IA et Nous
Attention

Sources :

- ***Attention is all you need Google 2017***
- *The illustrated Transformer Jay Alammar 2021*
- *Université Paris I - J. Rynkiewicz 2022*

Attention
Le Transformer

Le transformer a été proposé en 2017
dans l'article de Google
Attention is All You Need

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

Abstract

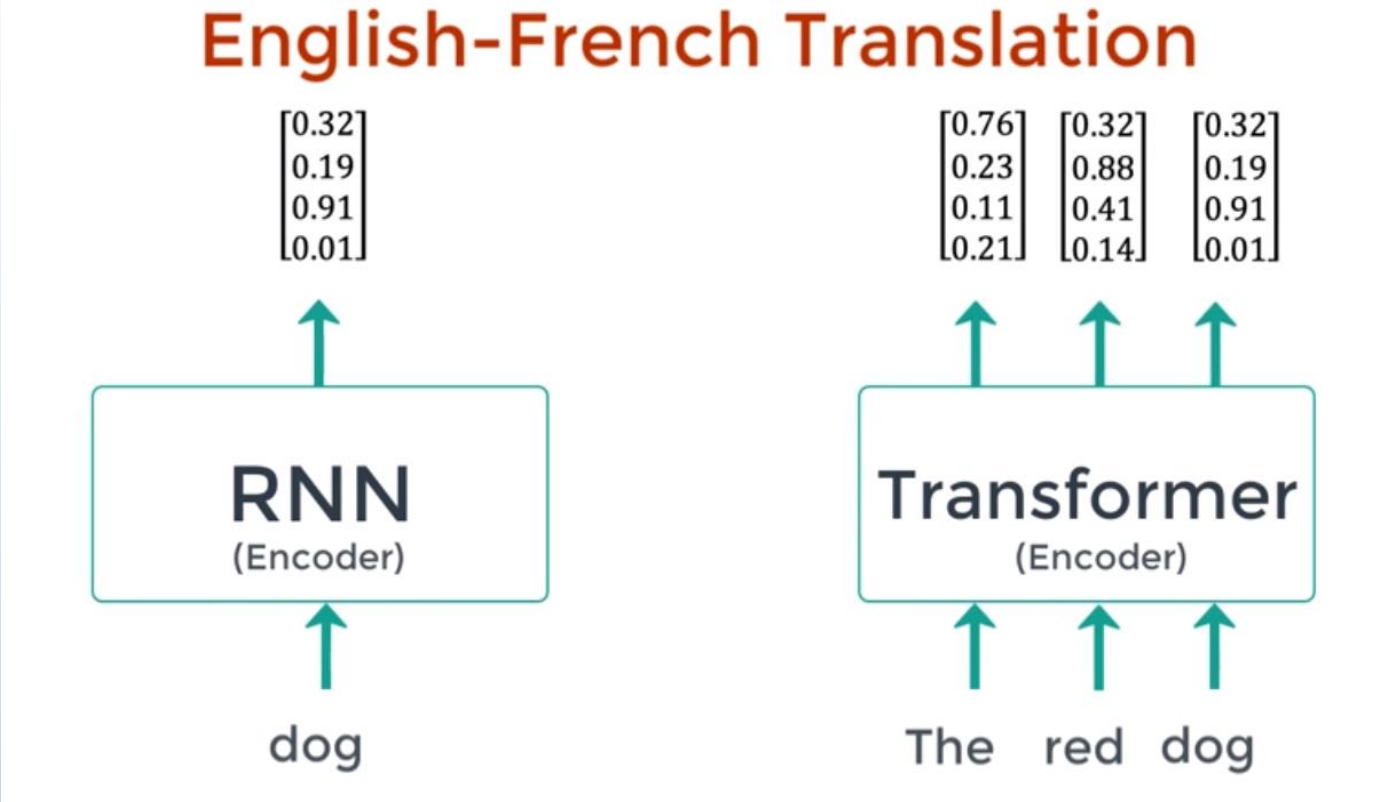
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, 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.

Transformer

On donne au réseau ***toute la phrase*** *en encodant la position de chaque mot*

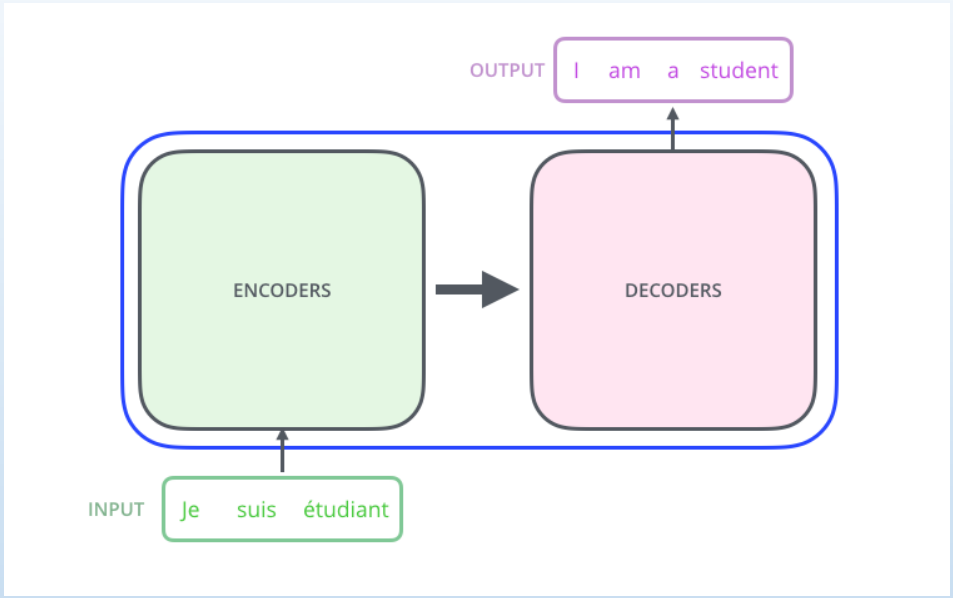
Ce qui leur permet de découper les traitements et de ***paralléliser les calculs de la phase d'apprentissage****.*

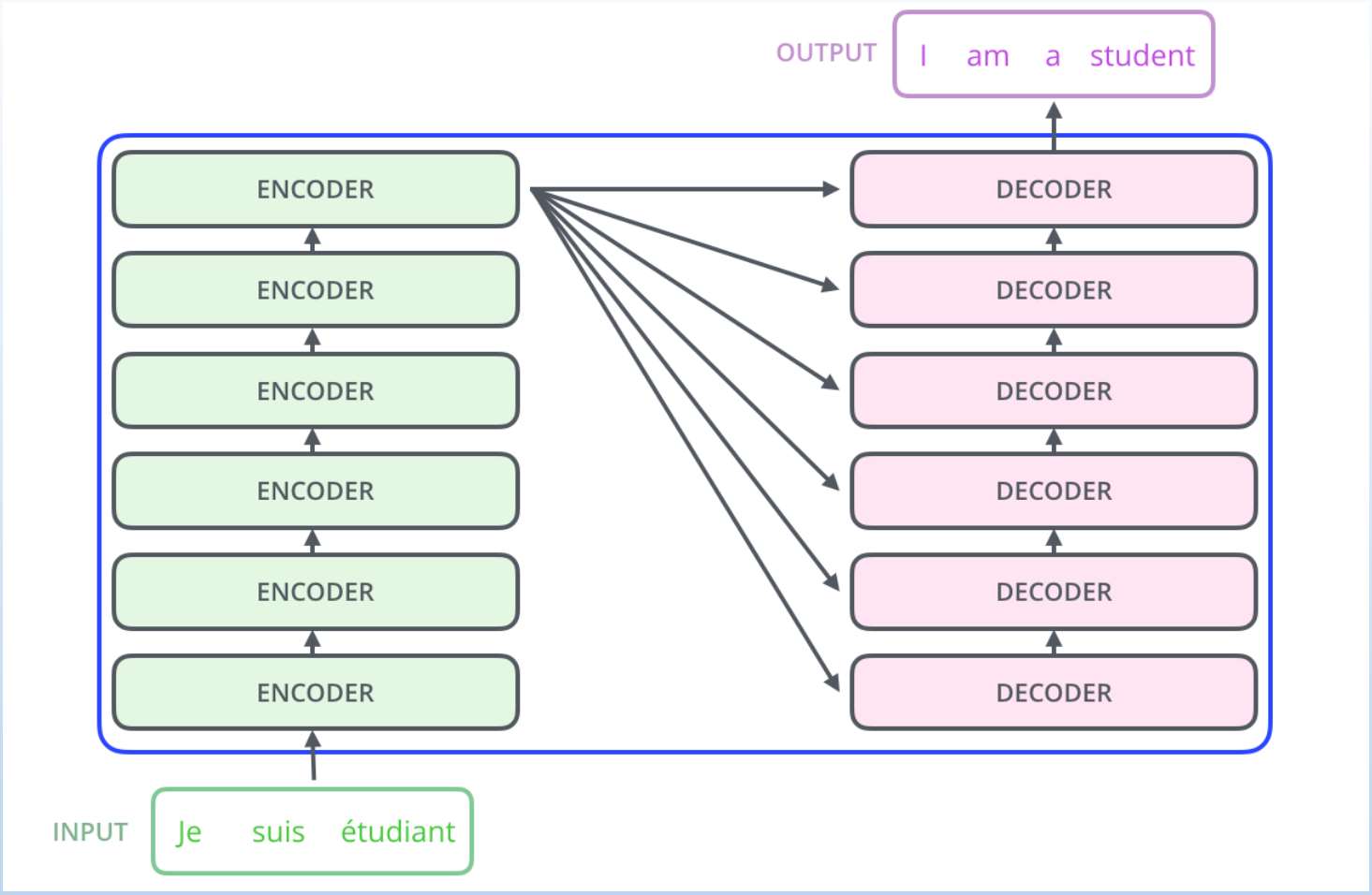
Résultat *: ils sont beaucoup* ***plus rapides à entraîner que les RNN.***



Attention
Le Transformer

Attention

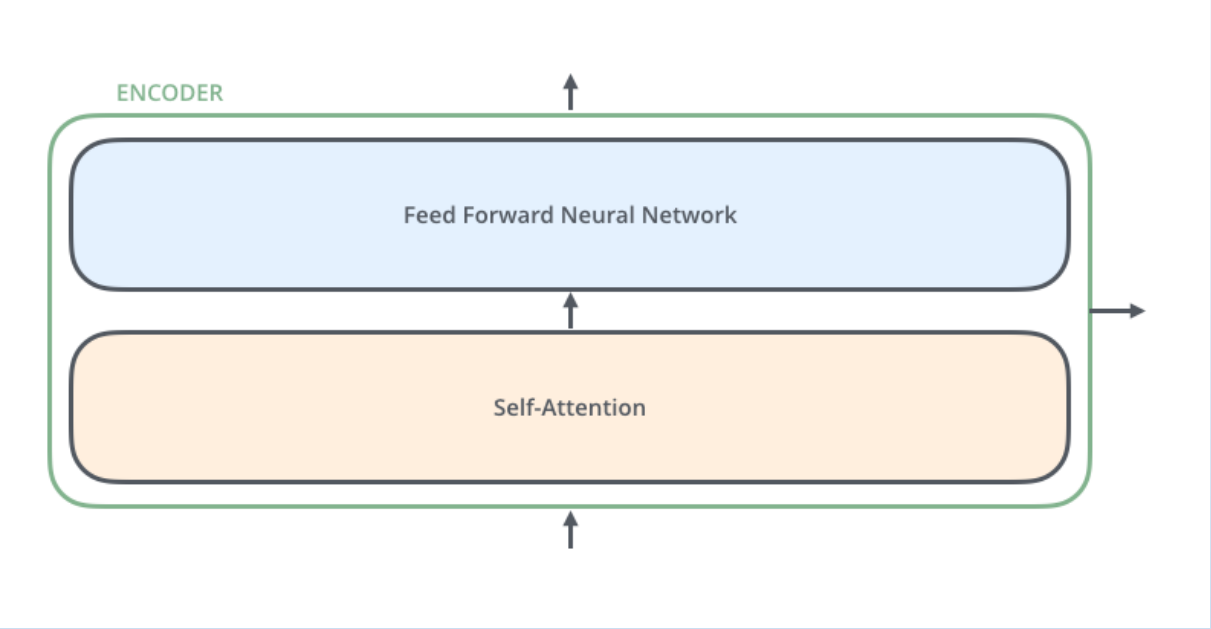




Le composant d'encodage est une pile d'encodeurs (6 dans l'article, 36 dans GPT2,….)

. Le composant de décodage est une pile de décodeurs du même nombre.

Attention

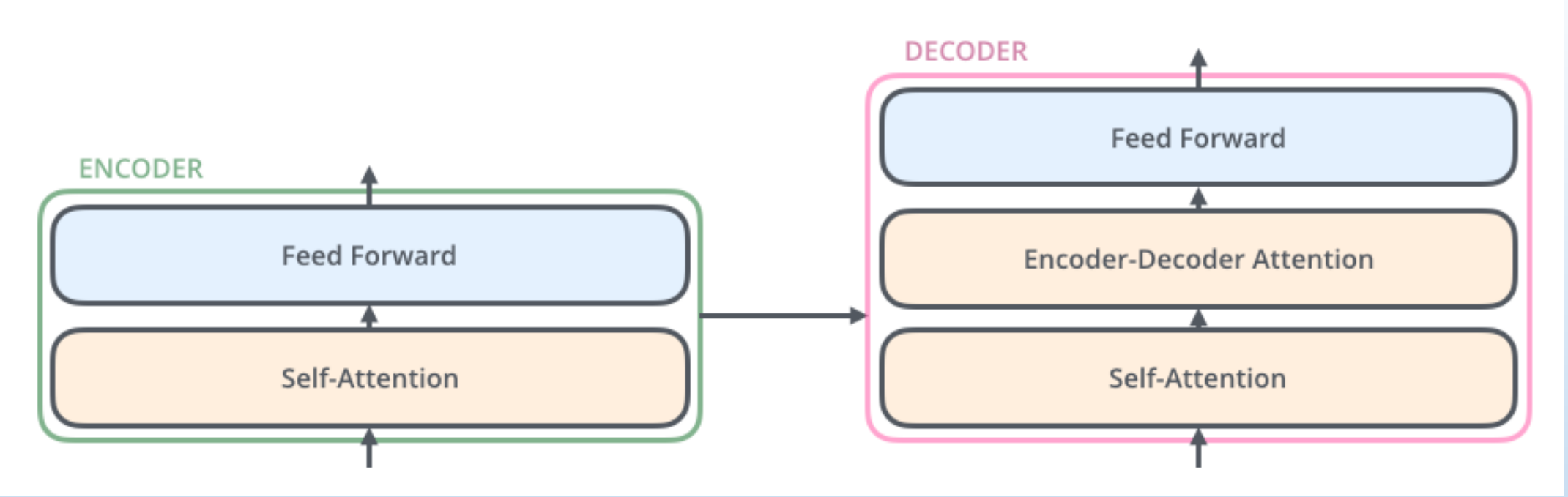


Encodeur

Les entrées de l'encodeur traversent d'abord une couche d'auto-attention – une couche qui aide l'encodeur à regarder d'autres mots dans la phrase d'entrée lorsqu'il code un mot spécifique.

Les sorties de la couche d'auto-attention sont transmises à un réseau neuronal de feed-forward. Le même réseau de feed-forward est appliqué indépendamment à chaque position.

Attention



Le décodeur a ces deux couches, mais entre elles se trouve une couche d'attention qui aide le décodeur à se concentrer sur les parties pertinentes de la phrase d'entrée.

C'est similaire à ce que l'attention fait dans les modèles seq2seq.

