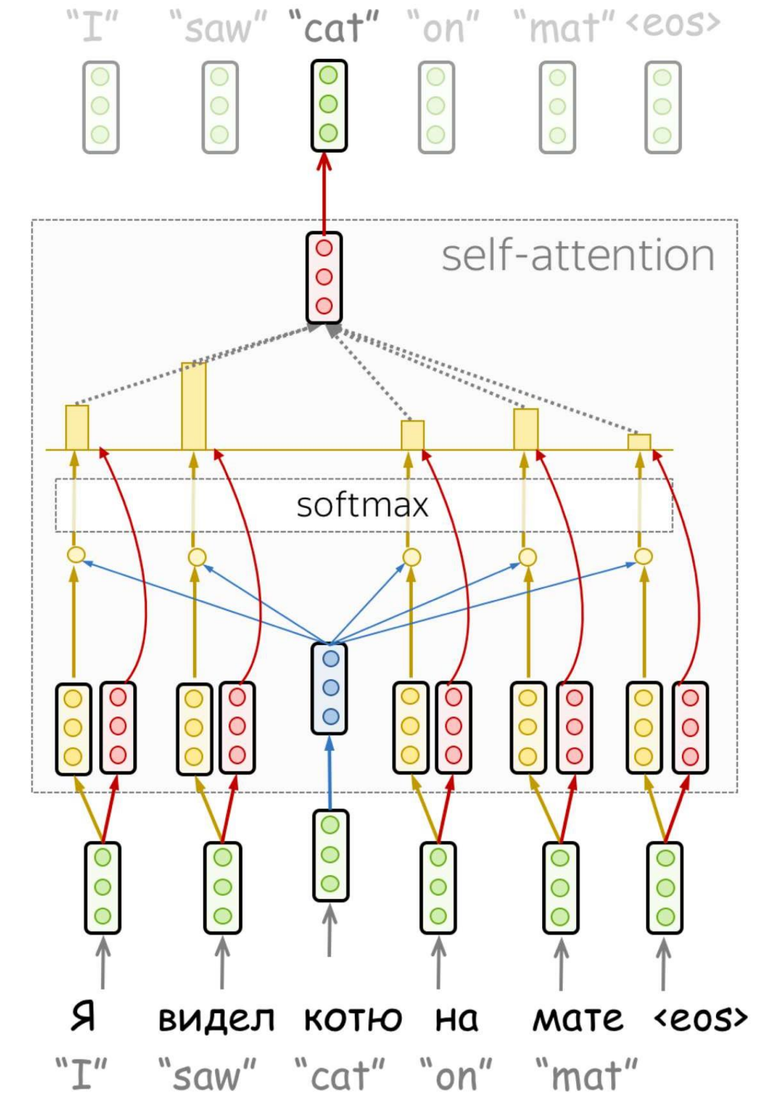
$$\text{score}(q, k) = \frac{qk^T}{\sqrt{d_k}}$$



Query, Key, Value

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

de vers



Query, Key, Value

$$\mathrm{X} \in \mathbb{R}^{n \times d} \qquad n = input\ length$$

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

de vers

$$q = XW^Q \qquad \mathrm{W^K} \in \mathbb{R}^{\mathrm{d} \times \mathrm{d_k}}$$

$$k = XW^K \qquad \mathrm{W^Q} \in \mathbb{R}^{\mathrm{d} \times d_k}$$

$$v = XW^K \qquad \mathrm{W^K} \in \mathbb{R}^{n \times d_v}$$

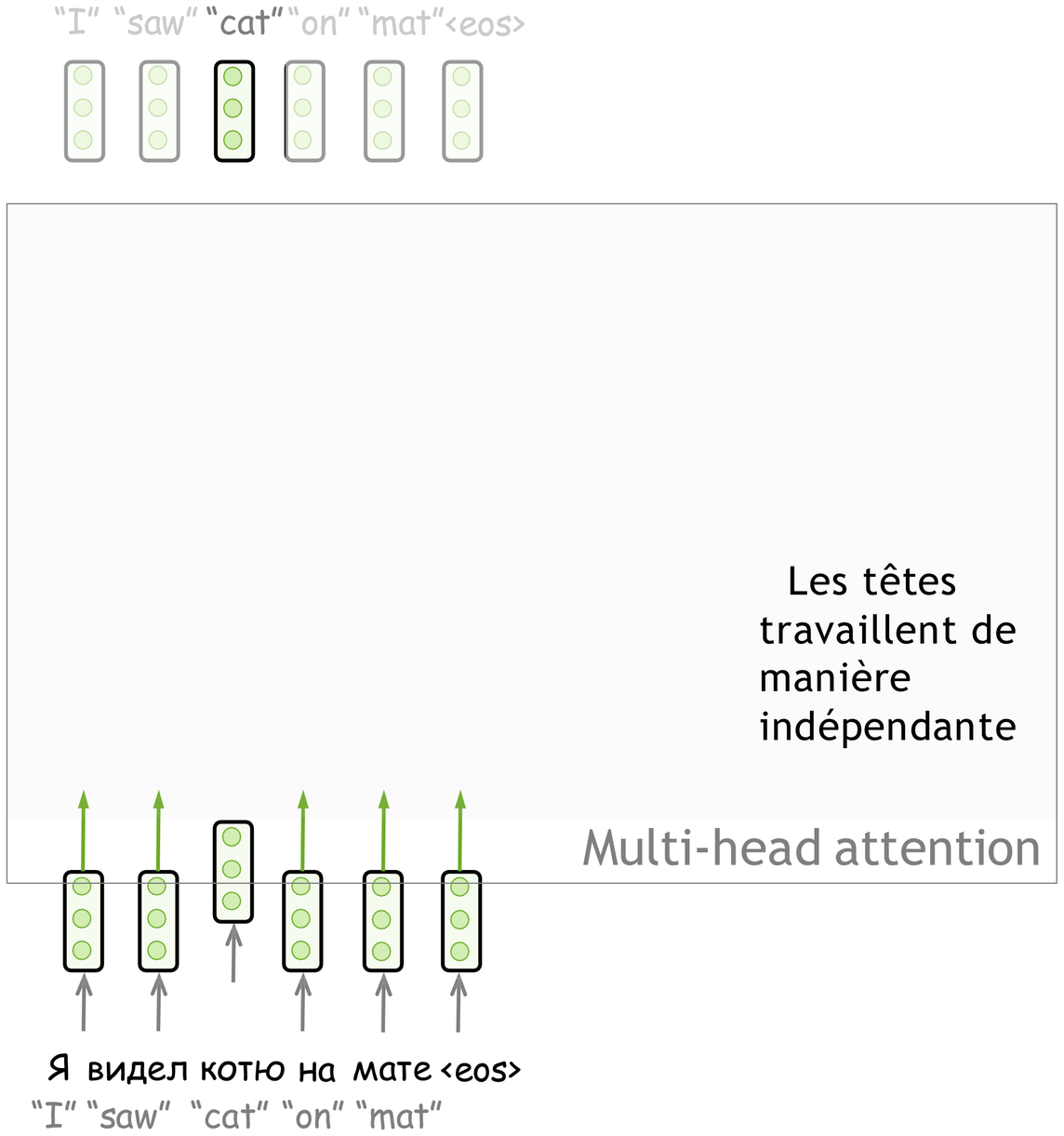
$$Attention(q, k, v) \in \mathbb{R}^{n \times d_v}$$

Attention Multi-Head

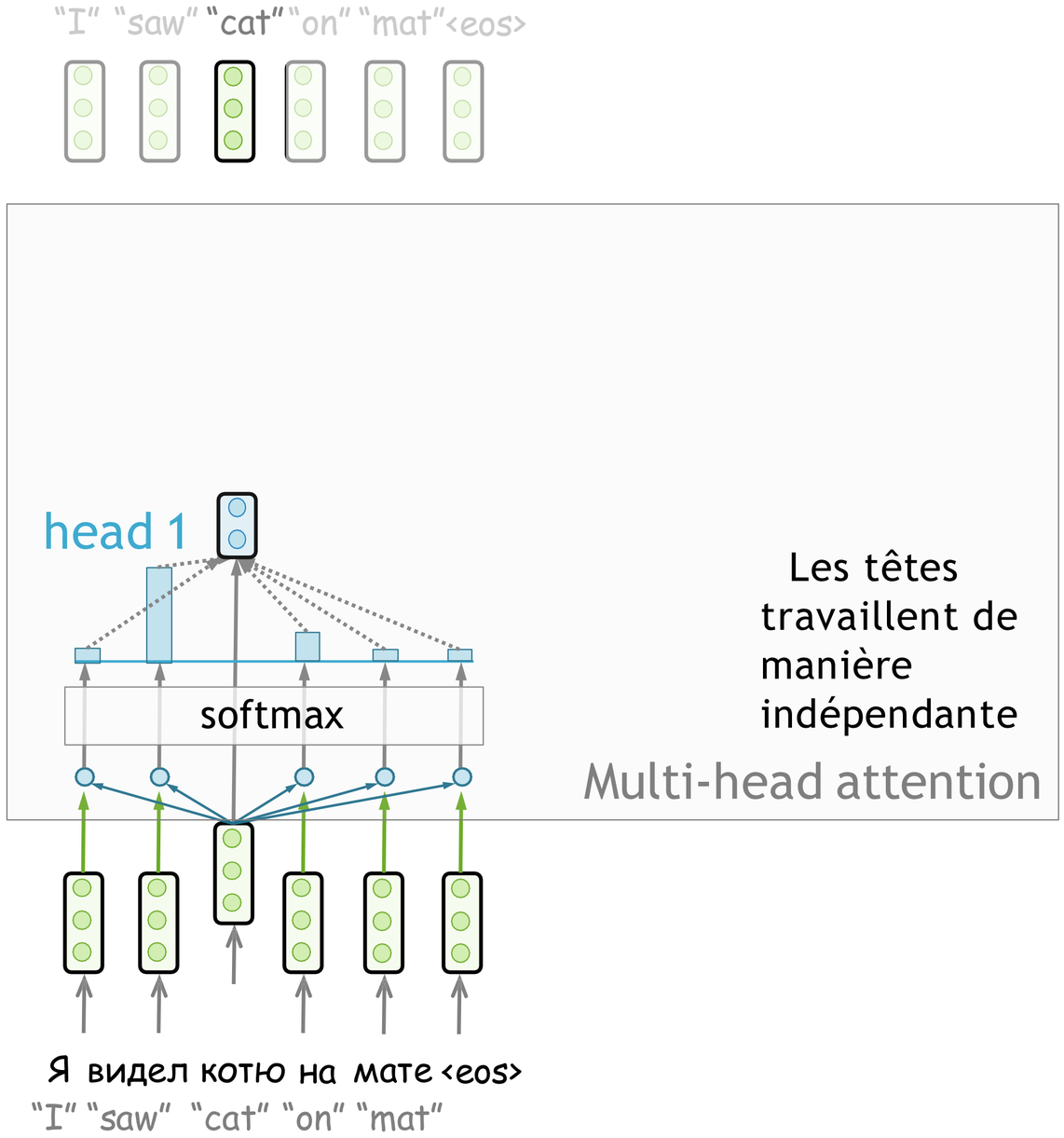
Nous devons suivre beaucoup de choses différentes à la fois !

- Gender agreement
- Case government
- Lexical preferences
- ...

