Introduction au Deep Learning

Les transformers (inpiré librement de http://jalammar.github.io/illustrated-transformer/)

J. Rynkiewicz

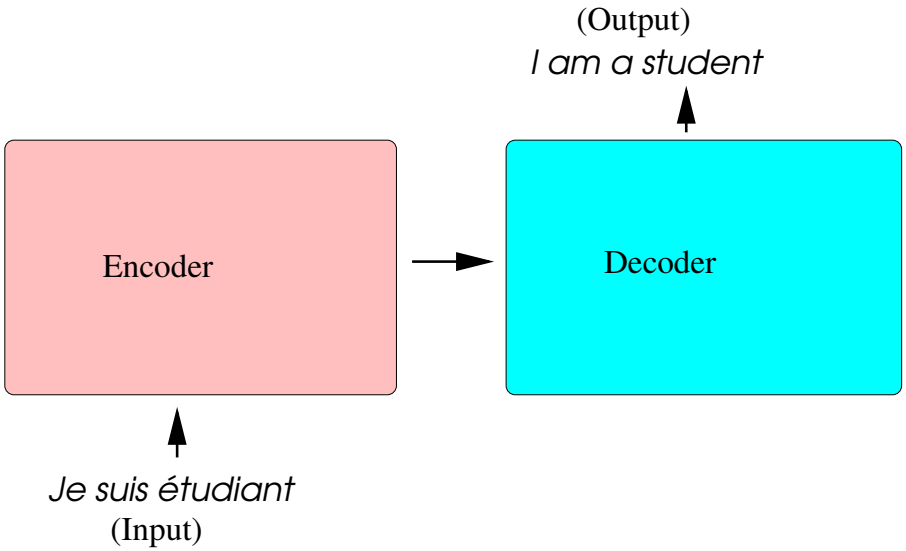
Université Paris 1

2022

Les transformers

Les réseaux de neurones transformers ont pour but de prévoir une séquence de longueur variable en fonction d'une autre séquence de longueur variable. Le principe est de tenir compte du contexte des observations à prévoir (concept d'attention) :

- Notons :
  - $(X_1, \cdots, X_{T_X})$, la séquence explicative.
  - $(Y_1, \cdots, Y_{T_Y})$ la séquence à prevoir. Remarquons que T_Y est aussi à prévoir.
  - θ le vecteur paramètre du modèle.
- Ce formalisme est adapté aux "Chatbot" ou bien au systèmes de traduction automatique.
- En général, l'architecture de ce réseau de neurones est composée d'un "encoder" et d'un "decoder" :

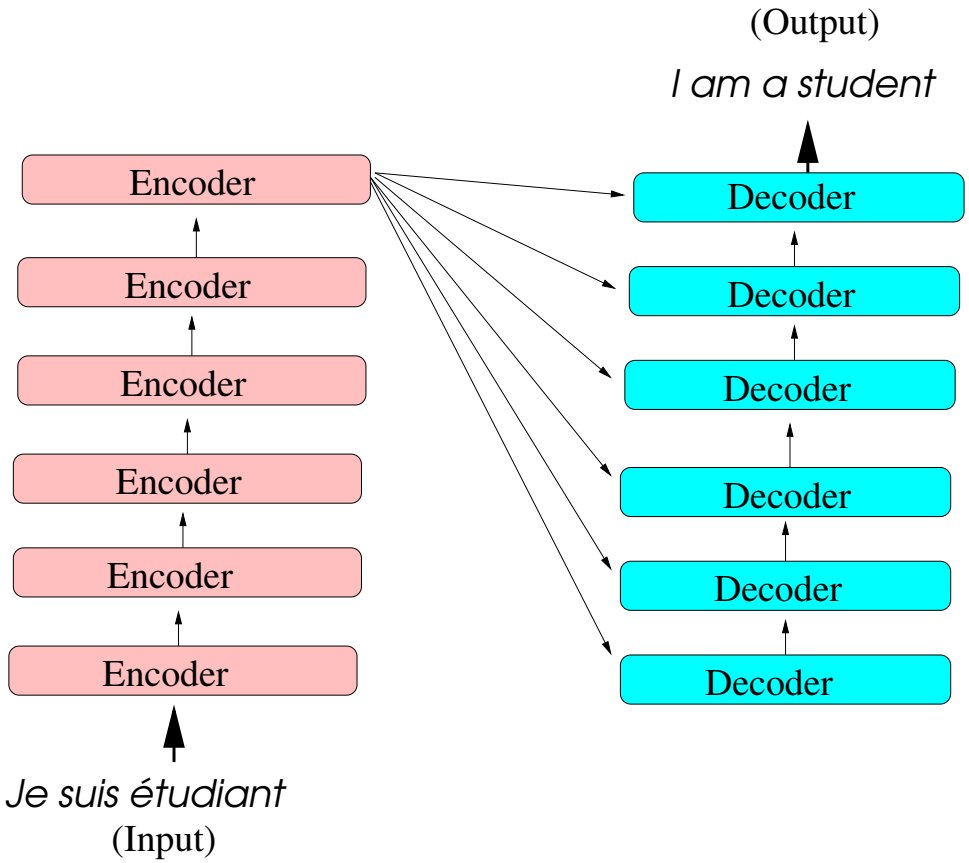


Tokenizer

- Tokenizer un texte consiste en le découpé en mots ou sous-mots. Il sont ensuite encodés en nombres (ids).
- Les trois techniques les plus employées sont : Byte-Pair Encoding (BPE), WordPiece et SentencePiece.
- Découper un texte en caractères ne permet pas d'obtenir des résultats proches de l'état de l'art.
- Mais découper le texte suivant les espaces n'est pas sastisfaisant. Il faut différencier la ponctuation et rendre possible la contruction de nouveaux mots.
- On mélange donc le codage des mots et des caractères : sous-mots.
- L'idée des sous-mots est de garder entier les mots très utilisés, mais de de découper les mots rares ou transformés par la grammaire (adverbe etc...)
- La tokenization en sous-mots permet de garder une taille raisonnable pour le vocabulaire (généralement de 50000 tokens).
- Les meilleurs ensembles de sous-mots sont contruits en gardant les sous-mots les plus fréquents (BPE) ou bien ceux qui maximisent la vraisemblance du texte (WordPiece).
- SentencePiece tokenize les textes en considérants les espaces comme des caractères, puis utilise BPE ou Wordpiece.

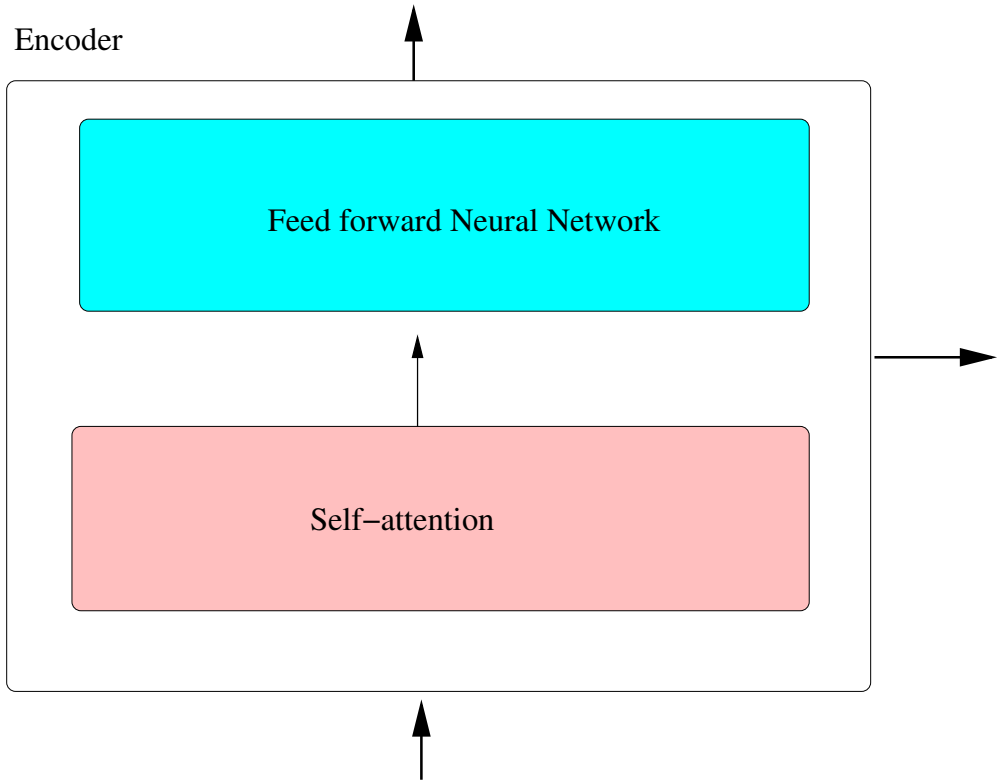
Encoder, decoder

L'encoder est une pile de N petits encoders, le decoder une pile de N petits decoder. Dans l'article original $N = 6$.