Attention

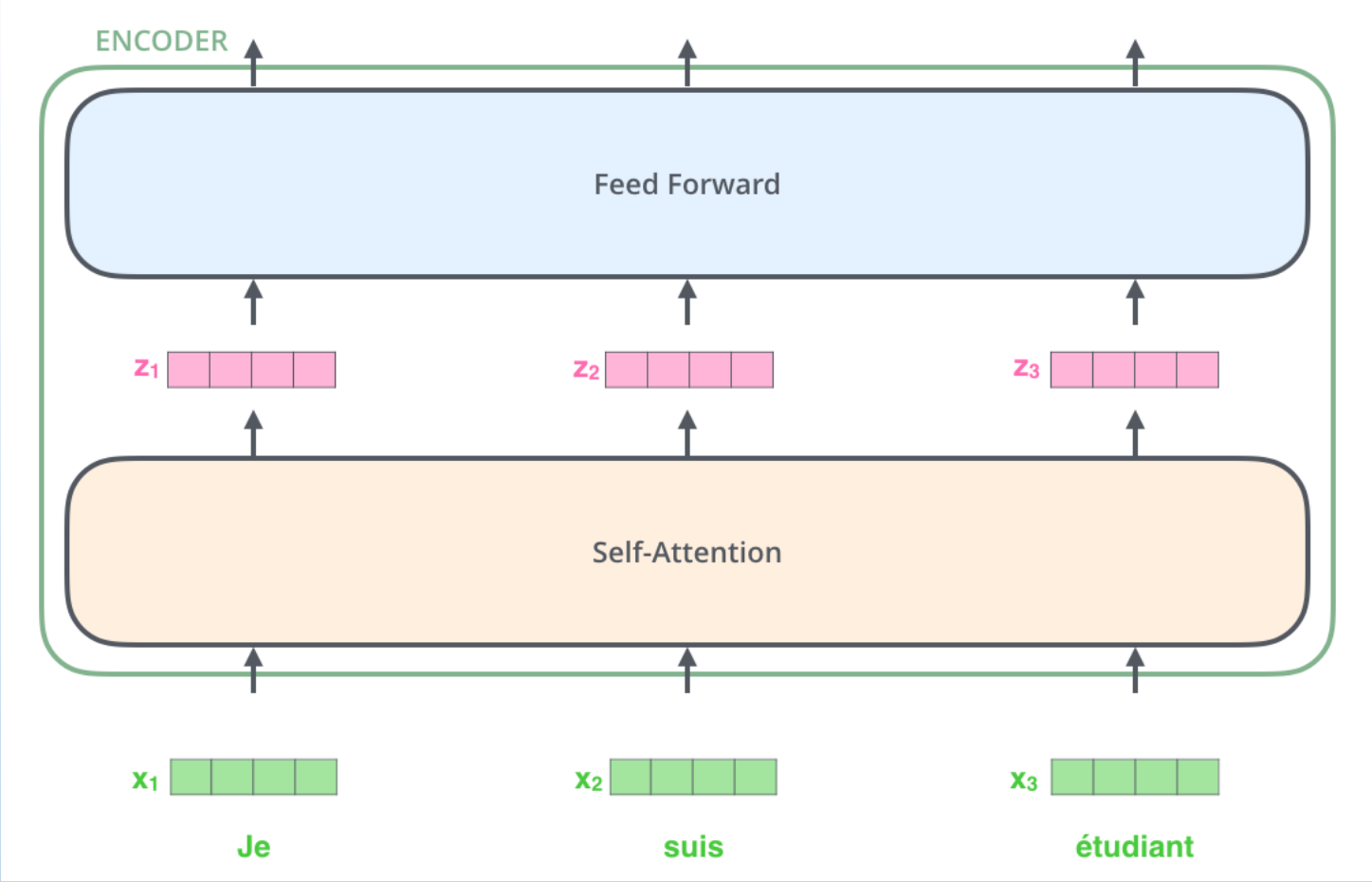
Embedding (Plongement, incorporation)

Chaque mot est incorporé dans un vecteur de taille 512.

Nous représenterons ces vecteurs avec ces boîtes simples, ici 4 cases pour 512 paramètres

Attention

Vers encodeur supérieur



Vecteurs d'embedding

Chaque mot traverse les deux couches de l'encodeur, et circule à travers son propre chemin.

Les dépendances entre ces chemins sont calculés dans la couche d'auto-attention.

Auto Attention

L'attention permet de tenir compte du contexte d'un mot.

*Pour traduire "La brebis n'a pas traversé la rue parce qu'**elle** était trop fatiguée"*

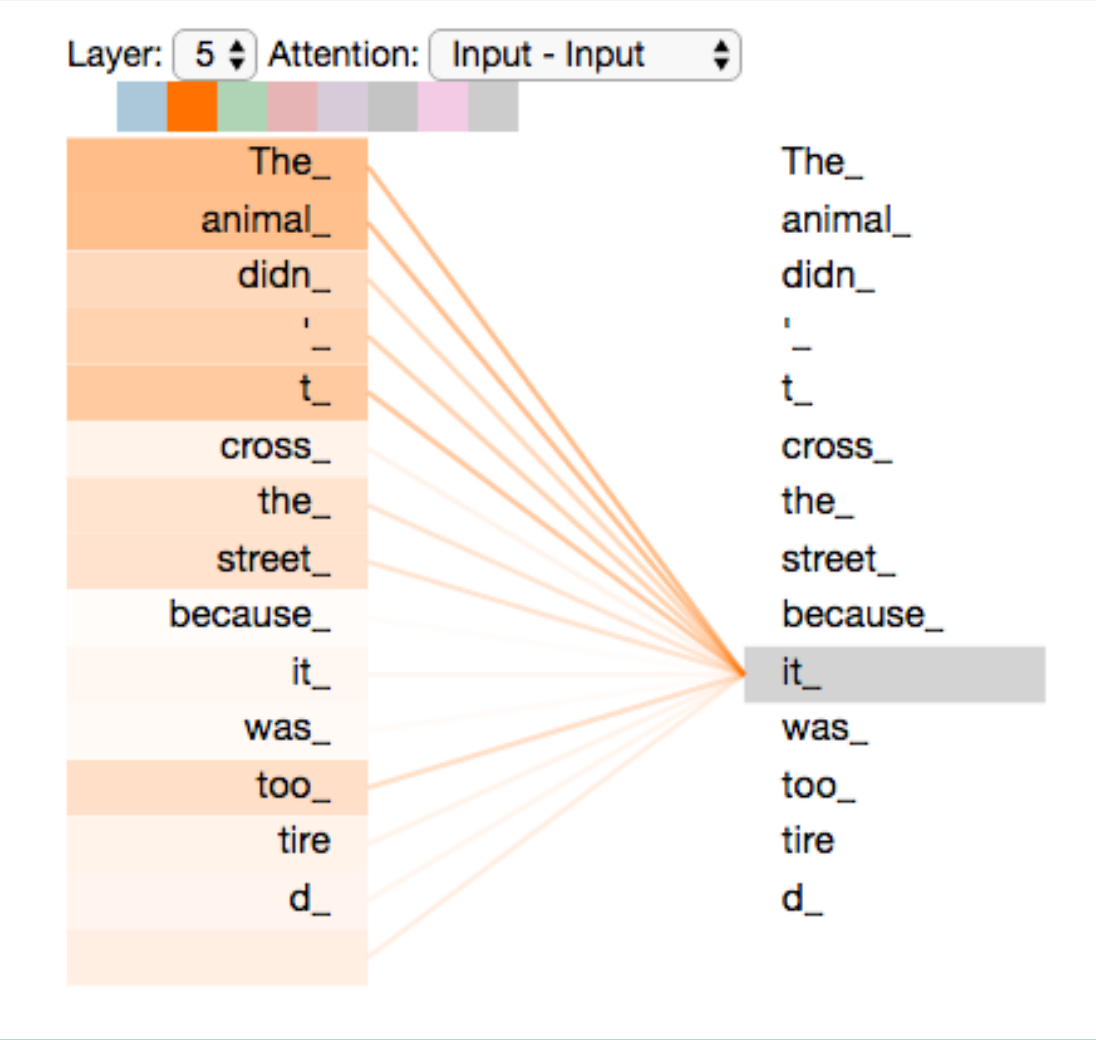
À quoi fait référence "elle" dans le texte ? La brebis ou bien la rue ?

C'est une question facile pour un être humain, mais difficile pour une machine.
La machine doit donc estimer si le mot "elle" est plus lié au mot "brebis" ou au mot "rue".

La couche de "Self-attention" des transformers permet cette estimation.

Attention

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



À quoi fait référence « it » dans cette phrase ?

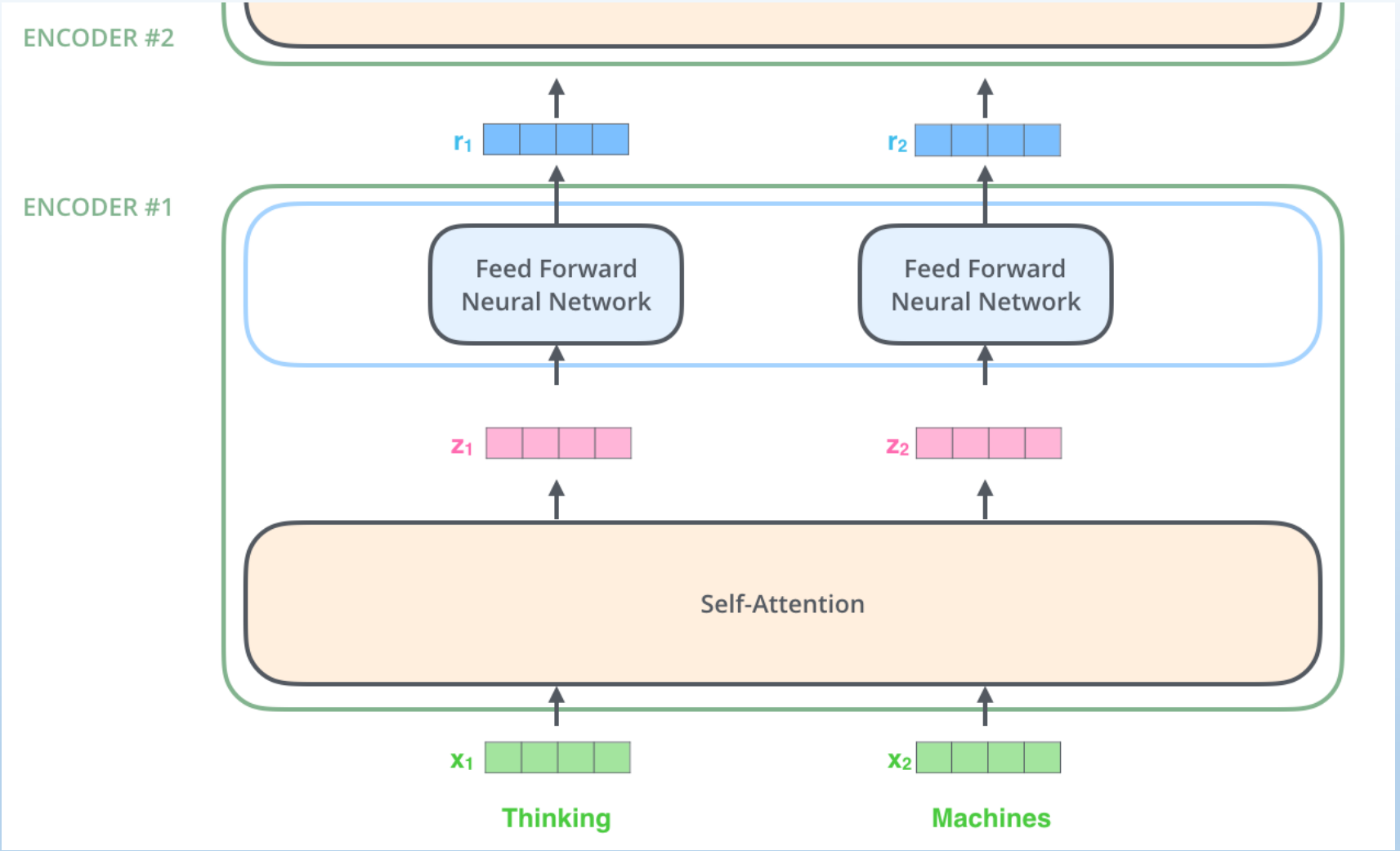
S'agit-il de la rue ou de l'animal?

Question simple pour un humain, mais pas aussi simple pour un algorithme.

Lorsque le modèle traite le mot « it », l'auto-attention lui permet d'associer « it » à « animal ».

Attention

Exemple avec deux mots



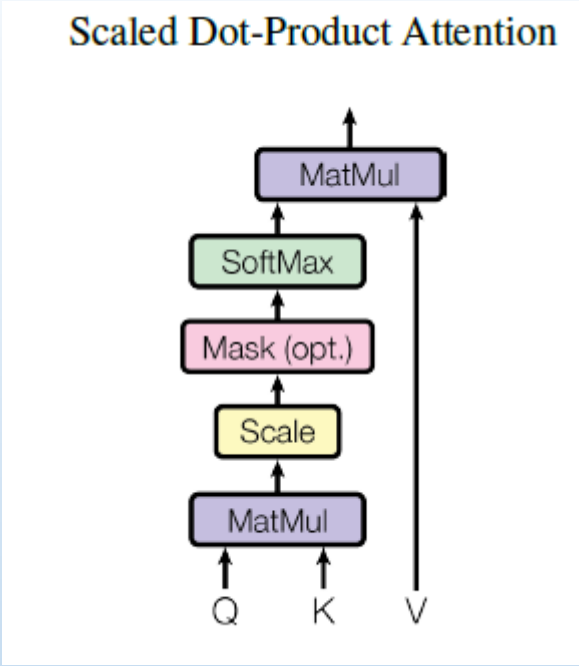
Auto Attention

QKV

Query Key Value

Que sont les vecteurs « requête », « clé » et « valeur » ?

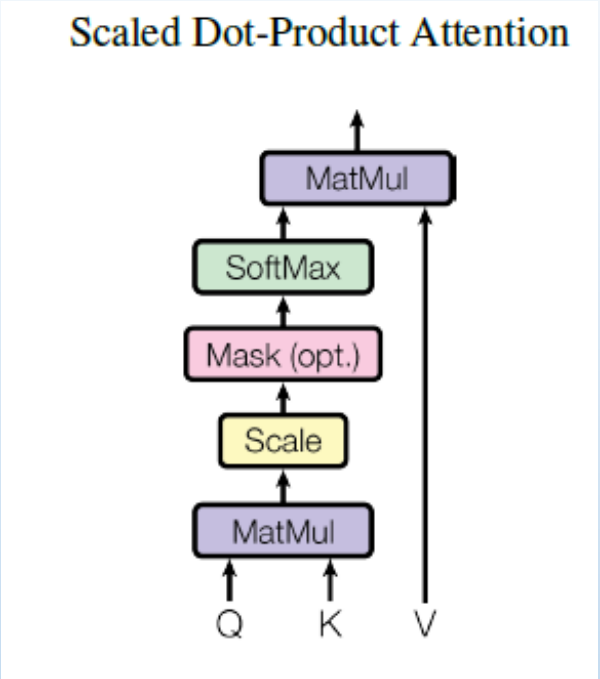
Ce sont des abstractions qui sont utiles pour calculer et penser à l'attention



Auto Attention

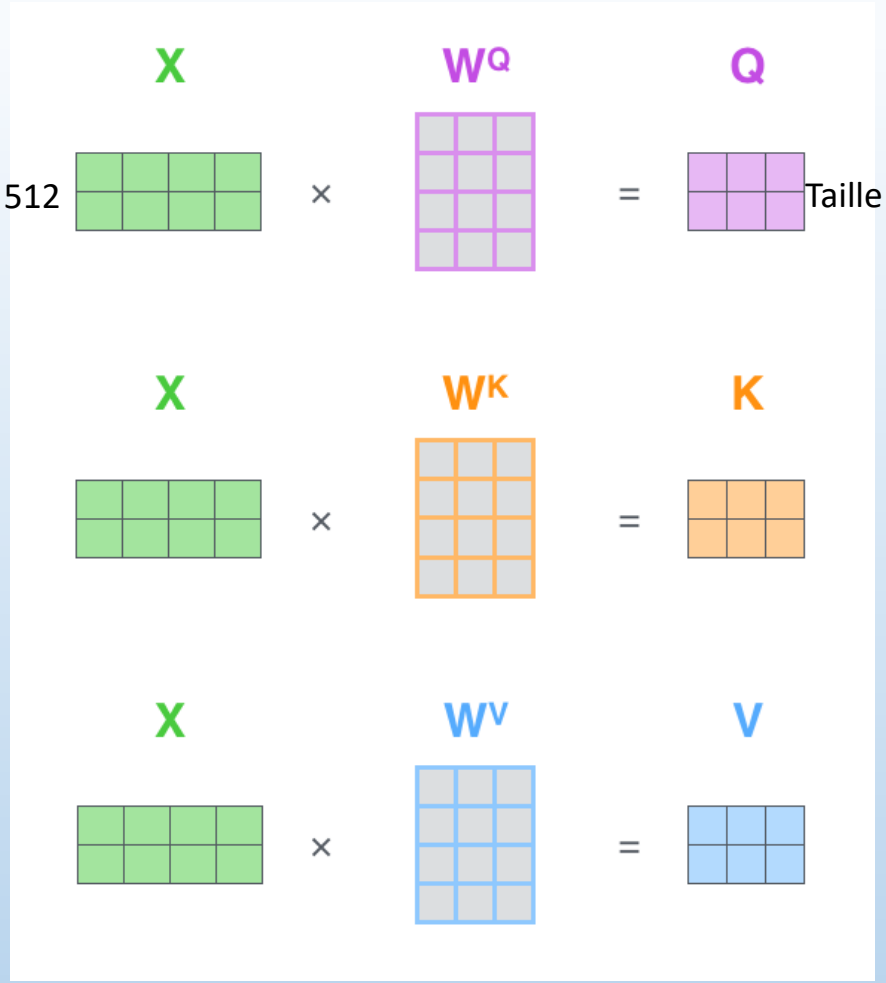
*Convertir **les jetons** (qui peuvent être un mot, une phrase ou un autre regroupement de texte) en **vecteurs** qui représentent **l'importance du jeton** dans la séquence d'entrée.*

