

Architecture de réseau de neurones

-

Le Transformer

Guillaume Bourmaud

PLAN

I. RNN vs Transformer

II. Couche d'attention à softmax

III. Équivariance par permutation et encodage de la position

IV. Application à des images

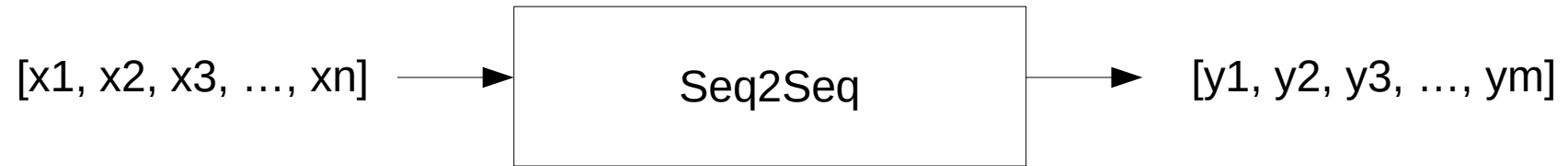
V. Limites

I) RNN vs Transformer

“Attention is All You Need”, NIPS 2017

1)

Sequence-to-sequence



```
"the cat sat on the mat" -> [Seq2Seq model] -> "le chat etait assis sur le tapis"
```

I)

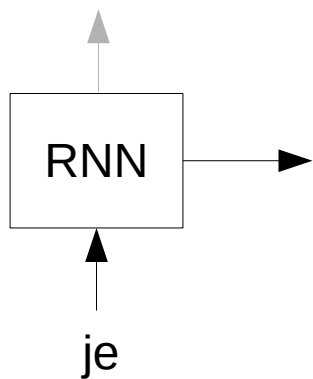
RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

I)

RNN - inférence

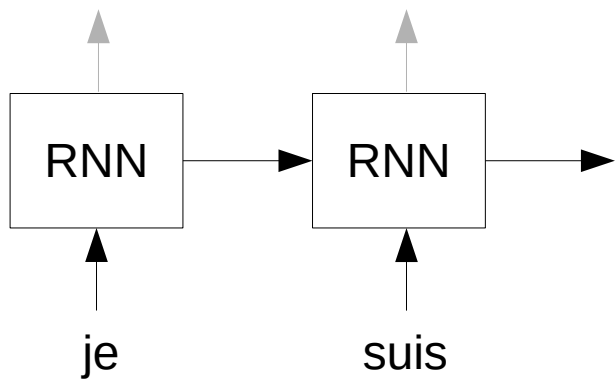
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



l)

RNN - inférence

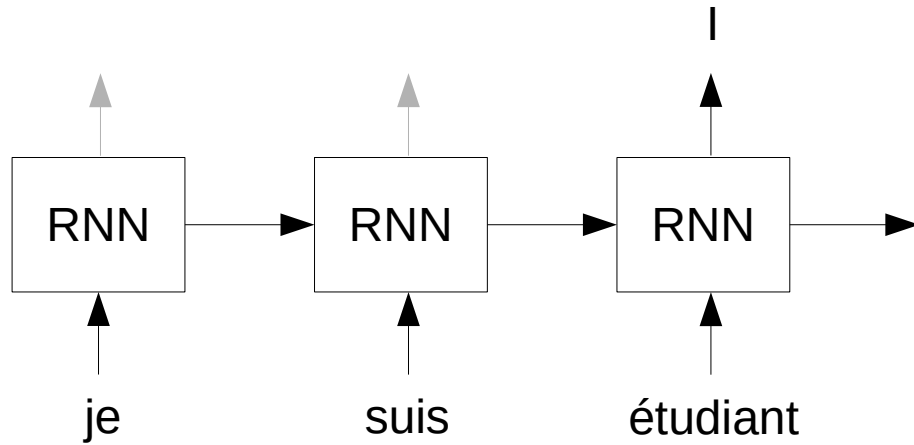
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

RNN - inférence

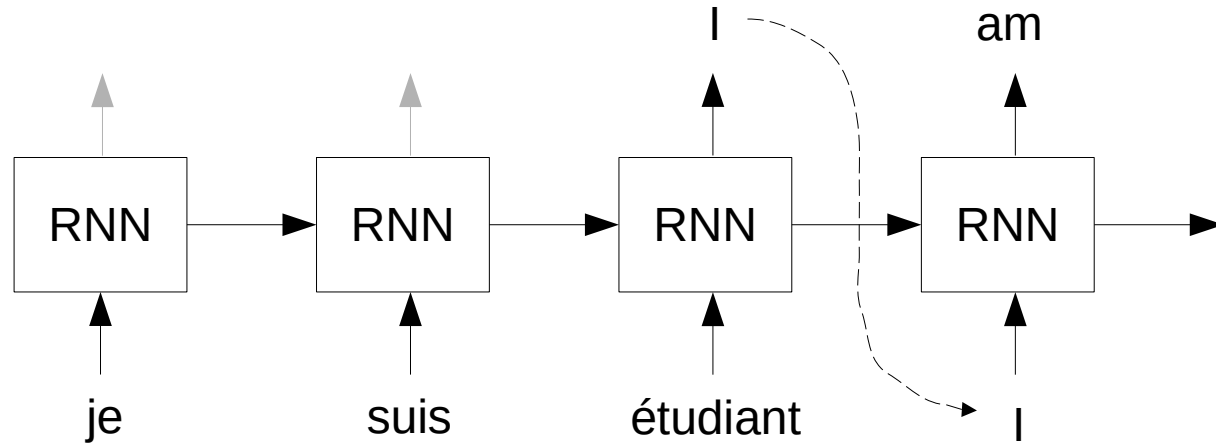
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

RNN - inférence

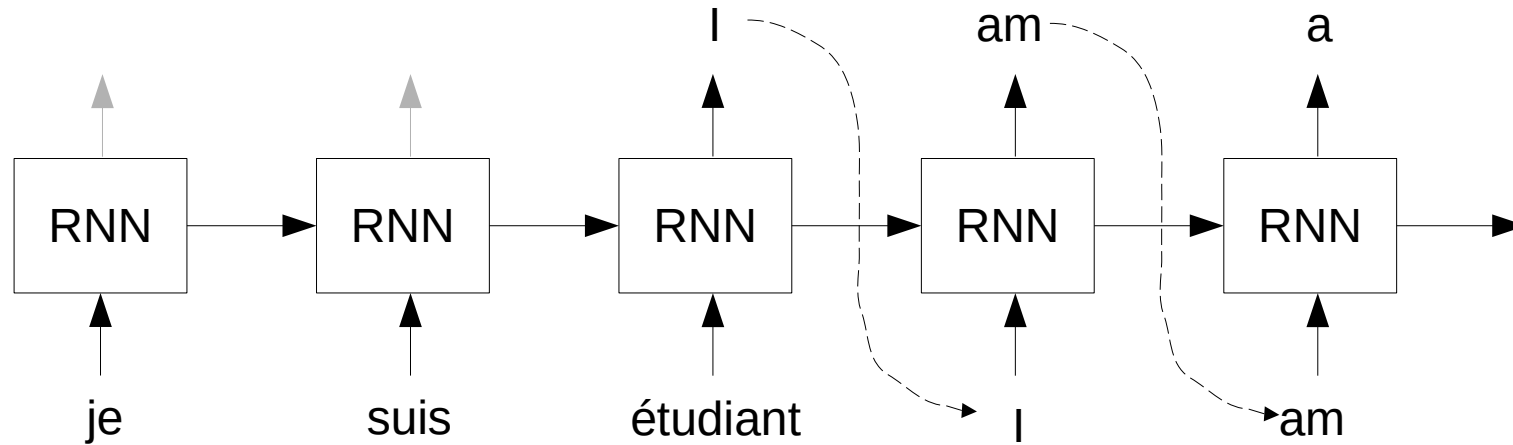
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



l)

RNN - inférence

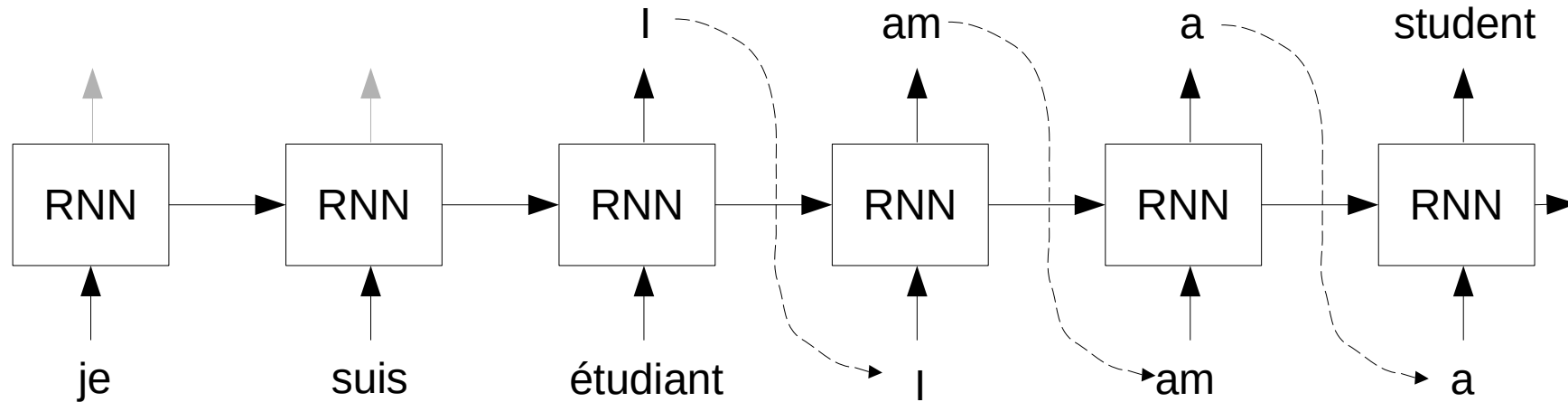
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