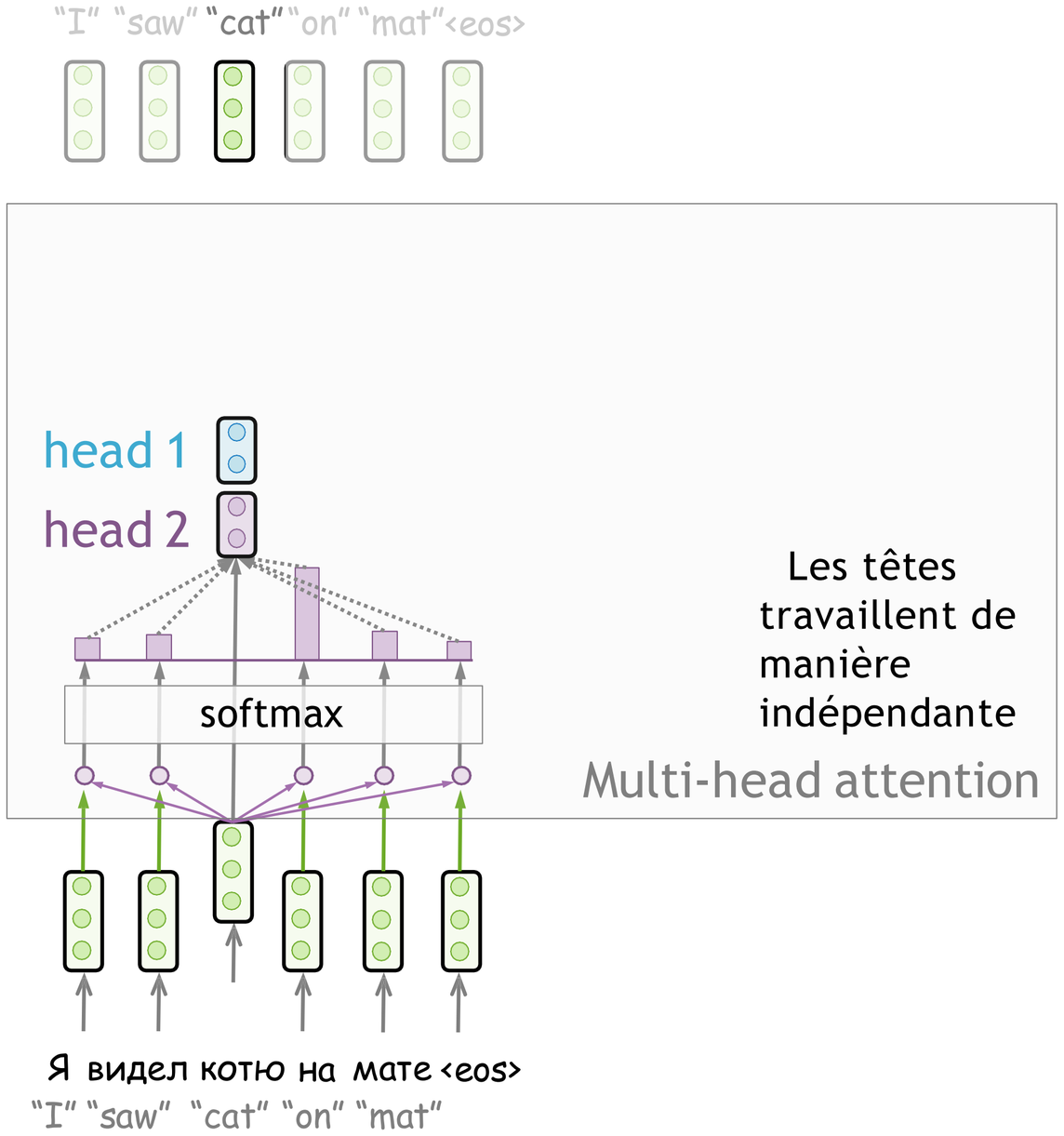
Attention Multi-Head



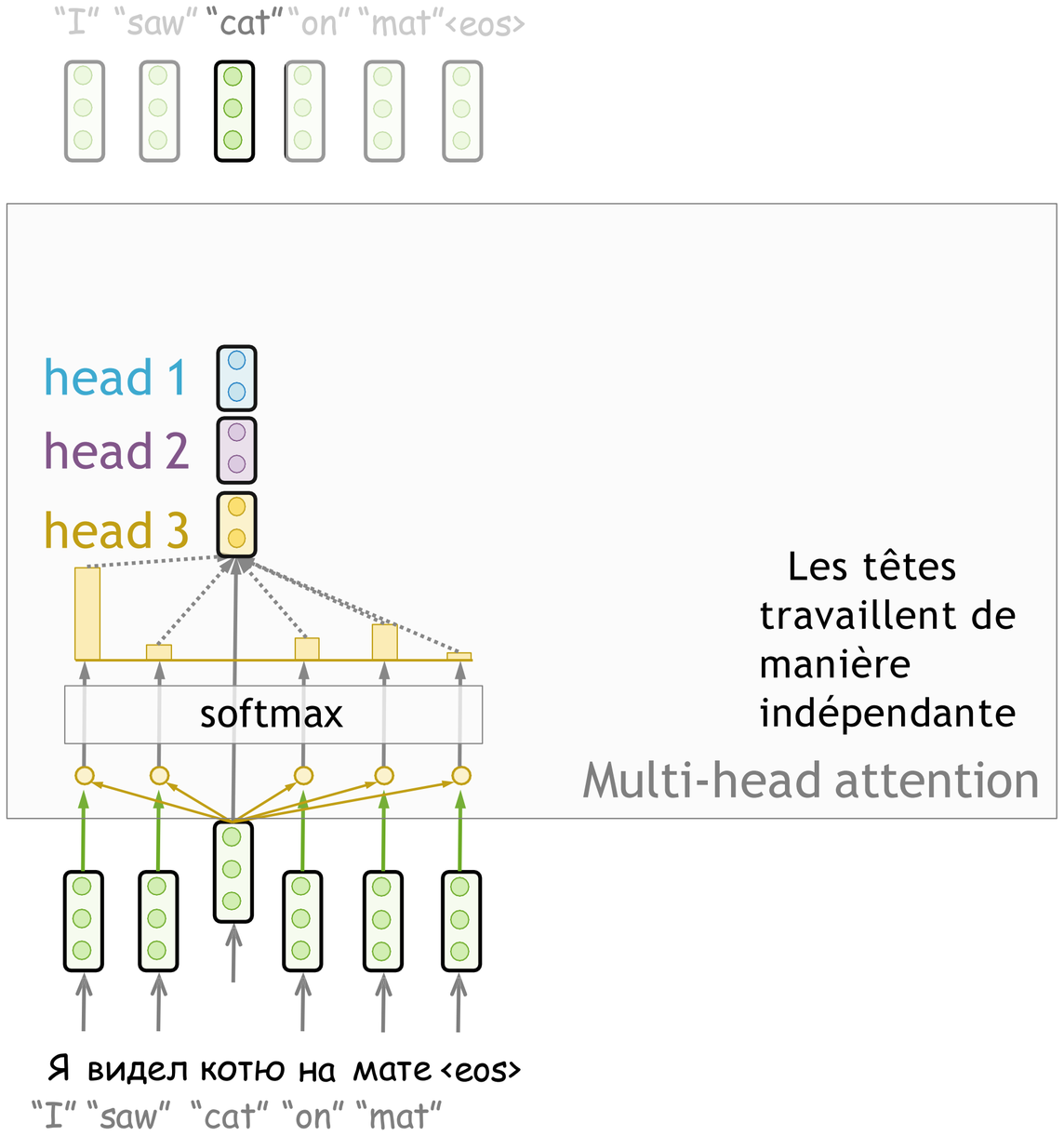
Attention Multi-Head



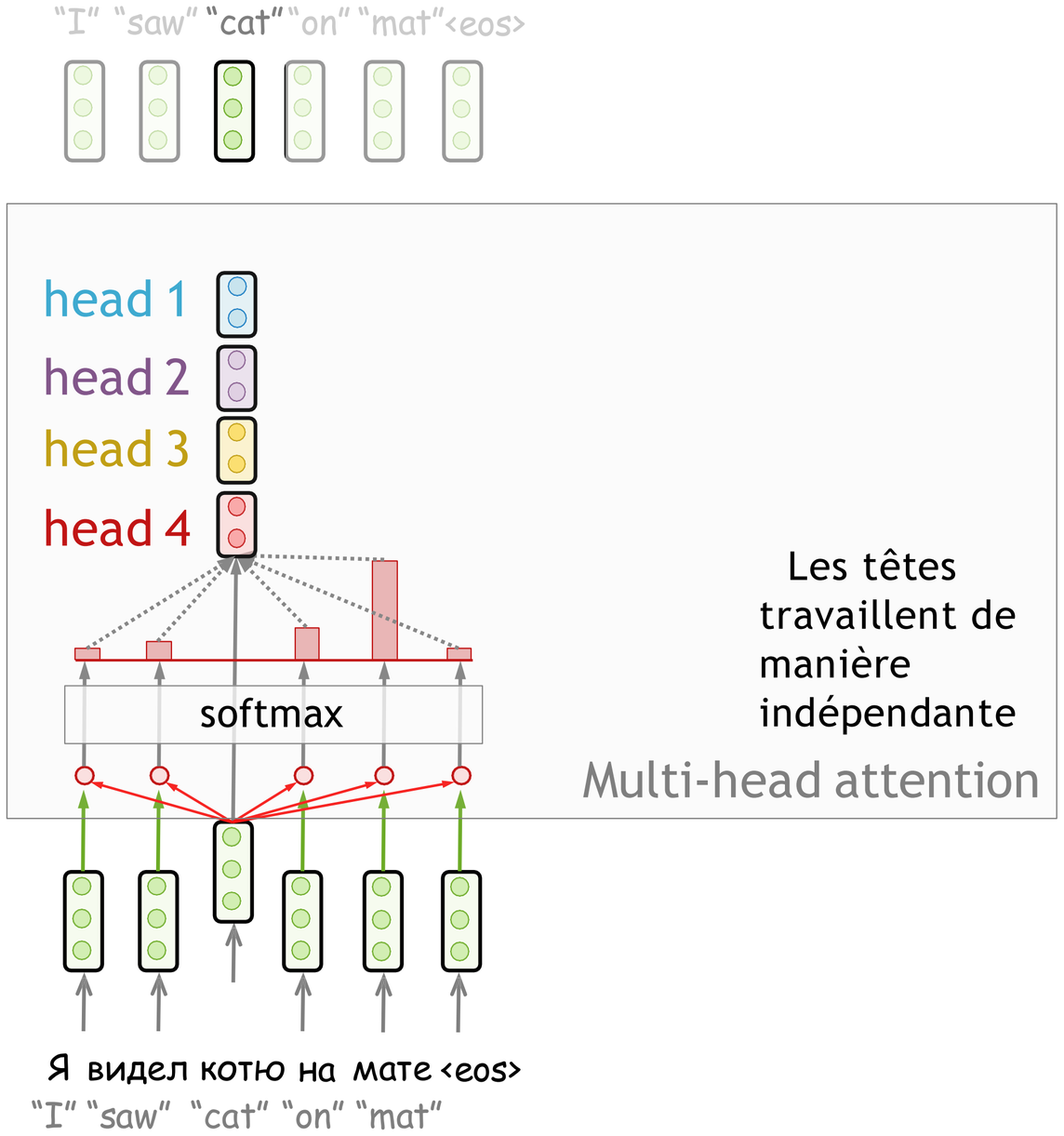
Attention Multi-Head



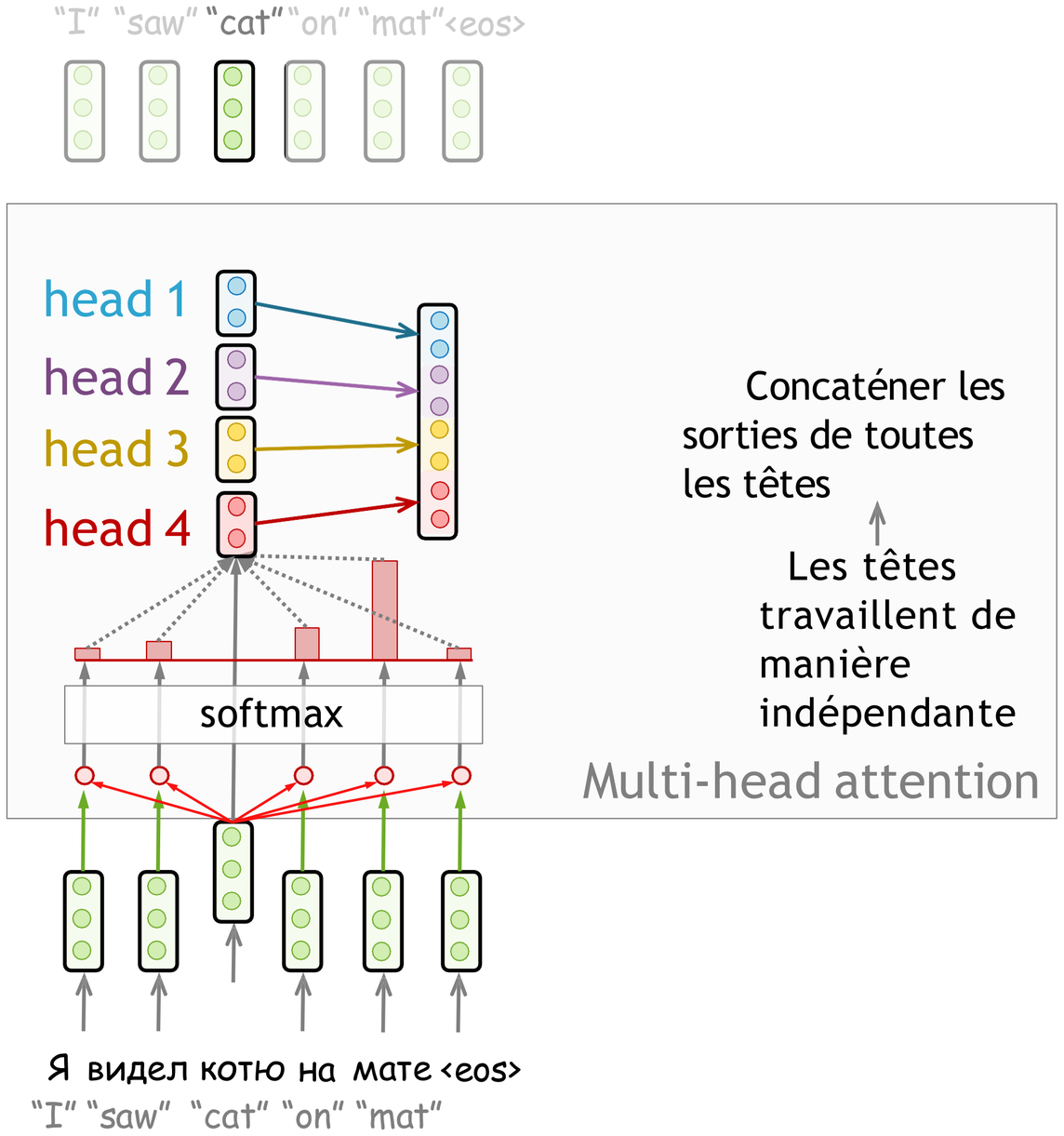
Attention Multi-Head



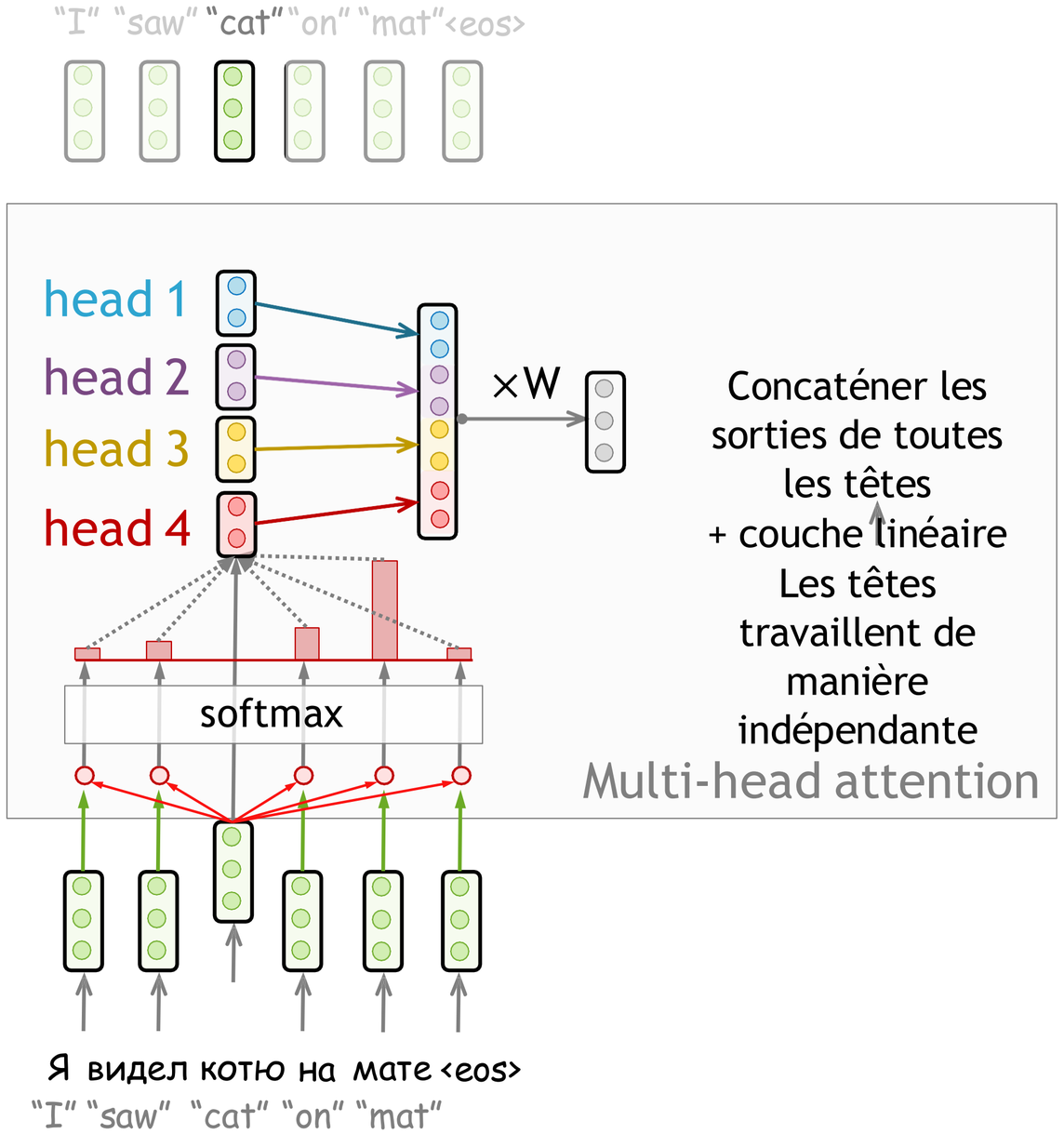
Attention Multi-Head



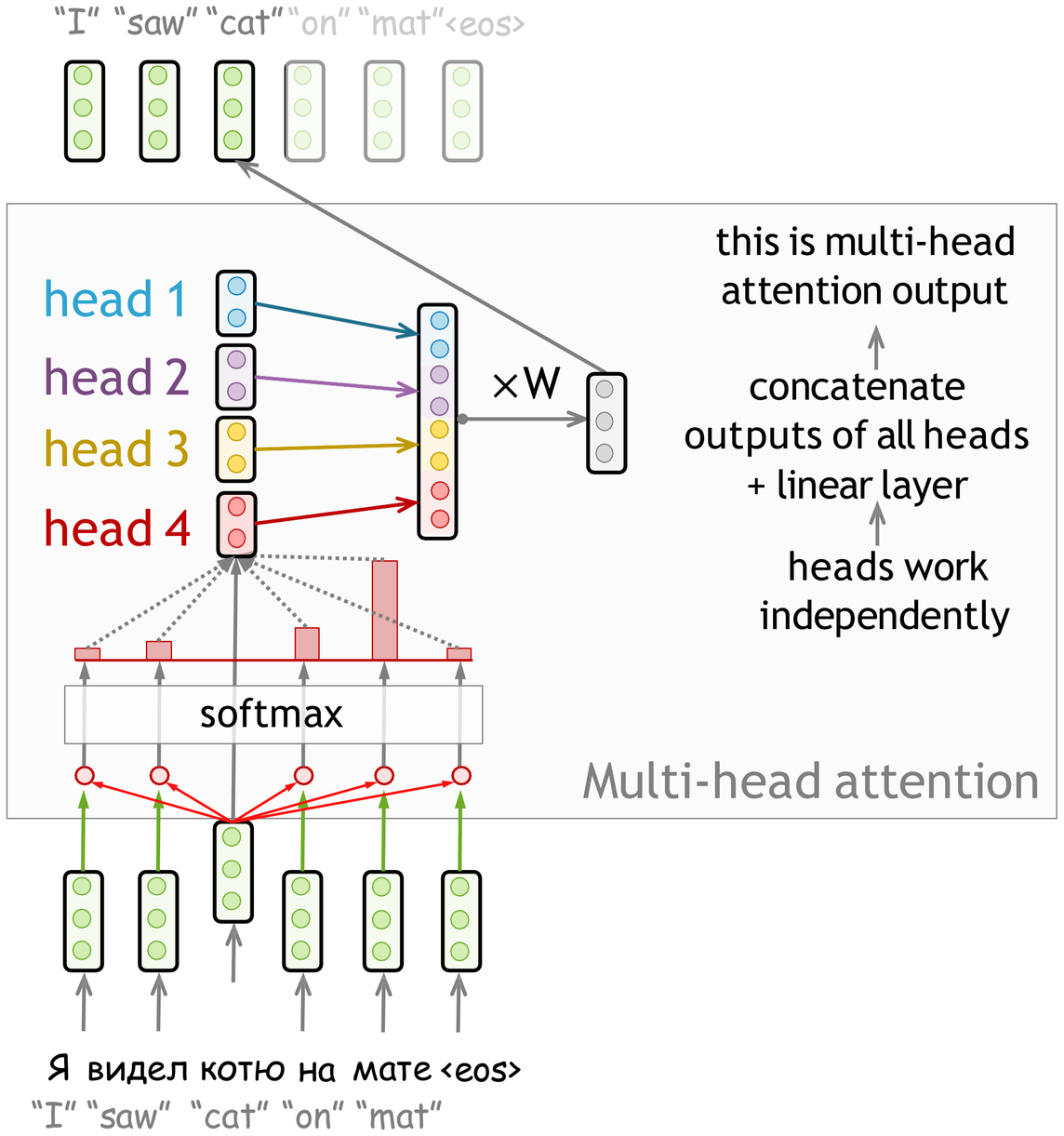
Attention Multi-Head



Attention Multi-Head



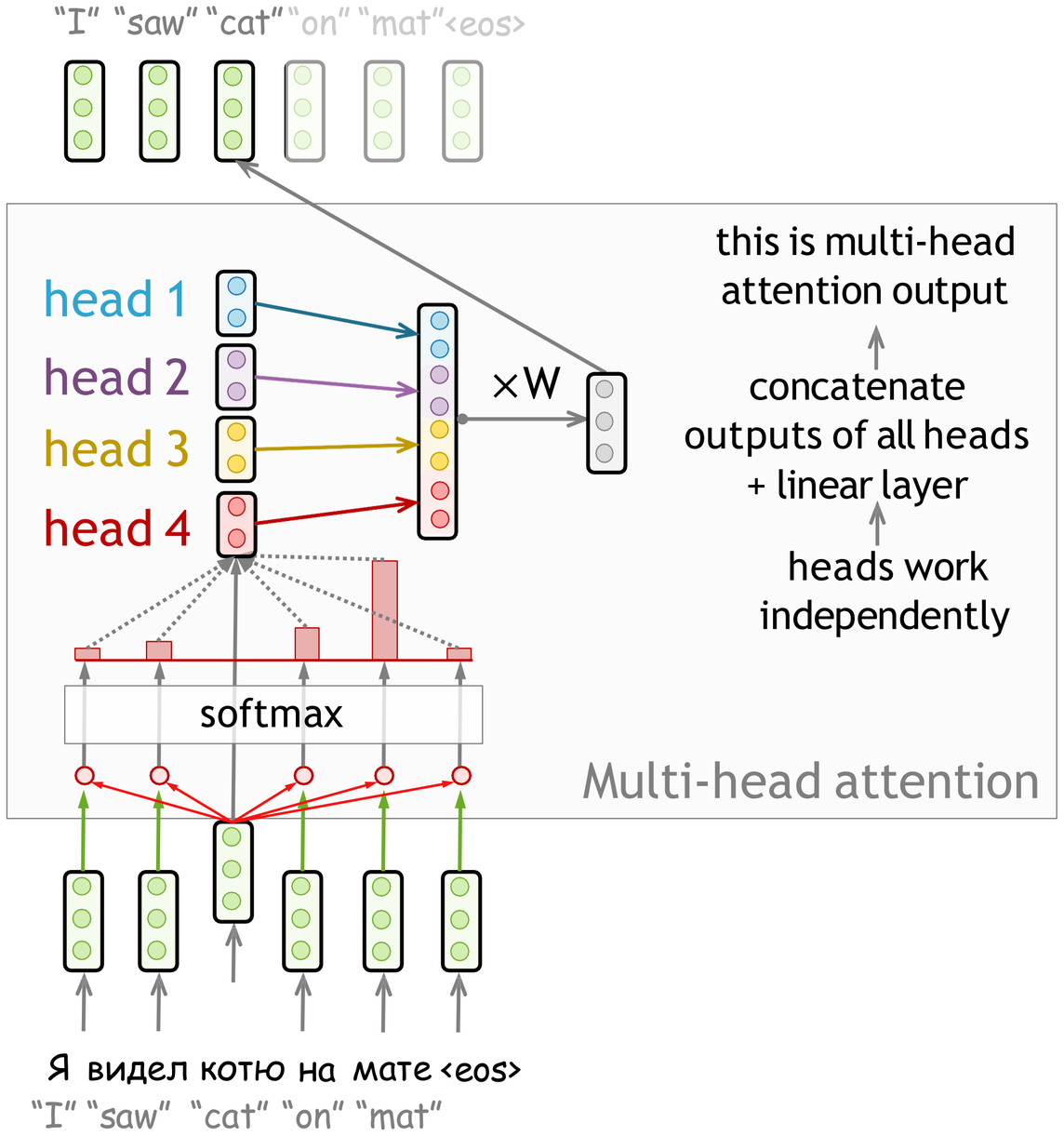
Attention Multi-Head



Attention Multi-Head

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_o,$$