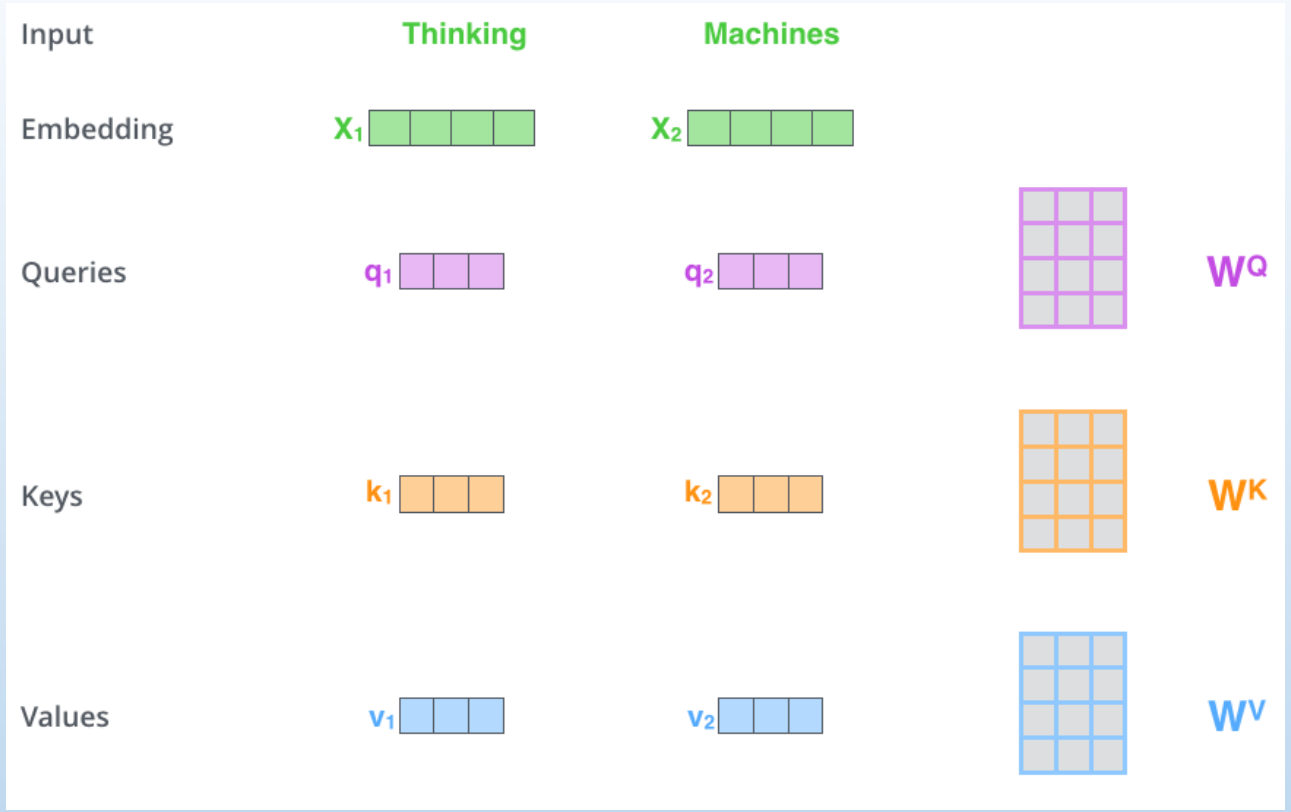
1. *Crée un vecteur de requête, de clé et de valeur pour chaque jeton de la séquence d'entrée.*
2. *Calcule la similarité entre le vecteur de **requête** de la première étape et le vecteur **clé** de tous les autres jetons en prenant le produit ponctuel des deux vecteurs.*
3. *Génère des **poids normalisés** en alimentant la sortie de l'étape 2 dans une **fonction softmax**.*
4. *Génère un **vecteur final**, représentant **l'importance du jeton dans la séquence** en multipliant les poids générés à l'étape 3 par les vecteurs de **valeur** de chaque jeton.*



Auto Attention

1 ère étape





Les vecteurs Q K V sont créés en multipliant l'incorporation (embedding) par trois matrices que nous avons formées au cours du processus d'apprentissage.

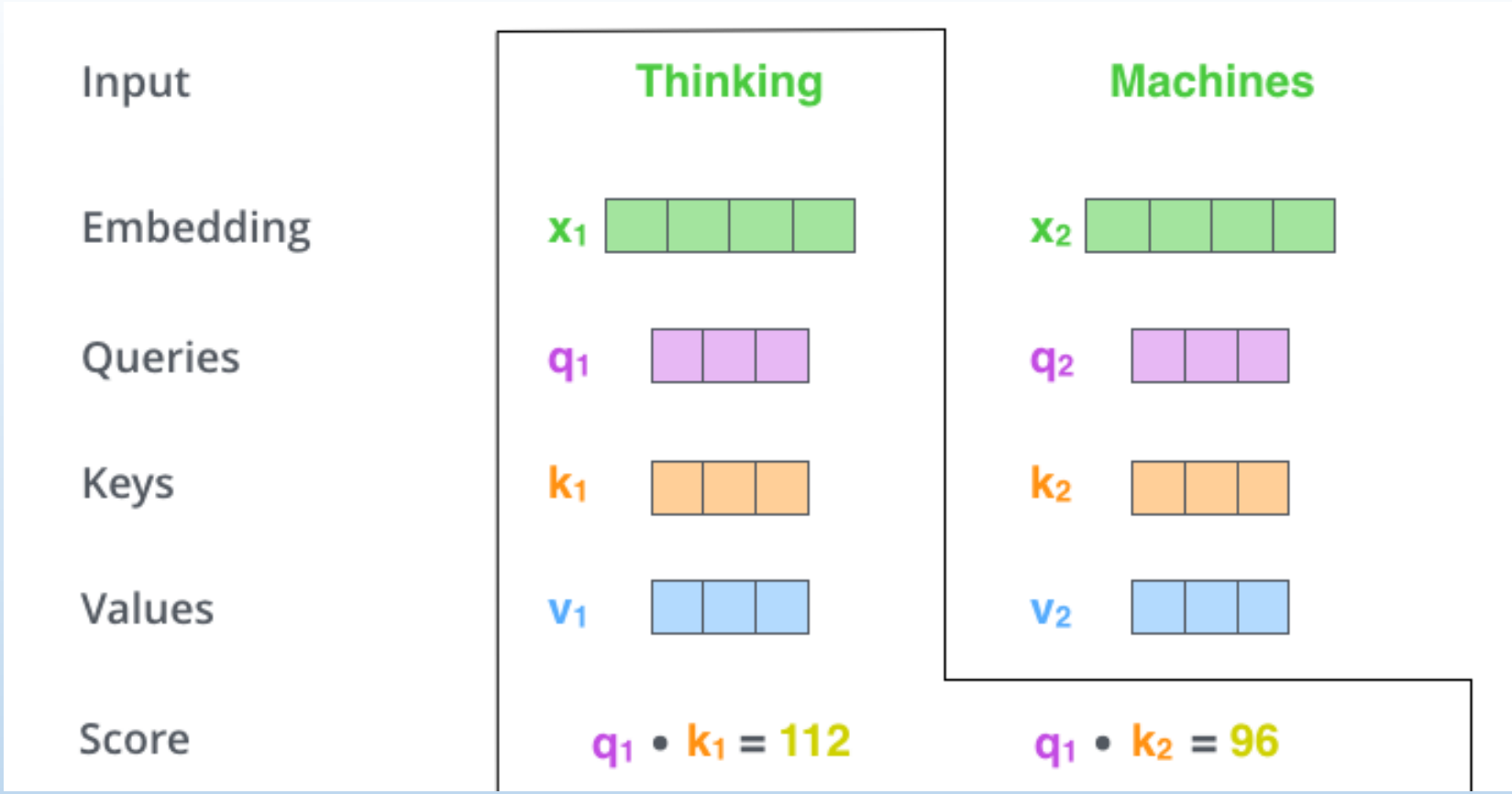
Auto Attention

2ème étape

Calcul des scores

Le score détermine le degré d'attention à accorder aux autres parties de la phrase d'entrée lorsque nous encodons un mot à une certaine position.

Le score est calculé en prenant le produit ponctuel du vecteur de requête avec le vecteur clé du mot respectif que nous notons.



Le premier score est le produit scalaire de **q1 et k1.**

Le deuxième score est le produit scalaire de **q1 et k2**.

Auto Attention

3ème et 4ème étapes

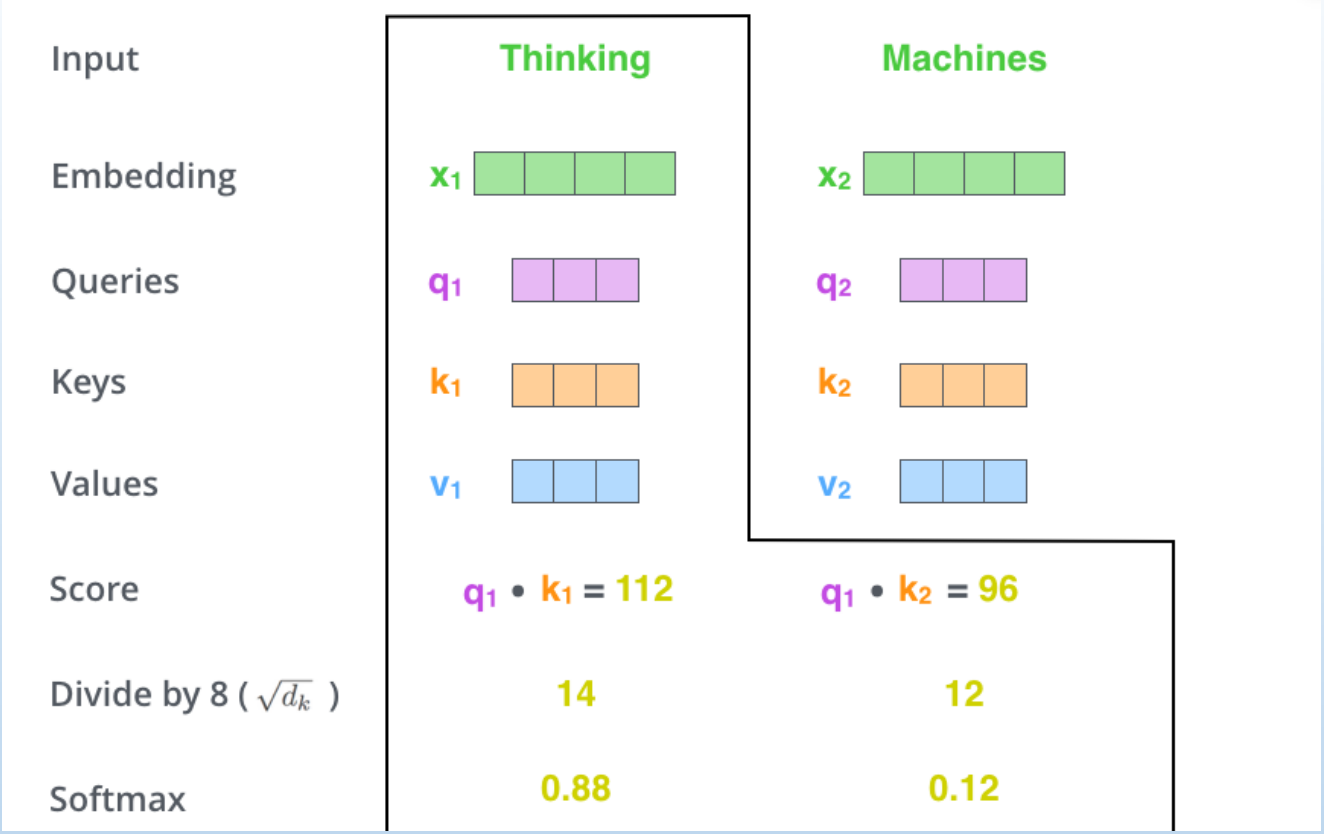
Normalisation pour stabiliser les gradients

Diviser par la racine carrée de la dimension des vecteurs clés utilisés dans l'article – 64.

Donc, ici ***8***

Puis passez le résultat via une opération softmax.

Softmax normalise les scores afin qu'ils soient tous positifs et totalisent 1.