RNN - inférence

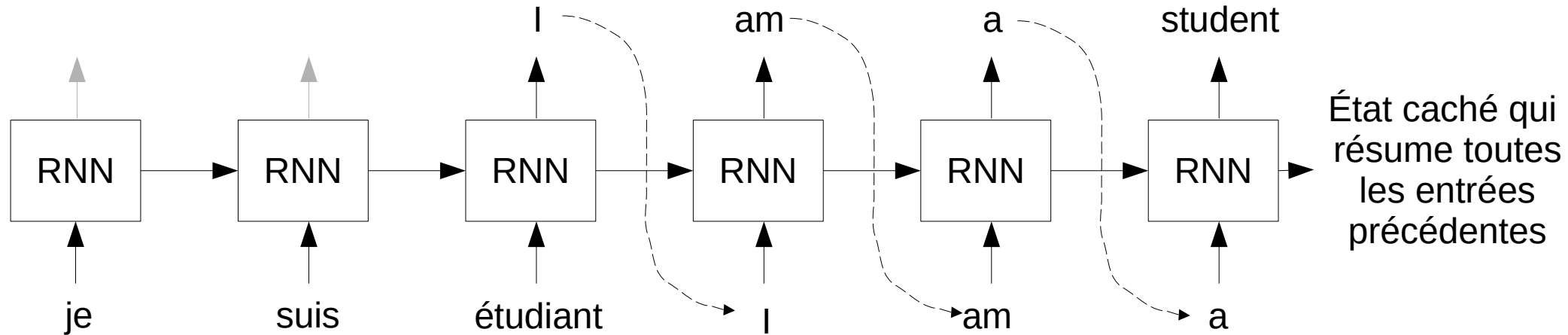
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

RNN - inférence

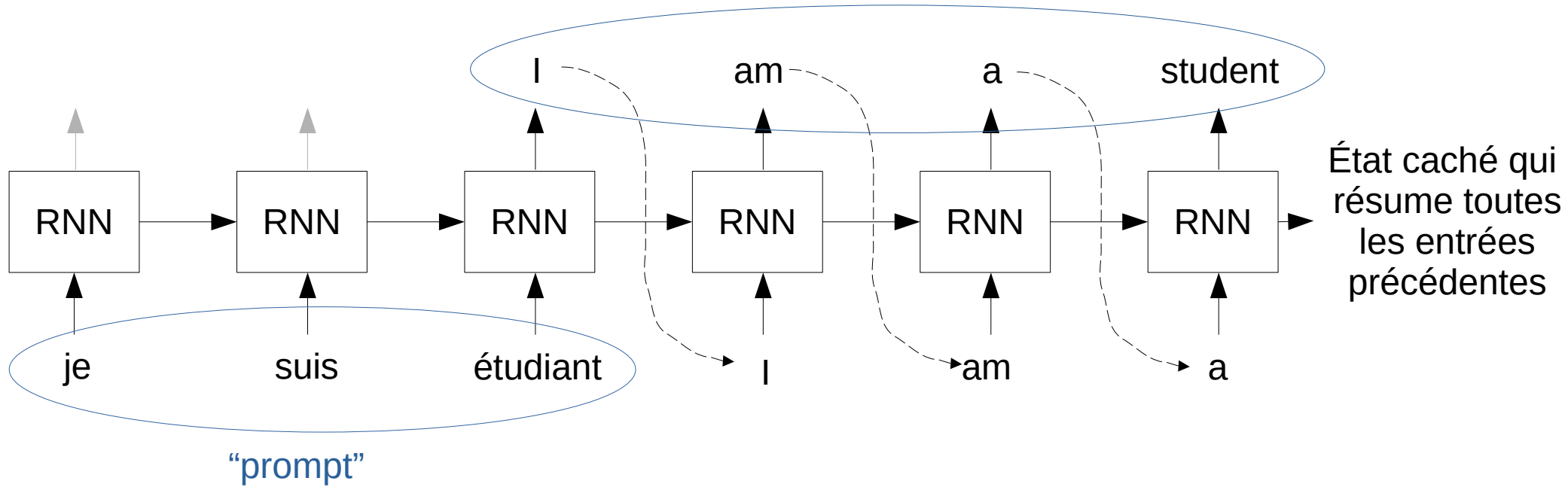
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

RNN - inférence

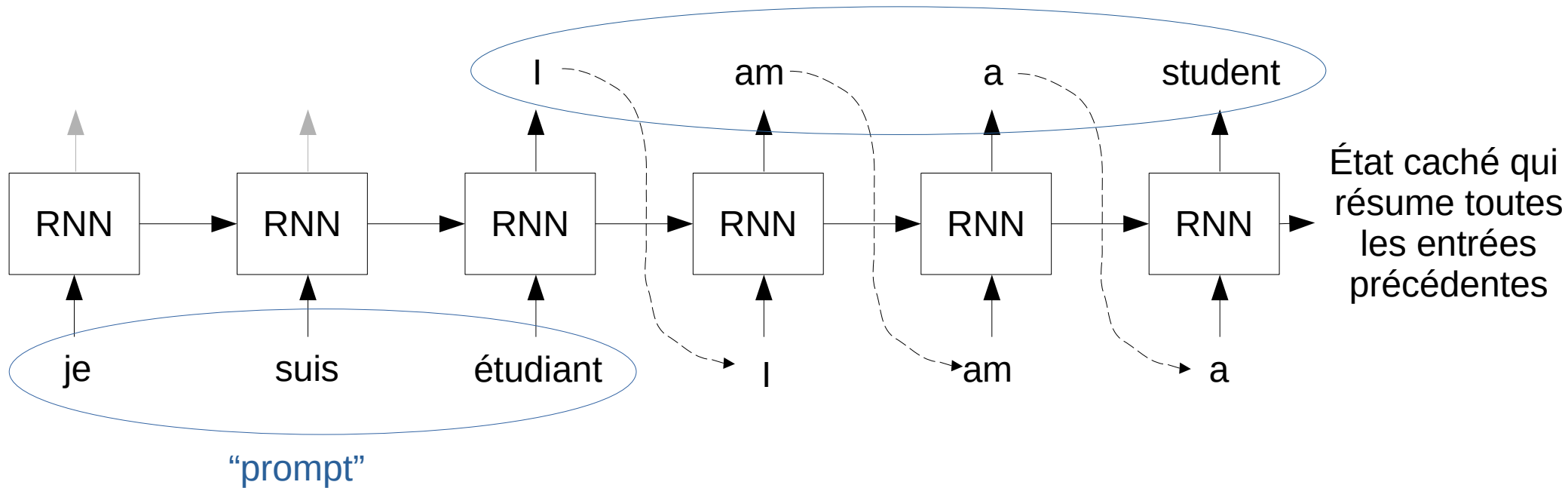
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”
réponse



1)

RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”
réponse

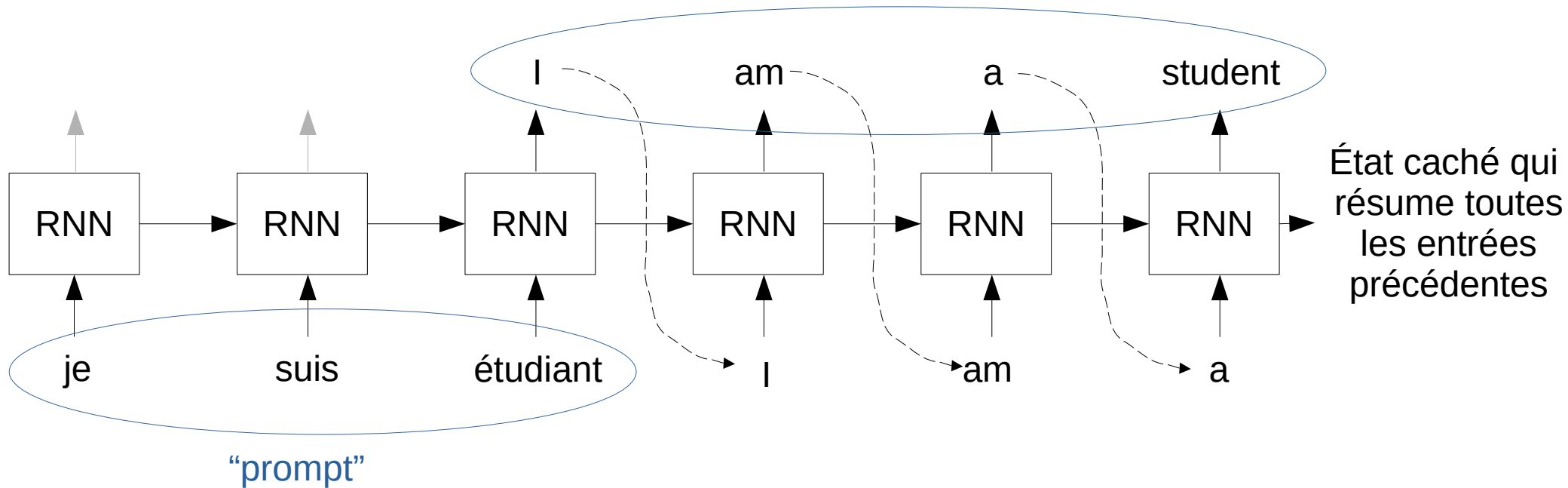


Inférence séquentielle (à cause du paradigme “next token prediction” pas de l'architecture RNN)

1)

RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”
réponse



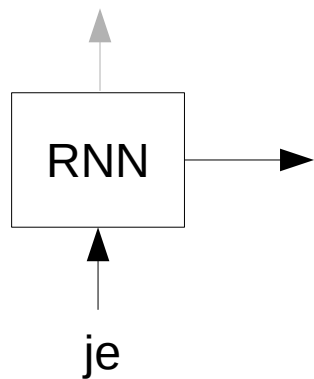
Inférence séquentielle (à cause du paradigme “next token prediction” pas de l’architecture RNN)

→ L’architecture Transformer ne va pas régler ce problème (ChatGPT est séquentiel)

I)

RNN - apprentissage

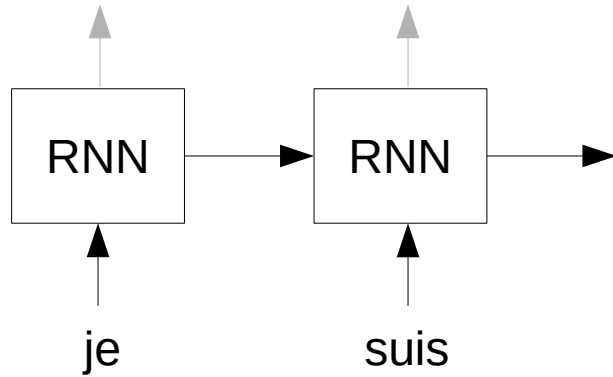
“Teacher forcing”



I)

RNN - apprentissage

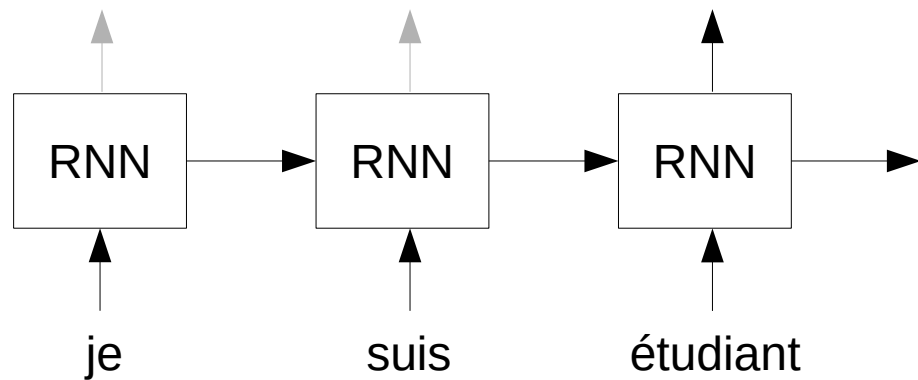
“Teacher forcing”



I)

RNN - apprentissage

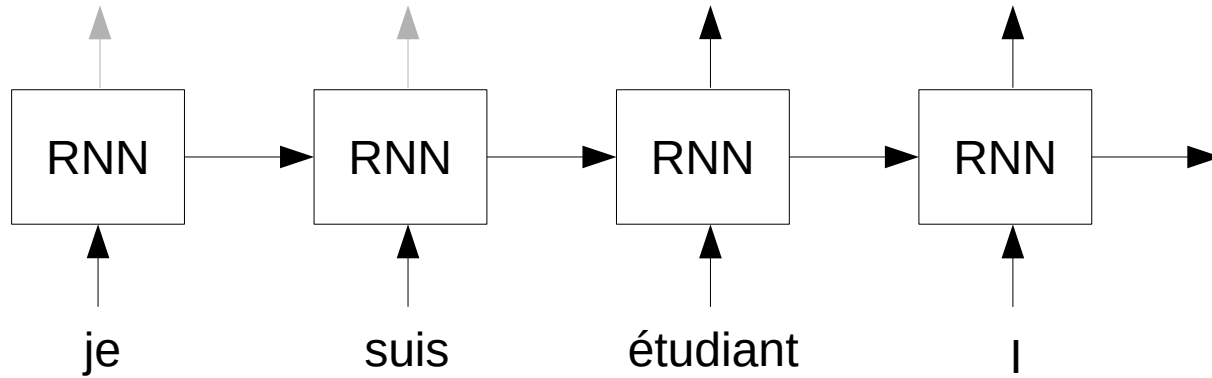
“Teacher forcing”



l)

RNN - apprentissage

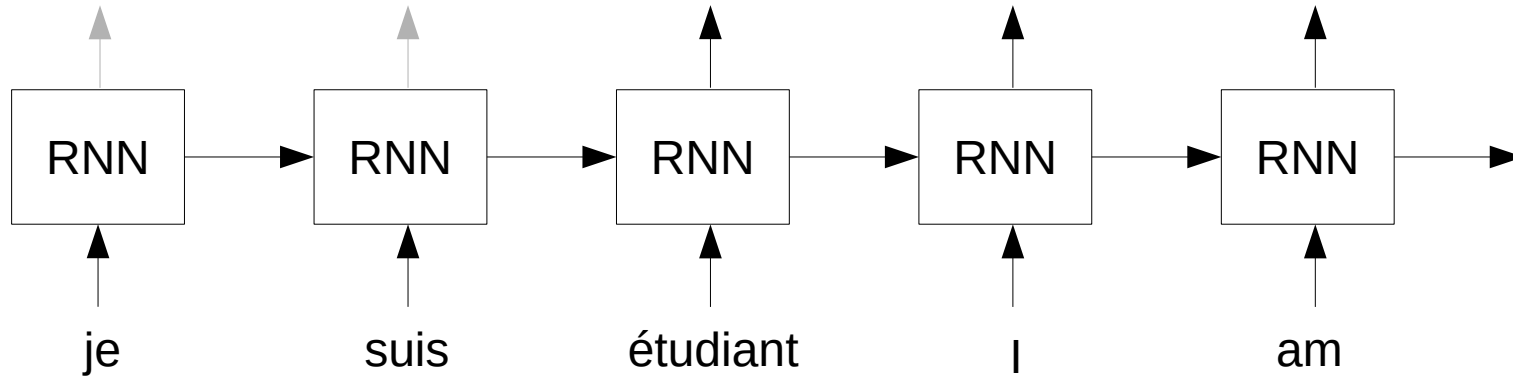
“Teacher forcing”



1)

RNN - apprentissage

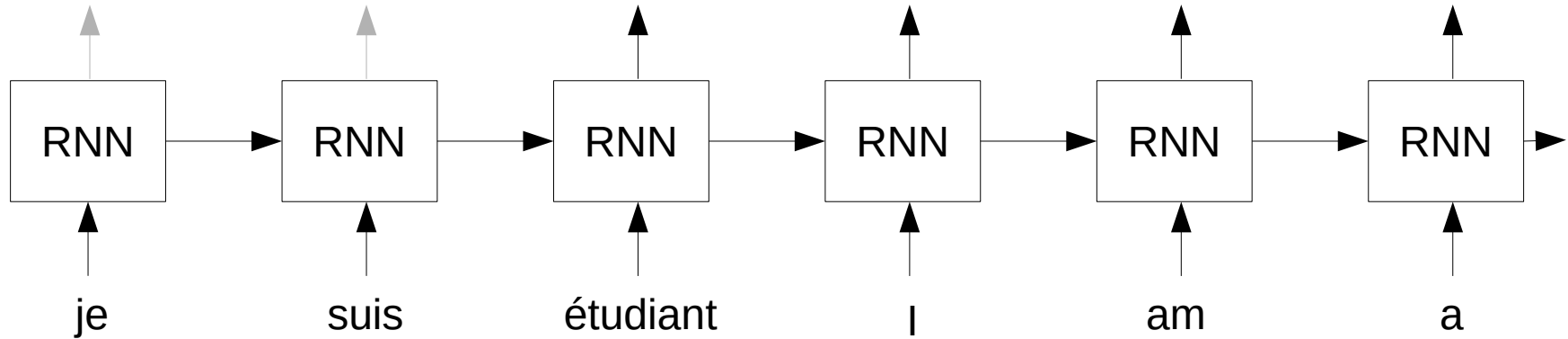
“Teacher forcing”