Petit encoder

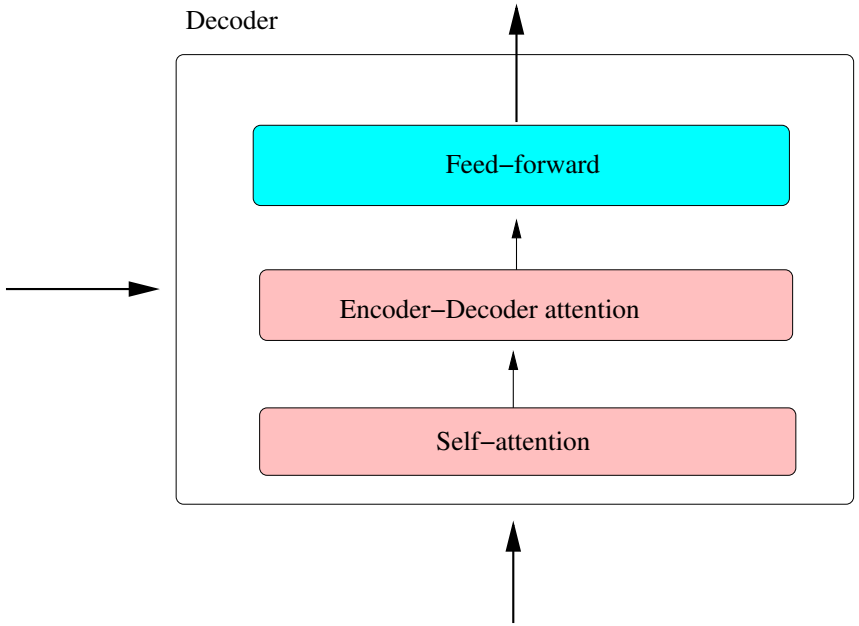
- Les petits encoders ont tous la même architecture (mais ils ne partagent pas leur paramètres).



- L'entrée du petit encoder passe d'abord par une couche de Self-attention (décrite plus tard).
- La sortie du petit encoder passe, auparavant, par un réseau feed-forward dont l'architecture est identique pour tous les encoders.

Petit decoder

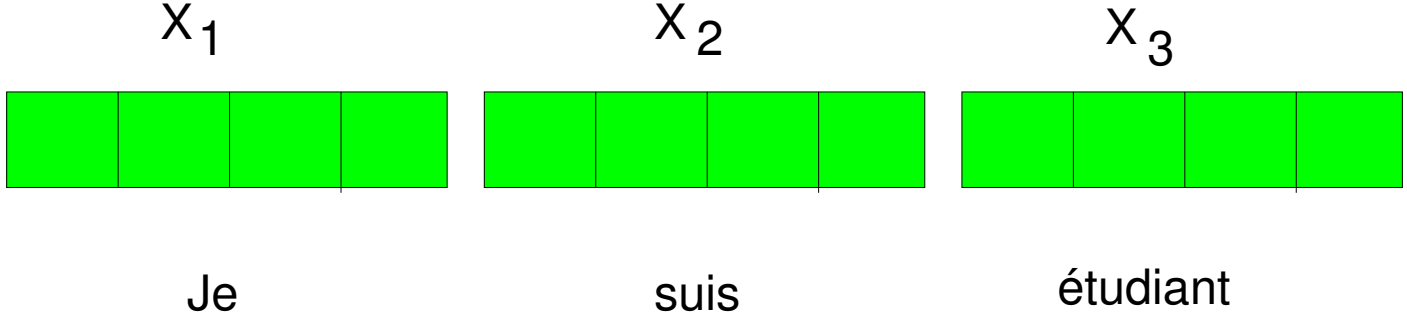
- Les petits decoders ont tous la même architecture (mais ils ne partagent pas leur paramètres).



- L'entrée du petit decoder passe d'abord par une couche de Self-attention (décrite plus tard).
- Il y a une couche intermédiaire pour se focaliser sur la séquence d'entrée et sa transformation par l'encoder.
- La sortie du petit encoder passe, auparavant, par un réseau feed-forward dont l'architecture est identique pour tous les decoders.

Représentation des mots

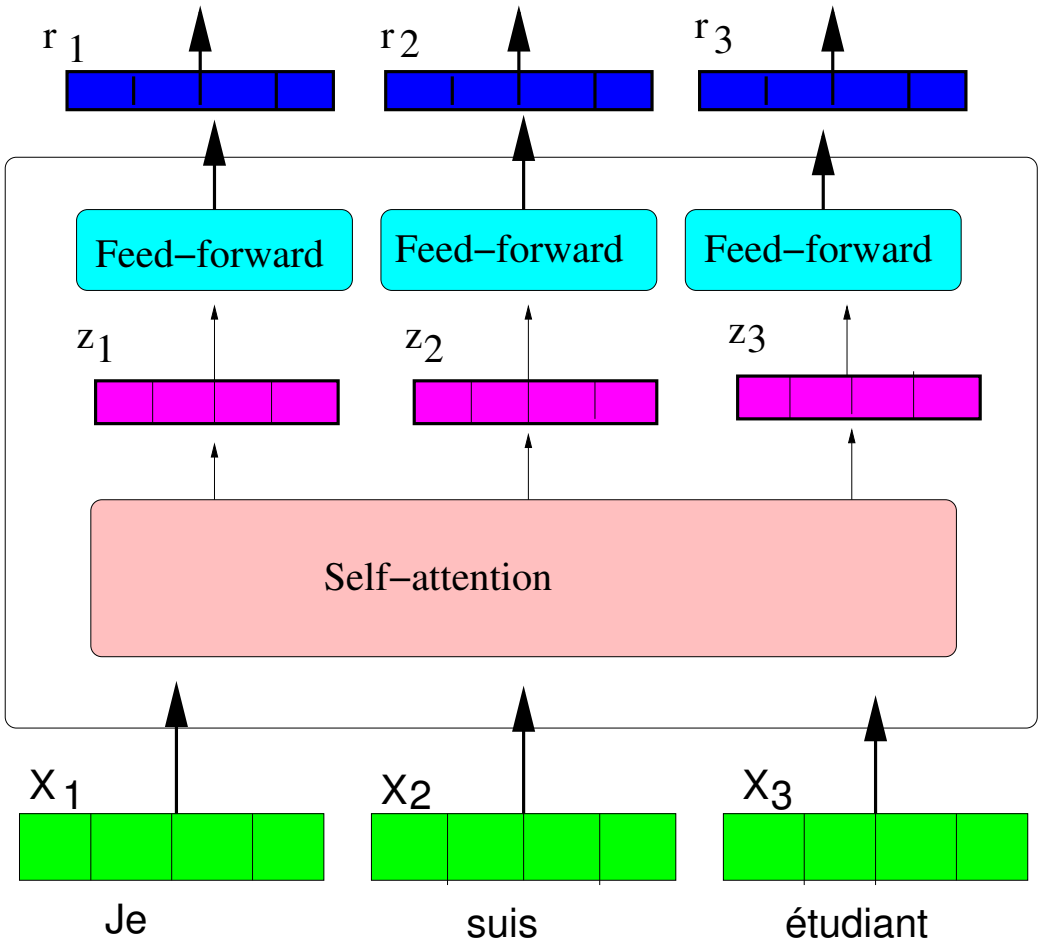
- On commence par plonger (embedding) les mots dans un espace continu ($\mathbb{R}^{512}$ dans l'article original).