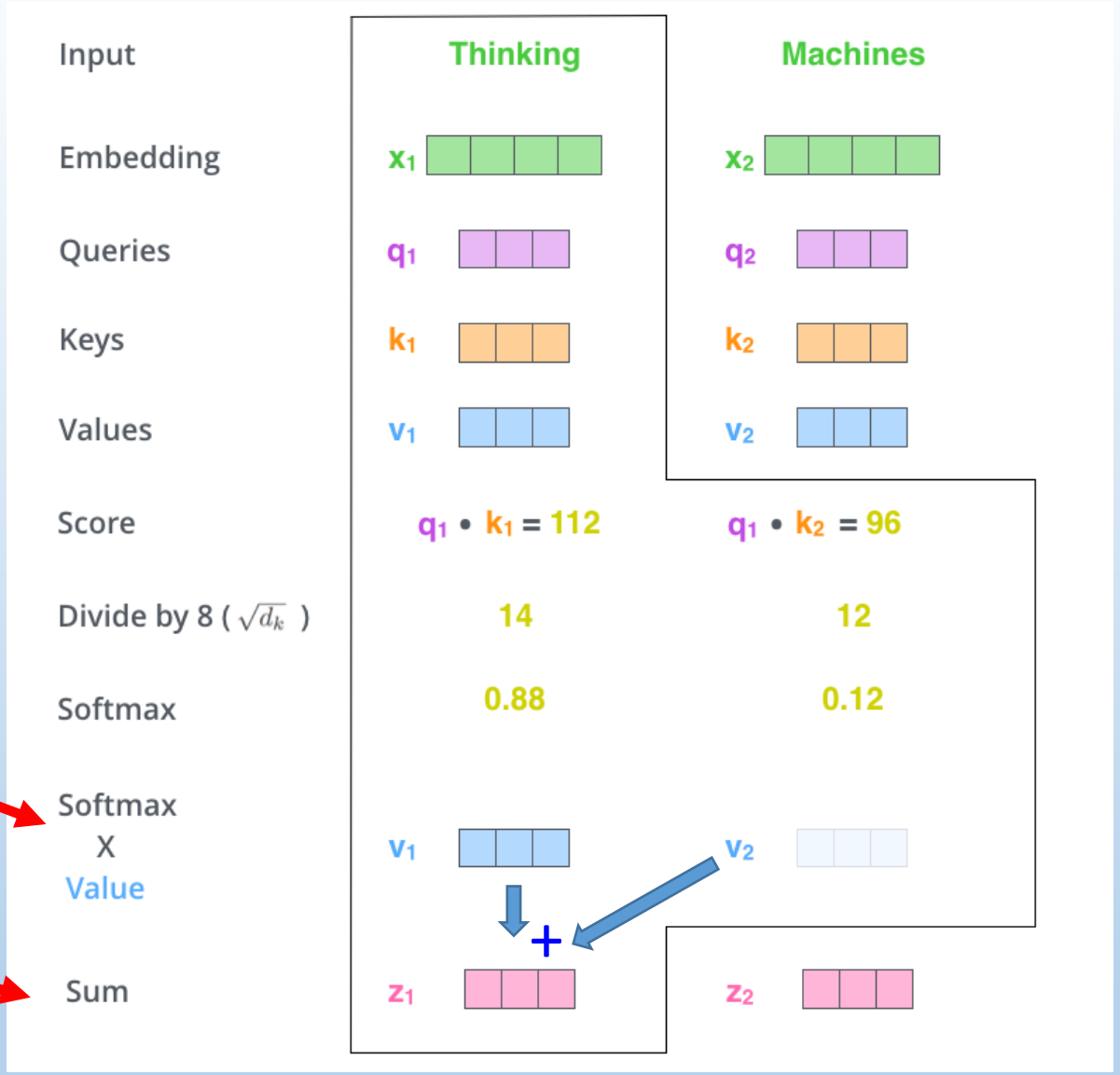
Auto Attention

5ème et 6ème étapes

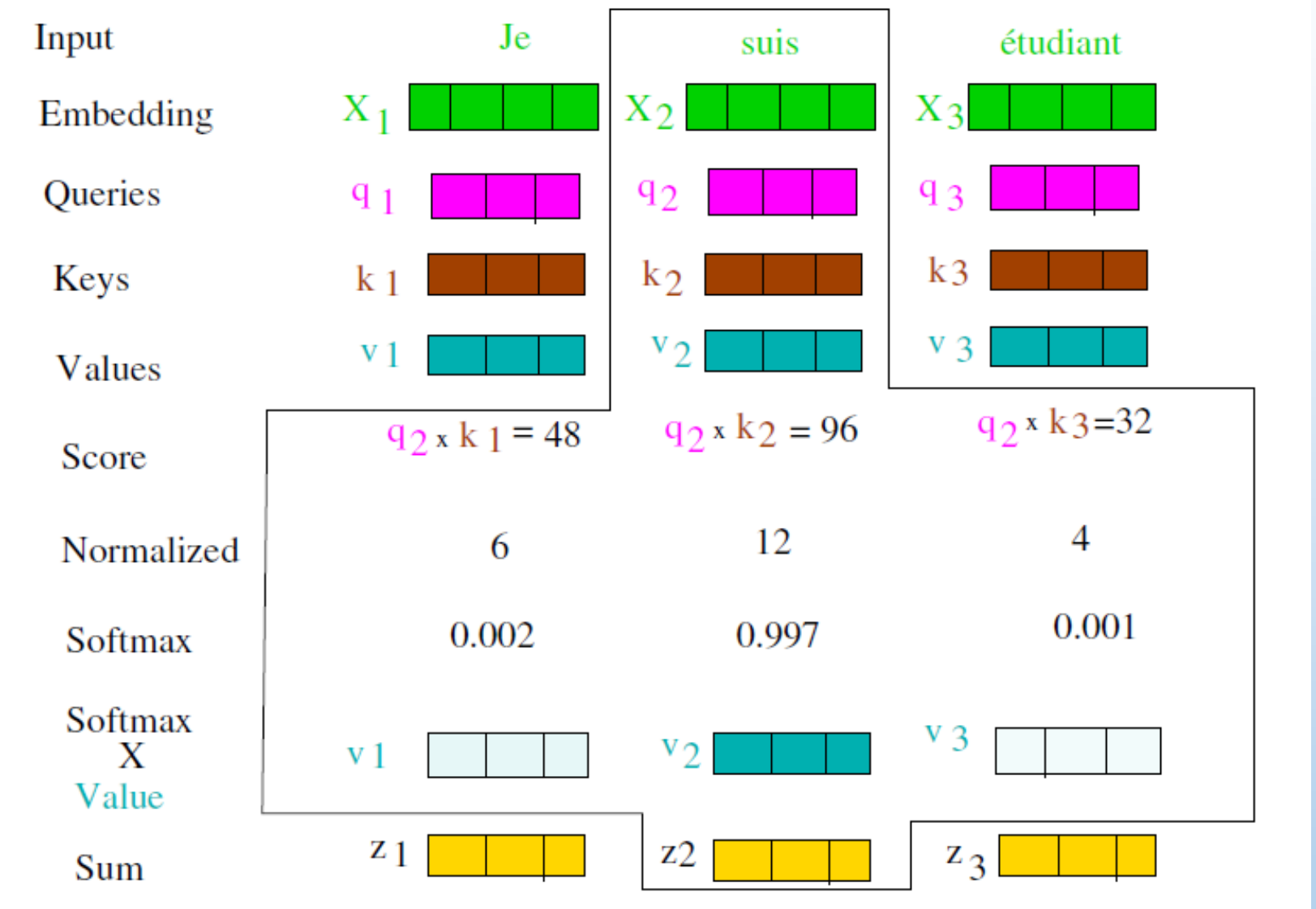
*La **cinquième étape** consiste à multiplier chaque vecteur de valeur par le score softmax*

L'intuition ici est de garder intactes les valeurs du ou des mots sur lesquels nous voulons nous concentrer, et de noyer les mots non pertinents (en les multipliant par de minuscules nombres comme 0,001, par exemple).

*La **sixième étape** consiste à additionner les vecteurs de valeurs pondérées. Cela produit la sortie de la couche d'auto-attention à cette position (pour le premier mot).*



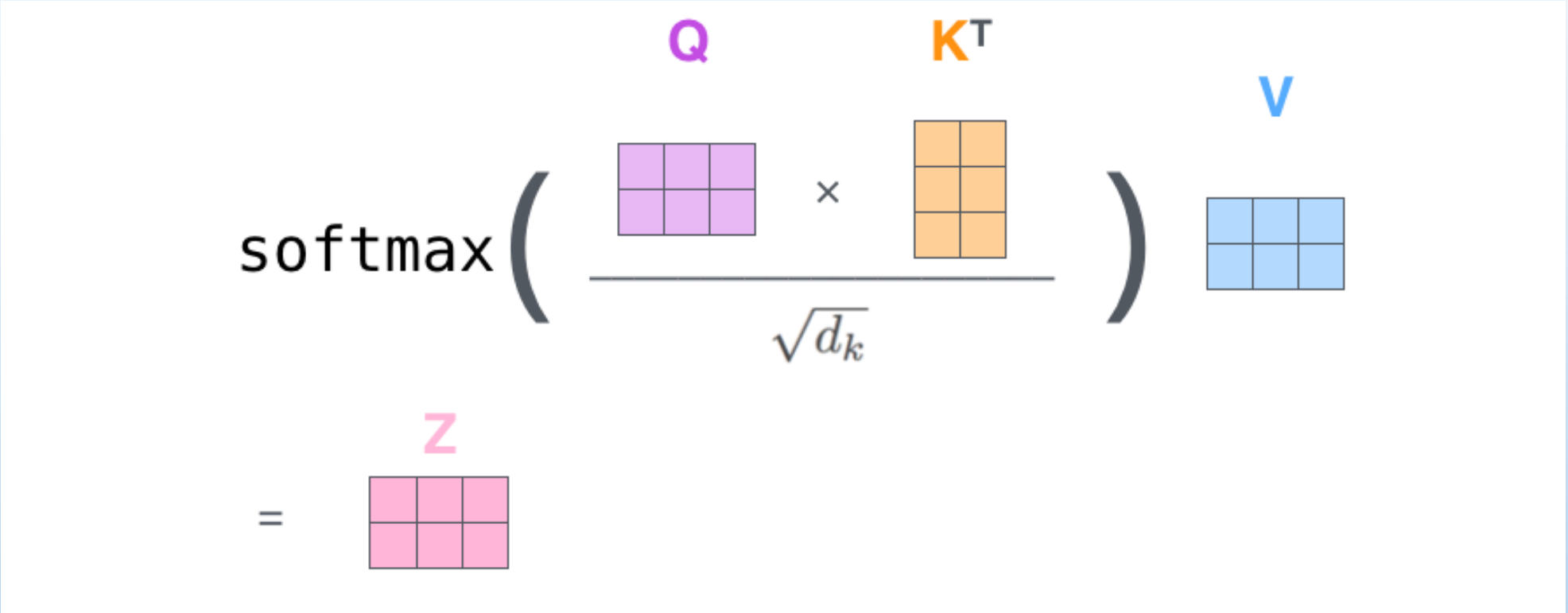
Auto Attention



La somme pondérés des « values » est la sortie "z" de la couche d'attention

Auto Attention

Le calcul de l'auto-attention sous forme matricielle



Auto Attention Multi Têtes
Multi Head Self Attention

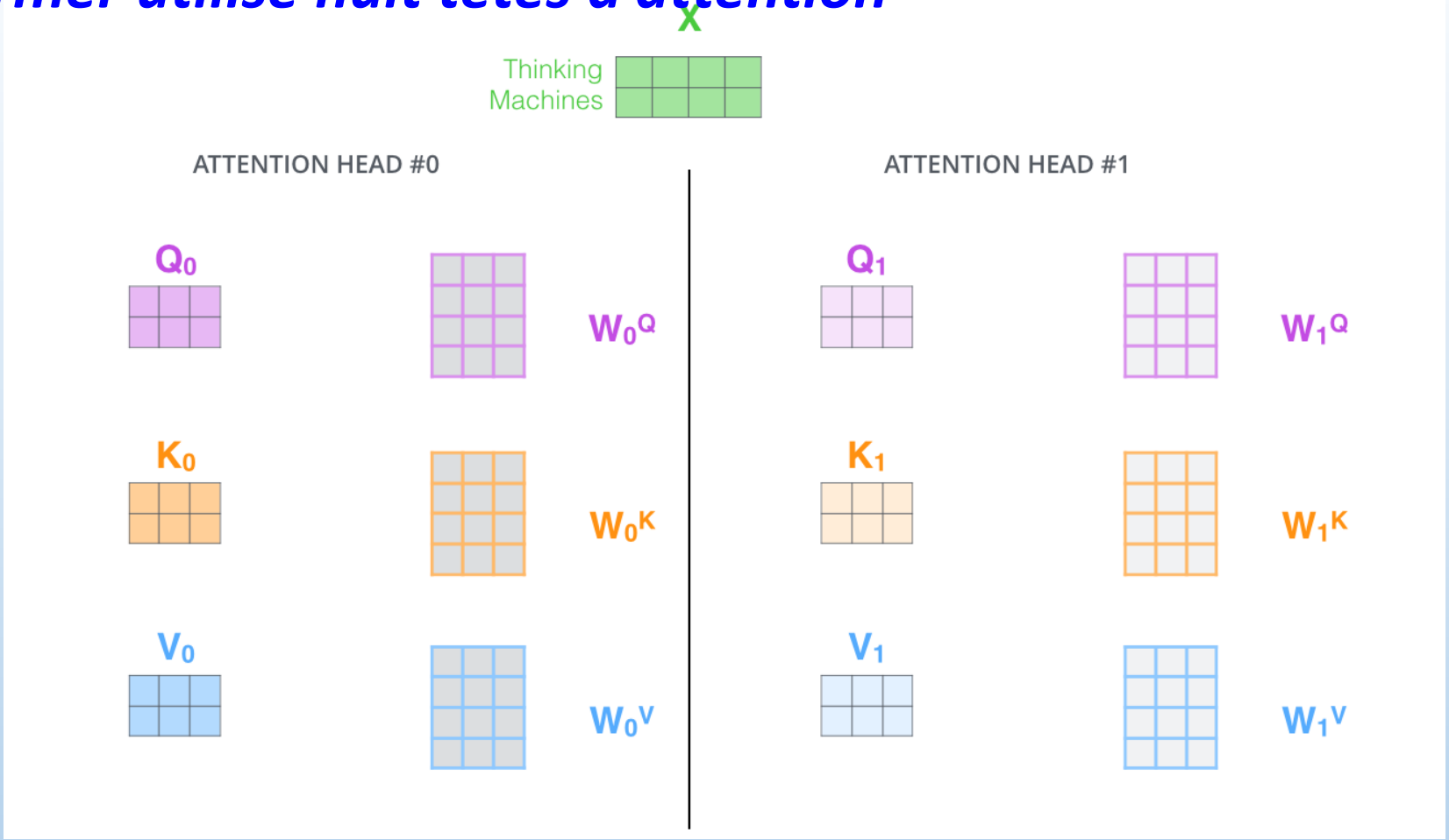
Le mécanisme d'attention « multi-tête » utilisé par GPT est une évolution de l'auto-attention.

Plutôt que d'effectuer les étapes 1 à 4 une fois, le modèle itére en parallèle ce mécanisme plusieurs fois, générant à chaque fois une nouvelle projection linéaire des vecteurs requête, clé et valeur.

En élargissant ainsi l'attention à soi, le modèle est capable de saisir des sous-significations et des relations plus complexes au sein des données d'entrée.

Auto Attention Multi têtes

Transformer utilise huit têtes d'attention

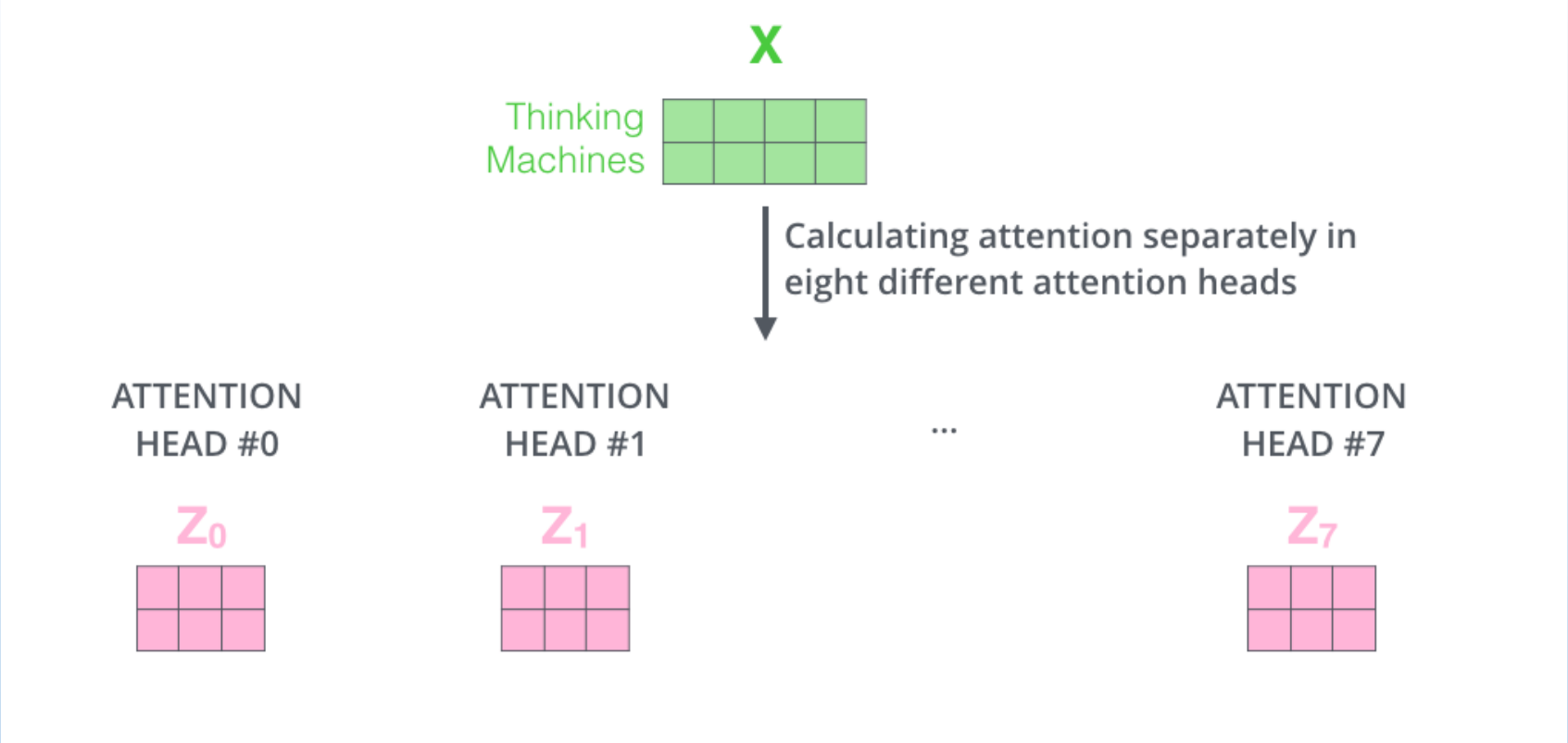


8 ensembles de matrices

Initialisées de manière aléatoire.

Ensuite, après l'entraînement, chaque ensemble est utilisé pour projeter les intégrations d'entrée dans un sous-espace de représentation différent.