I)

RNN - apprentissage

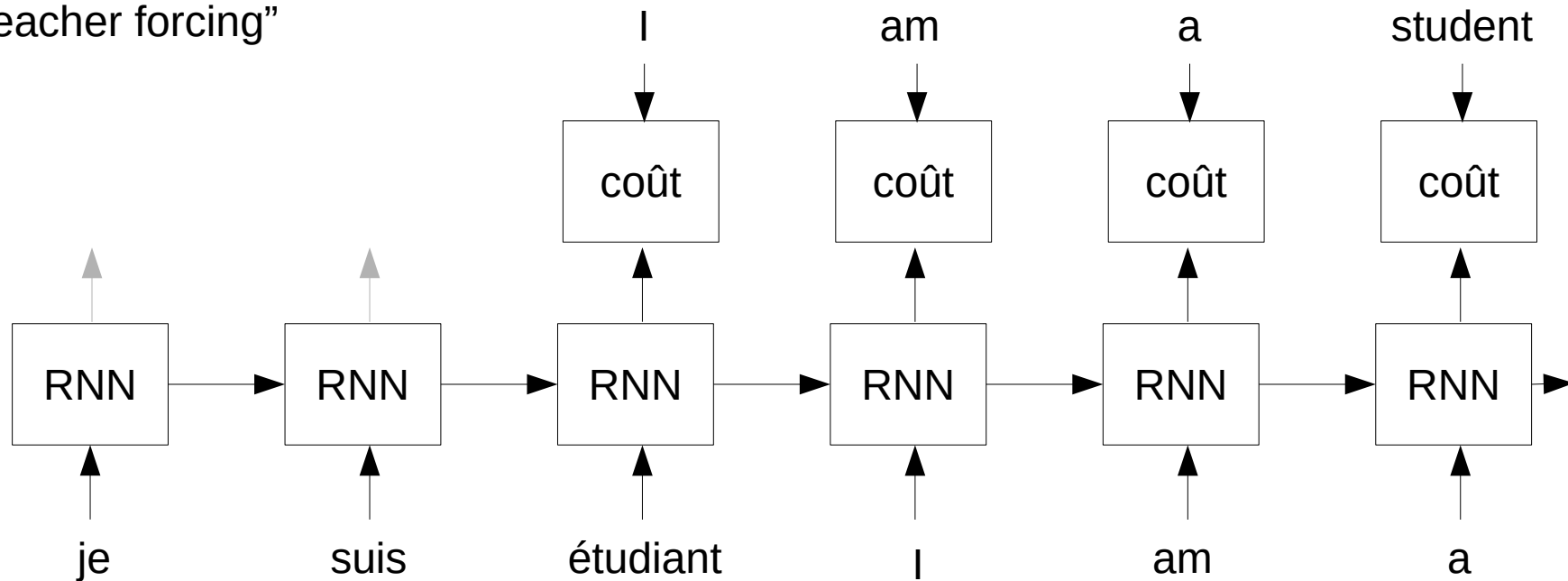
“Teacher forcing”



1)

RNN - apprentissage

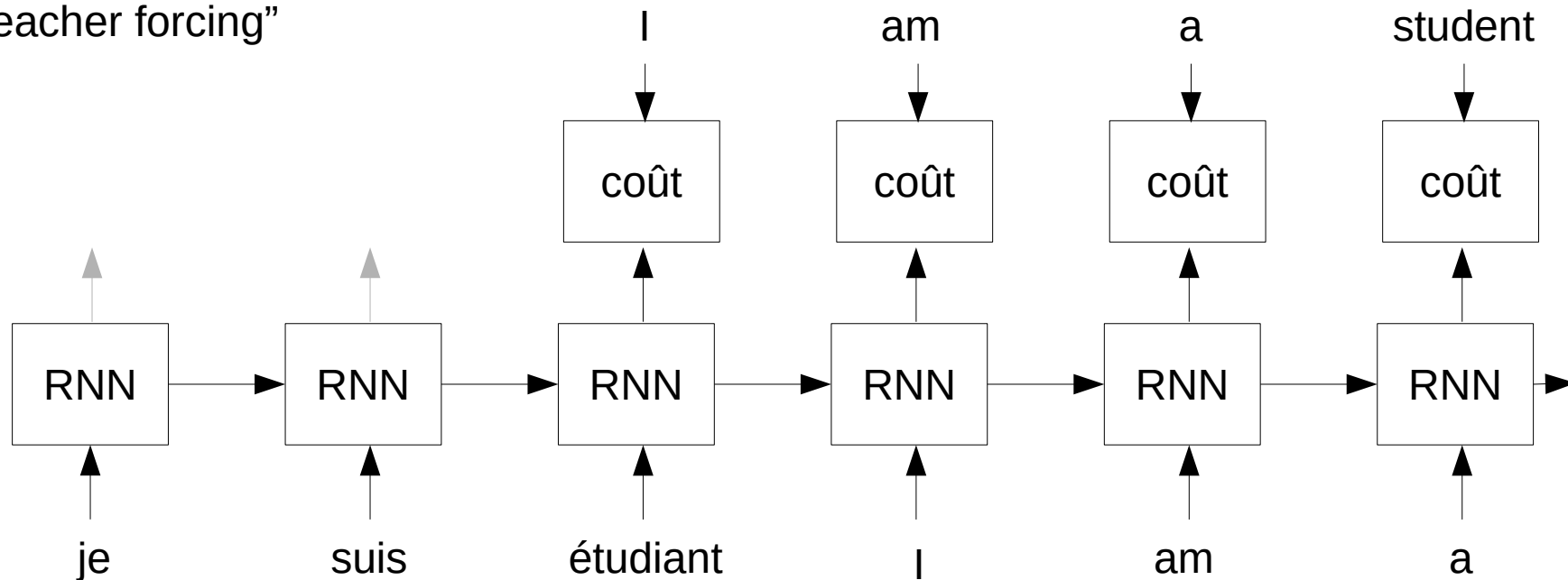
“Teacher forcing”



1)

RNN - apprentissage

“Teacher forcing”

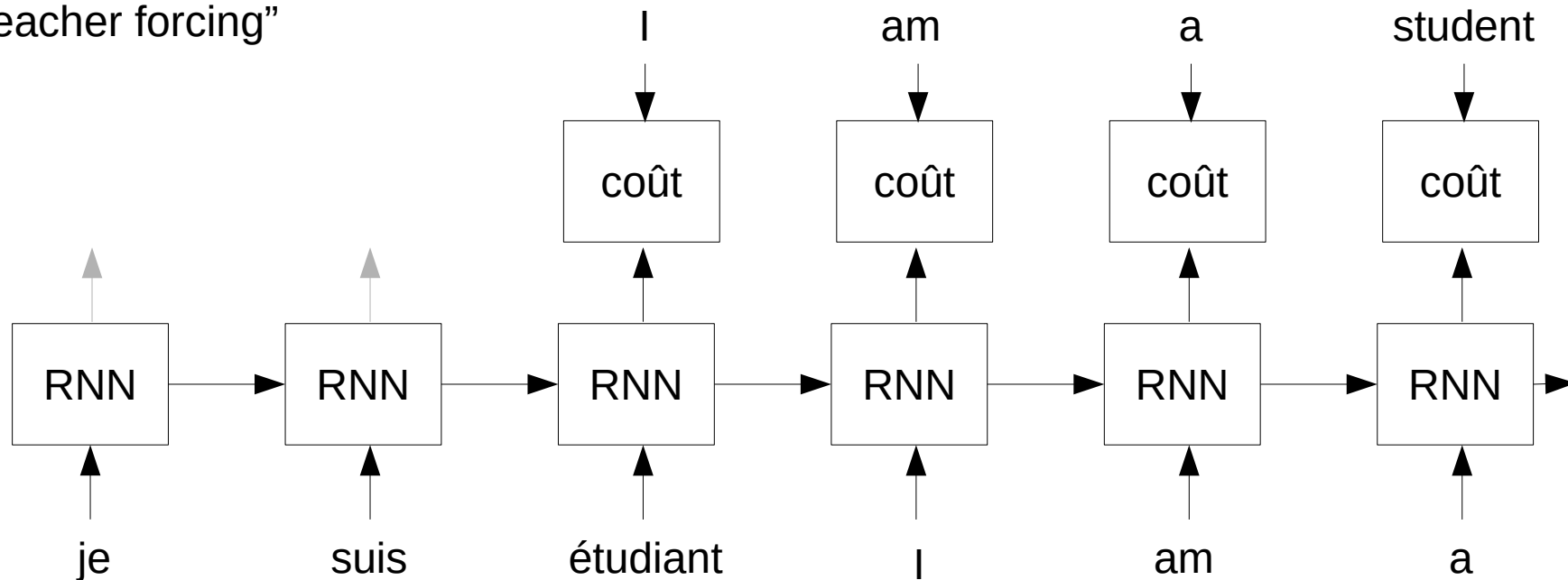


- Séquentiel (Non parallélisable) car il y a un **état caché** à propager.
- apprentissage lent (à cause de l'architecture RNN)