- Pour l'illustration, le plongement est dans $\mathbb{R}^4$.
- Le plongement a lieu uniquement pour l'entrée du premier petit encoder.
- Tous les autres petits encoder recoive la sortie du petit encoder précédent (de même taille que le plongement).
- Après le plongement, les vecteurs passent par les deux couches du premier petit encoder.

Passage dans le premier petit encoder

- Le petit encoder reçoit une liste de vecteurs et renvoi un liste de vecteurs de la même taille.
- Les vecteurs passent d'abord dans la couche de "Self-attention", puis dans les réseaux "Feed-forward".

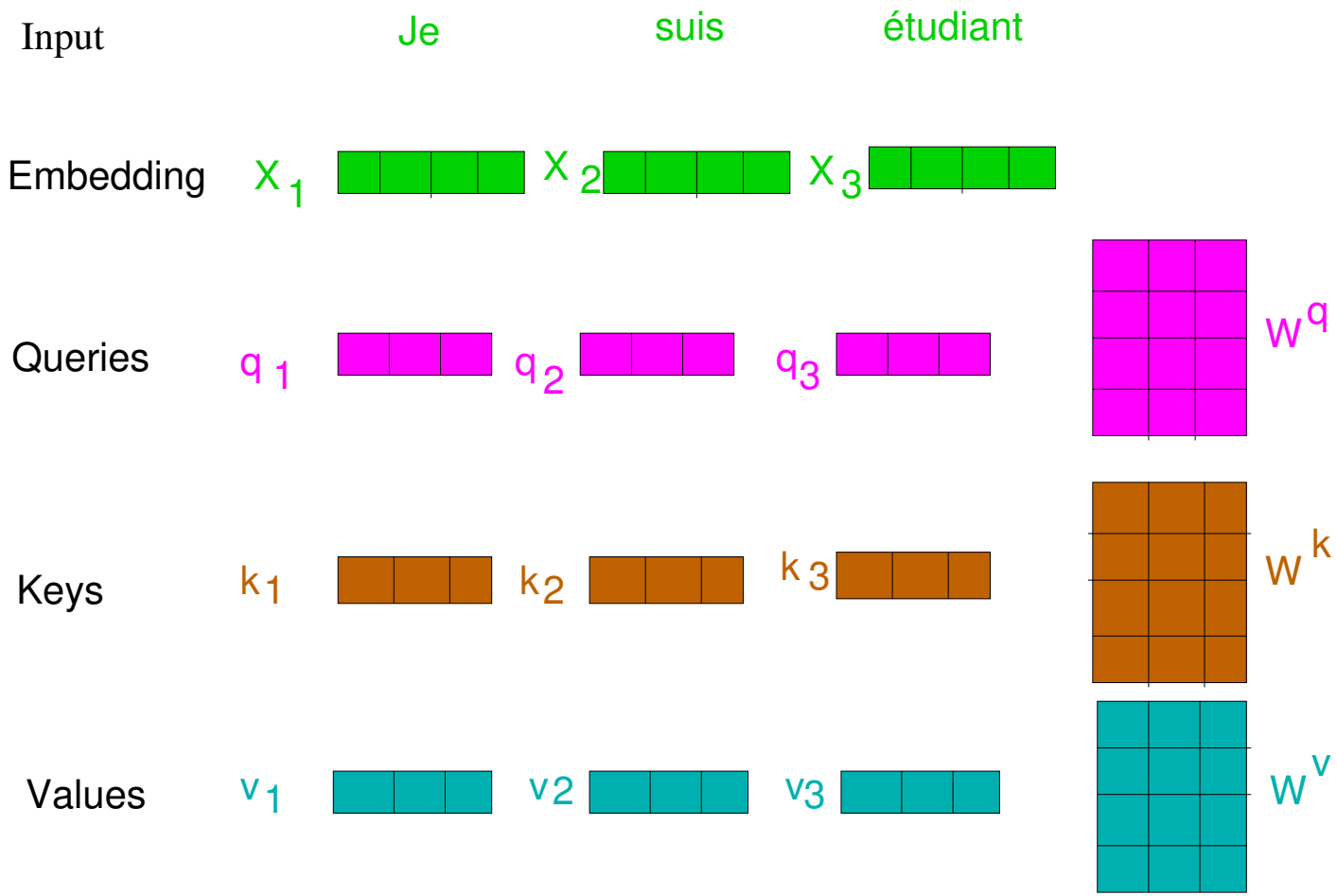


Le méchanisme d'attention

- L'attention permet de tenir compte du contexte d'un mot.
- Pour traduire la phrase "La brebis n'a pas traversé la rue parce qu'elle était trop fatiguée"
- À quoi fait réfèrence "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 proposent une méthode pour permettre cette estimation.

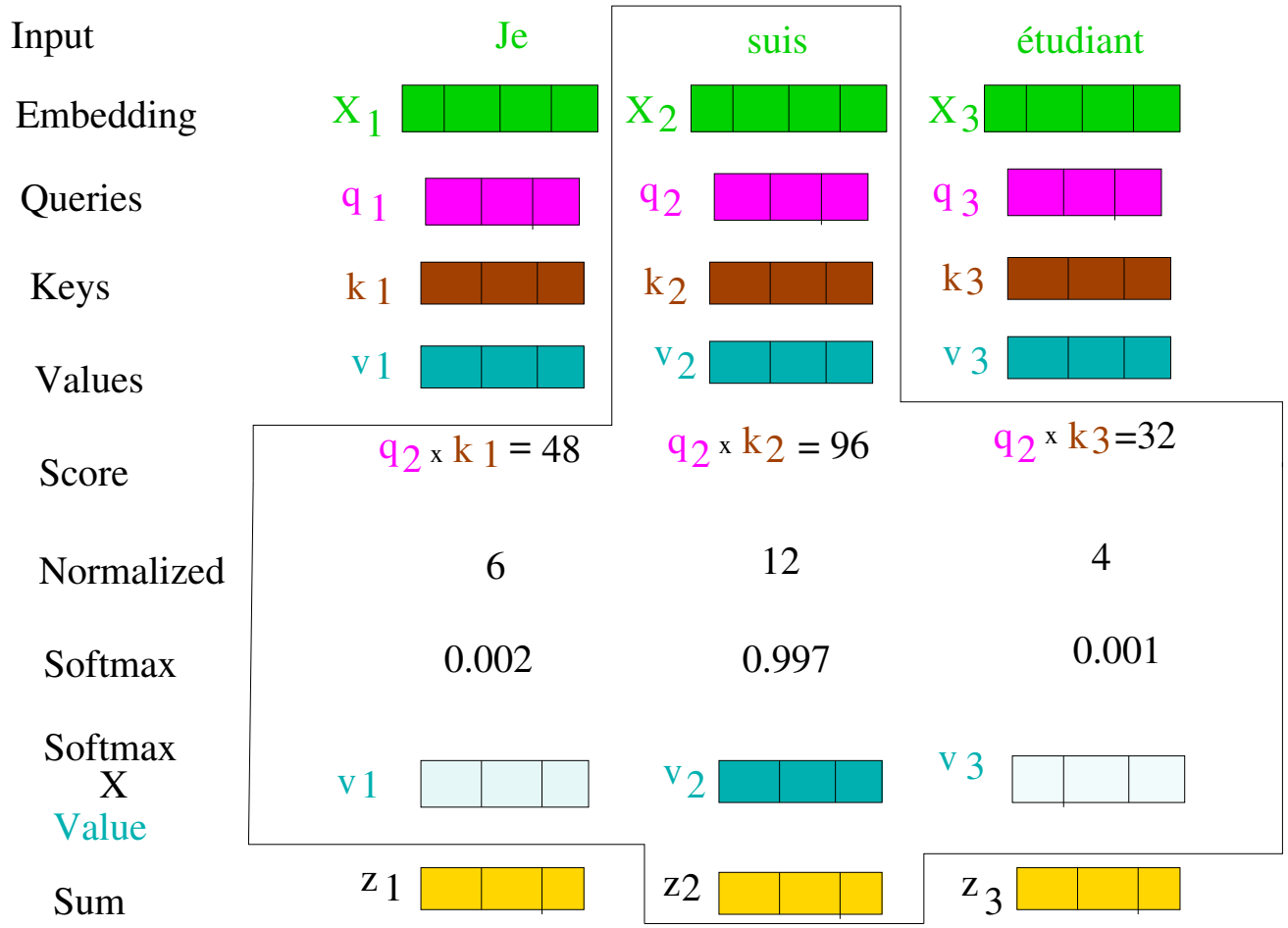
Détail de la couche de “Self-attention” (1)

- On commence par définir trois vecteurs : Le vecteur “Query”, le vecteur “Key” et le vecteur “Value”.
- Ces vecteurs sont crées en multipliant l”Embedding” par une matrice de poids (de dimension 64x512 dans l’article originel).