Auto Attention Multi têtes



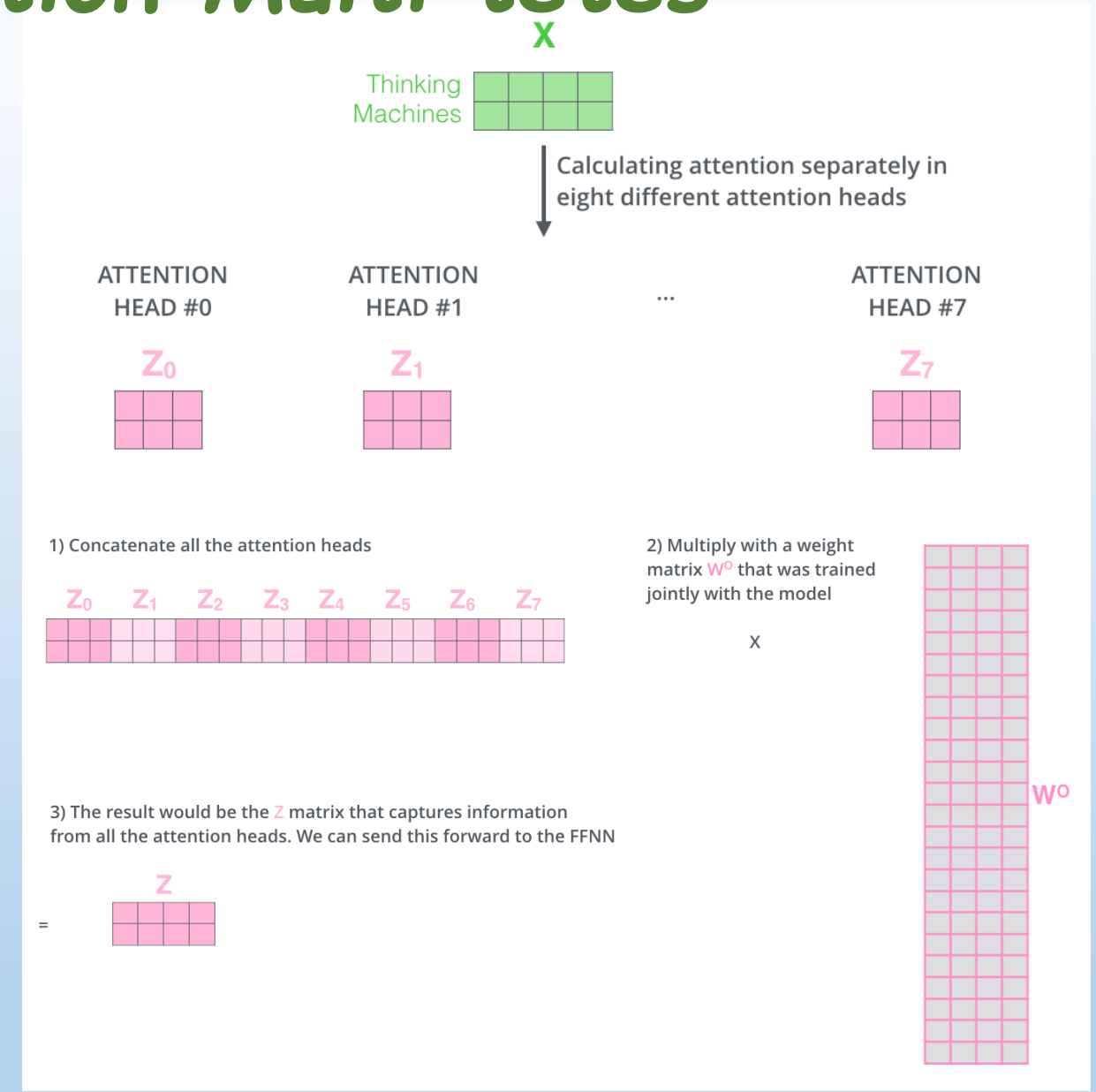
Nous faisons le même calcul d'auto-attention que précédemment, huit fois différentes avec des matrices de poids différentes.
Nous obtenons huit matrices Z différentes

Auto Attention Multi têtes

La couche supérieure attend une seule matrice
Comment pouvons-nous faire cela?

Nous concaténons les matrices puis les multiplions par une matrice de poids supplémentaire WO apprenable.

Nous obtenons une seule matrice Z .



Auto Attention Multi têtes

1) This is our input sentence*

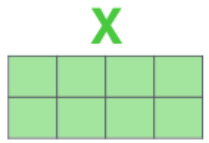
2) We embed each word*

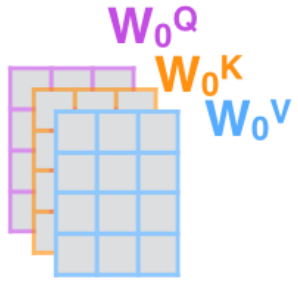
3) Split into 8 heads. We multiply X or R with weight matrices

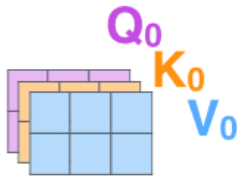
4) Calculate attention using the resulting Q/K/V matrices

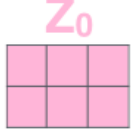
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

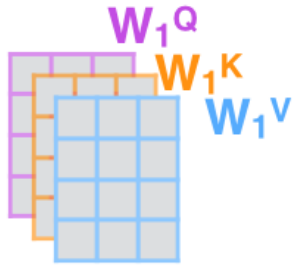
Thinking Machines

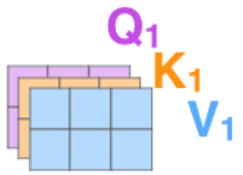


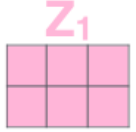


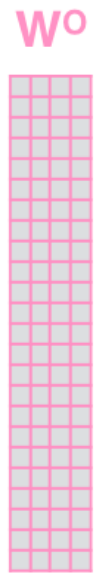


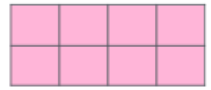




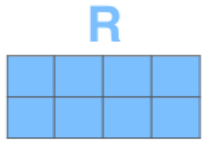


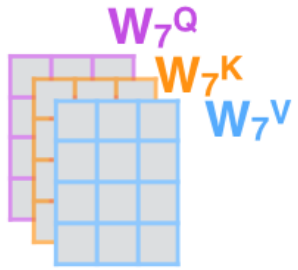


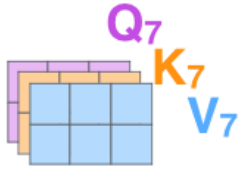


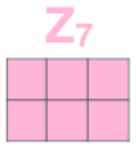


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

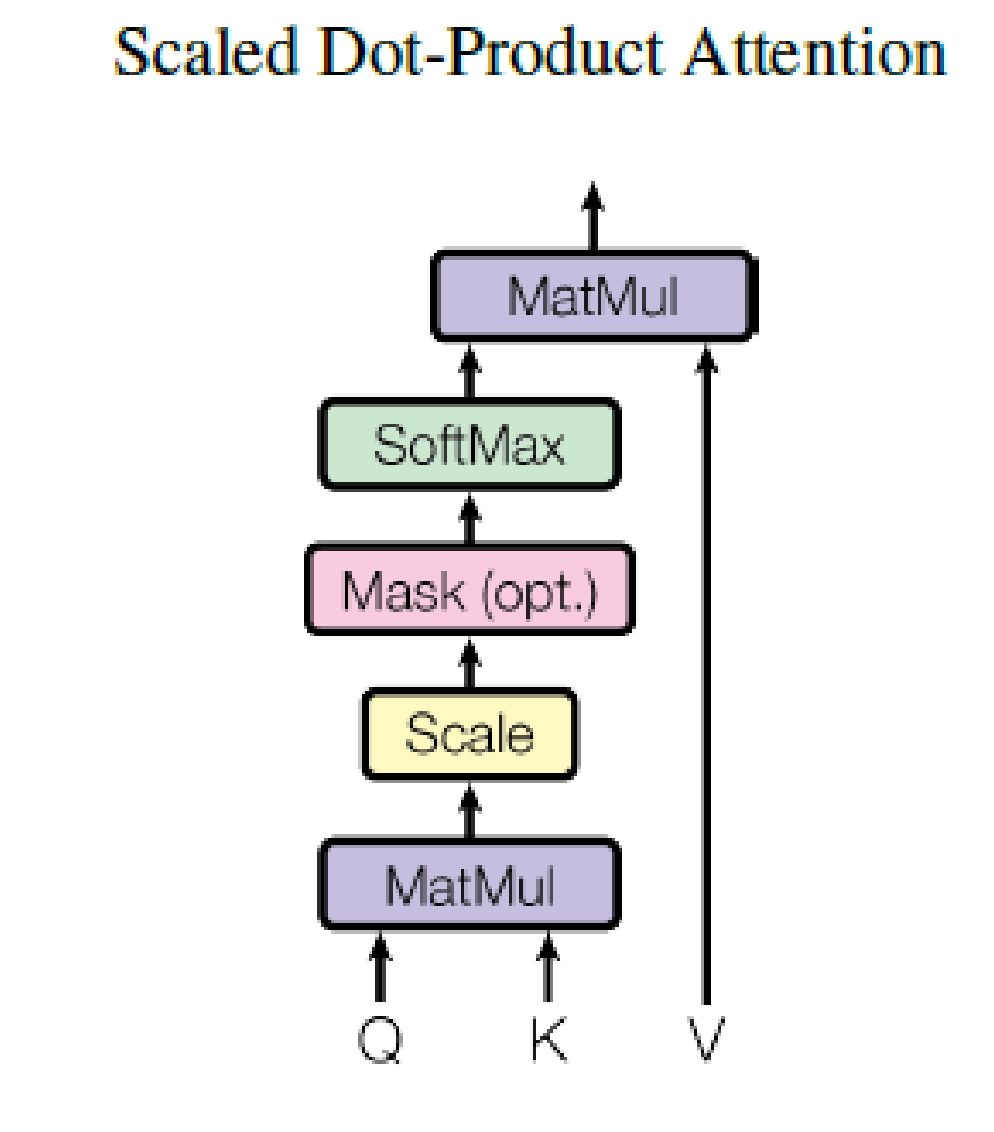


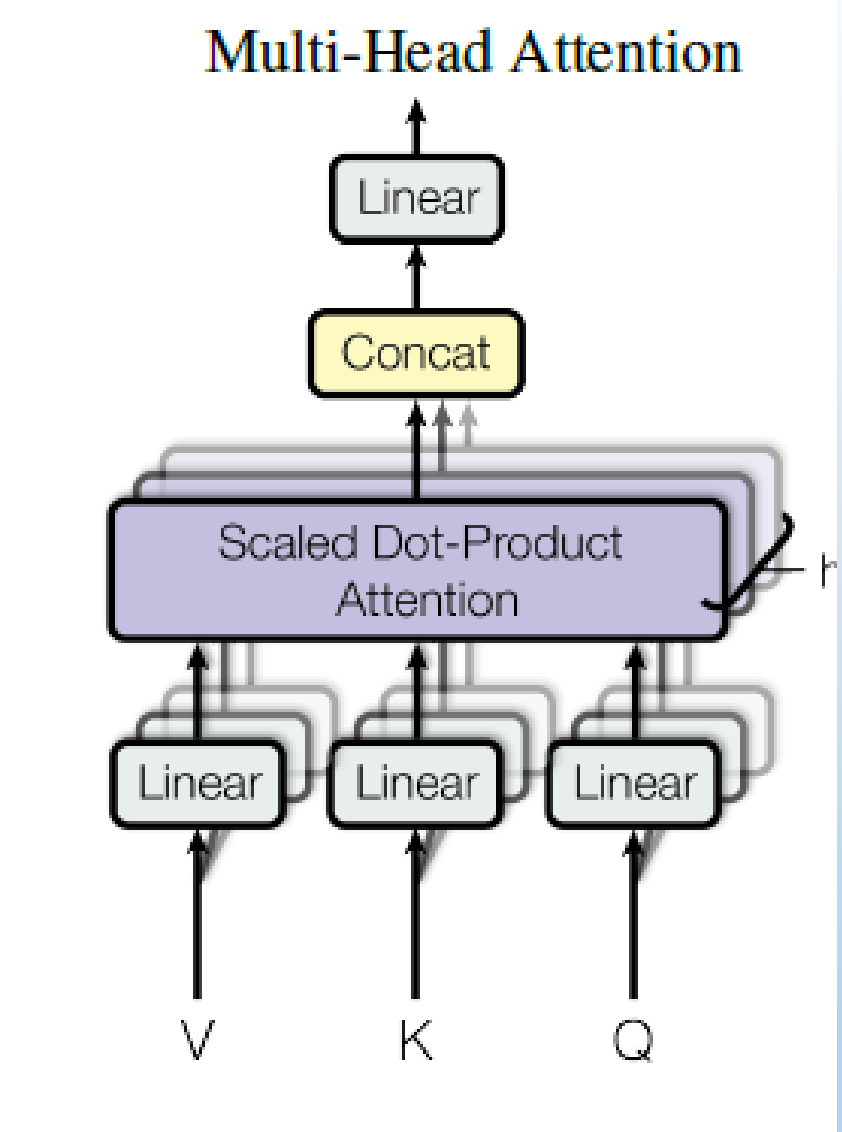




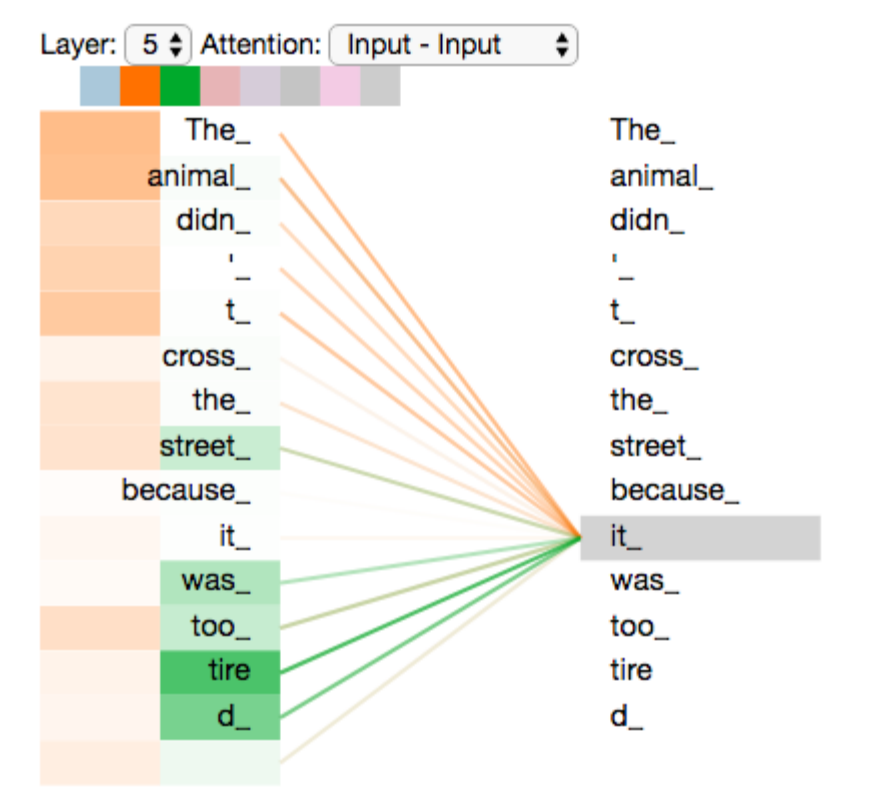


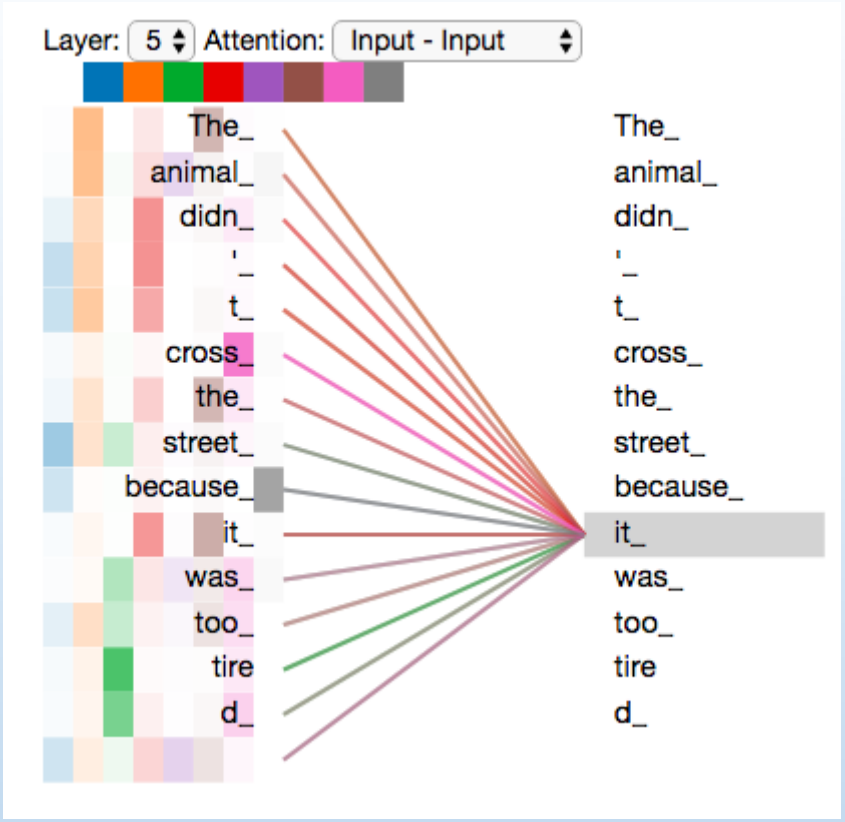
Auto Attention Multi têtes





Auto Attention Multi têtes





*Lorsque nous encodons le mot **« it »**, une tête d'attention se concentre le plus sur **« l'animal »**, tandis qu'une autre se concentre sur **« fatigué »***

Si nous ajoutons toutes les têtes d'attention à l'image, les choses peuvent être plus difficiles à interpréter:

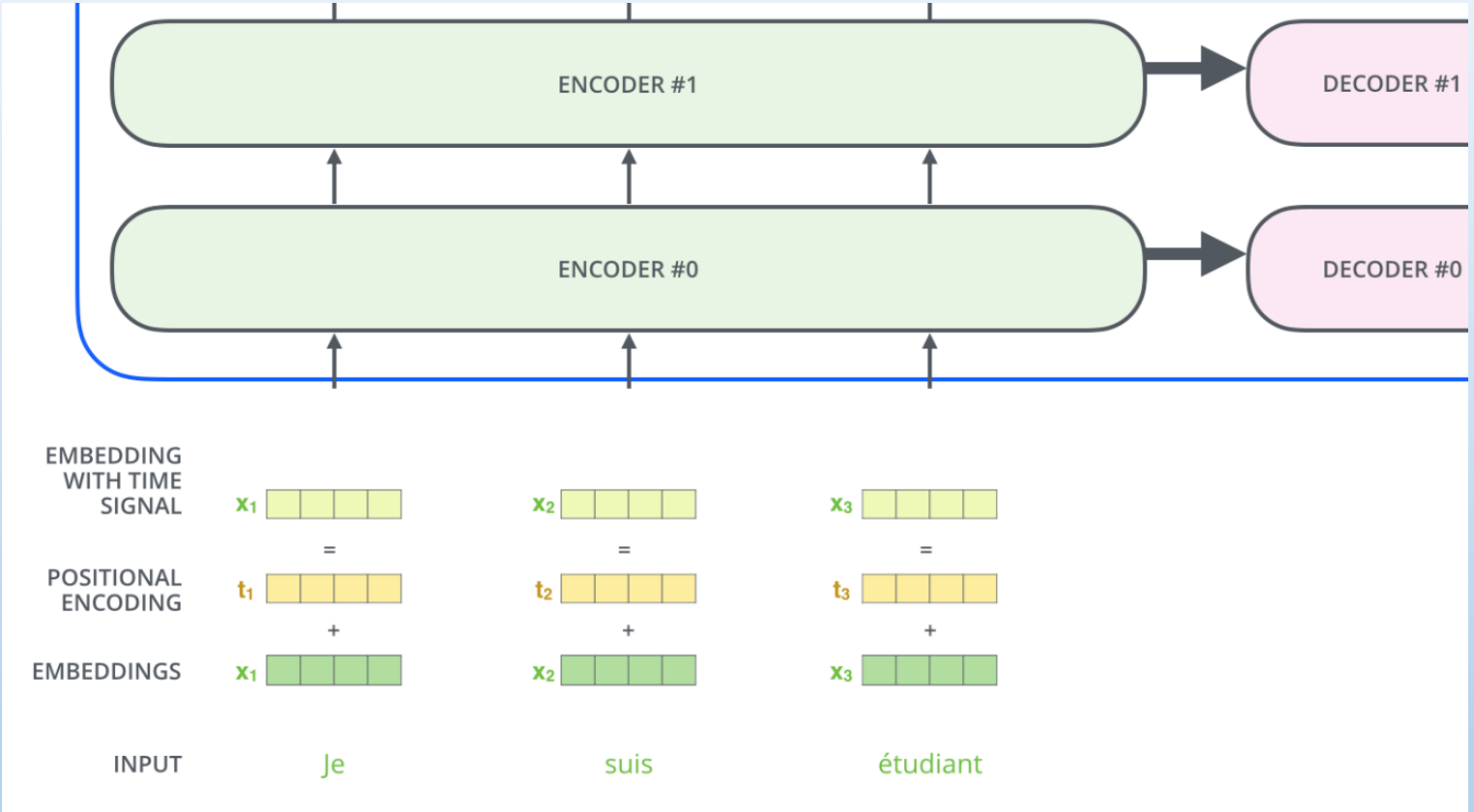
Ordre des mots dans la séquence

Une chose qui manque au modèle tel que nous l'avons décrit jusqu'à présent est un moyen de rendre compte de l'ordre des mots dans la séquence d'entrée.

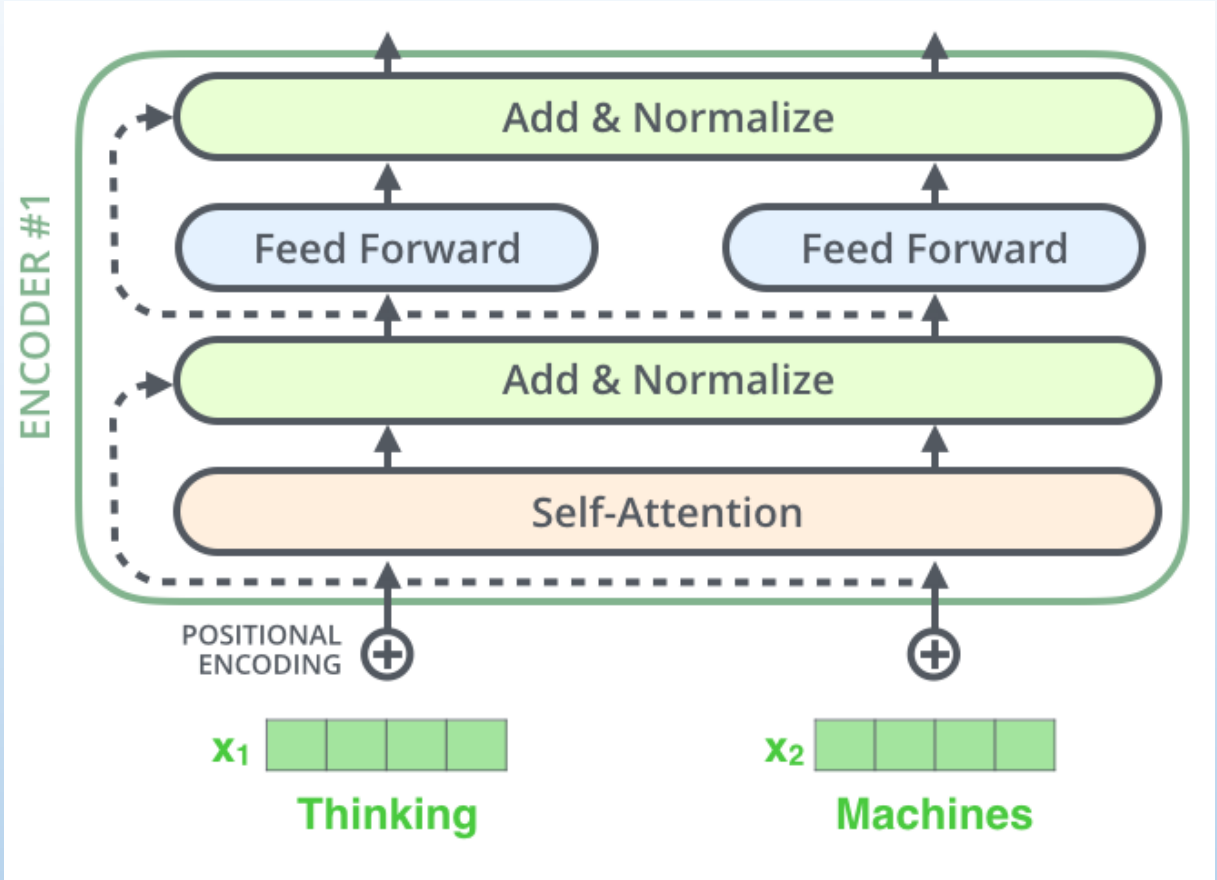
Le transformer ajoute un vecteur à chaque intégration d'entrée.

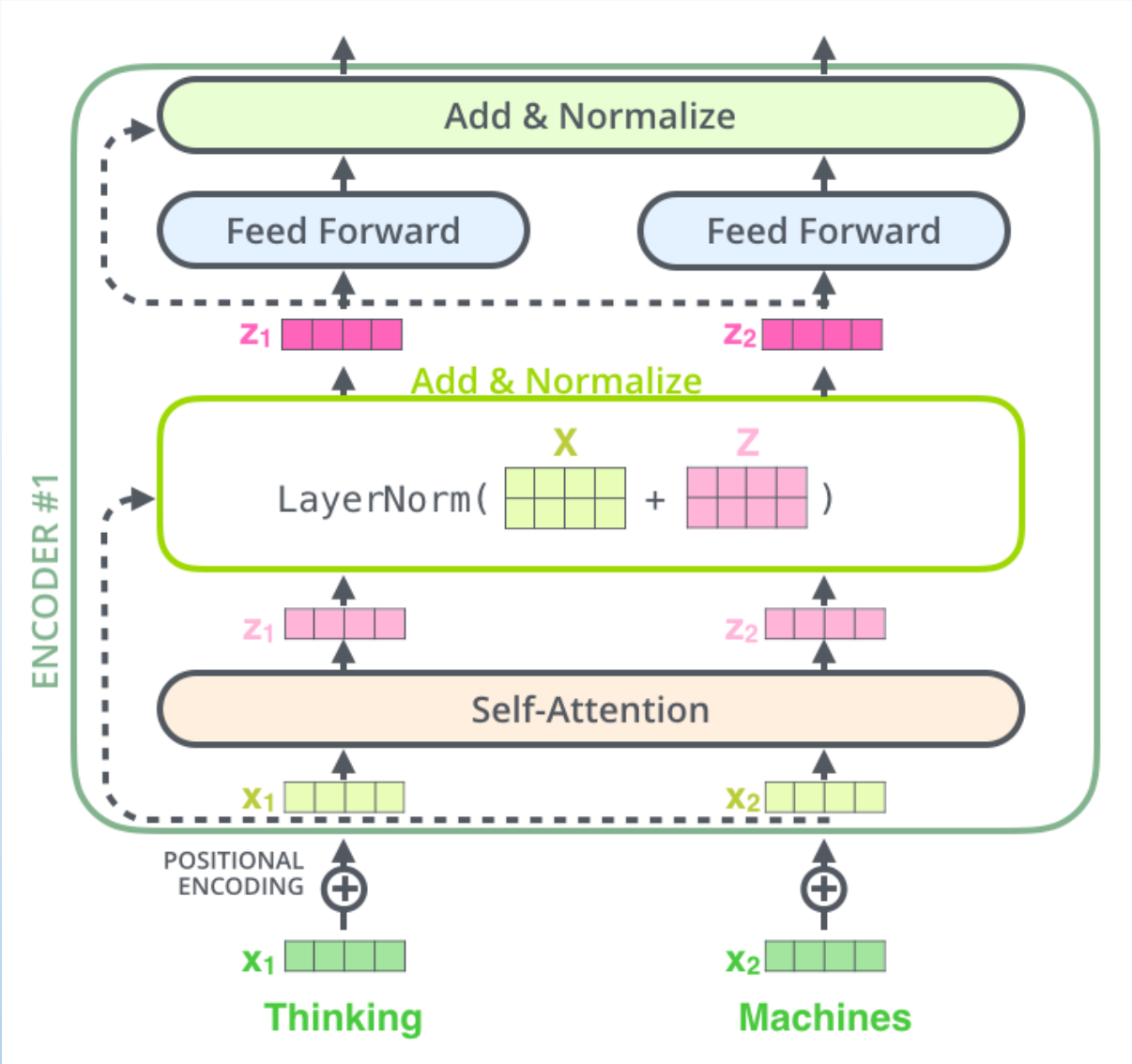
.

Ces vecteurs suivent un modèle spécifique que le modèle apprend, ce qui l'aide à déterminer la position de chaque mot ou la distance entre les différents mots de la séquence



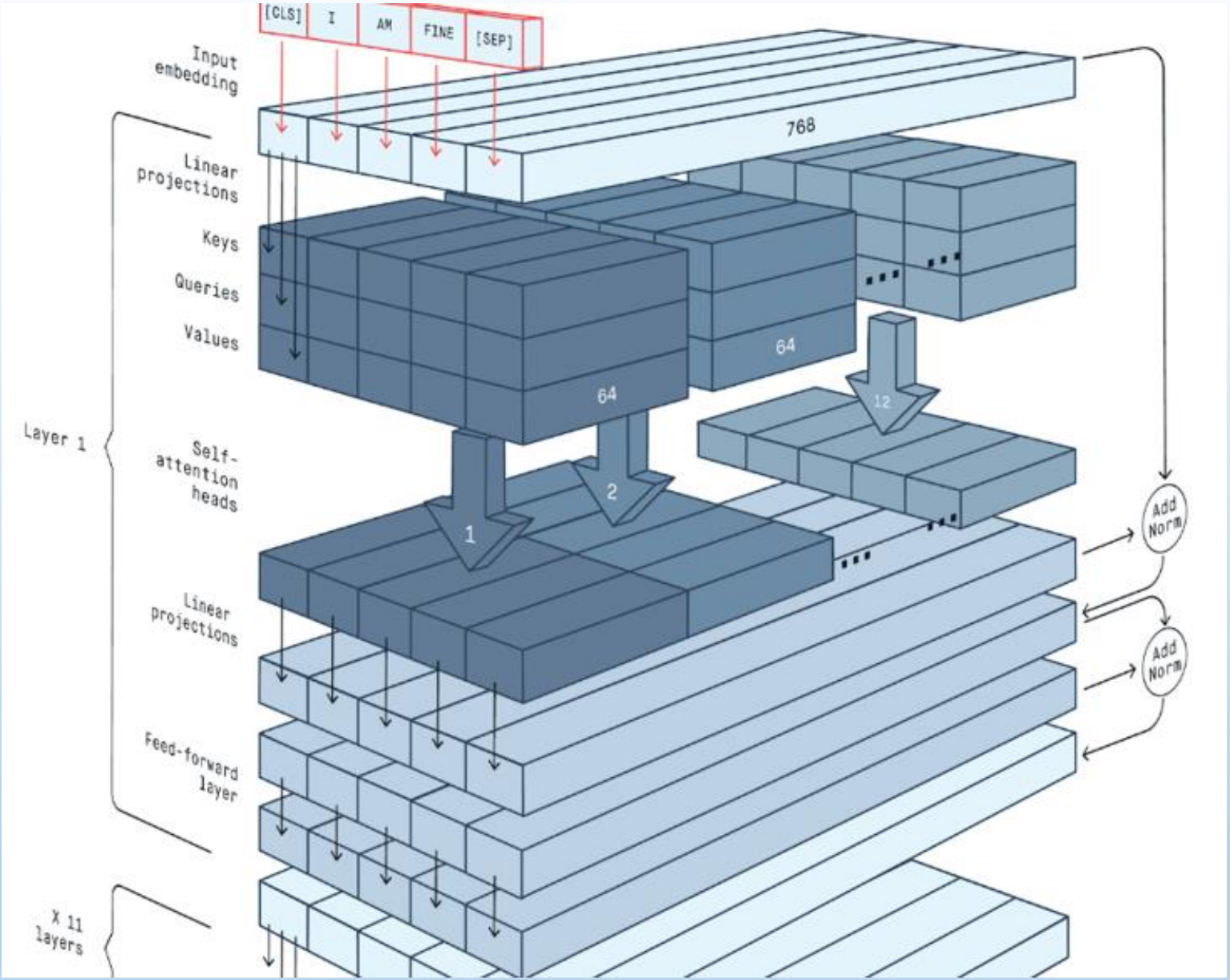
Encodeur



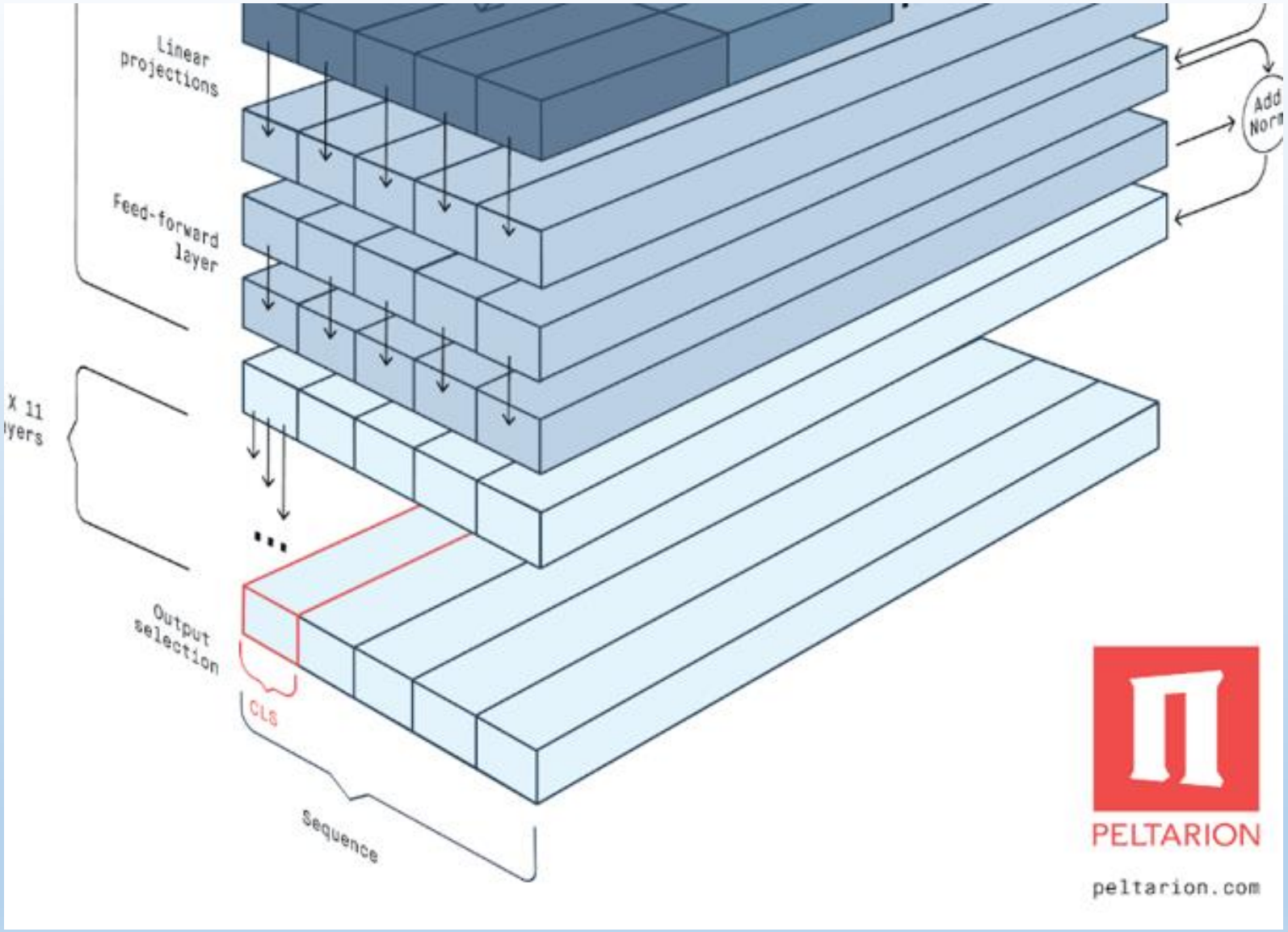


Chaque sous-couche (auto-attention, ffnn) dans chaque codeur a une connexion résiduelle autour d'elle, et est suivie d'une étape de normalisation de couche ***Résidus***