I)

RNN - apprentissage

“Teacher forcing”



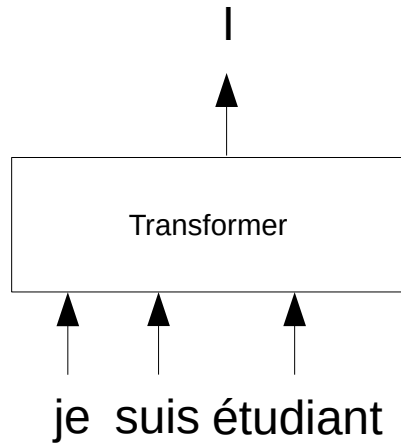
- Séquentiel (Non parallélisable) car il y a un **état caché** à propager.
→ apprentissage lent (à cause de l'architecture RNN)

- Doit apprendre à résumer à chaque instant toutes les entrées précédentes dans l'**état caché**
→ problème des longues dépendances (à cause de l'architecture RNN)

l)

Transformer (“decoder only”) - inférence

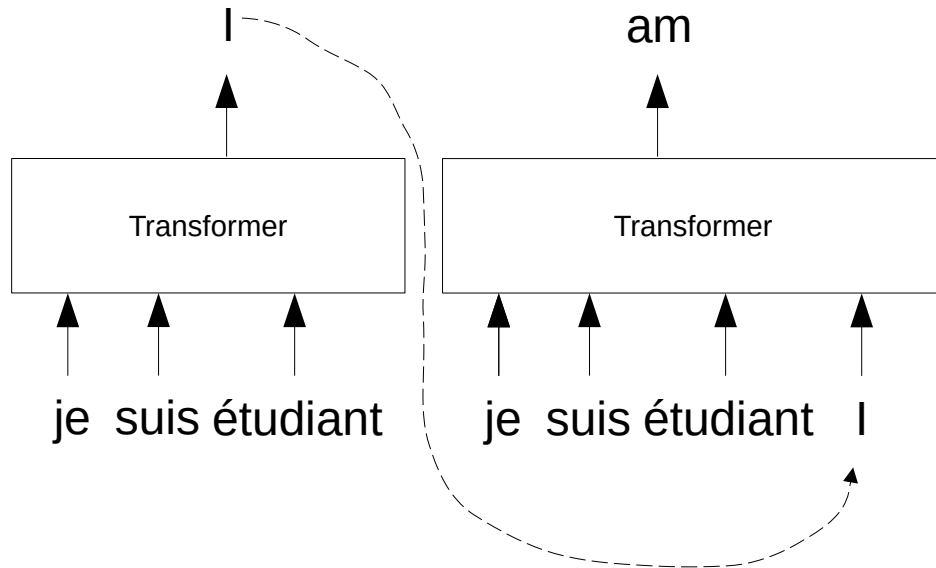
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

Transformer (“decoder only”) - inférence

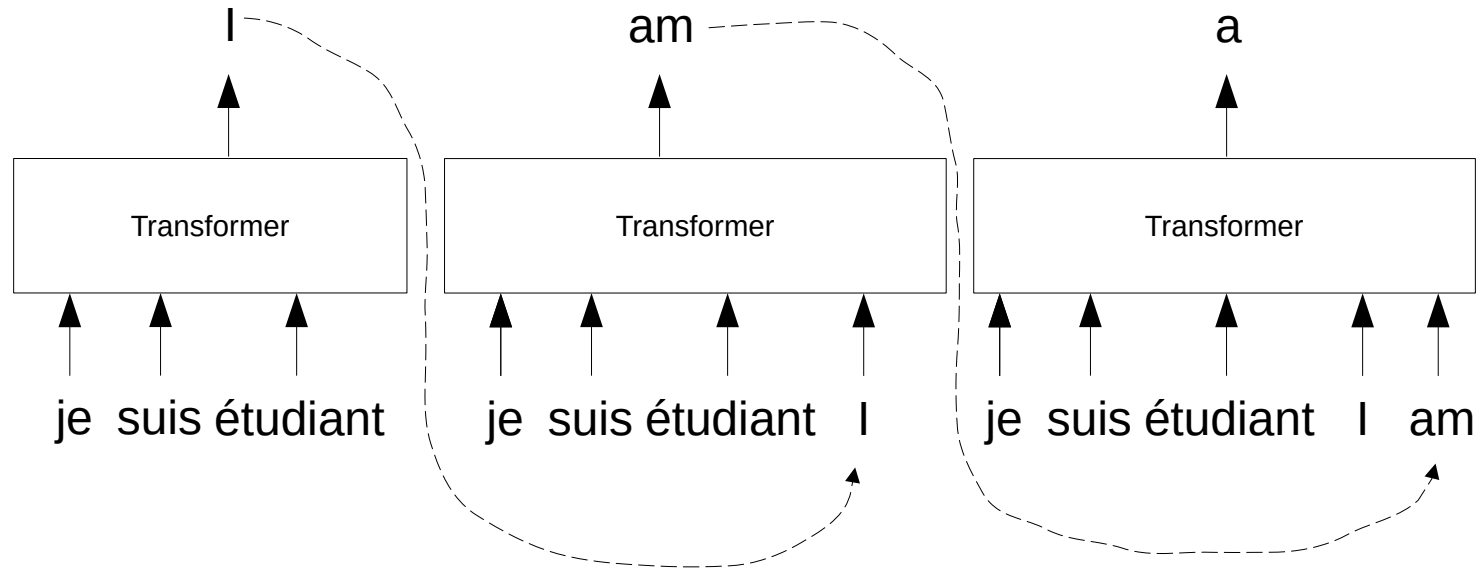
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

Transformer (“decoder only”) - inférence

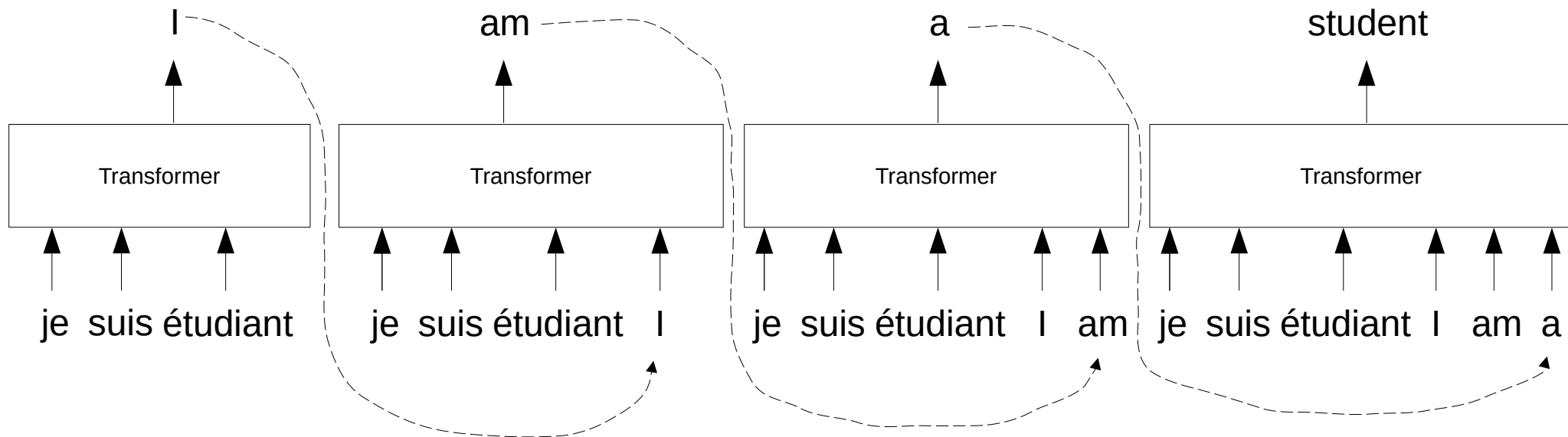
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



1)

Transformer (“decoder only”) - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