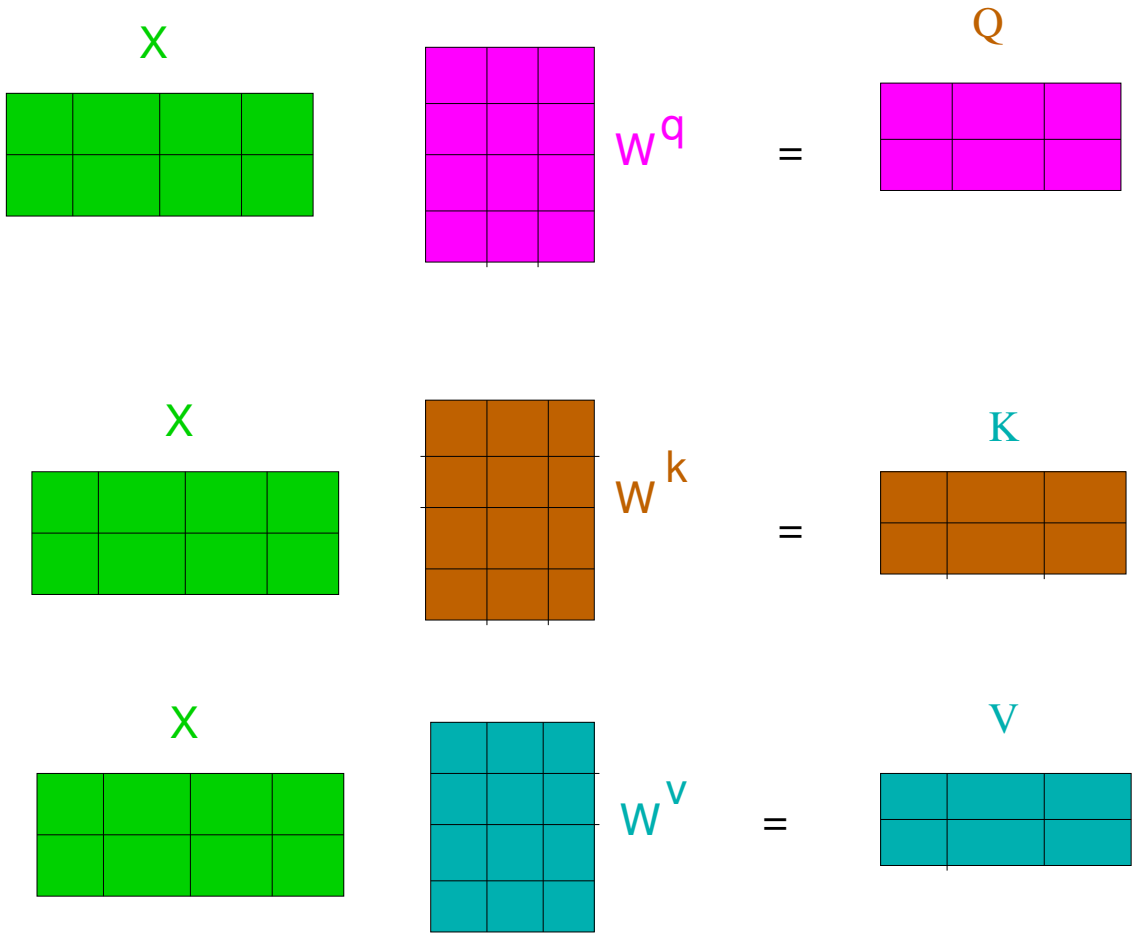
Détail de la couche de “Self-attention” (2)

- L’embedding est transformé en “query”, “key” et “value” puis chaque “value” est pondérée par un softmax du score induit par toutes les keys.
- La somme pondérés des values est la sortie “z” de la couche d’attention.



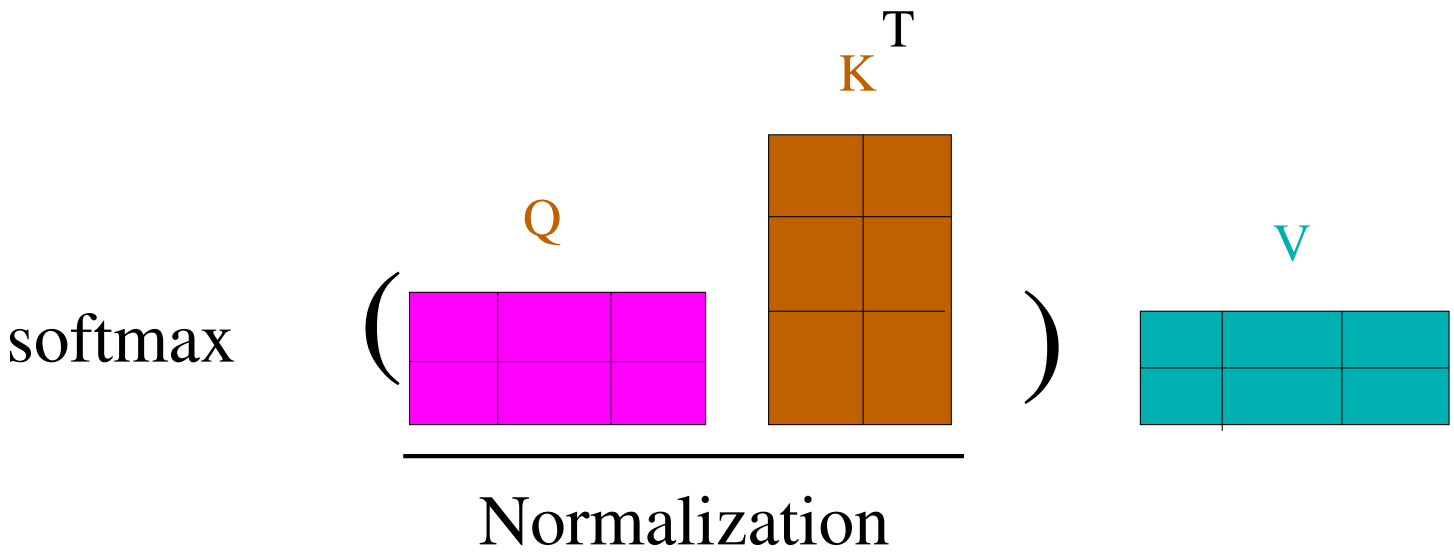
Calcul matriciel pour l'attention (1)

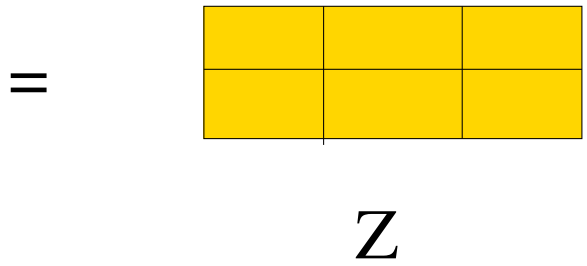
- La forme matriciel permet de paralléliser les calculs.