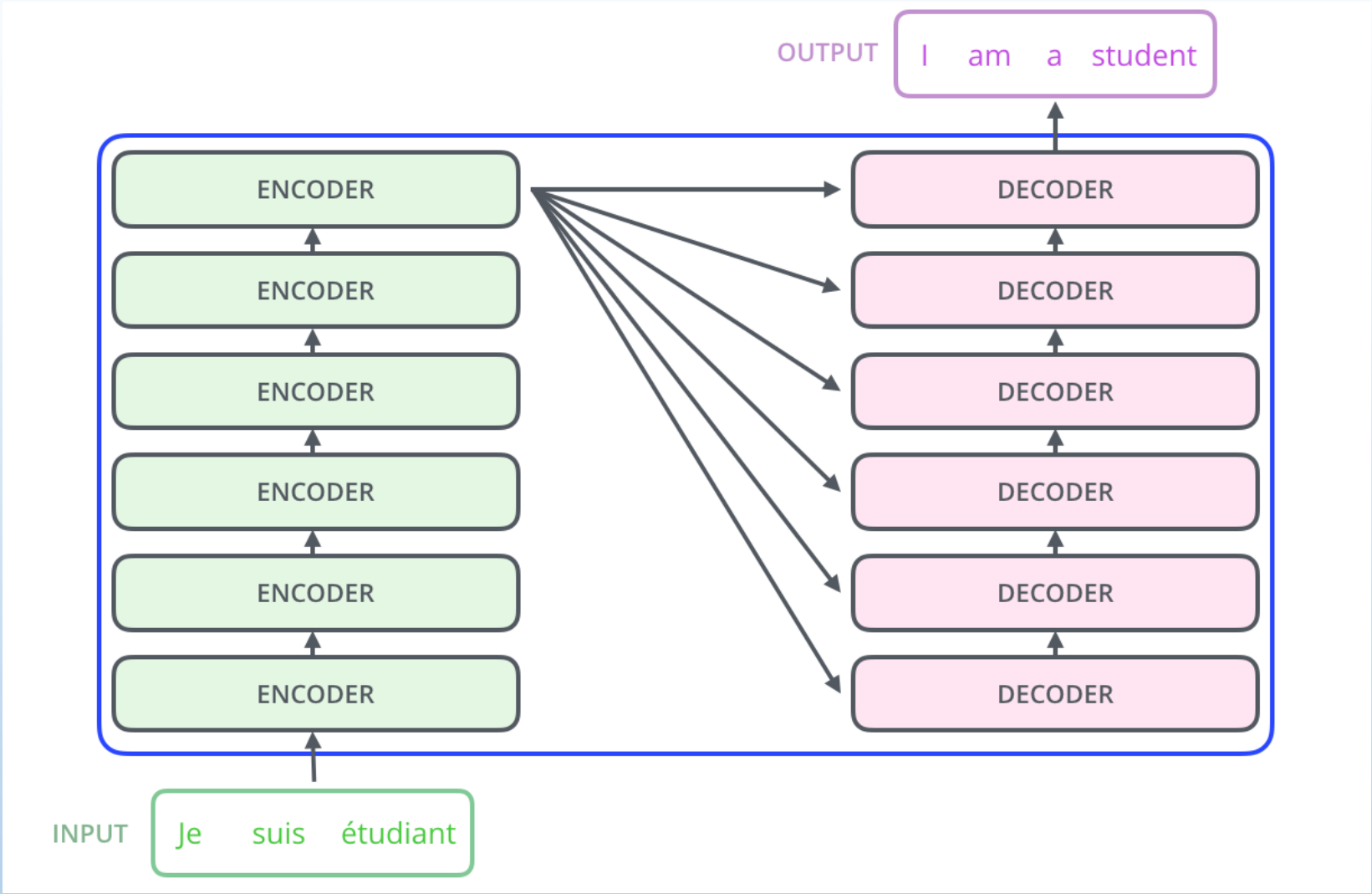
Transformer Encodeur



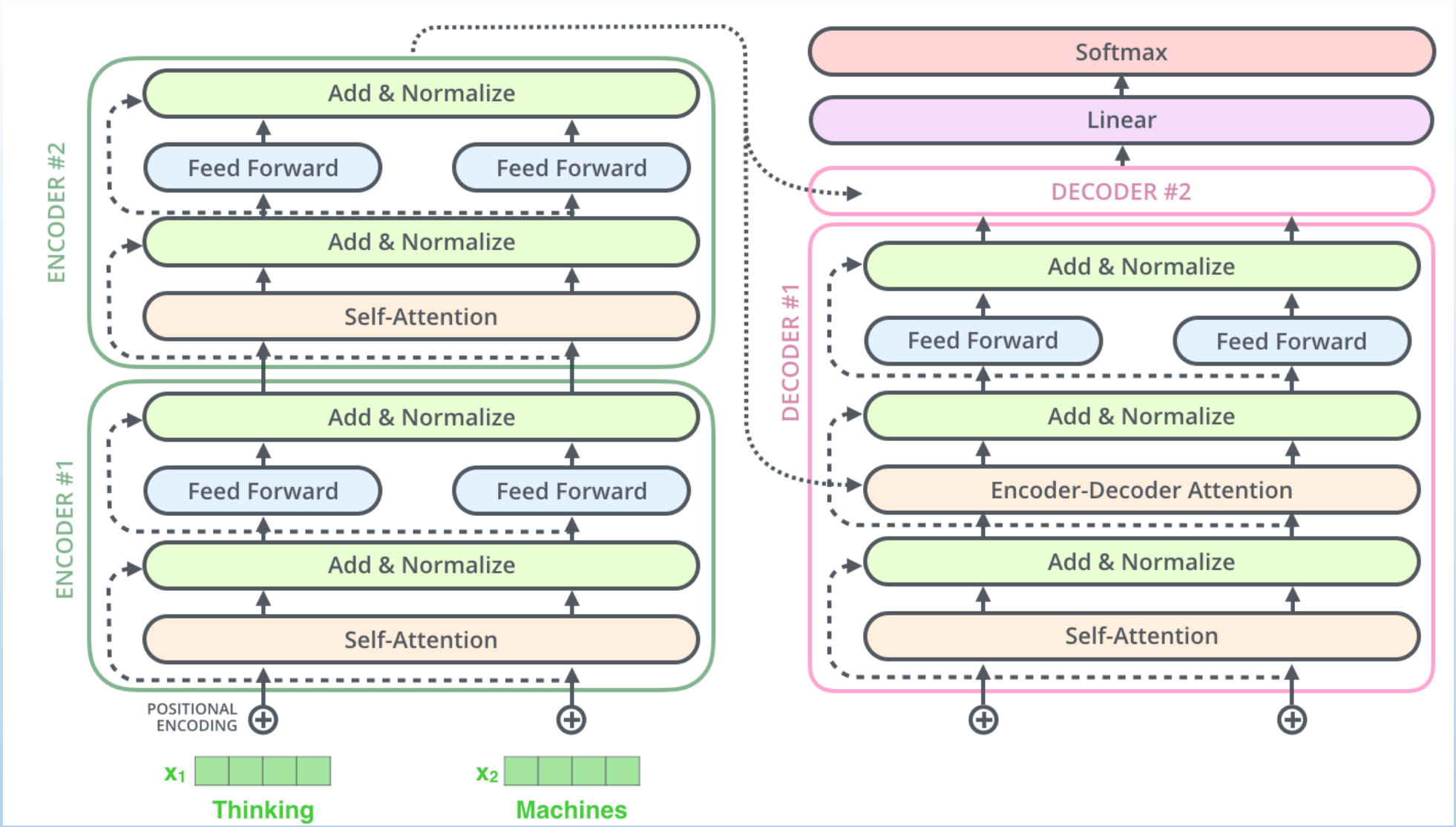
Transformer Encodeur



Encodeur Décodeur

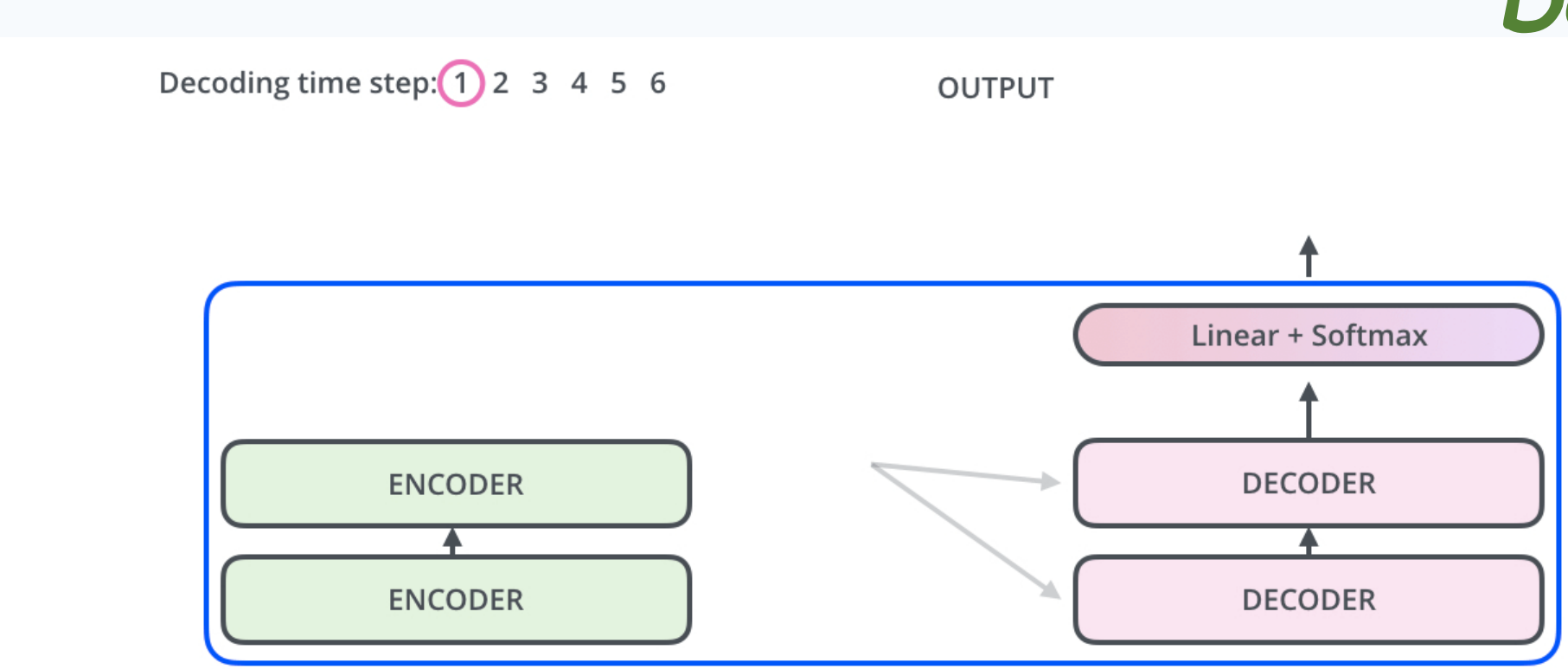


Décodeur



Le décodeur a une couche d'attention supplémentaire pour tenir compte de la sortie de l'encoder.

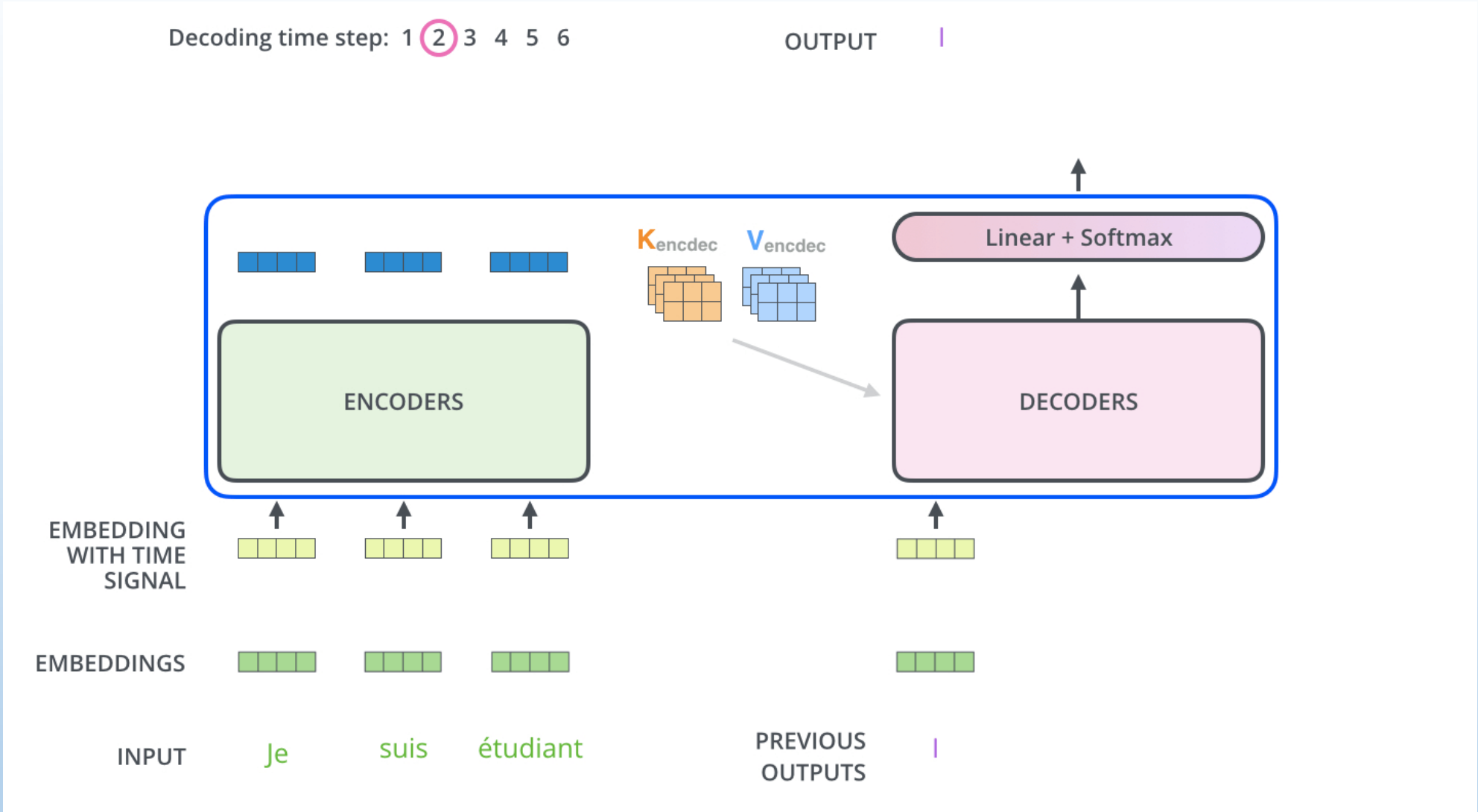
Décodeur



La sortie du codeur supérieur est ensuite transformée en un ensemble de vecteurs d'attention K et V.