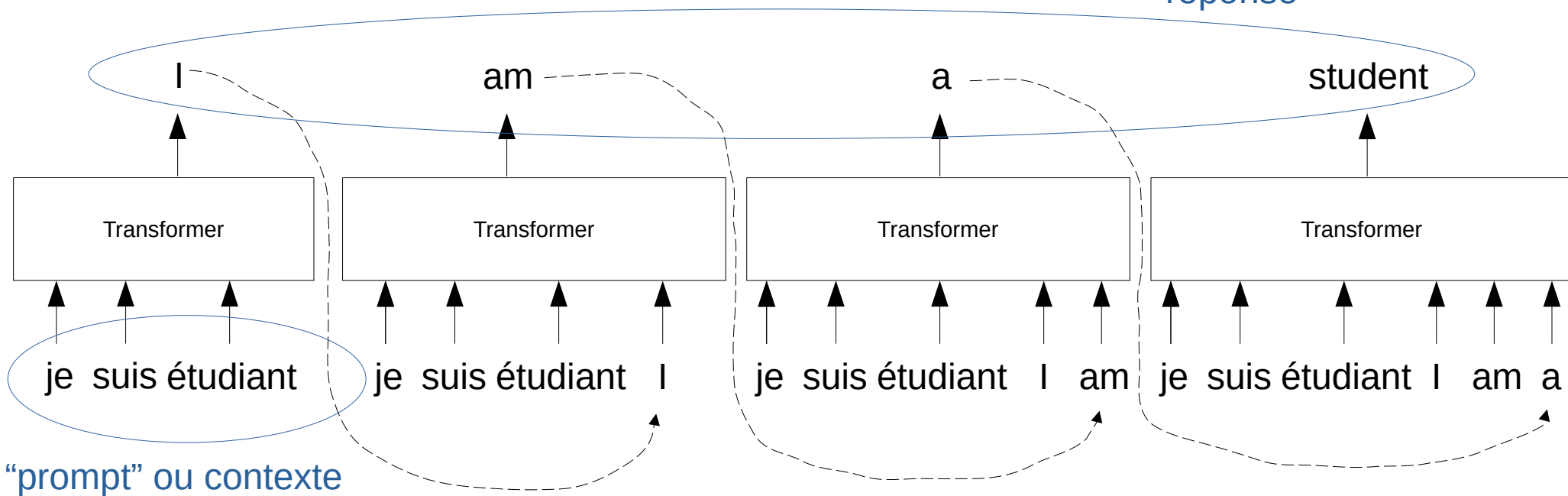


1)

Transformer (“decoder only”) - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

réponse

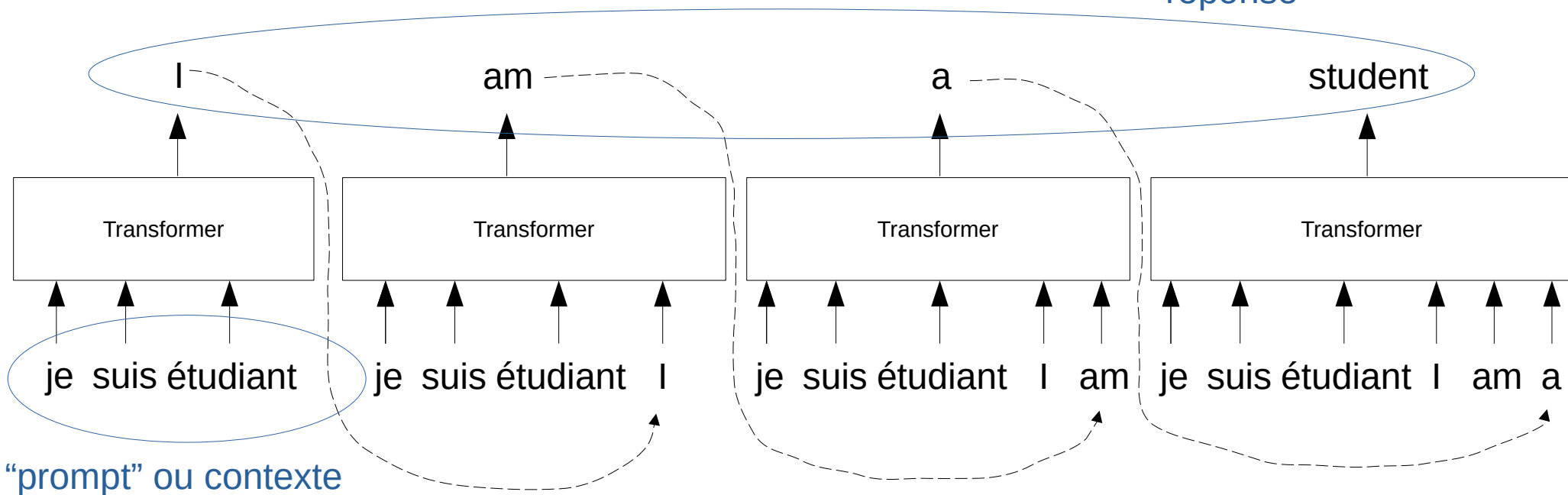


1)

Transformer (“decoder only”) - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

réponse



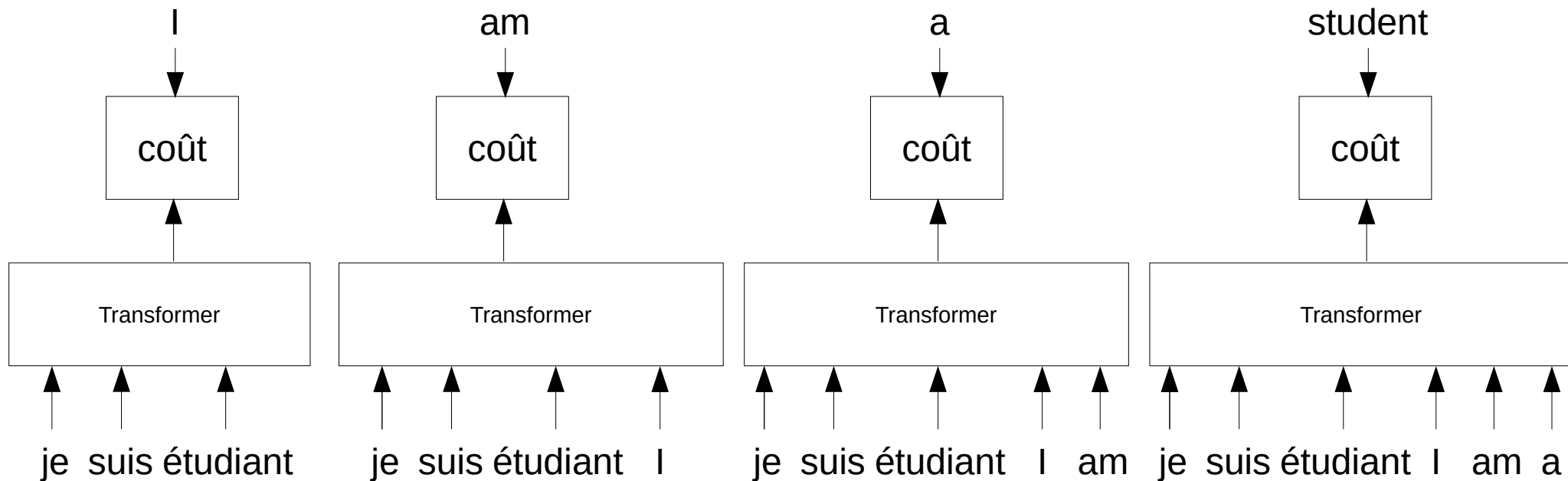
Remarque : Pas d'état caché à propager !

Inférence séquentielle (à cause du paradigme “next token prediction”)
 → Le Transformer ne règle pas ce problème (ChatGPT est séquentiel)

1)

Transformer (“decoder only”) - apprentissage

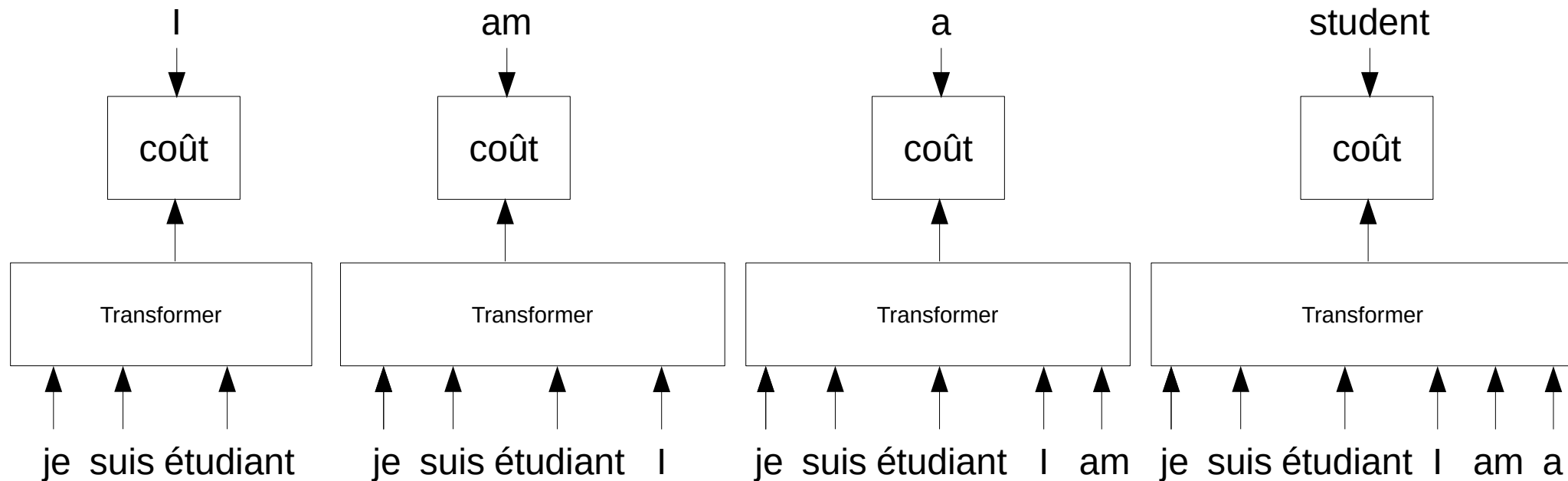
“Teacher forcing”