Calcul matriciel pour l'attention (2)

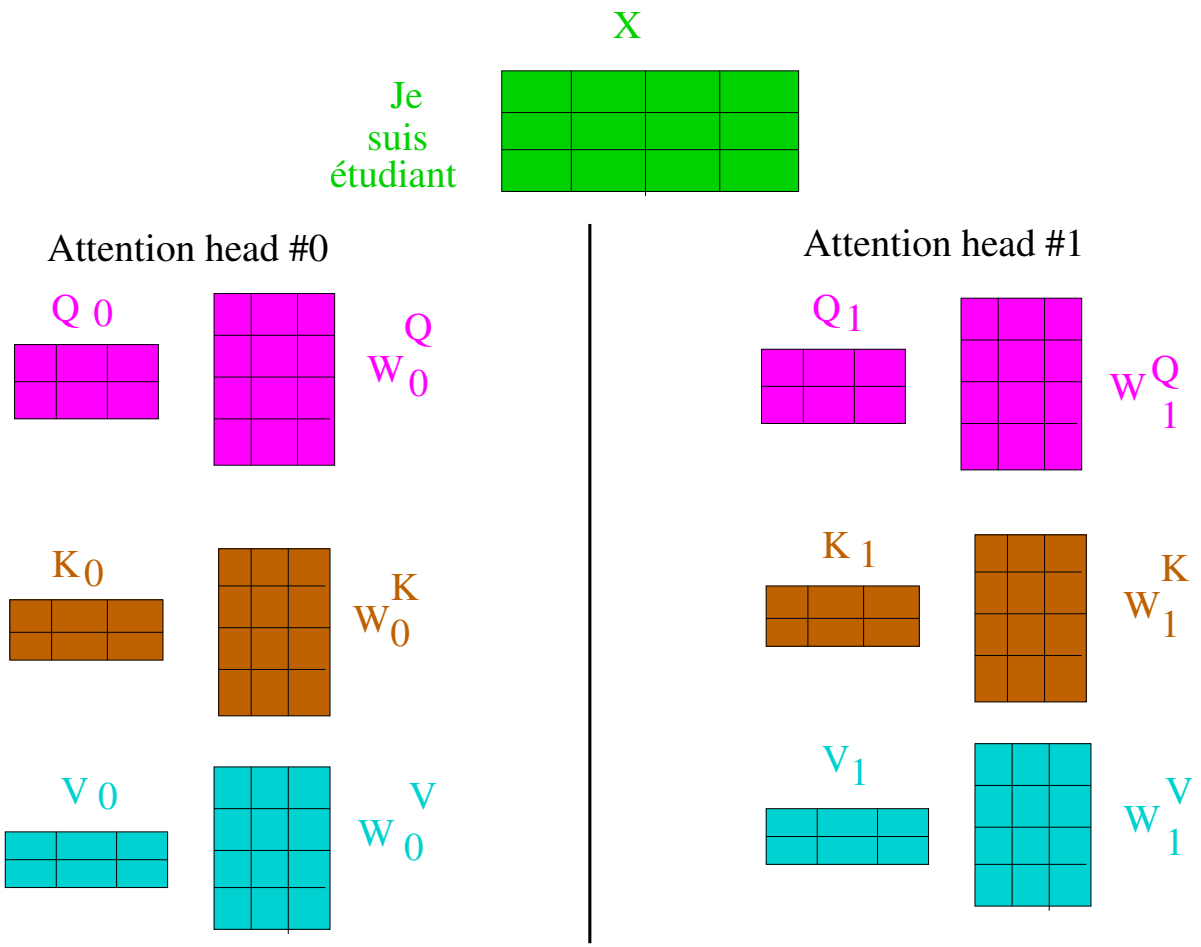
- On peut résumer le calcul détaillé de l'attention par l'équation matricielle suivante :





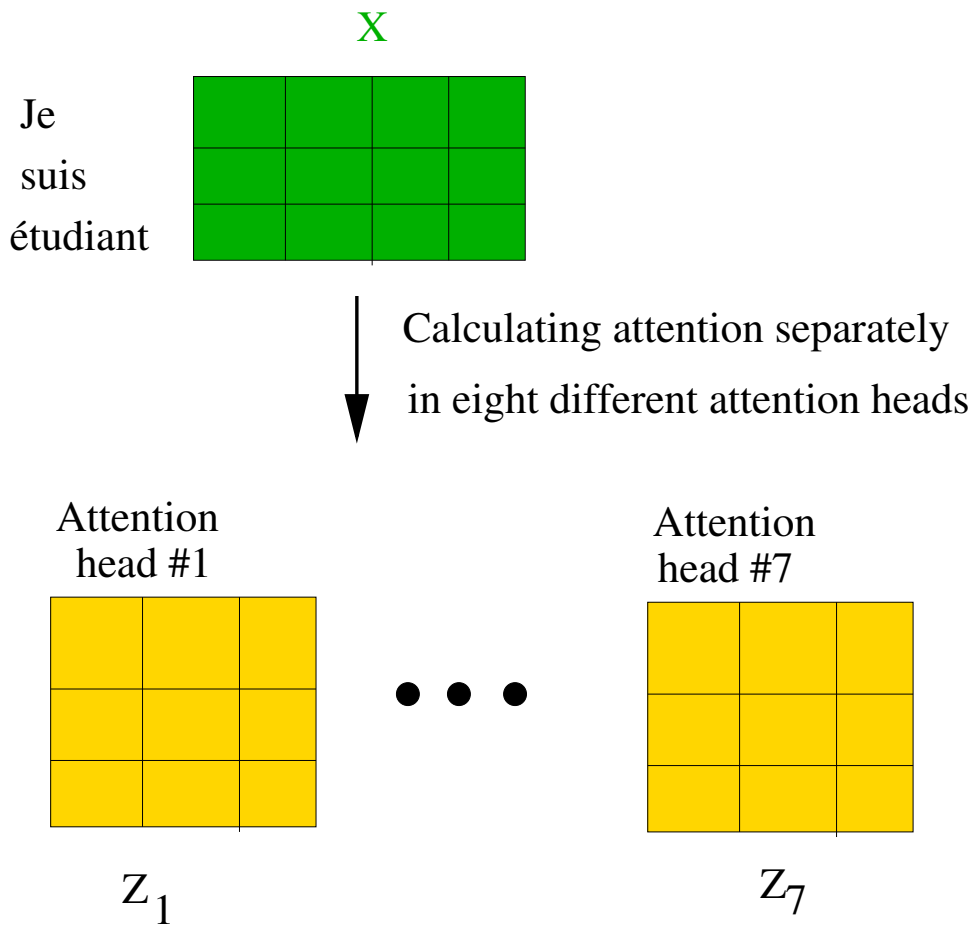
Couche d'attention "multi-headed"

- Dans l'article originel, il y a plusieurs couches d'attention en parallèle.
- Cela permet au modèle d'avoir un plus grand espace de représentation du contexte.