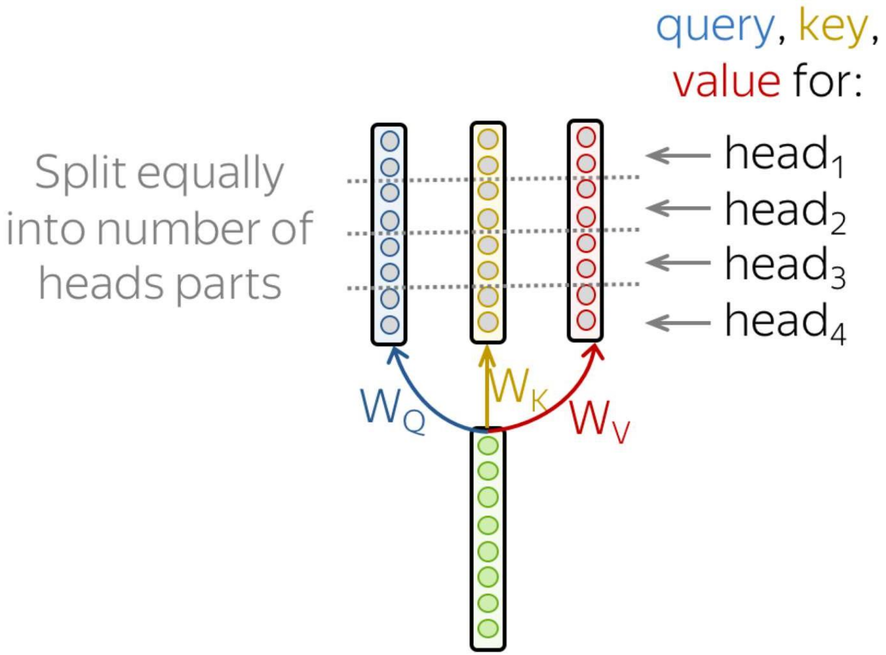
$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

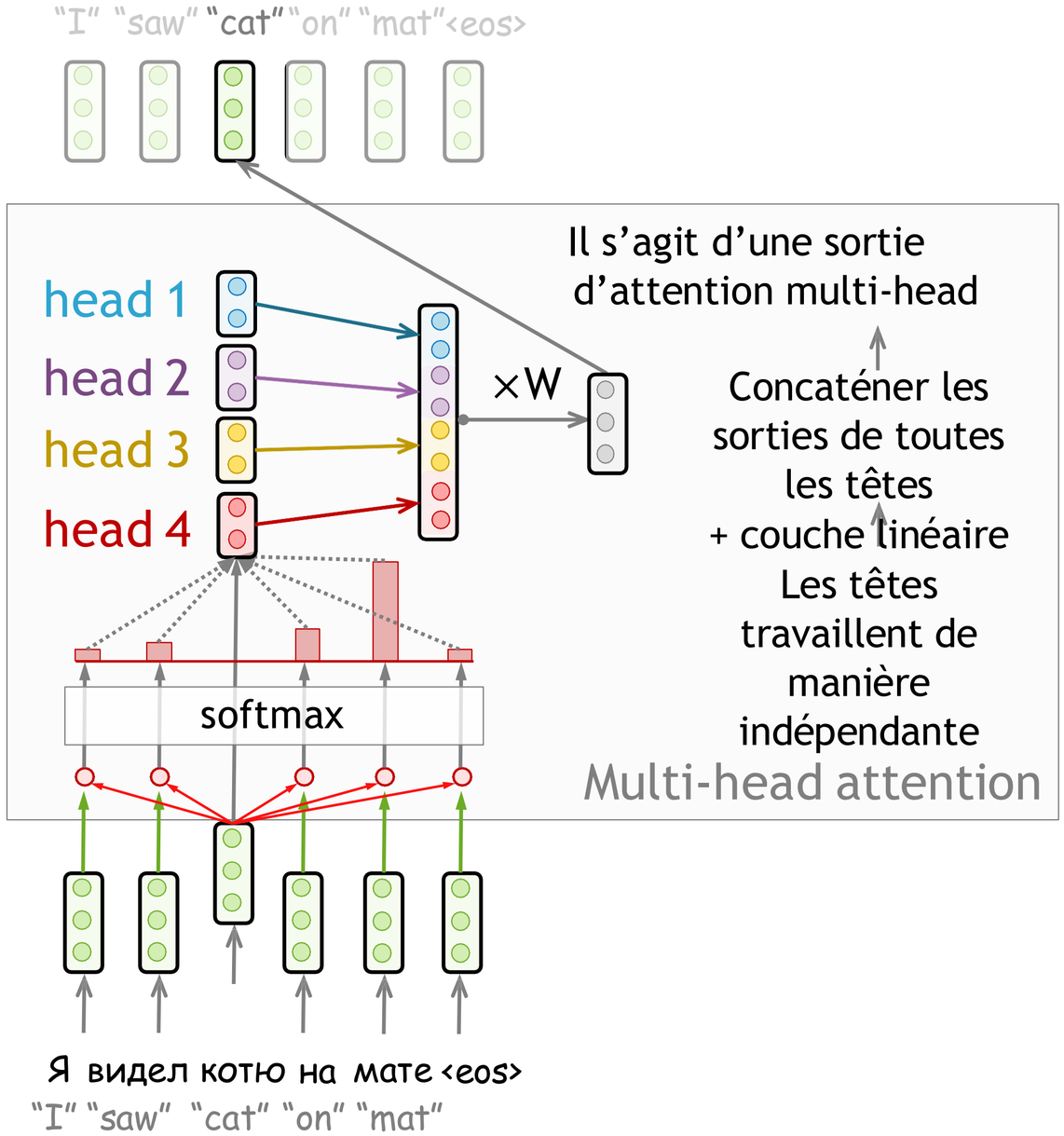


Attention Multi-Head

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_o,$$

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

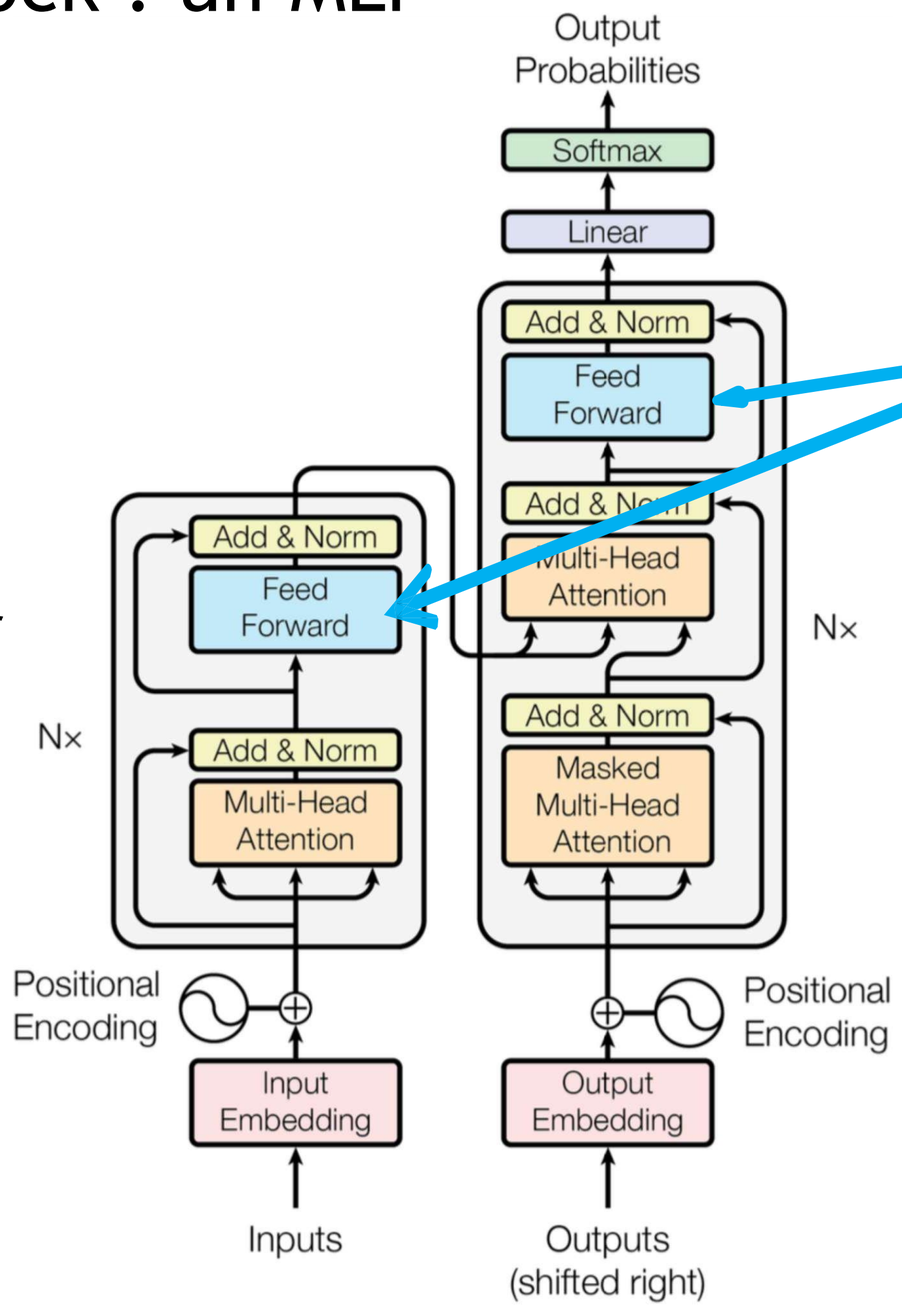


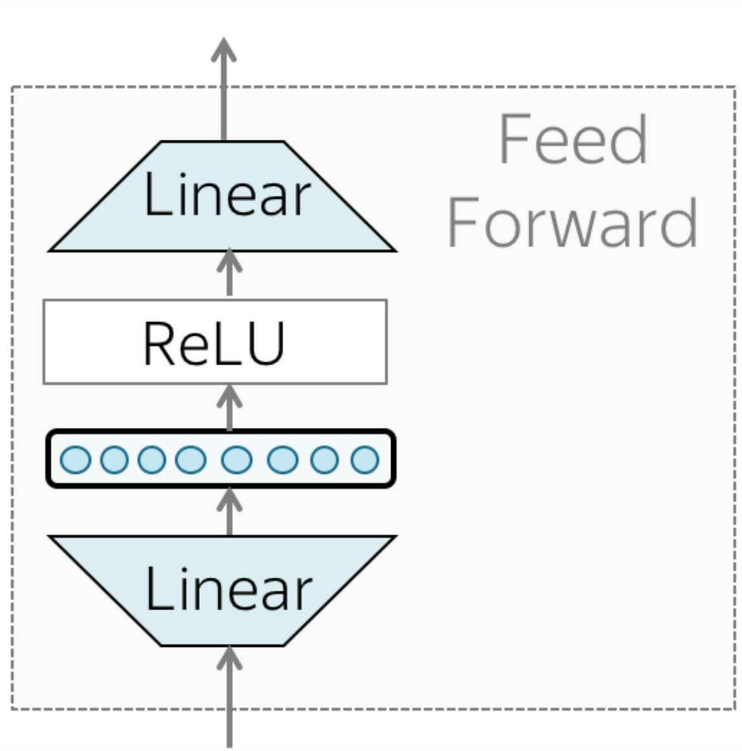


Feed Forward Block : un MLP

Feed-forward network:

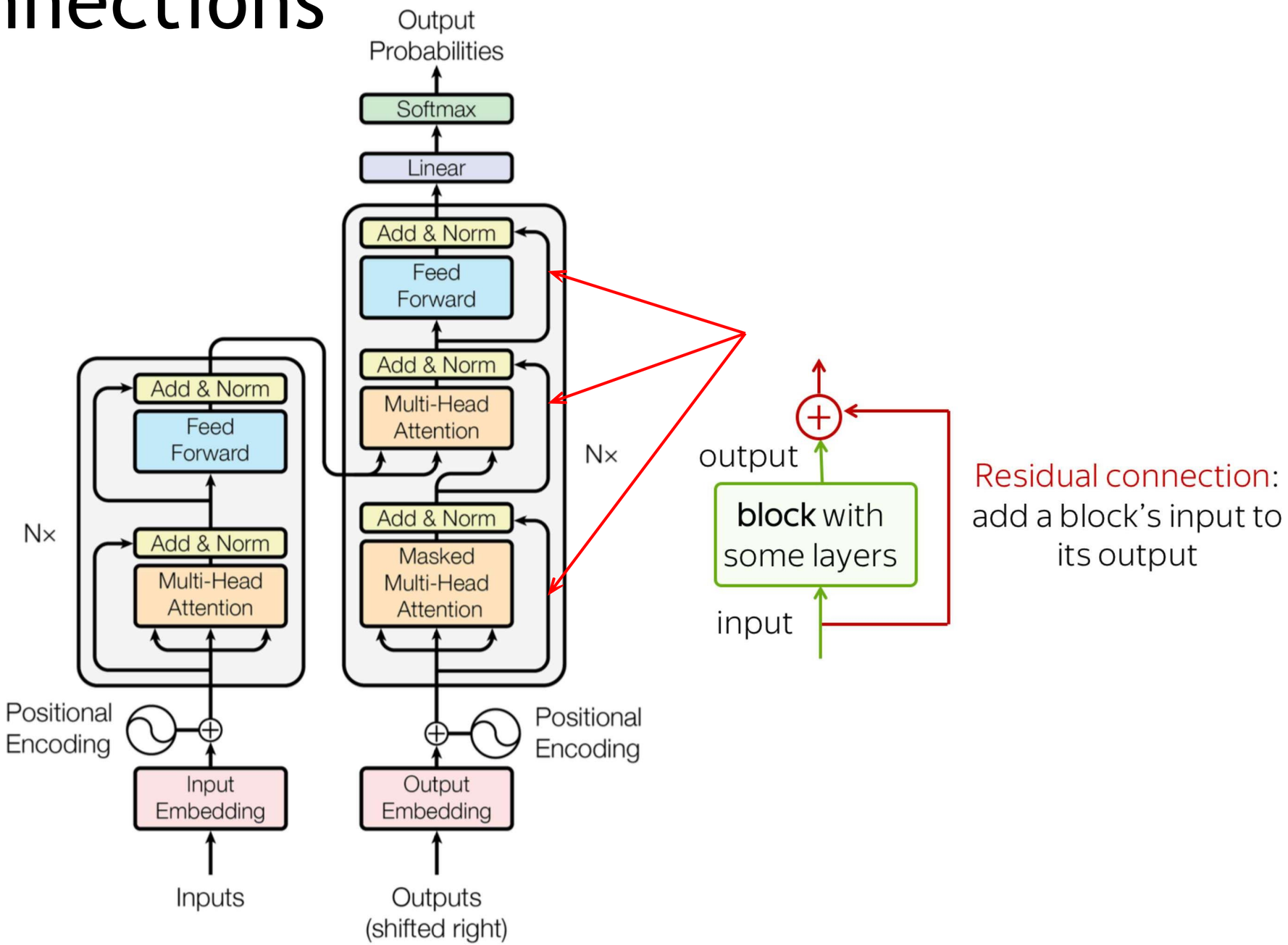
Après avoir pris des informations à partir d'autres jetons, prends un moment pour penser et traiter ces informations