1)

Transformer (“decoder only”) - apprentissage

“Teacher forcing”

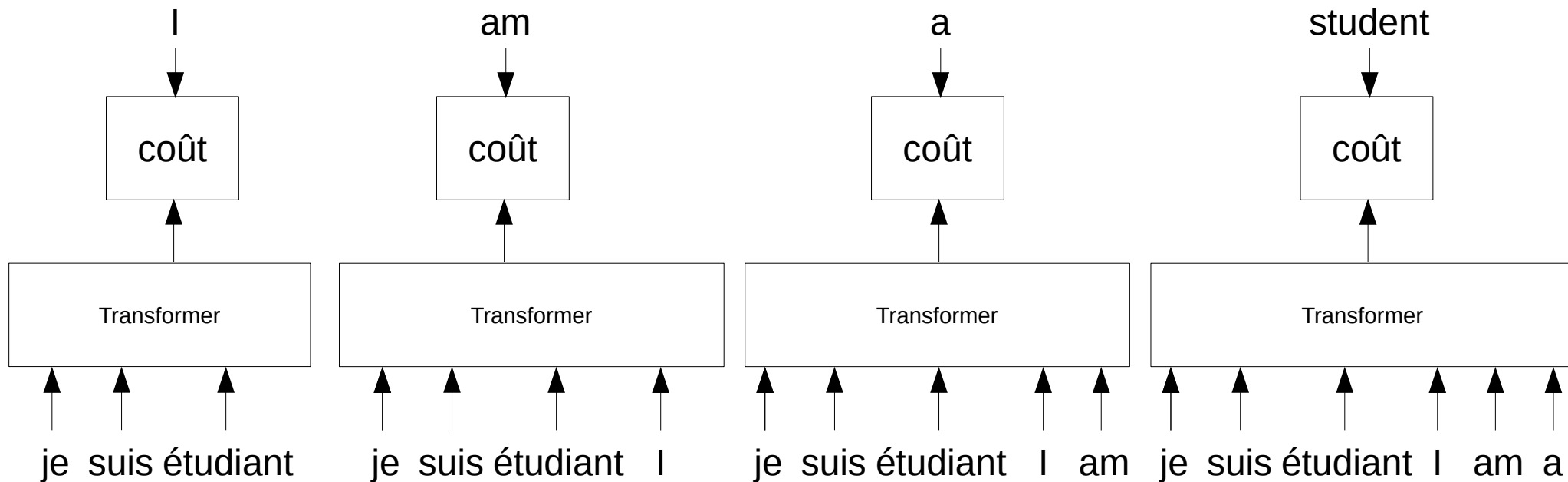


- Parallélisable à l'apprentissage car il n'y a plus d'état caché d'un instant à l'autre.

1)

Transformer (“decoder only”) - apprentissage

“Teacher forcing”



- Parallélisable à l'apprentissage car il n'y a plus d'état caché d'un instant à l'autre.

- Plus de problèmes de longue dépendance car toutes les entrées passées sont placées en entrée à chaque instant (il n'y a plus d'état caché !)

II) Couche d'attention à softmax

II)

Attention utilisant la fonction softmax

Entrées

\mathbf{x} : vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

II)

Attention utilisant la fonction softmax

Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction $\frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\sum_i \exp(\mathbf{x}\mathbf{y}_i^\top)}$

Produit scalaire + exp :

$\gg 1$ si \mathbf{x} est “attiré” par \mathbf{y}_j
 $= 1$ si \mathbf{x} orthogonal à \mathbf{y}_j
 ≈ 0 si \mathbf{x} est “repoussé” par \mathbf{y}_j

II)

Attention utilisant la fonction softmax


Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction $\frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\sum_i \exp(\mathbf{x}\mathbf{y}_i^\top)}$

Produit scalaire + exp :

$\gg 1$ si \mathbf{x} est “attiré” par \mathbf{y}_j
 $= 1$ si \mathbf{x} orthogonal à \mathbf{y}_j
 ≈ 0 si \mathbf{x} est “repoussé” par \mathbf{y}_j

\mathbf{y}_j “attire l’attention de” \mathbf{x}



II)

Attention utilisant la fonction softmax

Entrées

\mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathbf{y}_k^\top)} \quad \leftarrow \text{Softmax}$$

Attention utilisant la fonction softmax

Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathbf{y}_k^\top)} \mathbf{y}_j$$

Combinaison linéaire des $\{\mathbf{y}_i\}_{i=1 \dots N_y}$

Les poids les plus élevés de cette combinaison linéaire correspondent aux vecteurs ayant le plus “attiré l’attention” de \mathbf{x}

II)

Attention utilisant la fonction softmax

Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \sum_{j=1}^{N_y} \frac{\exp(\mathbf{xQ}(\mathbf{y}_j\mathbf{K})^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{xQ}(\mathbf{y}_k\mathbf{K})^\top)} \mathbf{y}_j$$

Pour pouvoir apprendre à “attirer l’attention” → introduction de paramètres à optimiser

Paramètres \mathbf{Q} : matrice “query” de taille $D \times L$
 \mathbf{K} : matrice “key” de taille $D \times L$

Attention utilisant la fonction softmax

Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction
$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}Q(\mathbf{y}_jK)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}Q(\mathbf{y}_kK)^\top)} \mathbf{y}_jV$$

En pratique, la combinaison linéaire est elle-même transformée linéairement, et suivie d'une connection résiduelle.

Paramètres Q : matrice "query" de taille $D \times L$
 K : matrice "key" de taille $D \times L$
 V : matrice "value" de taille $D \times D$

II)

Attention utilisant la fonction softmax

Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction
$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}Q(\mathbf{y}_jK)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}Q(\mathbf{y}_kK)^\top)} \mathbf{y}_jV$$

Paramètres Q : matrice “query” de taille $D \times L$
 K : matrice “key” de taille $D \times L$
 V : matrice “value” de taille $D \times D$

Terminologie : “Dot-product attention”, plus rarement “softmax attention”

II)

Couche d'inter-attention softmax (“Cross-attention”)

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X} : matrice de taille $N_x \times D$
 $\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D \longrightarrow \mathbf{Y} : matrice de taille $N_y \times D$

II)

Couche d'inter-attention softmax ("Cross-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X} : matrice de taille $N_x \times D$
 $\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D \longrightarrow \mathbf{Y} : matrice de taille $N_y \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X}' : matrice de taille $N_x \times D$

II)

Couche d'inter-attention softmax ("Cross-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X} : matrice de taille $N_x \times D$
 $\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D \longrightarrow \mathbf{Y} : matrice de taille $N_y \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X}' : matrice de taille $N_x \times D$

$$\mathbf{M} = \mathbf{XQ}(\mathbf{YK})^\top$$

$$\mathbf{S} = \text{softmax}(\mathbf{M}, \text{dim}=1)$$

$$\mathbf{X}' = \mathbf{X} + \mathbf{SYV}$$

Calcul réalisé en parallèle sur
les vecteurs $\{\mathbf{x}_i\}_{i=1\dots N_x}$

II)

Couche d'inter-attention softmax ("Cross-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X} : matrice de taille $N_x \times D$
 $\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D \longrightarrow \mathbf{Y} : matrice de taille $N_y \times D$