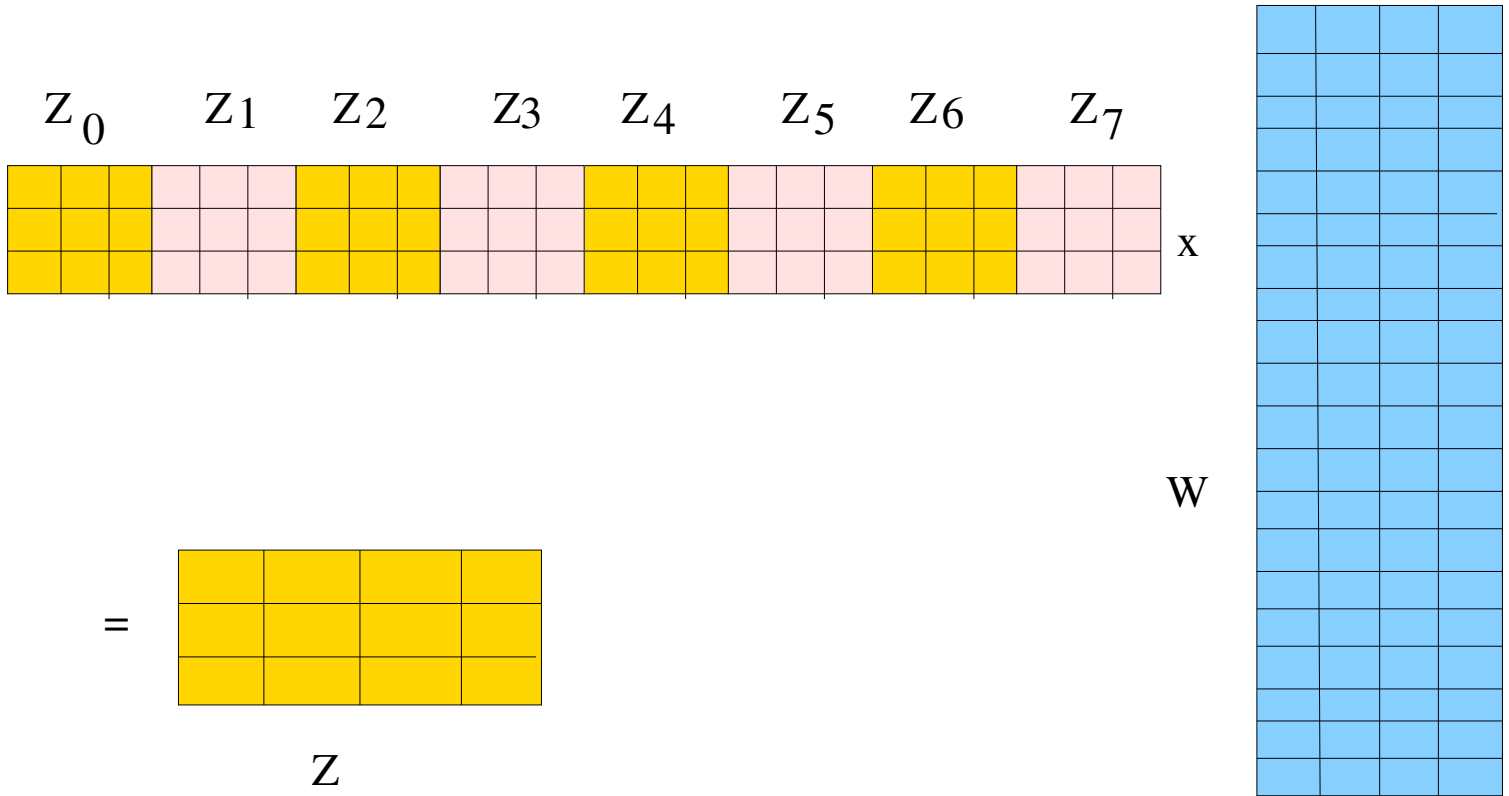
Calculs de la couche d'attention "multi-headed"

- Chaque couche d'attention a des poids différents.
- Dans l'article originel, les auteurs calculent 8 matrices Z



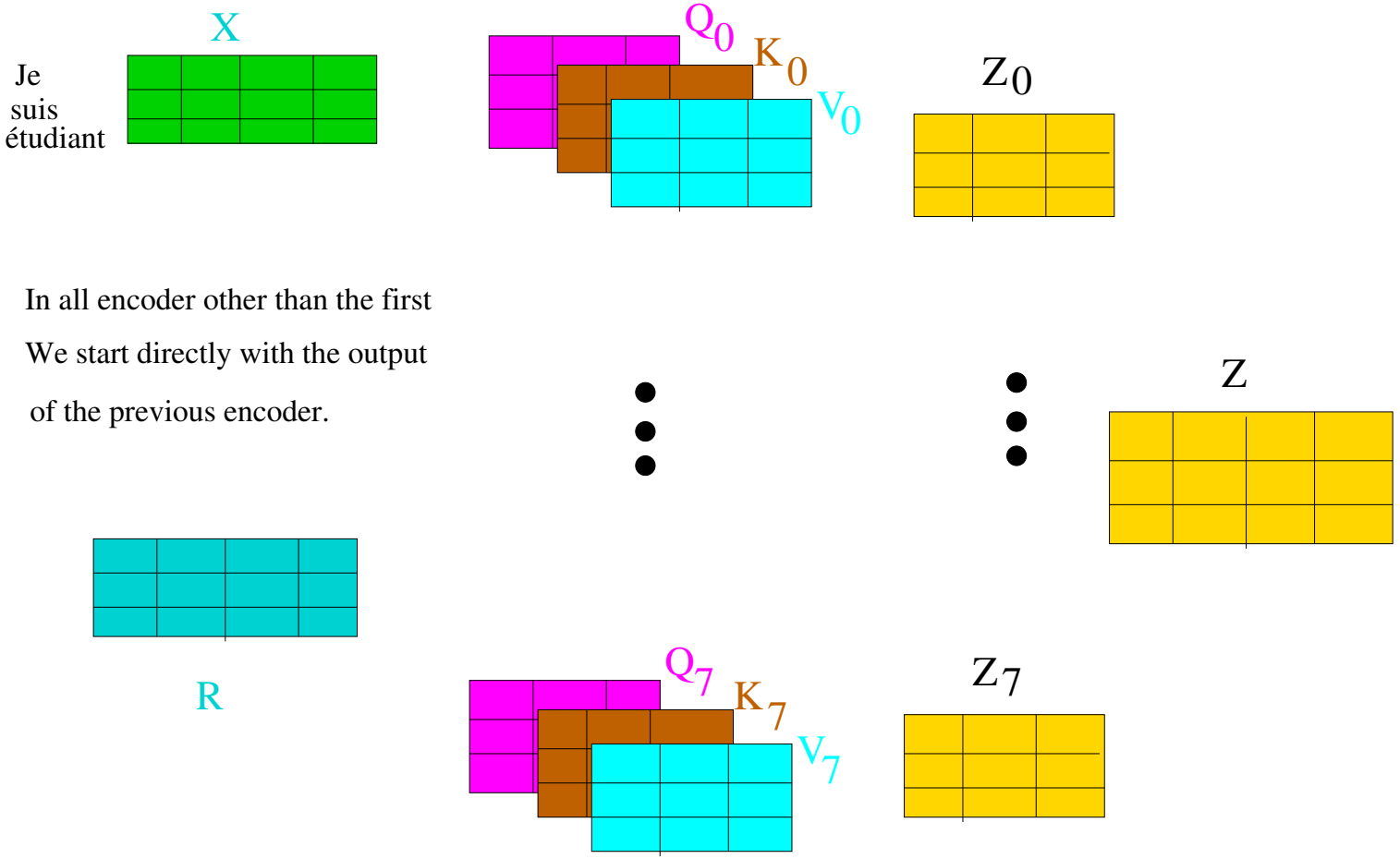
Calcul de l'attention finale

- Pour obtenir l'attention finale, on concatène les sorties des couches d'attention.
- Puis, on multiplie ce vecteur par une matrice de poids W qui sera à estimer.