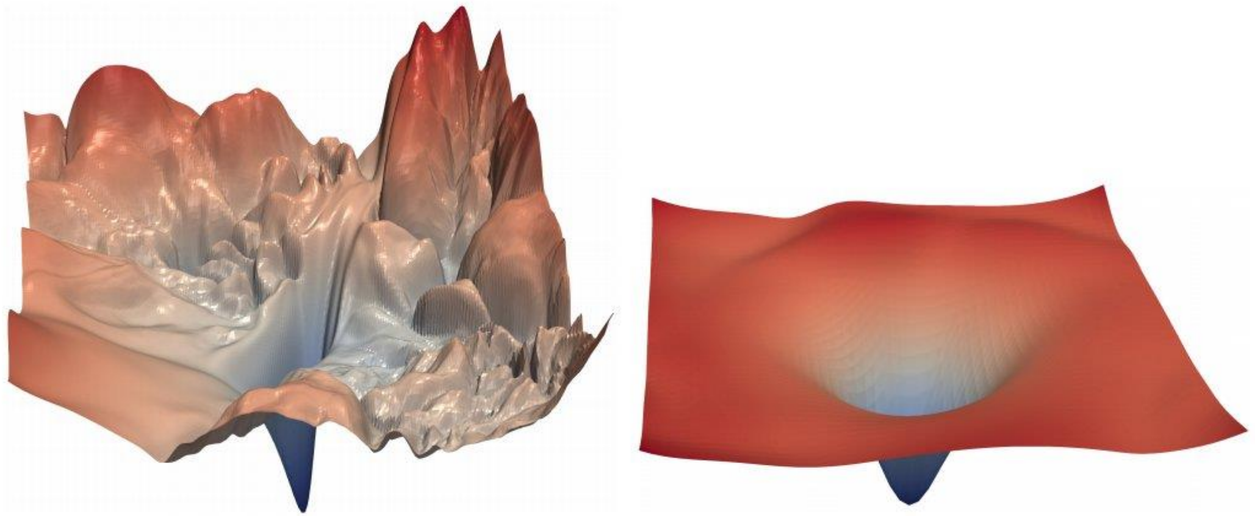


$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Residual Connections



Residual Connections

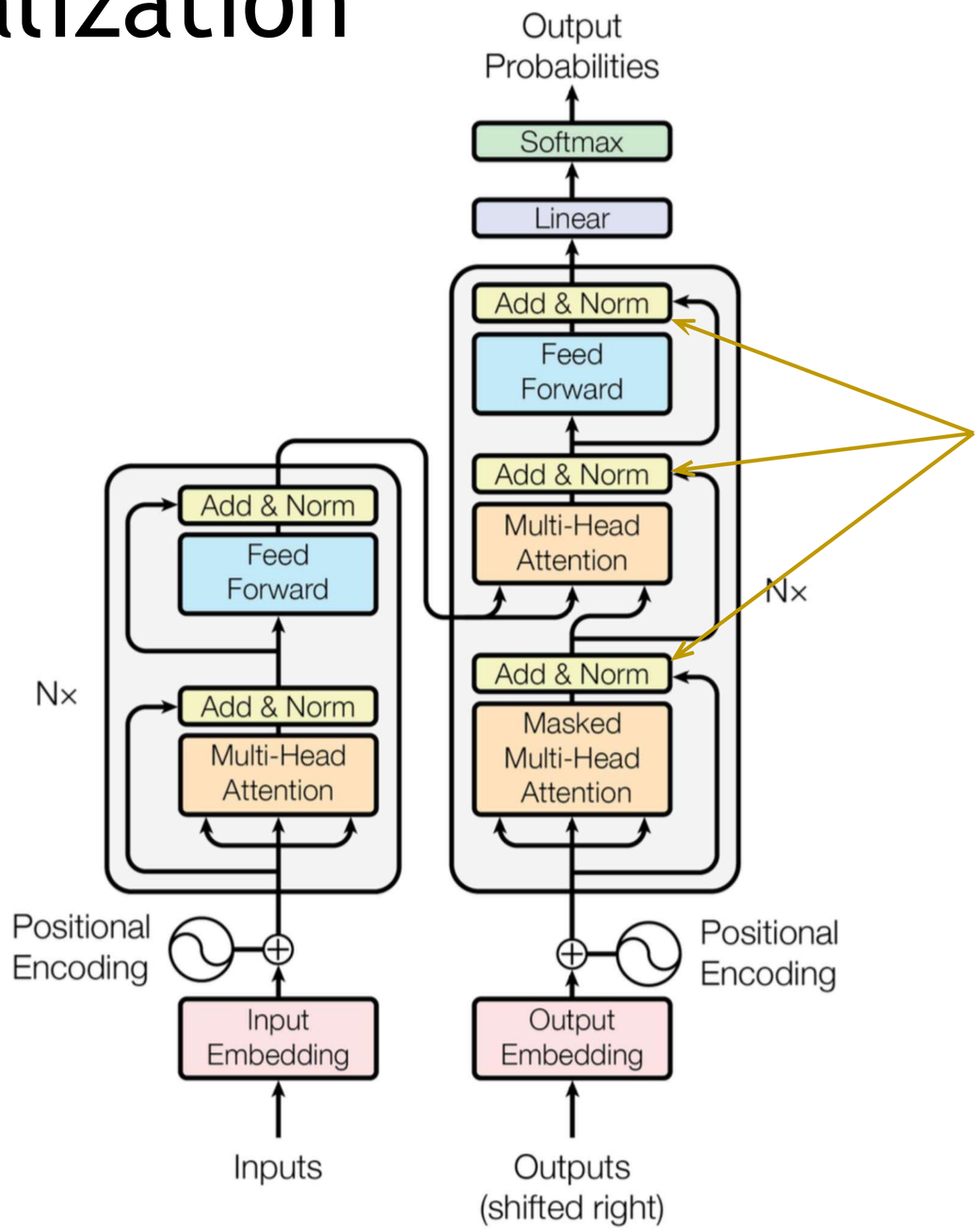


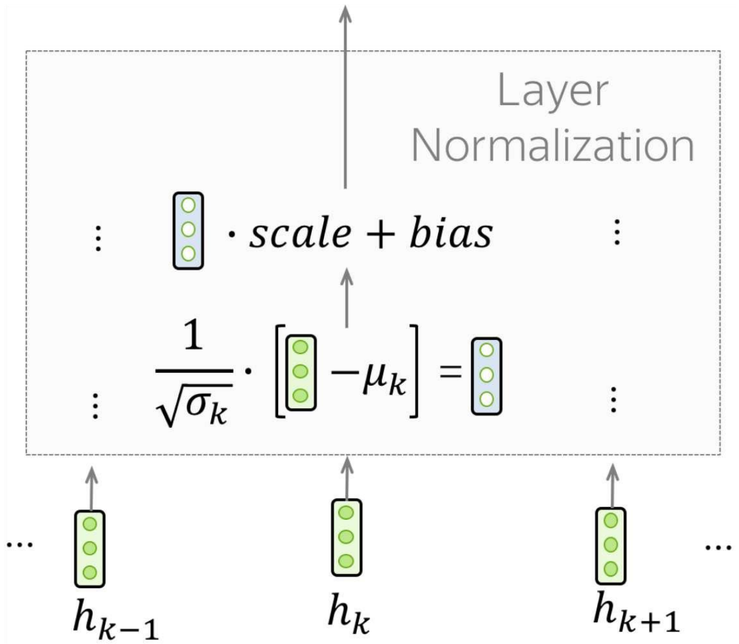
(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The vertical axis is logarithmic to show dynamic range. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

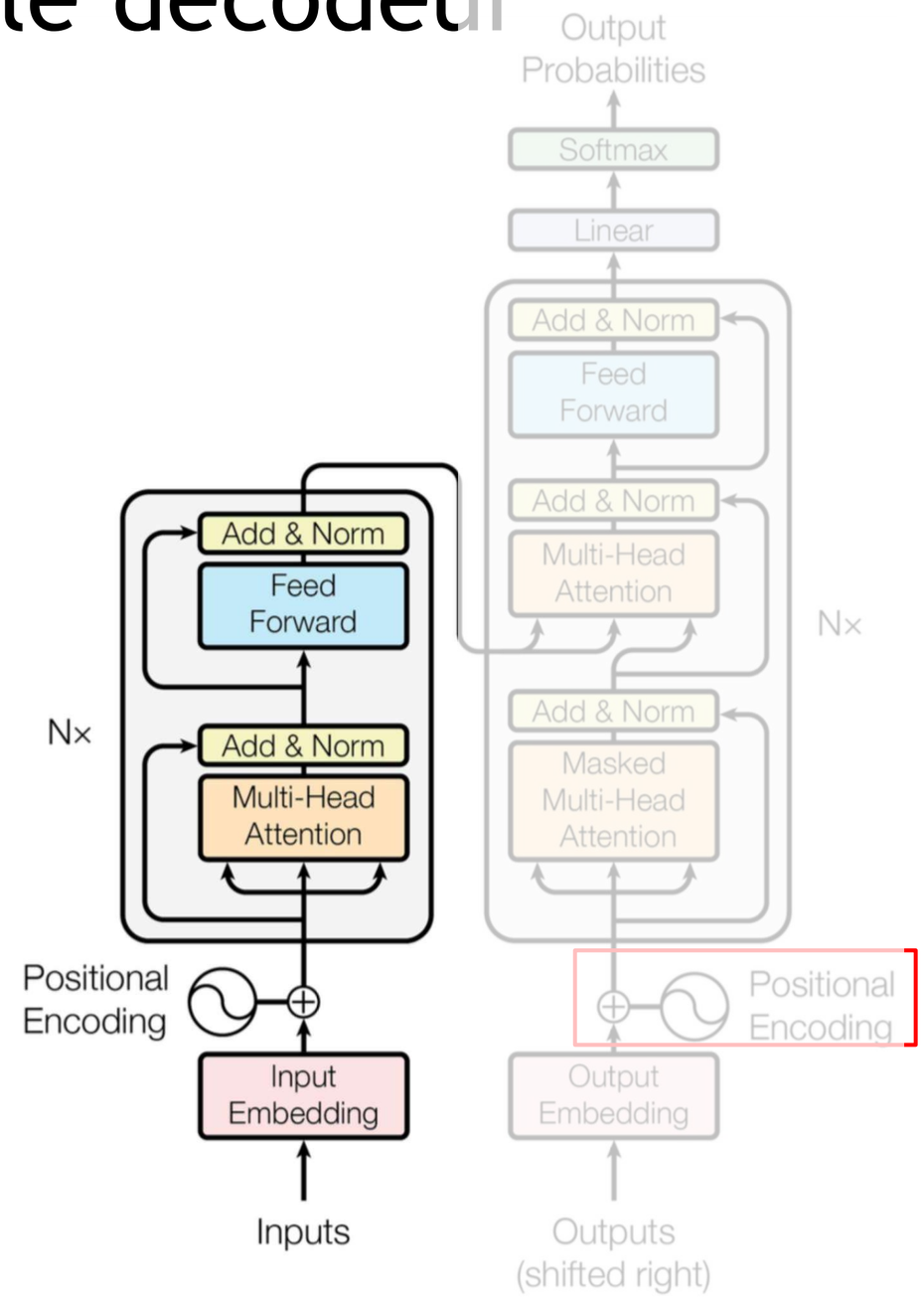
Entrainer des réseaux très profonds

Layer Normalization

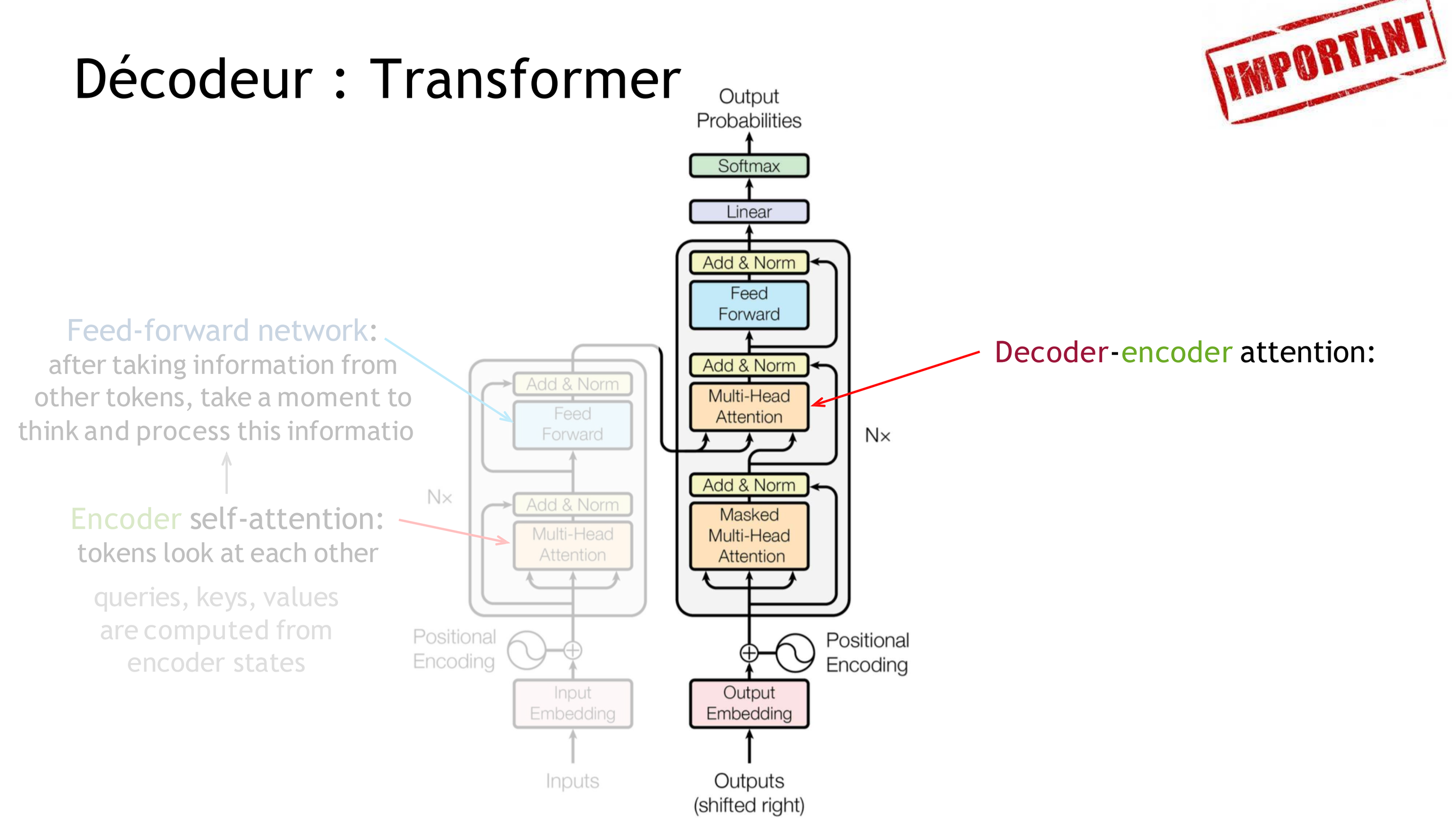




Comprendre le decodeur



Décodeur : Transformer



Feed-forward network:
after taking information from other tokens, take a moment to think and process this informatio

Encoder self-attention:
tokens look at each other

queries, keys, values are computed from encoder states

Decoder-encoder attention:

Cross Attention : Query, Key, Value

$$X \in \mathbb{R}^{n \times d} \qquad n = input\ length$$

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

décodeur — encodeur

$$q = XW^Q \qquad W^K \in \mathbb{R}^{d \times d_k}$$

$$k = XW^K \qquad W^Q \in \mathbb{R}^{d \times d_k}$$

$$v = XW^K \qquad W^K \in \mathbb{R}^{n \times d_v}$$

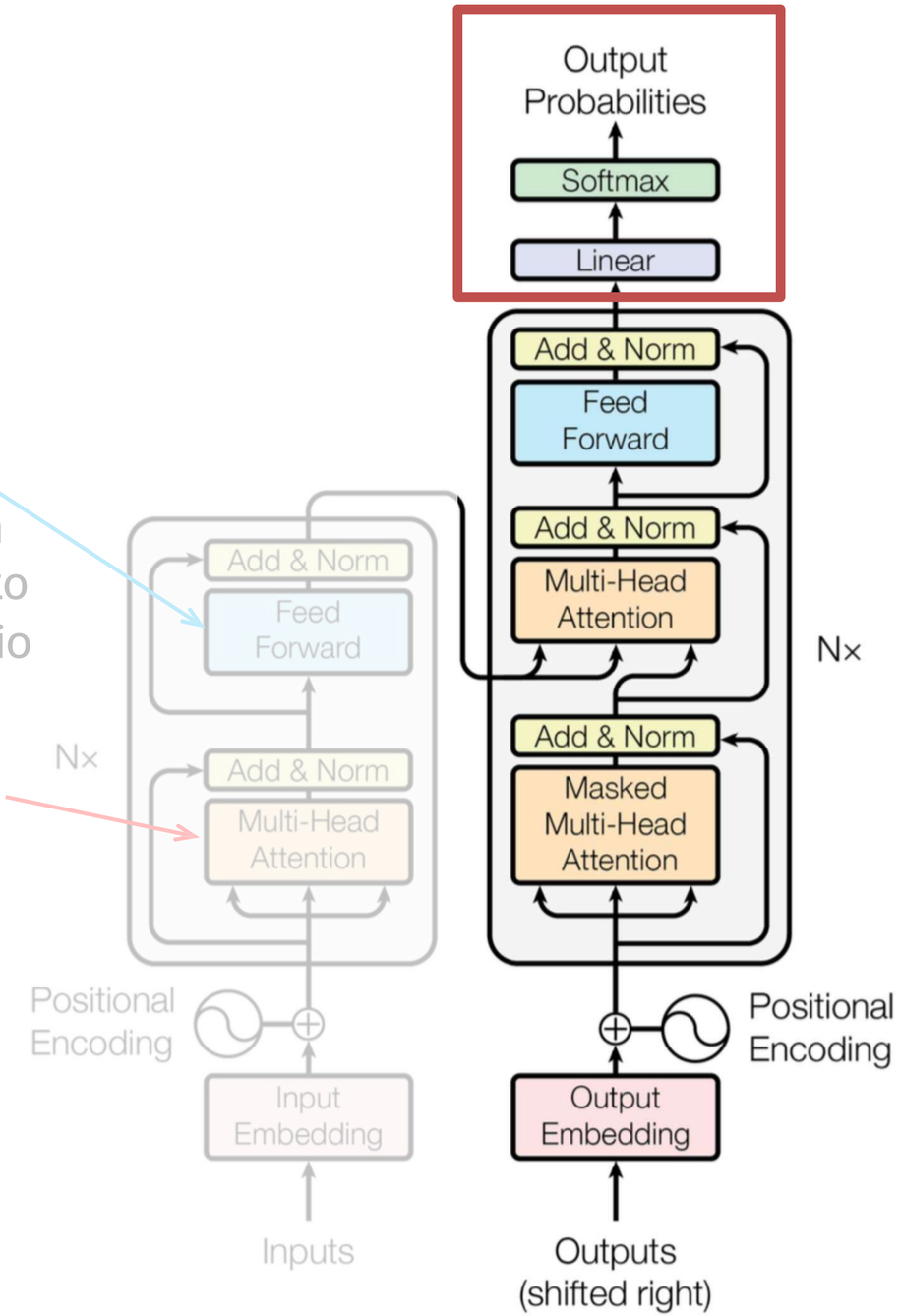
$$Attention(q, k, v) \in \mathbb{R}^{n \times d_v}$$

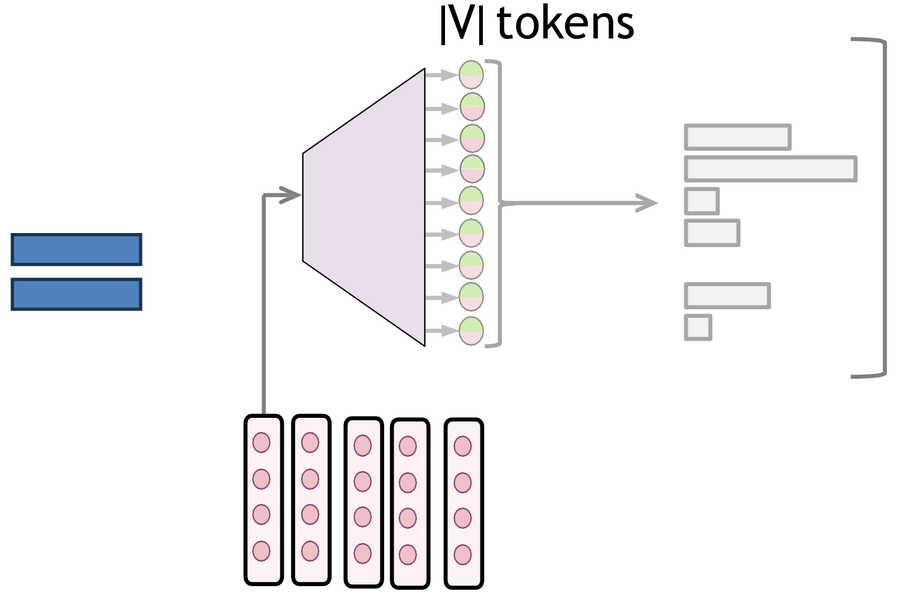
Prédiction

Feed-forward network:
after taking information from other tokens, take a moment to think and process this informatio

Encoder self-attention:
tokens look at each other

queries, keys, values are computed from encoder states



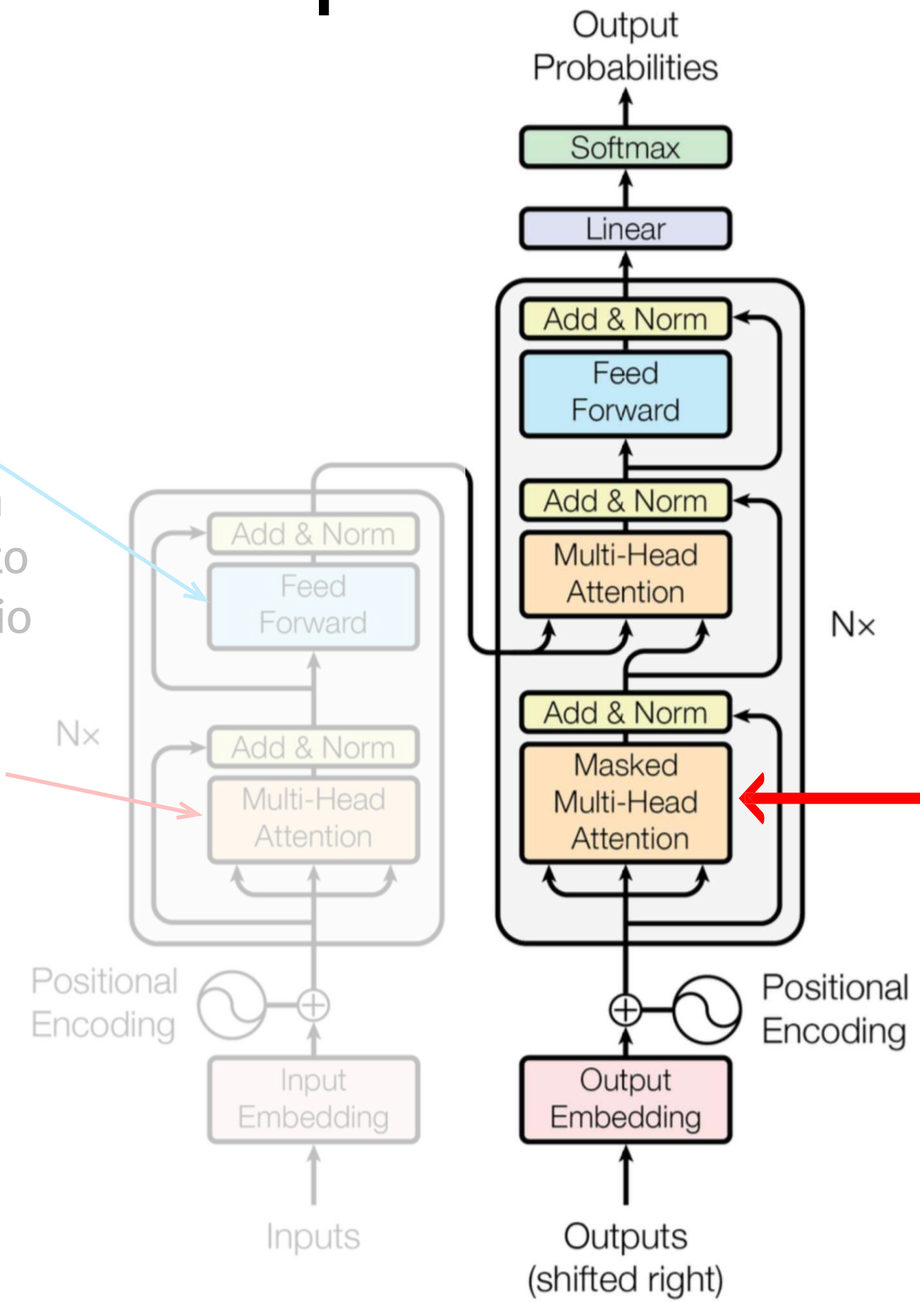


Self Attention masqué

Feed-forward network:
after taking information from other tokens, take a moment to think and process this informatio

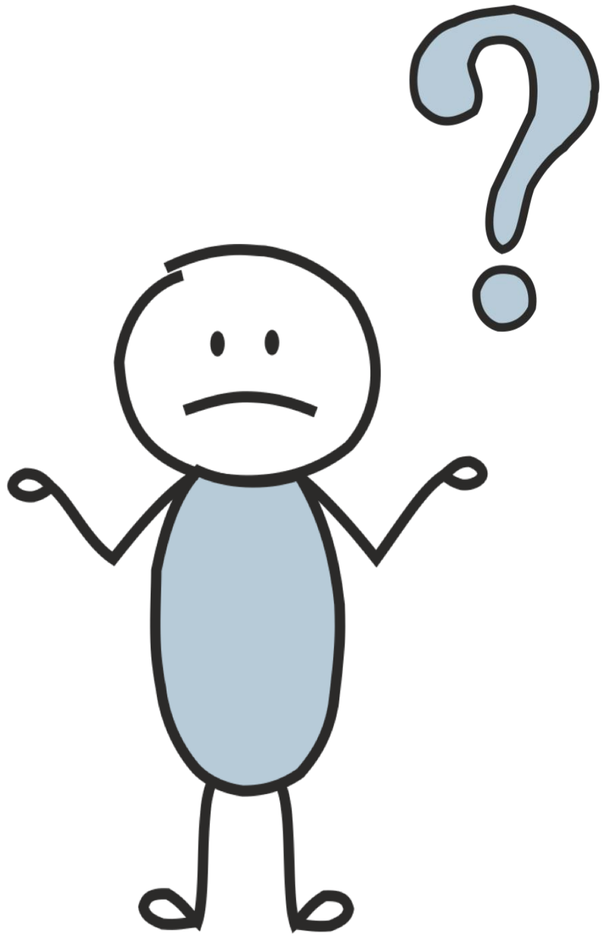
Encoder self-attention:
tokens look at each other

queries, keys, values are computed from encoder states



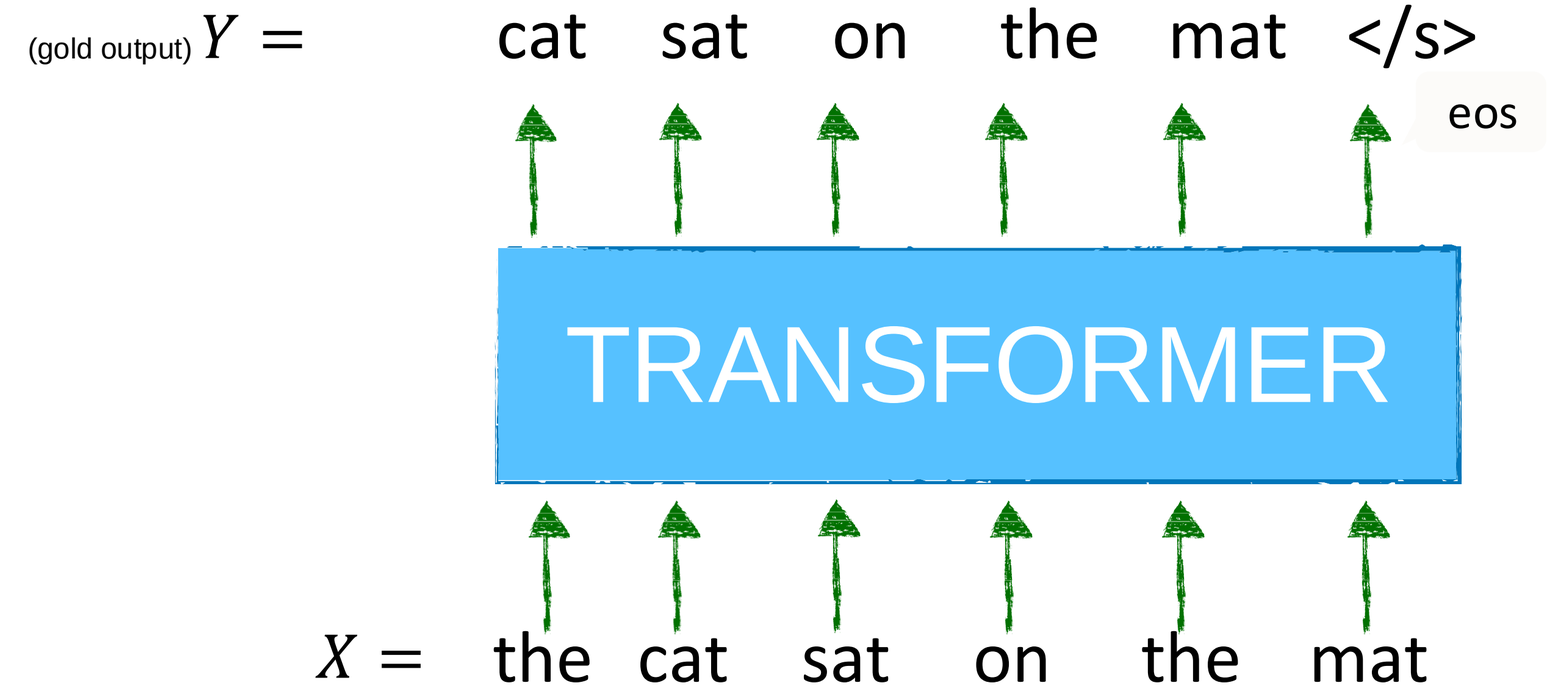
Decoder self-attention (masqué):

Des idées ?



Entraînement d'un modèle de langage Transformer

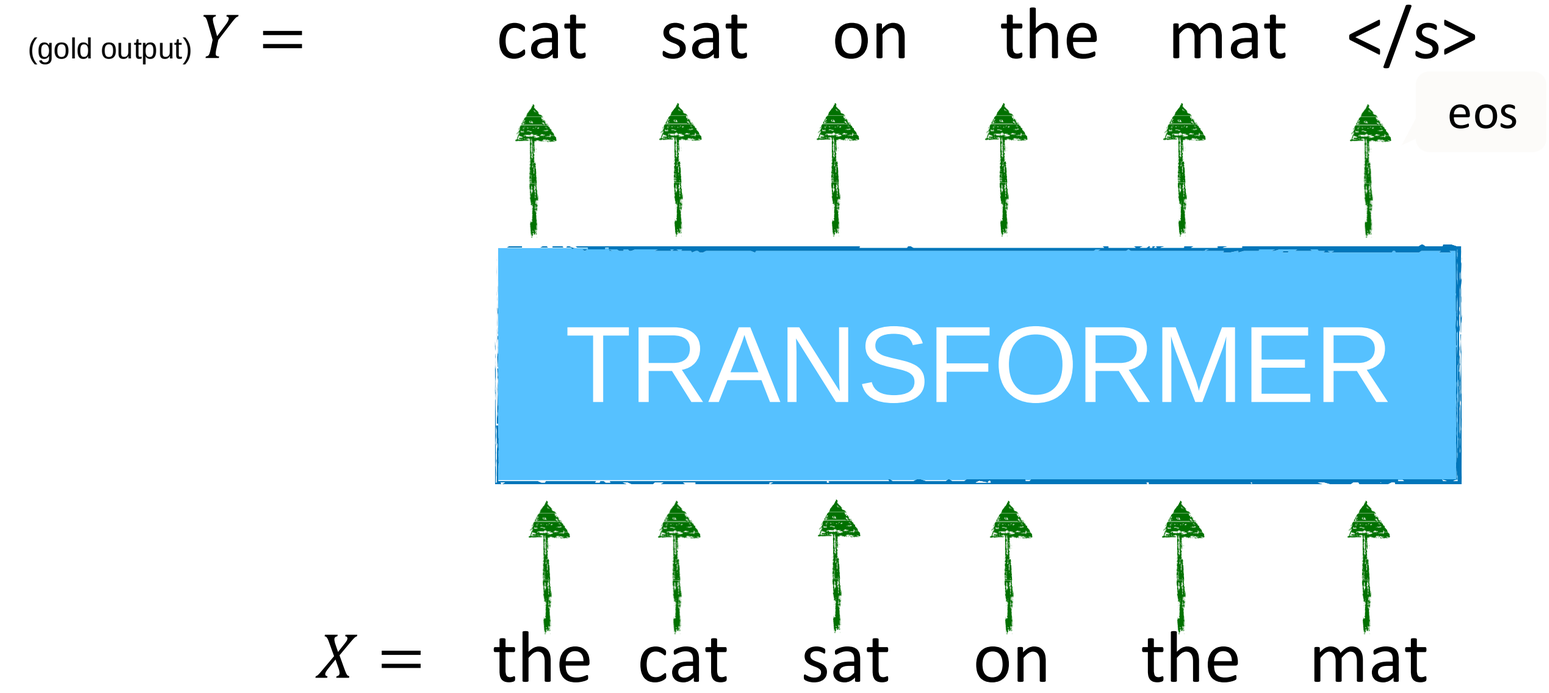
- **Goal:** Entraînez un Transformer pour la modélisation du langage (c'est-à-dire la prédiction du mot suivant).
- **Approach:** Entraînez-le de manière à ce que chaque position soit un prédicteur du token suivant (à droite).