Ceux-ci sont utilisés par chaque décodeur dans sa couche « attention codeur-décodeur » qui aide le décodeur à se concentrer sur les endroits appropriés dans la séquence d'entrée

Décodeur


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
K_{encdec}
V_{encdec}
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Décodeur

Le décodeur utilise la sortie mais aussi les vecteurs “keys” et “values” finaux de l’encoder pour calculer les probabilités conditionnelles de mots en sorties.
La couche « Encodeur-Décodeur Attention » fonctionne comme l’auto-attention multi-tête, avec les “keys” et “values” finaux de l’encoder et elle crée ses Queries à partir de la couche située en dessous

Le décodeur va prédire la probabilité conditionnelle des mots cibles les uns après les autres en fonction des sorties de l’encoder et des mots cibles ***précédents.***

Pour cela, il masque les mots cibles qui ***suivent*** *le mot à prévoir en assignant une probabilité nulle aux coordonnées correspondantes à ceux-ci dans son vecteur “value*

Décodeur

Exemple :

Notre modèle connait 10 000 mots anglais uniques appris.

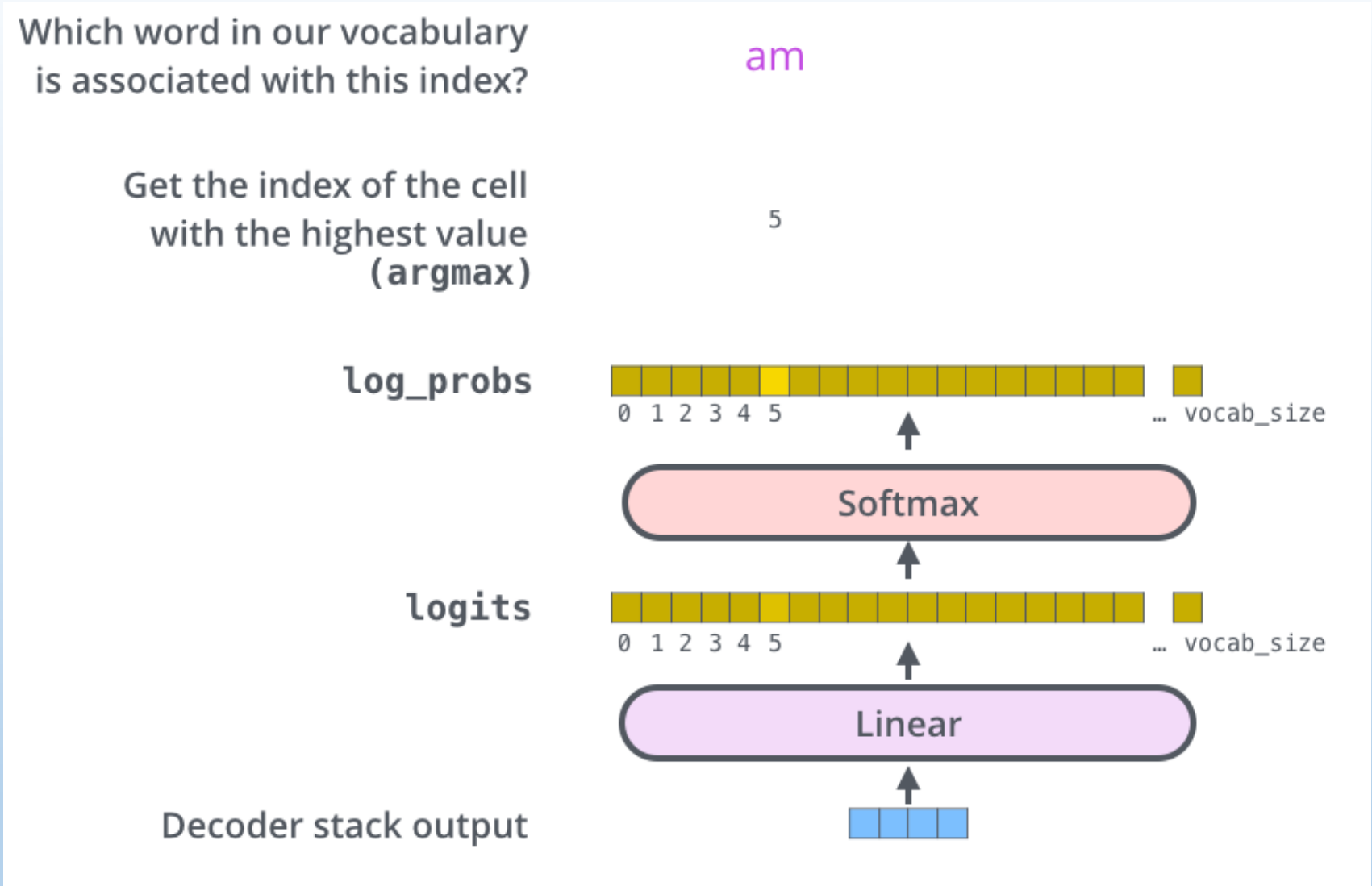
.

Le vecteur logits 10 000 cellules

Chaque cellule correspondant au score d'un mot unique.

.

La couche finale linéaire et softmax



Entrainement du modèle

Pendant l'entraînement, un modèle non entraîné passerait exactement par les mêmes phases

Nous l'entraînons sur un jeu de données d'apprentissage étiqueté et nous pouvons donc comparer sa sortie avec la sortie correcte réelle.

Pour visualiser cela, supposons que notre vocabulaire de sortie ne contient que six mots

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

One-hot encoding of the word "am"

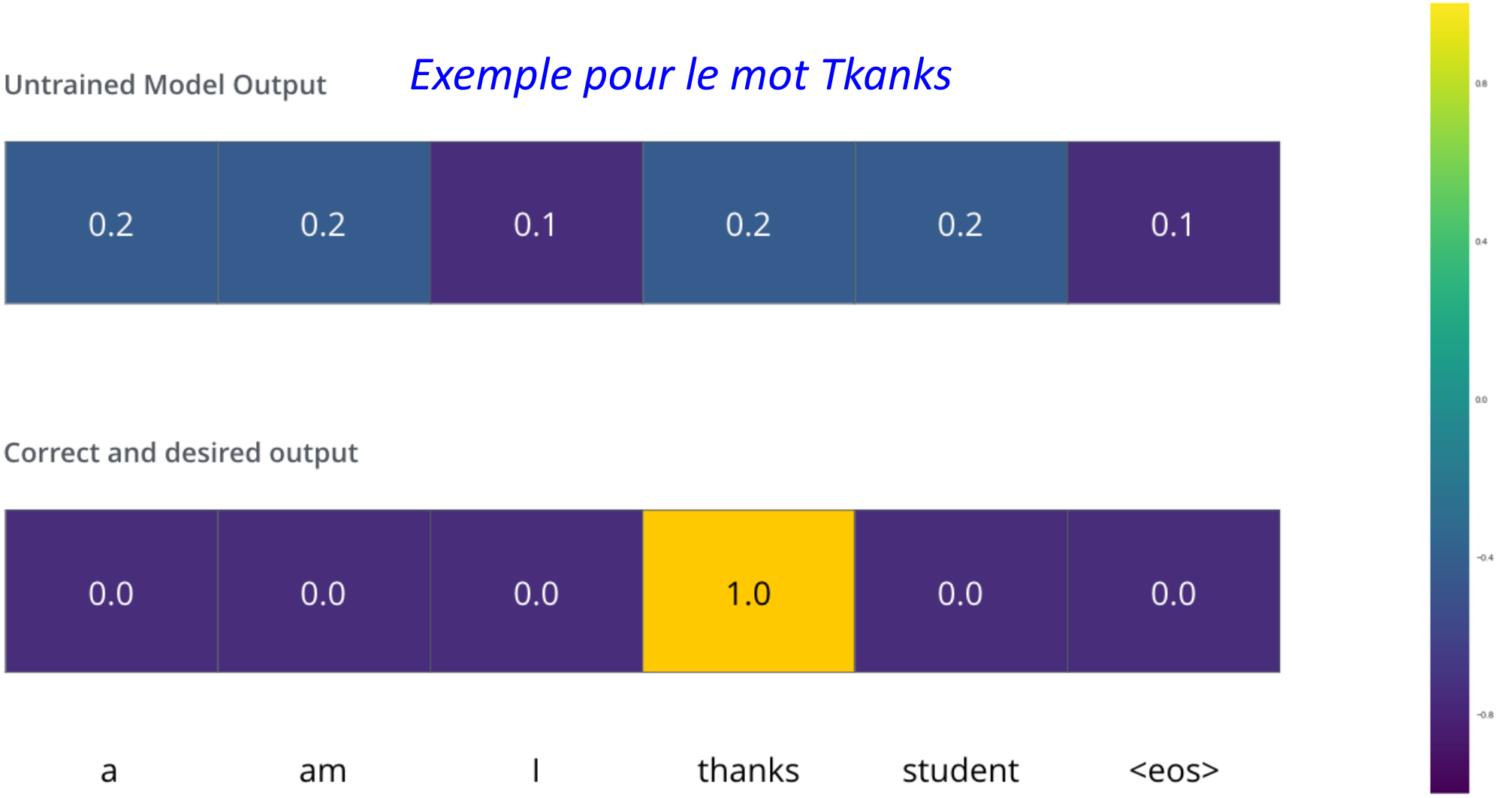
0.0	1.0	0.0	0.0	0.0	0.0

One hot encoding

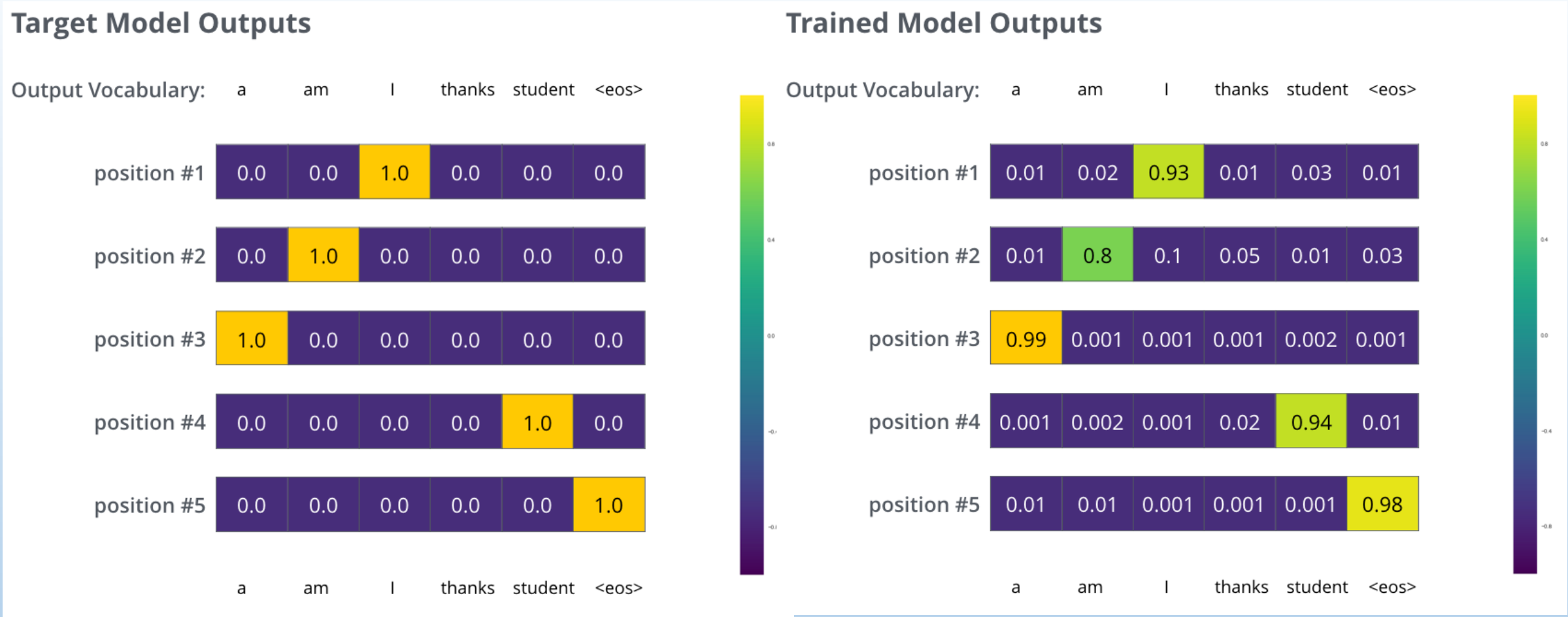
Vecteur de la taille du vocabulaire

Entrainement du modèle

Exemple pour le mot Tkanks



Entrainement du modèle



Après avoir entraîné le modèle pendant suffisamment de temps sur un ensemble de données suffisamment volumineux, nous espérons que les distributions de probabilité produites ressembleront à ceci:

Modèles

Depuis 2018 des modèles ont des poids pré-entrainés sur de grandes bases de données (livres, wikipedia, etc...)

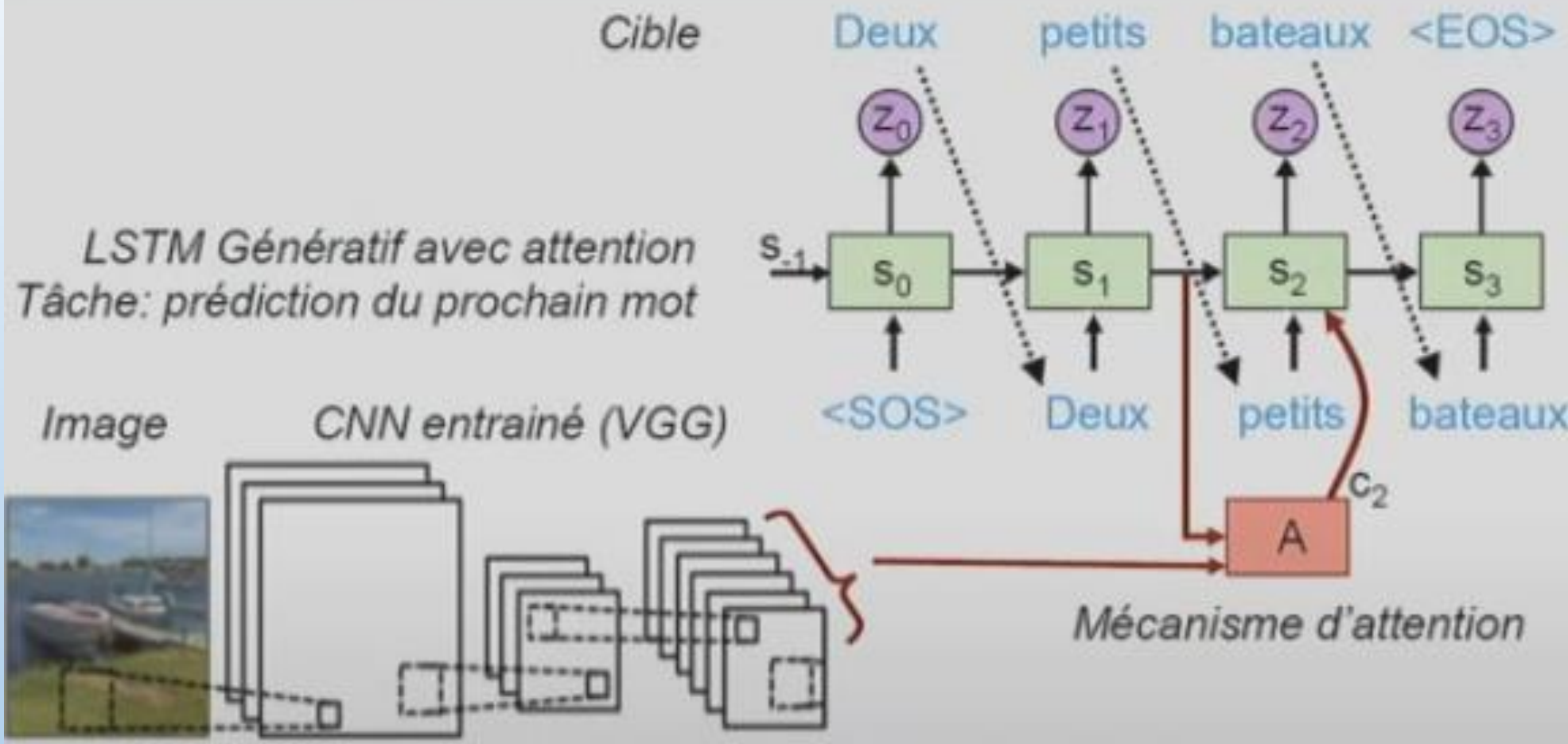
.

On utilise ces modèles pré-entrainés et on calibre finement leur poids pour une tâche spécifique (Fine Tuning et RLHF Reinforcement Learning by Human feedback)

Exemples : BERT et les modèles GPT (OpenAI-GPT, GPT2, GPT3 et GPT4, PaLm puis BARD de Google etc.)

Génération de légende

Entraînement: Génération du troisième mot (a partir de « petits »)



Ku et al., *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, 2015

45/56