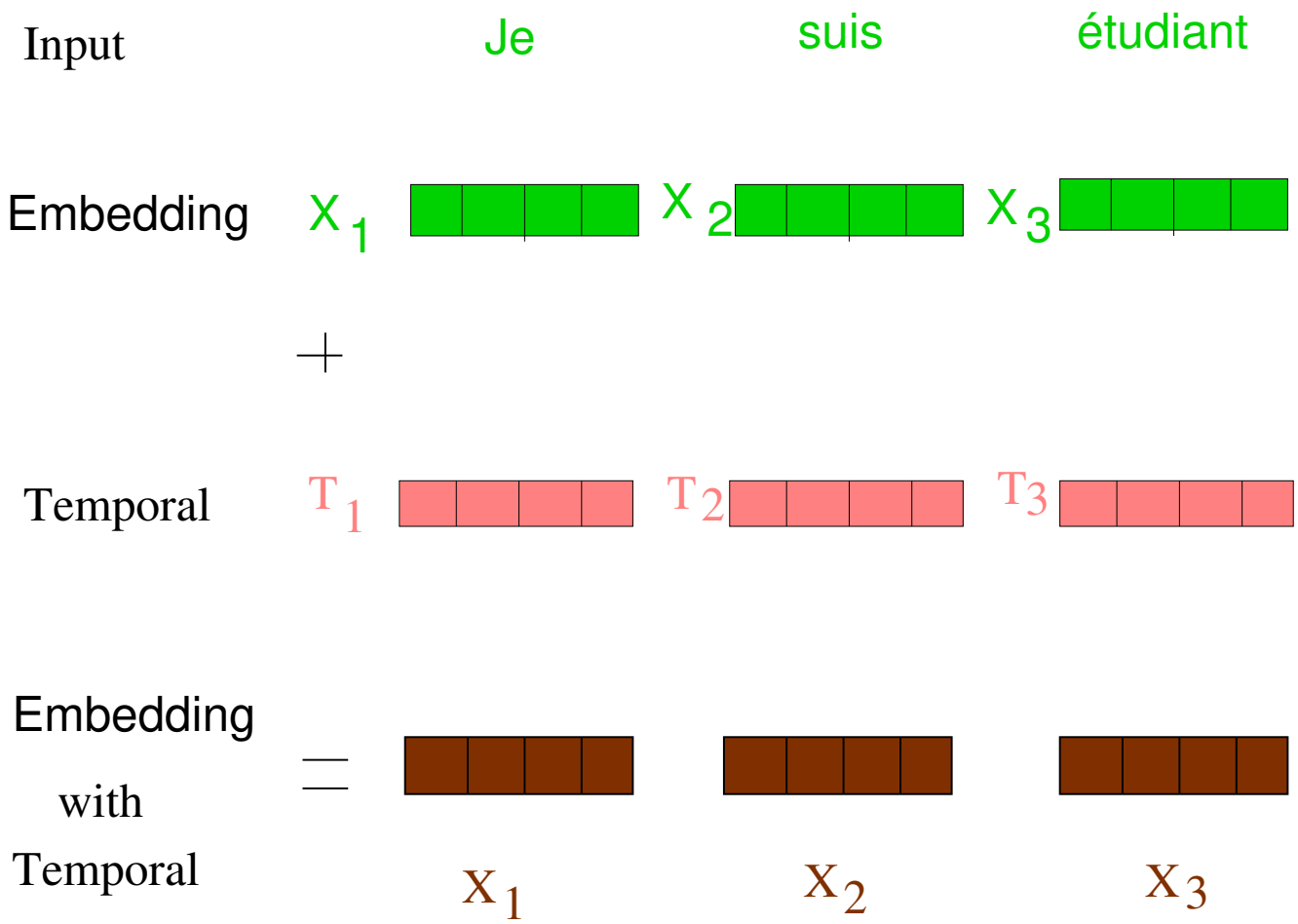
Pour résumer la couche d'attention multihead

- Le vecteur final d'attention Z sera obtenu par les calculs des slides précédents.



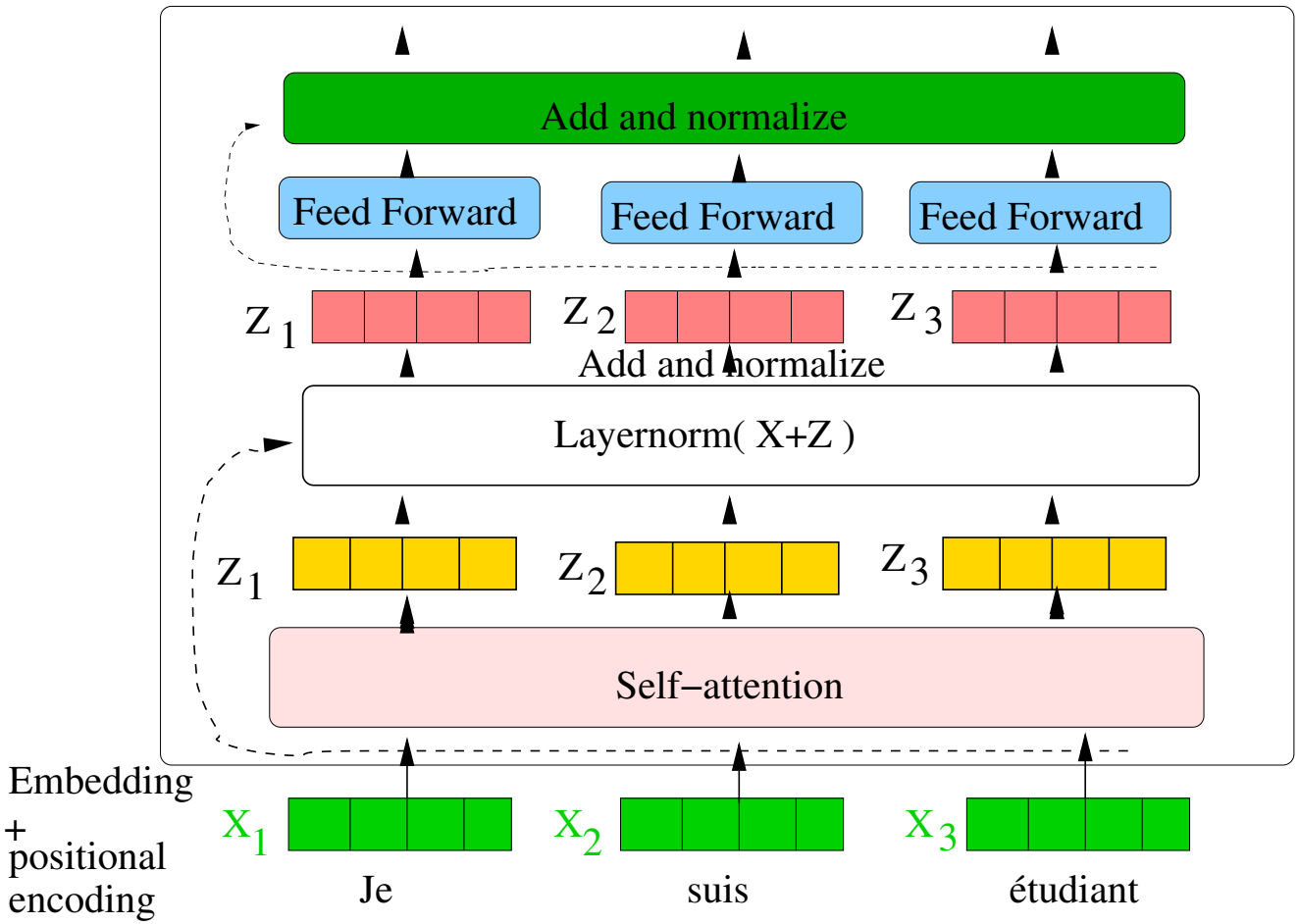
Encodage de la position pour représenter l'ordre

- On veut pouvoir tenir compte de l'ordre des mots dans la phrase.
- Pour cela on va plonger un signal périodique (un cosinus) dans un espace de même taille que celui où on a plongé les mots.
- Ensuite on va additionner les deux plongements.