[Slide credit: Arman Cohan]

Entraînement d’un modèle de langage Transformer

- **Goal:** Entraînez un Transformer pour la modélisation du langage (c’est-à-dire la prédiction du mot suivant).
  - **Approach:** Nous décalons simplement l’entrée vers la droite d’une unité, et l’utilisons comme labels

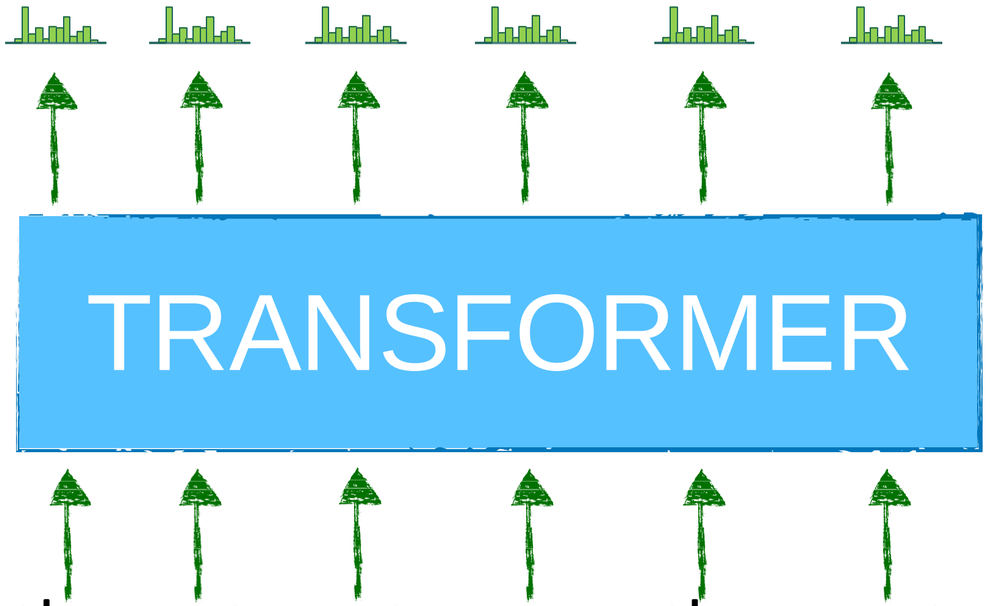


[Slide credit: Arman Cohan]

Entraînement d’un modèle de langage Transformer

- Pour chaque position, calculez leur distribution correspondante sur l’ensemble du vocabulaire.

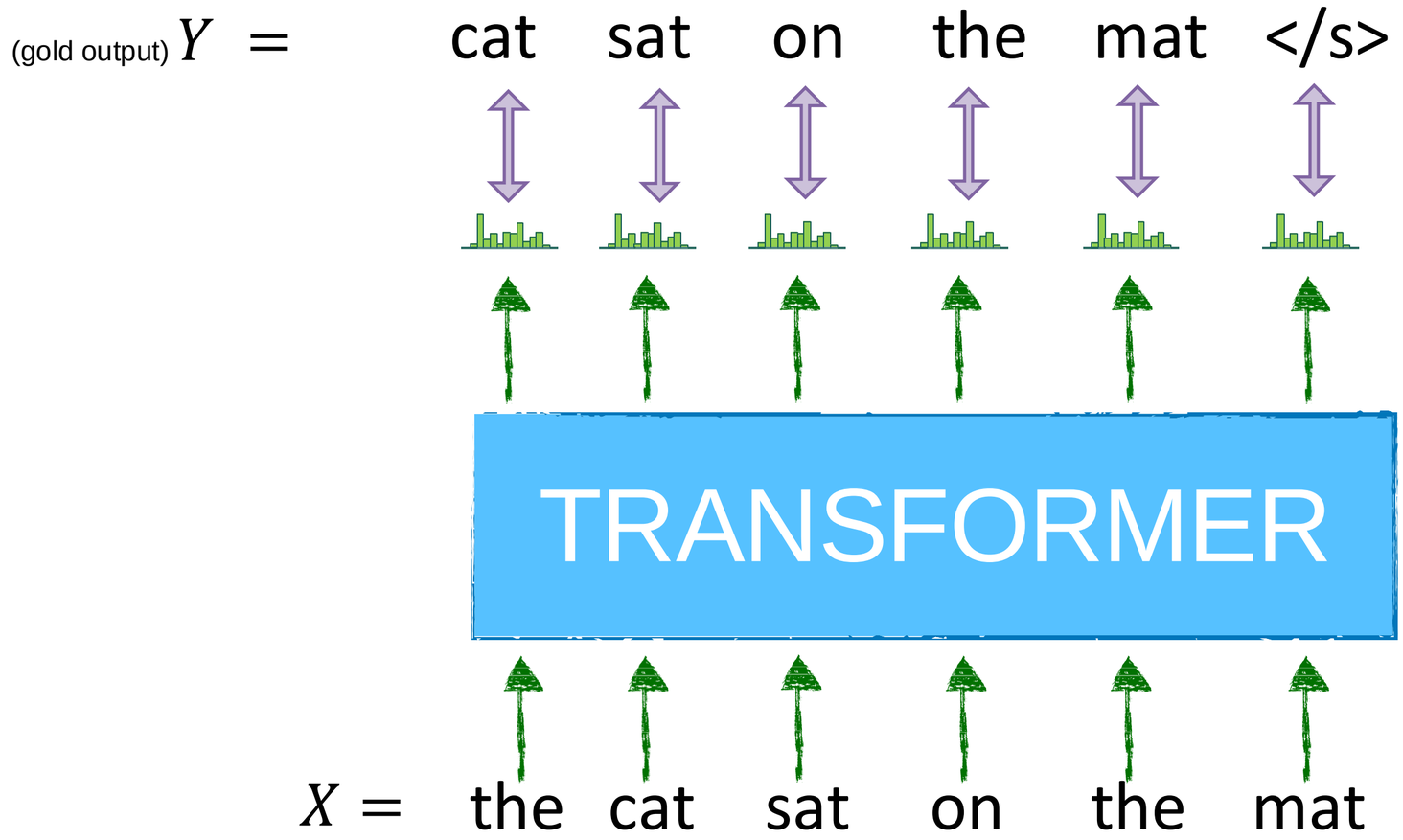
(gold output) $Y =$ cat sat on the mat </s>



$X =$ the cat sat on the mat

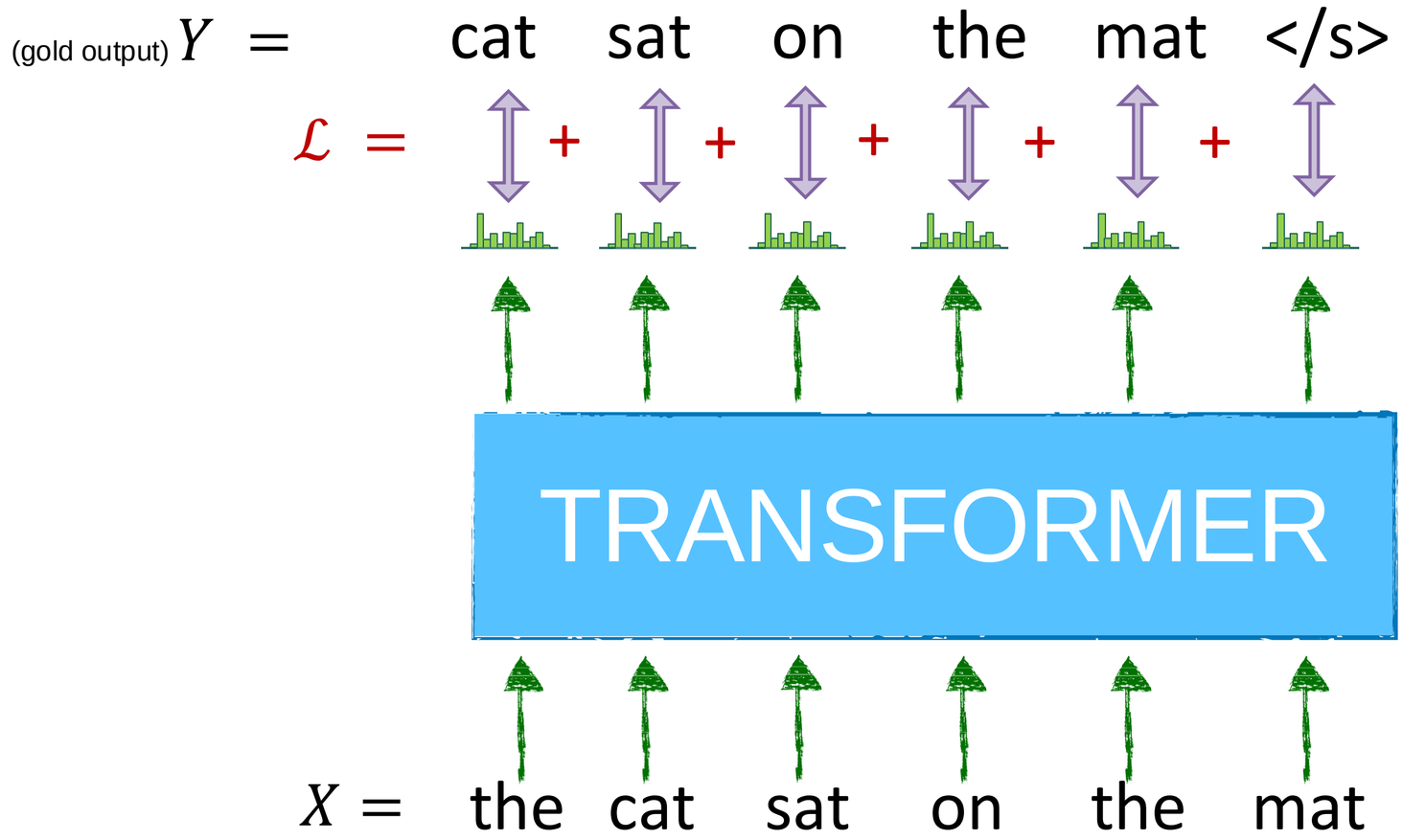
Entraînement d’un modèle de langage Transformer

- Pour chaque position, calculez la perte entre la distribution et l’étiquette réelle.



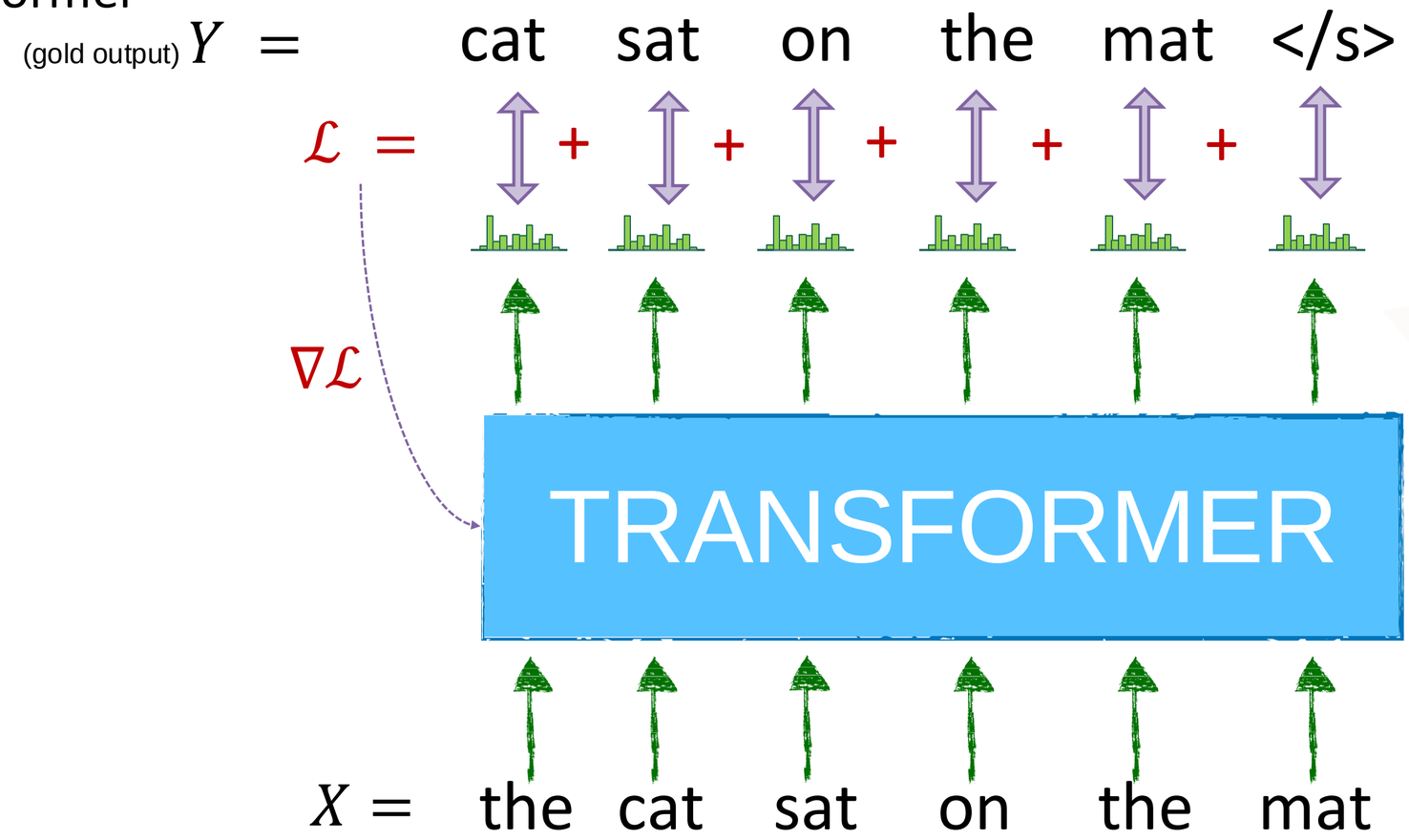
Entraînement d’un modèle de langage Transformer

- Additionnez les valeurs de perte par position pour obtenir une loss globale.



Entraînement d'un modèle de langage Transformer

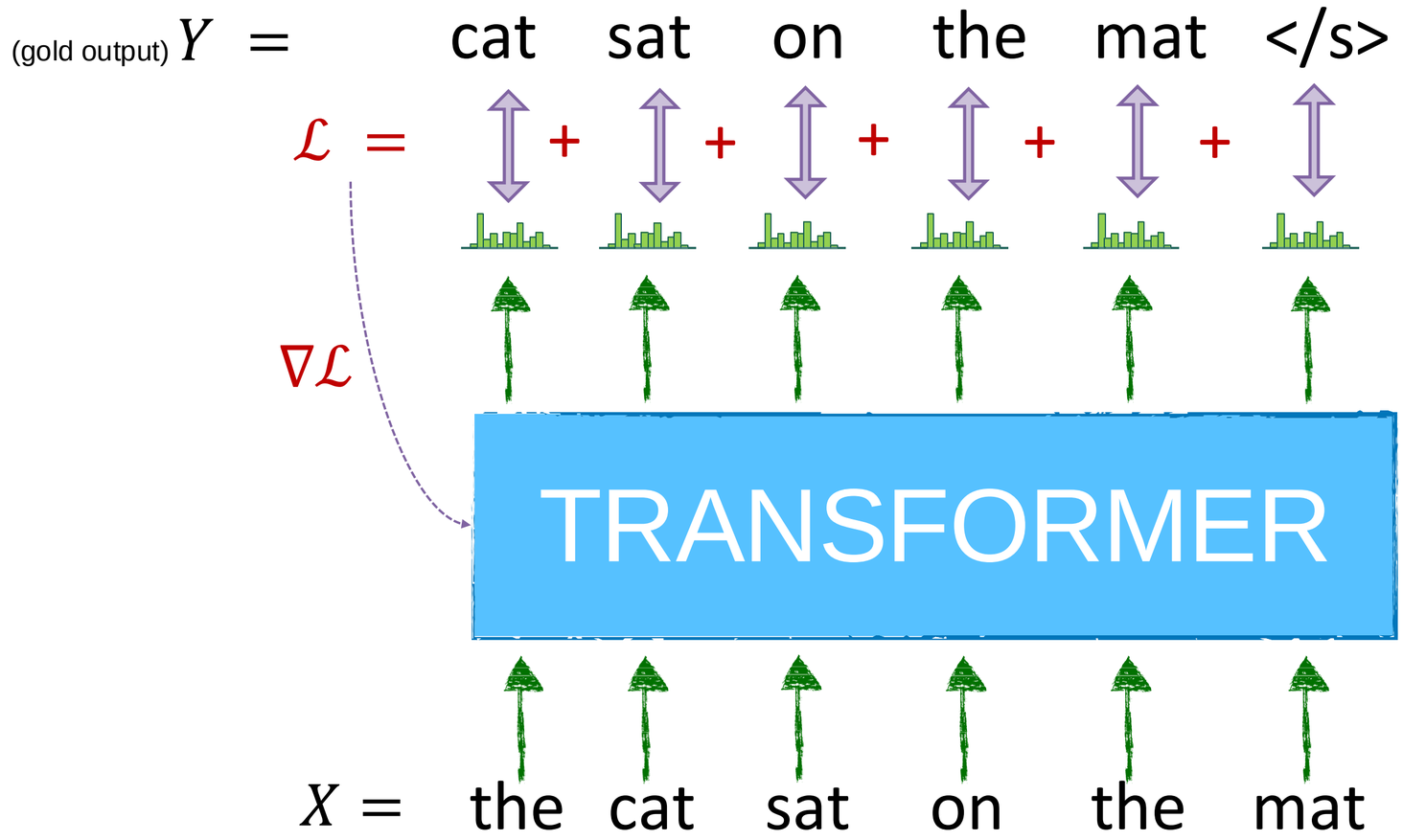
- En utilisant cette loss, rétropopager le gradient and **mettre à jour** les paramètres du Transformer



Eh bien, ce n'est pas tout à fait exact 🤡 ...

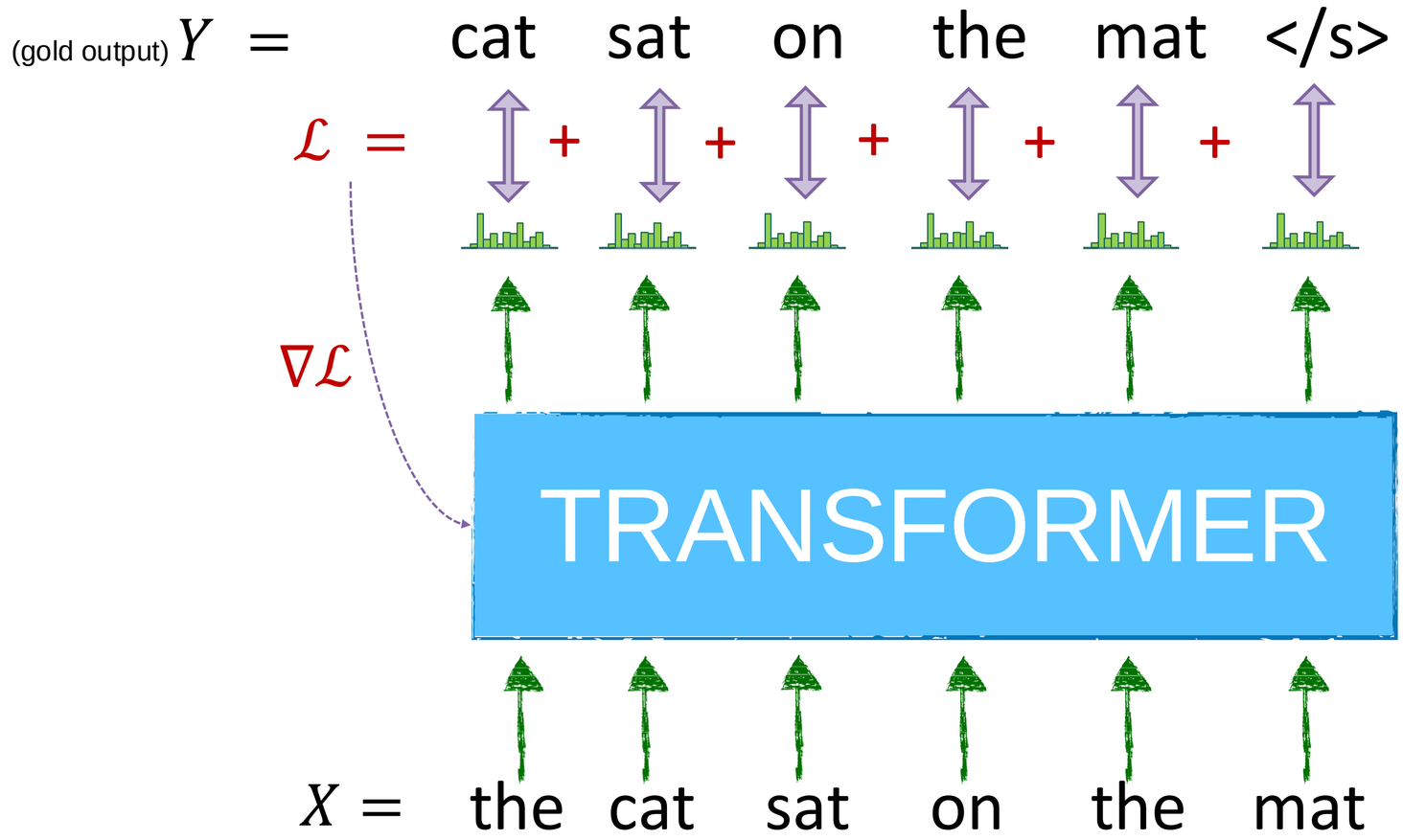
Entraînement d’un modèle de langage Transformer

- Le modèle résoudrait la tâche en copiant le jeton suivant à sortir (fuite de données).
- **N’apprend rien d’utile**



Entraînement d'un modèle de langage Transformer

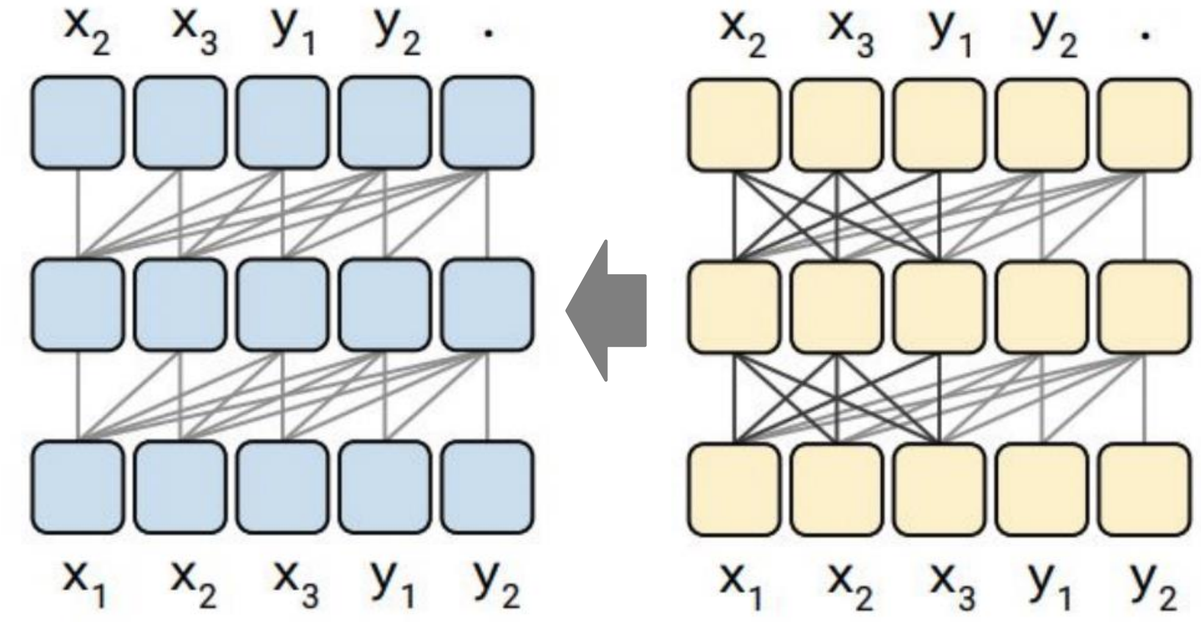
- Nous devons **éviter les fuites d'informations** des tokens futurs ! Comment?



Attention mask

Attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	1.68	-0.36	0.64	0.36	0.27	0.66
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-0.31	1.54	1.66	1.14	0.58	-1.44
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-0.96	-0.17	-0.9	-1.57	0.22
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-0.7	-0.04	1.54	0.81
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	0.3	0.57	0.74
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	0.5	-0.3
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-0.43
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

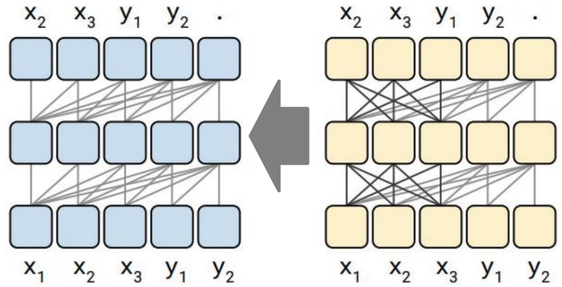


Ce que nous voulons

Ce que nous avons

Slide credit: Arman Cohan

Attention mask

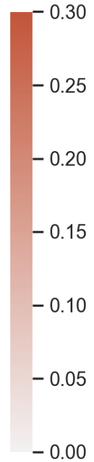


Attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	1.68	-0.36	0.64	0.36	0.27	0.66
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-0.31	1.54	1.66	1.14	0.58	-1.44
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-0.96	-0.17	-0.9	-1.57	0.22
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-0.7	-0.04	1.54	0.81
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	0.3	0.57	0.74
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	0.5	-0.3
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-0.43
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

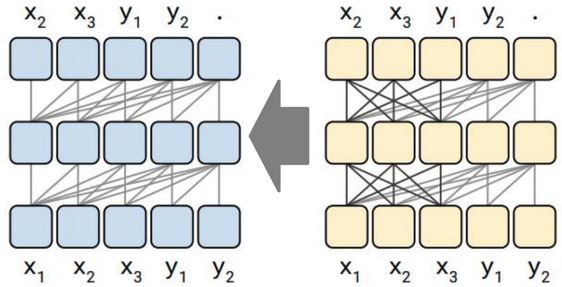
Attention mask

	0	1	2	3	4	5	6	7	8	9	10	11
0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
1	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
2	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
3	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
4	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf
5	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf
6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf
7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf
9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf
10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf
11	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0



Slide credit: Arman Cohan

Attention mask



Attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	1.24	0.69	-0.98	1.43	-0.6	0.7	0.16	0.93	1.28	-1.61	-1.1
1	-0.09	-0.0	-0.7	0.06	0.25	0.23	0.26	0.18	0.78	-0.21	-1.01	1.01
2	0.86	1.19	1.59	0.86	-0.13	-0.15	-2.13	-0.98	-0.87	-1.72	1.87	-0.72
3	0.12	-0.03	-0.02	0.88	-0.46	-0.7	0.54	-0.42	-1.89	-0.38	0.04	-0.84
4	0.51	0.17	0.13	-1.64	0.24	-0.02	1.68	-0.36	0.64	0.36	0.27	0.66
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-0.31	1.54	1.66	1.14	0.58	-1.44
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-0.96	-0.17	-0.9	-1.57	0.22
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-0.7	-0.04	1.54	0.81
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	0.3	0.57	0.74
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	0.5	-0.3
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-0.43
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

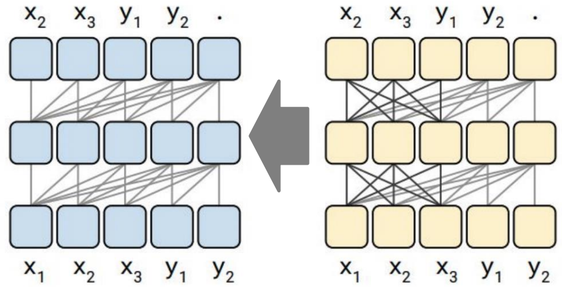
X

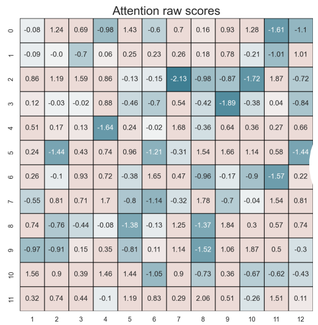
Attention mask

	0	1	2	3	4	5	6	7	8	9	10	11
0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
1	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
2	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
3	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
4	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf
5	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf	-inf
6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf	-inf
7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf	-inf
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf	-inf
9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf	-inf
10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-inf
11	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

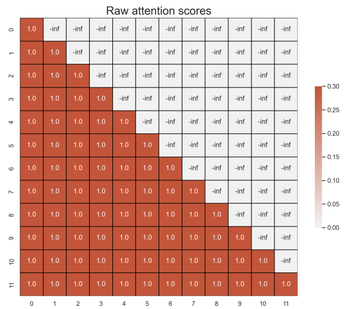
La multiplication matricielle des notes est assez rapide dans les GPU.

Attention mask





x



=

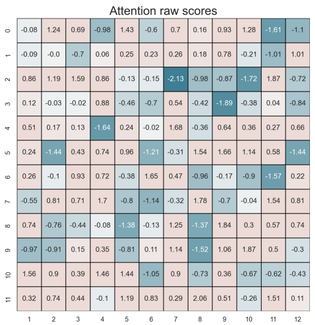
Masked attention raw scores

	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.08	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
1	-0.09	-0.0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
2	0.86	1.19	1.59	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
3	0.12	-0.03	-0.02	0.88	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
4	0.51	0.17	0.13	-1.64	0.24	-inf	-inf	-inf	-inf	-inf	-inf	-inf
5	0.24	-1.44	0.43	0.74	0.96	-1.21	-inf	-inf	-inf	-inf	-inf	-inf
6	0.26	-0.1	0.93	0.72	-0.38	1.65	0.47	-inf	-inf	-inf	-inf	-inf
7	-0.55	0.81	0.71	1.7	-0.8	-1.14	-0.32	1.78	-inf	-inf	-inf	-inf
8	0.74	-0.76	-0.44	-0.08	-1.38	-0.13	1.25	-1.37	1.84	-inf	-inf	-inf
9	-0.97	-0.91	0.15	0.35	-0.81	0.11	1.14	-1.52	1.06	1.87	-inf	-inf
10	1.56	0.9	0.39	1.46	1.44	-1.05	0.9	-0.73	0.36	-0.67	-0.62	-inf
11	0.32	0.74	0.44	-0.1	1.19	0.83	0.29	2.06	0.51	-0.26	1.51	0.11

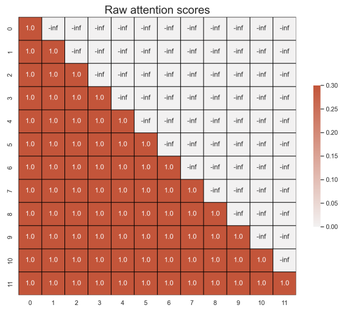
Slide credit: Arman Cohan

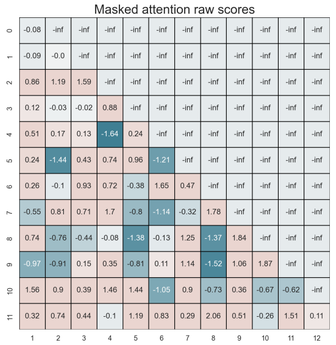
Attention mask

L’effet est plus que simplement élaguer certains des câblages dans le bloc d’auto-attention.