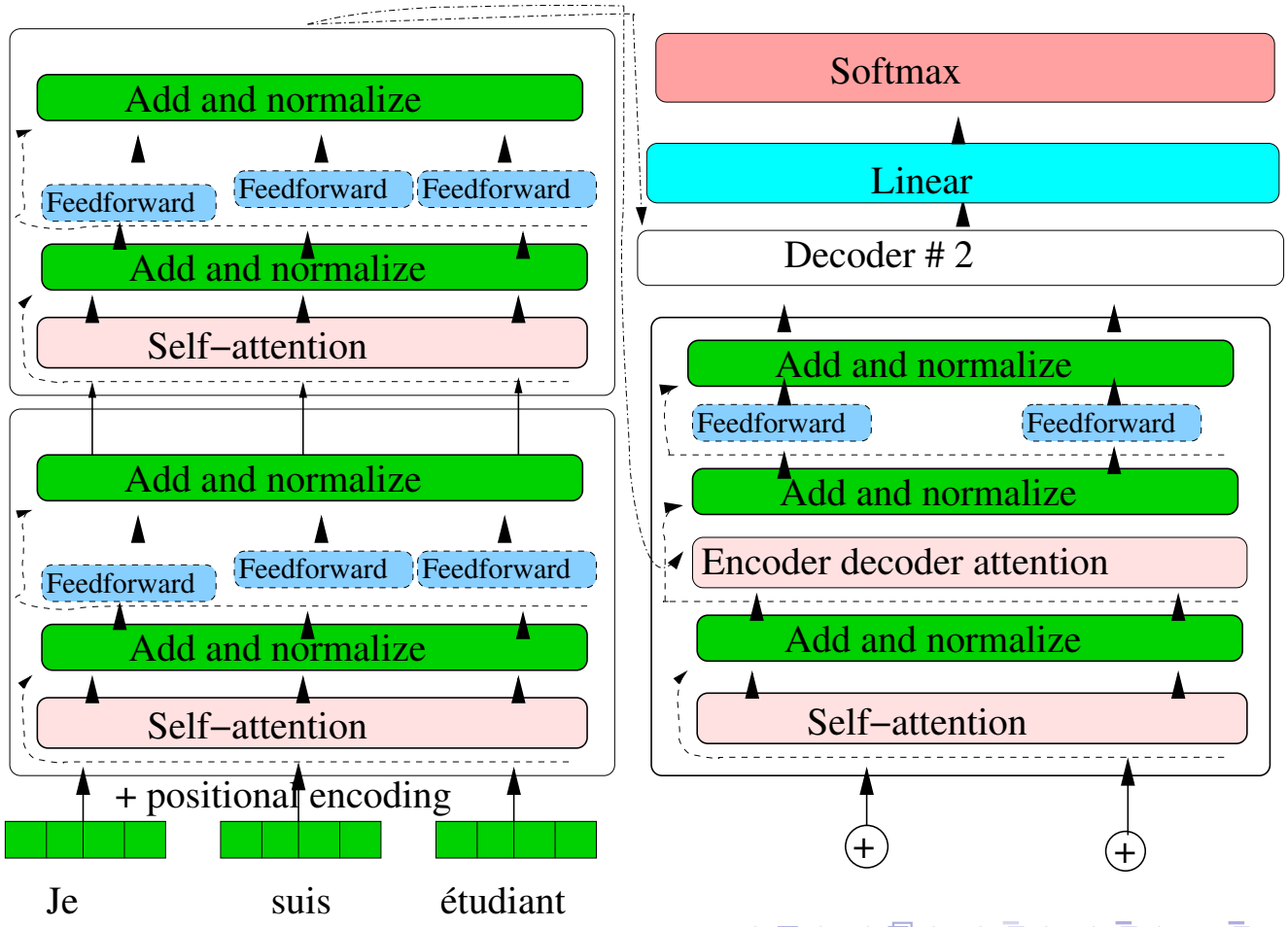
Connexion résiduelle

- Chaque sous-couche à une connexion résiduelle (copie de l'entrée).
- Cela permet d'améliorer la transmission de l'information et le calcul du gradient.



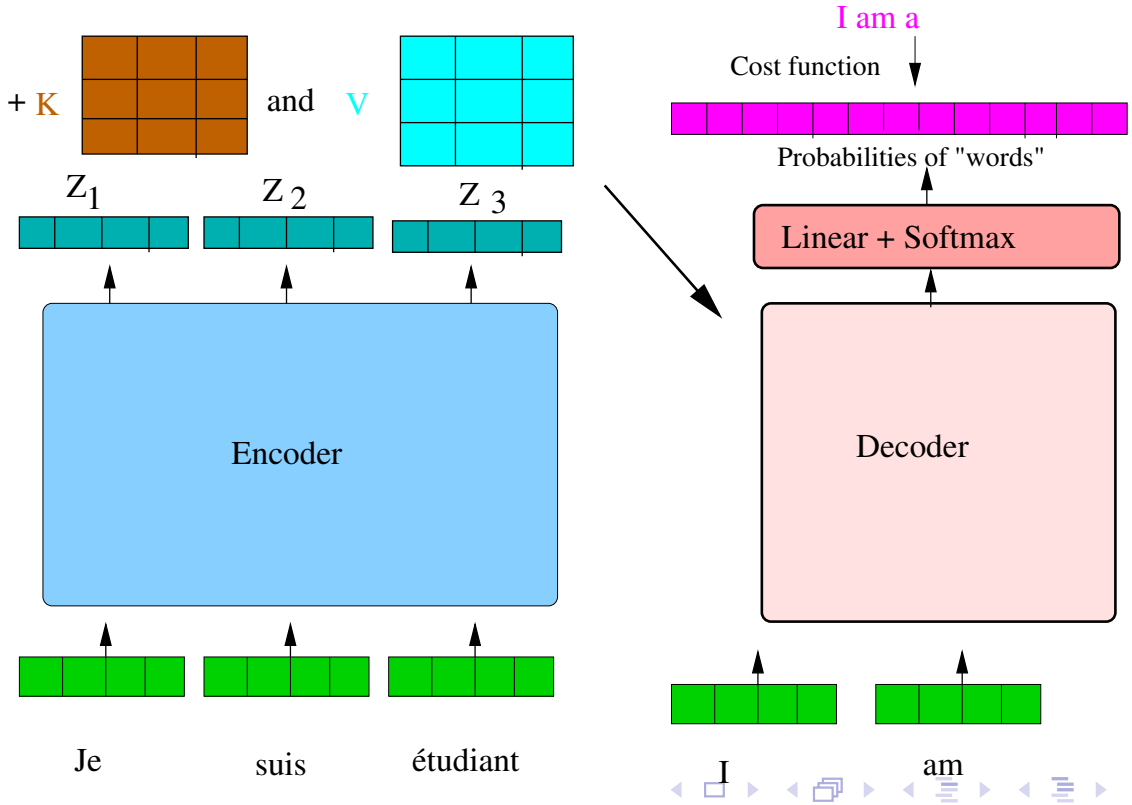
Lien entre l'encoder et le decoder

- On représente ici un encoder fait de deux petits encoders, idem pour le decoder.
- L'architecture du decoder a une couche d'attention supplémentaire pour tenir compte de la sortie de l'encoder.



Estimation du Transformer

- Le decoder utilise la sortie mais aussi les vecteurs "keys" et "values" finaux de l'encoder pour calculer les probabilités conditionnelles de mots en sorties.
- Il 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".



Génération de la séquence par le decoder

- Le decoder commence avec pour entrée la sortie de l'encoder. Il génère alors le mot (ou signe de ponctuation) le plus probable selon lui (ses poids).
- Le decoder utilise alors la sortie de l'encoder et le mot qu'il vient de générer pour générer le mot suivant.
- La couche de self attention encoder decoder utilise les "keys" et "values" finales de l'encoder en plus des caractères déjà générer par le décodeur.
- Le decoder arrête de générer des caractères lorsqu'il émet le caractère spécial "end of sentence".
- On remarquera que la taille du vocabulaire est forcément fini. En pratique c'est quelques dizaines de milliers.
- Le decoder ne peut pas inventer de "mots" nouveaux.

Transfert learning pour le NLP

- Depuis 2018 des modèles ont des poids pré-entrainée sur de grande base de données (livres, wikipedia, etc...).
- On utilise donc ces modèles pré-entrainés et on calibre finement leur poids pour une tâche spécifique.
- Les deux plus connus (mais cela change vite) sont BERT et les modèles GPT (OpenAI-GPT, GPT2 et GPT3).
- BERT tient compte de tout le contexte (le passé et le futur de la phrase), il est surtout utile pour :
  - L'analyse du sentiment (phrases positives ou négatives).
  - Plus généralement, classification de phrase (spam, non spam etc...)
  - Question/réponse.
- GPT a été entrainé à prévoir le mot suivant d'une phrase. Ce modèle n'utilise que le decoder du transformer, il est surtout utile pour :
  - Chatbot.
  - Génération de texte en général.

Génération de texte

GPT2 (maintenant GPT3) a été entrainé à prévoir le mot suivant d'une phrase. Il est le modèle de base pour générer du texte. Cependant, il calcule seulement la loi de probabilité du prochain sous-mot, il faut une méthode pour le tirer au hasard parmi tous les sous-mots. Les principales méthodes sont :

- Greedy search : On choisit simplement le sous-mot le plus probable. Cela a tendance à créer des séquences répétitives.
- Beam search : On calcule les probabilités des suites de mots de longueur NB, et on choisit le suite de mot la plus probable. NB est un hyperparamètre à régler. Cette méthode a toujours tendance à créer des répétitions et le langage naturel semble plus aléatoire.
- Top K sampling : On tire aléatoirement parmi les K sous-mots les plus probables (en renormalisant la loi de probabilité sur ces K sous-mots). Cependant l'hyperparamètre K n'est peut-être pas adapté à toutes les lois de probabilités.
- Top P sampling : On choisit K telle que la somme des probabilités de K sous-mots soit égale au moins à P.
- Dans la pratique, on mixte les deux méthodes précédentes avec un K maximum.