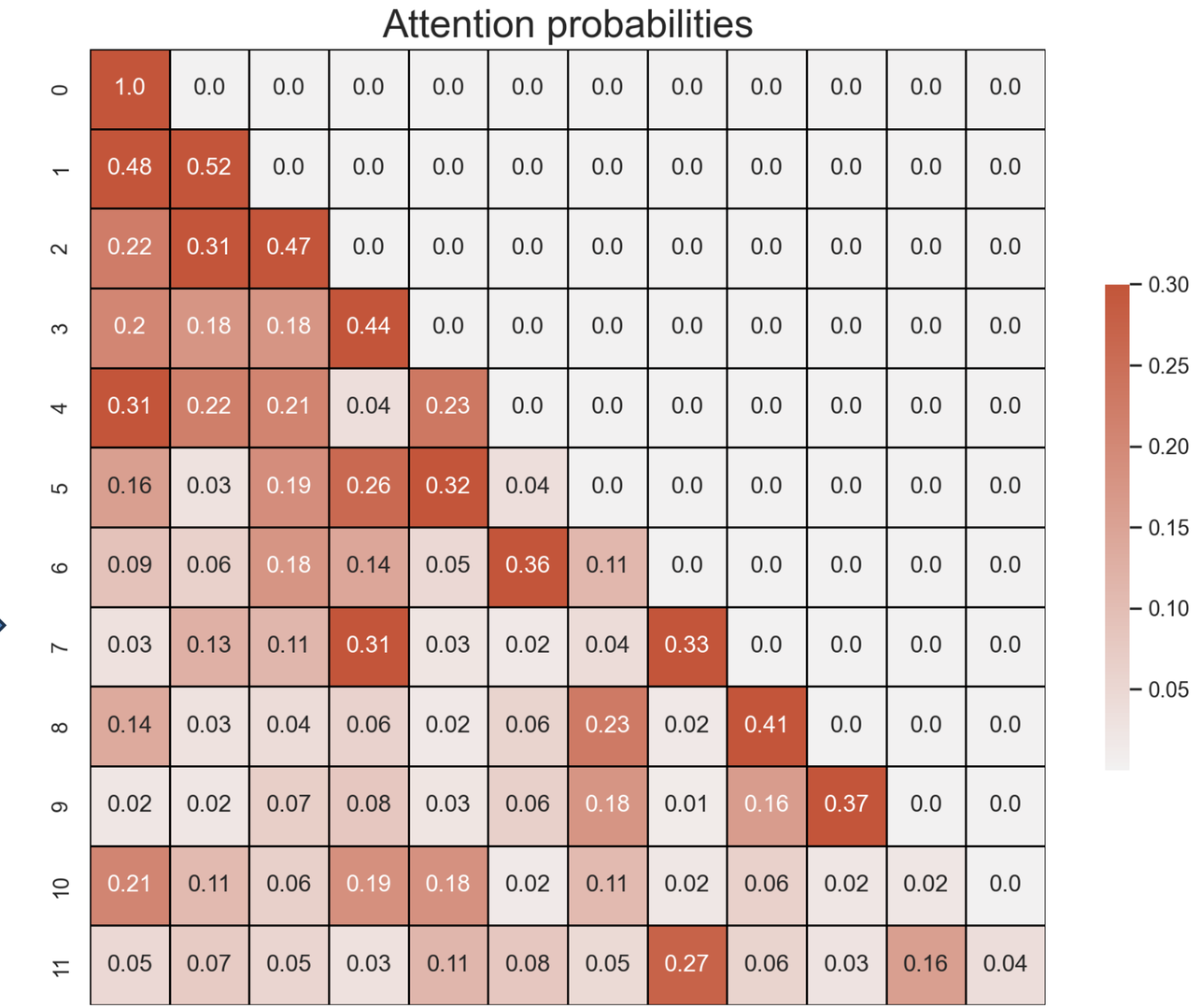


X





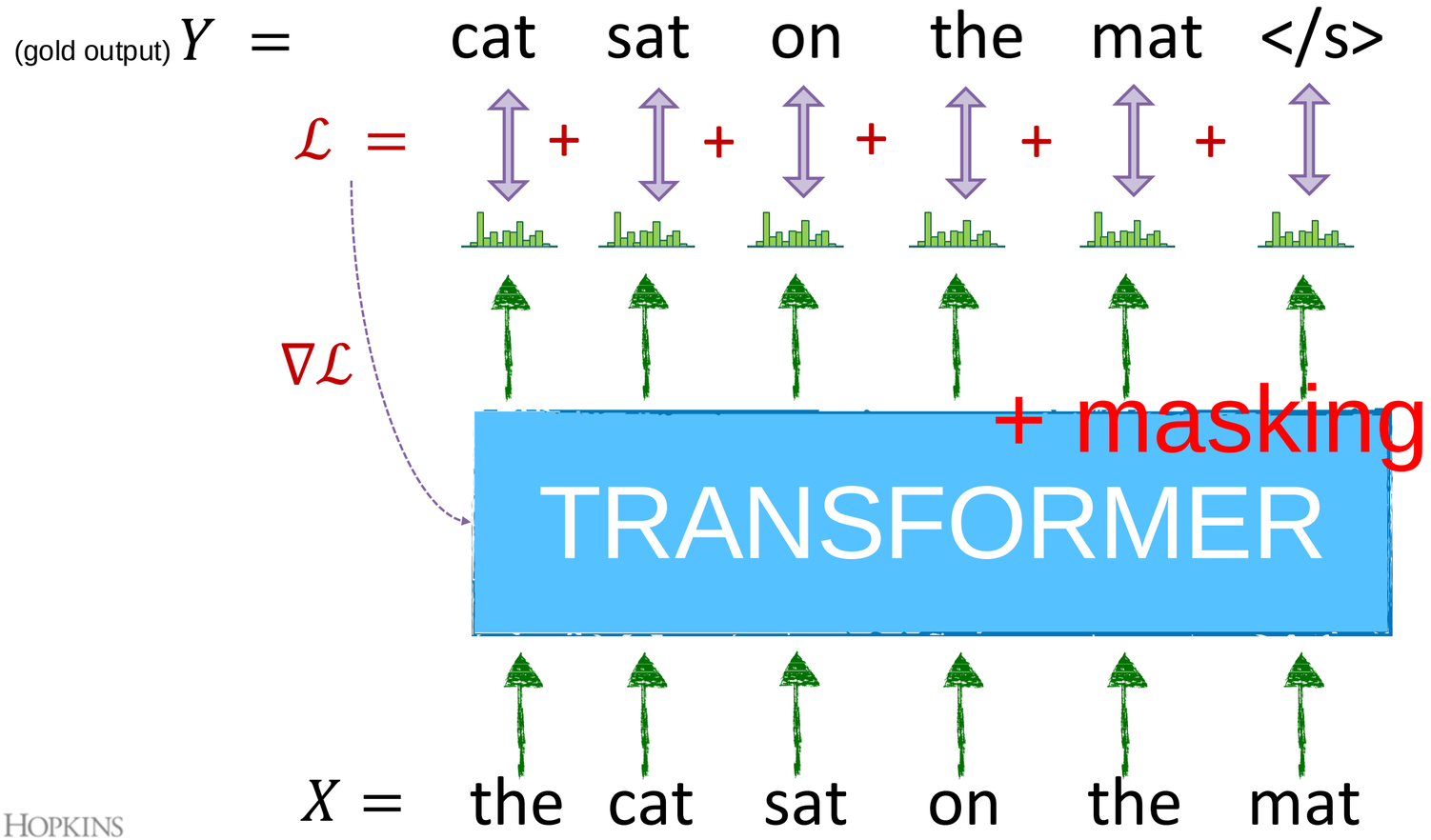
softmax



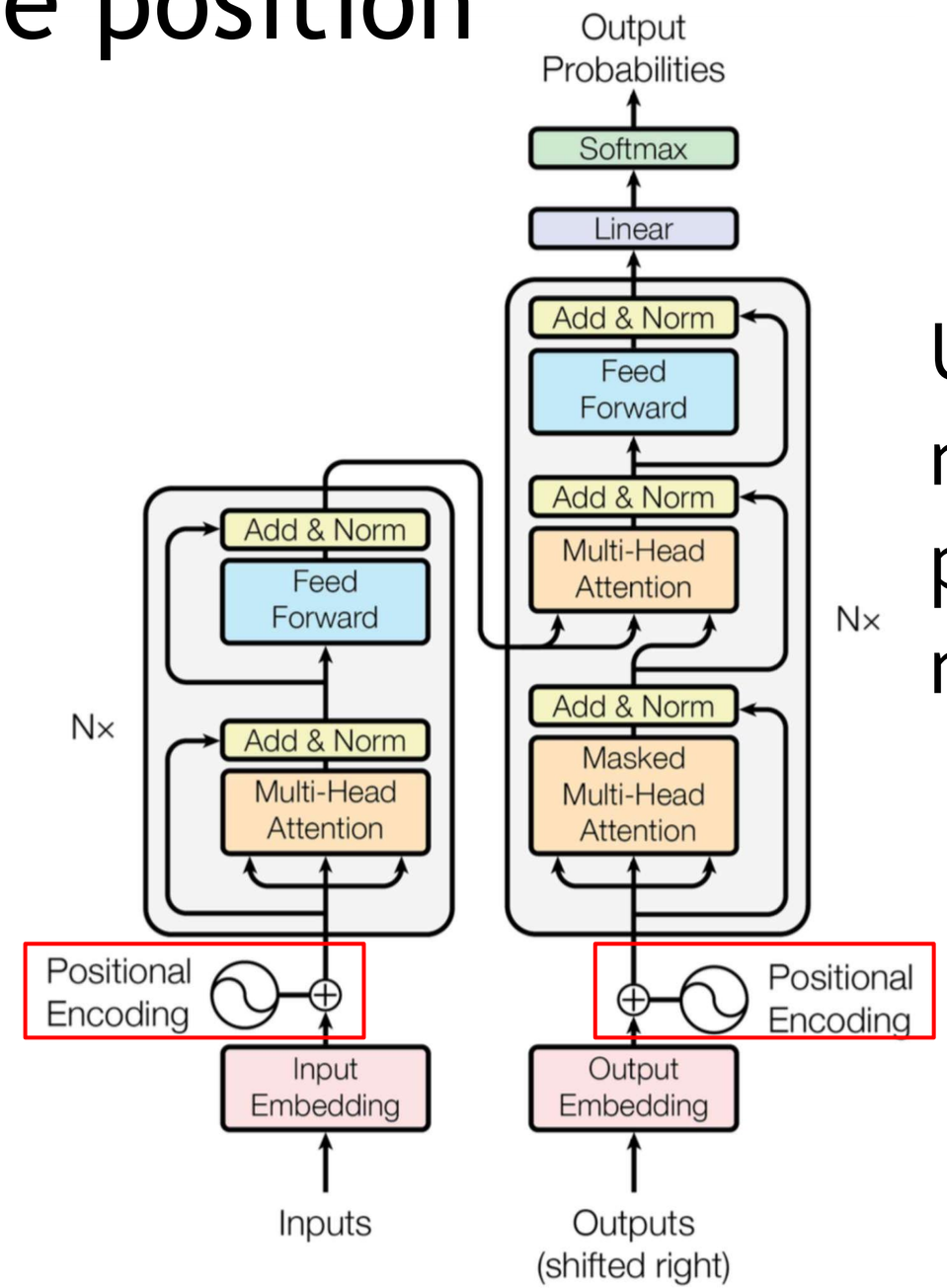
Slide credit: Arman Cohan

Training a Transformer Language Model

- Nous devons empêcher les fuites d'informations des futurs tokens ! Comment?



Encodage de position



Un problème : le modèle ne connaît pas les positions/l'ordre des mots.

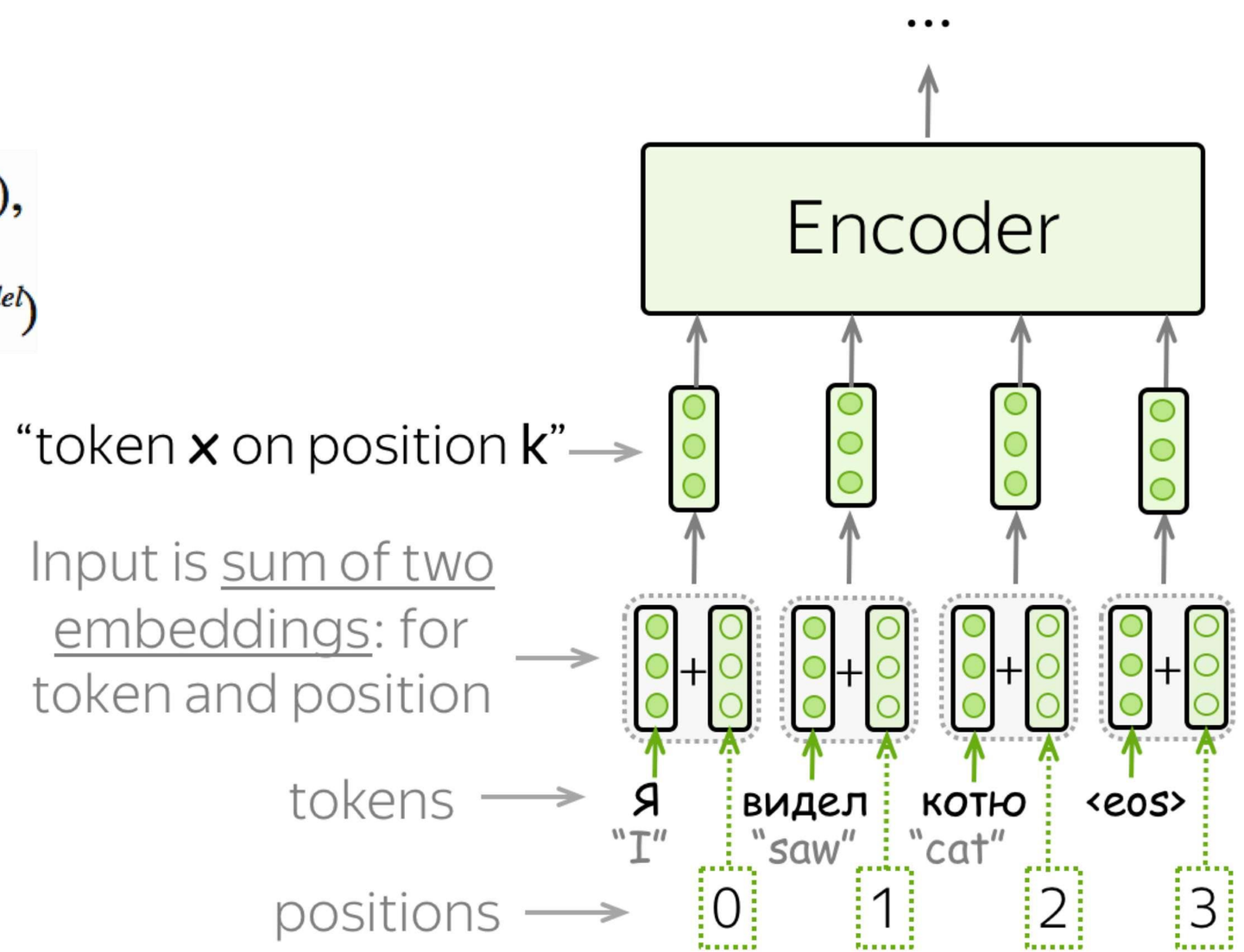
Encodage des positions

Fixed encodings:

$$\mathrm{PE}_{pos,2i} = \sin(pos/10000^{2i/d_{model}}),$$

$$\mathrm{PE}_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

pos – position, i - dimension



Positional Encoding

$$P(k, 2i) = \sin\left(\frac{k}{n^{\frac{2i}{d}}}\right) \qquad P(k, 2i) = cos\left(\frac{k}{n^{\frac{2i}{d}}}\right)$$

k: Position of an object in the input sequence, $0 \leq \mathrm{k} < \mathrm{L}/2$

d: Dimension of the output embedding space

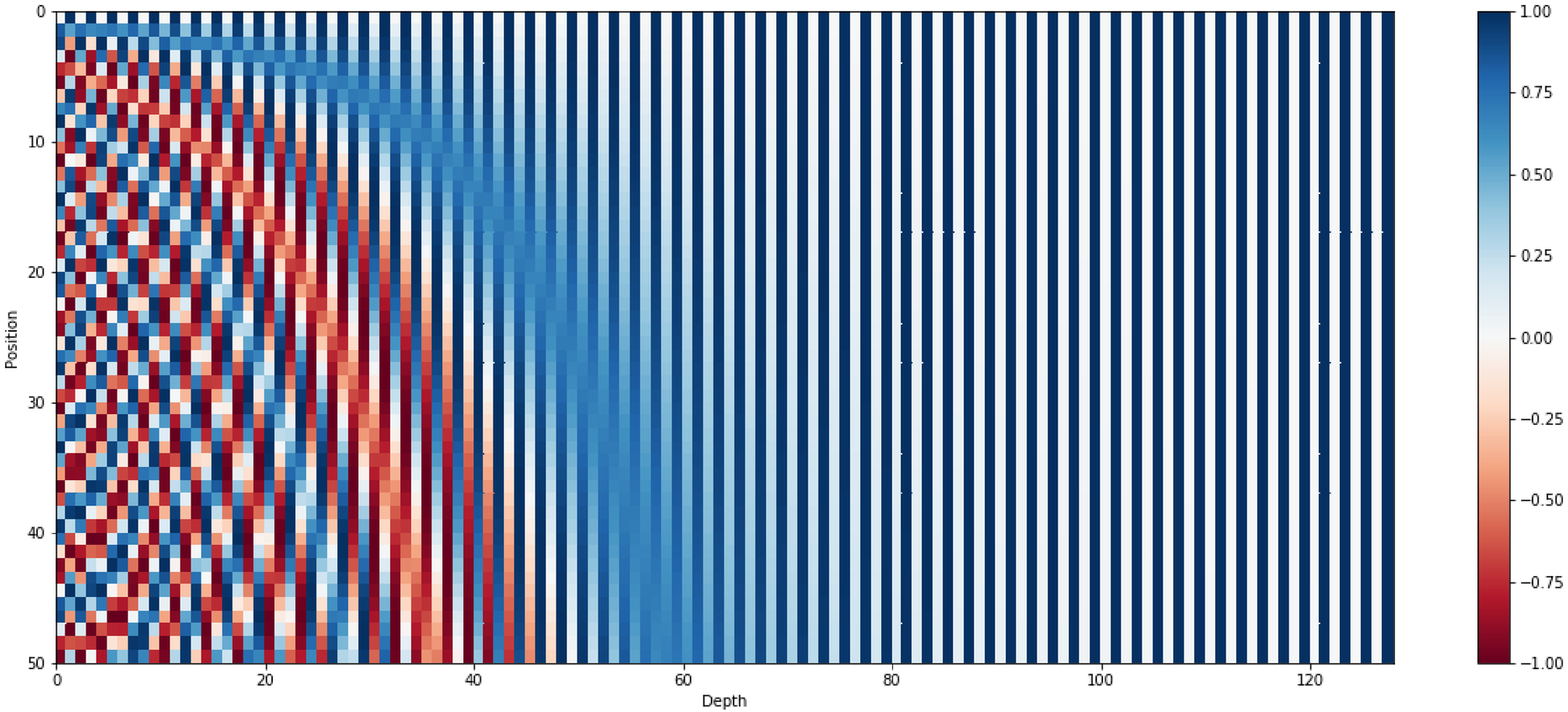
$\mathrm{P}(\mathrm{k}, \mathrm{j})$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

i: Used for mapping to column indices $0 \leq \mathrm{i} < \mathrm{d}/2$, with a single value of i maps to both sine and cosine functions

Positional Encoding

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



A retenir

Les RNNs, même avec l’attention souffrent des carences en exploitations

Le mécanisme d’attention est suffisant pour apprendre

Le transformer est composé d’un ensemble de blocs simples, paralellisables

Prochain cours

TDs

BPE

Attentions

Merci !

Mohamed Abbas KONATE

mohamed-abbas.konate@michelin.com