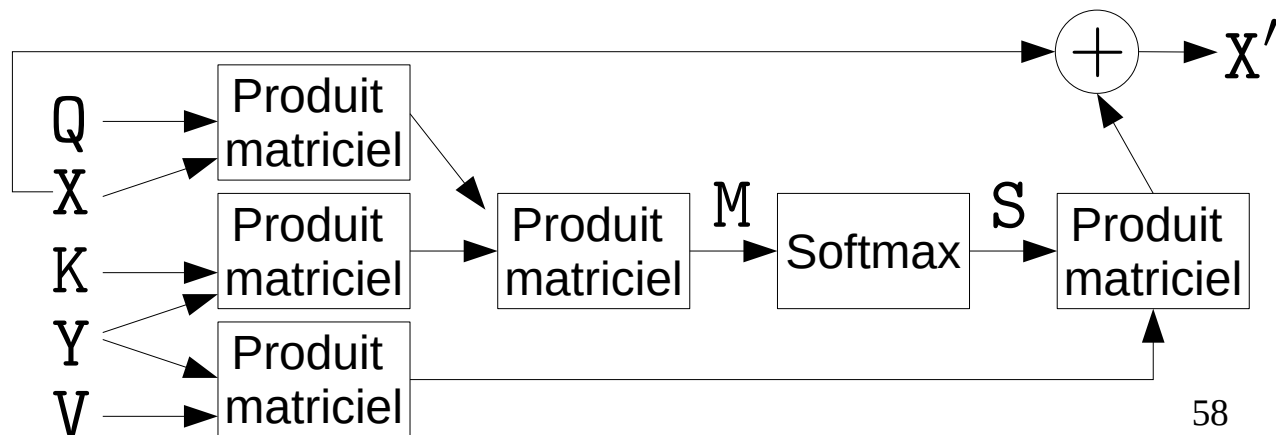
Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X}' : matrice de taille $N_x \times D$

$$\mathbf{M} = \mathbf{XQ}(\mathbf{YK})^\top$$

$$\mathbf{S} = \text{softmax}(\mathbf{M}, \text{dim}=1)$$

$$\mathbf{X}' = \mathbf{X} + \mathbf{SYV}$$

Calcul réalisé en parallèle sur
les vecteurs $\{\mathbf{x}_i\}_{i=1\dots N_x}$



II)

Couche d'inter-attention softmax ("Cross-attention") (suite)

$$\mathbf{X} : N_x \times D \quad \mathbf{Y} : N_y \times D \quad \mathbf{X}' : N_x \times D$$

$$\mathbf{M} = \mathbf{XQ}(\mathbf{YK})^\top : N_x \times N_y \quad \mathbf{S} = \text{softmax}(\mathbf{M}, \text{dim}=1) : N_x \times N_y$$

II)

Couche d'inter-attention softmax ("Cross-attention") (suite)

$$\mathbf{X} : N_x \times D \quad \mathbf{Y} : N_y \times D \quad \mathbf{X}' : N_x \times D$$

$$\mathbf{M} = \mathbf{XQ}(\mathbf{YK})^\top : N_x \times N_y$$

$$\mathbf{S} = \text{softmax}(\mathbf{M}, \text{dim}=1) : N_x \times N_y$$

Calcul

- Nombre d'opérations
potentiellement très élevé

+ Parallélisable

Stockage

- Mémoire requise pour stocker \mathbf{M} et \mathbf{S}
potentiellement très élevée

II)

Couche d'inter-attention softmax ("Cross-attention") (suite)

$$\mathbf{X} : N_x \times D \quad \mathbf{Y} : N_y \times D \quad \mathbf{X}' : N_x \times D$$

$$\mathbf{M} = \mathbf{XQ}(\mathbf{YK})^\top : N_x \times N_y \quad \mathbf{S} = \text{softmax}(\mathbf{M}, \text{dim}=1) : N_x \times N_y$$

Calcul

- Nombre d'opérations
potentiellement très élevé

+ Parallélisable

Stockage

- Mémoire requise pour stocker \mathbf{M} et \mathbf{S}
potentiellement très élevée

Exemple : $N_x = N_y = 640 \times 480 \approx 3.10^5$ pixels
 $N_x \times N_y \approx 9.10^{10}$ flottants (32 bits) \rightarrow 360Go

II)

Cas particulier : Couche d'auto-attention ("Self-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X} : matrice de taille $N_x \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X}' : matrice de taille $N_x \times D$

II)

Cas particulier : Couche d'auto-attention ("Self-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X} : matrice de taille $N_x \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D \longrightarrow \mathbf{X}' : matrice de taille $N_x \times D$

$$\mathbf{M} = \mathbf{XQ}(\mathbf{XK})^\top : N_x \times N_x$$

$$\mathbf{S} = \text{softmax}(\mathbf{M}, \text{dim}=1) : N_x \times N_x$$

$$\mathbf{X}' = \mathbf{X} + \mathbf{SXV} : N_x \times D$$

Transfert d'information depuis \mathbf{X} vers lui-même, le tout stocké dans \mathbf{X}'

II)

Attention softmax à têtes multiples

"Multi-head dot-product attention"

Entrées

\mathbf{x} : vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

1 tête = 1 façon
"d'attirer l'attention"

$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}Q(\mathbf{y}_jK)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}Q(\mathbf{y}_kK)^\top)} \mathbf{y}_jV$$

II)

Attention softmax à têtes multiples

"Multi-head dot-product attention"

Entrées

\mathbf{x} : vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

1 tête = 1 façon
"d'attirer l'attention"

$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}Q(\mathbf{y}_jK)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}Q(\mathbf{y}_kK)^\top)} \mathbf{y}_jV$$

H têtes = H façons
"d'attirer l'attention"

$$\mathbf{x}' = \mathbf{x} + \sum_{h=1}^H \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}Q_h(\mathbf{y}_jK_h)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}Q_h(\mathbf{y}_kK_h)^\top)} \mathbf{y}_jV_h$$

II)

Attention softmax à têtes multiples (suite)

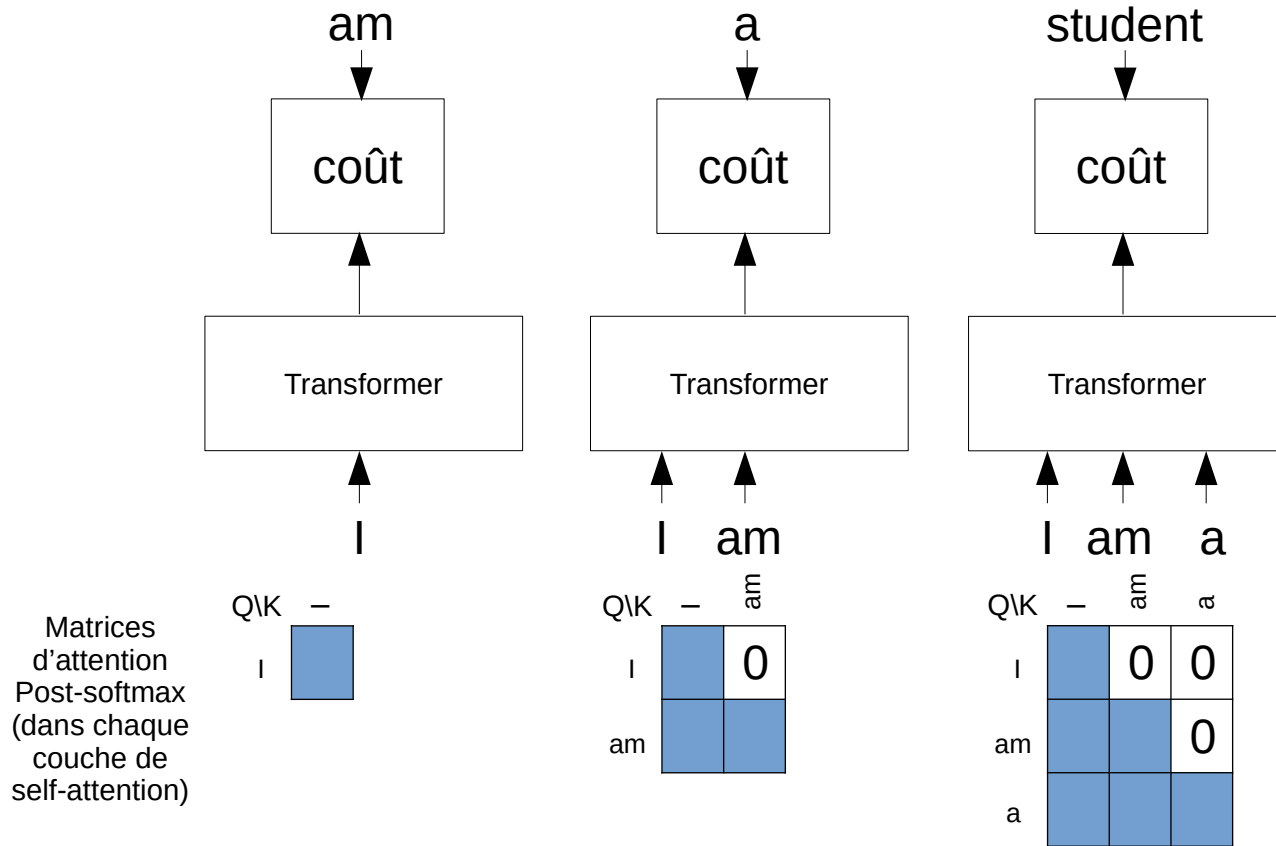
"Multi-head dot-product attention"

Entrées \mathbf{x} : vecteur de dimension $1 \times D$
 $\{\mathbf{y}_i\}_{i=1 \dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction
$$\mathbf{x}' = \mathbf{x} + \sum_{h=1}^H \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x} \mathbf{Q}_h (\mathbf{y}_j \mathbf{K}_h)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x} \mathbf{Q}_h (\mathbf{y}_k \mathbf{K}_h)^\top)} \mathbf{y}_j \mathbf{V}_h \mathbf{W}_h$$

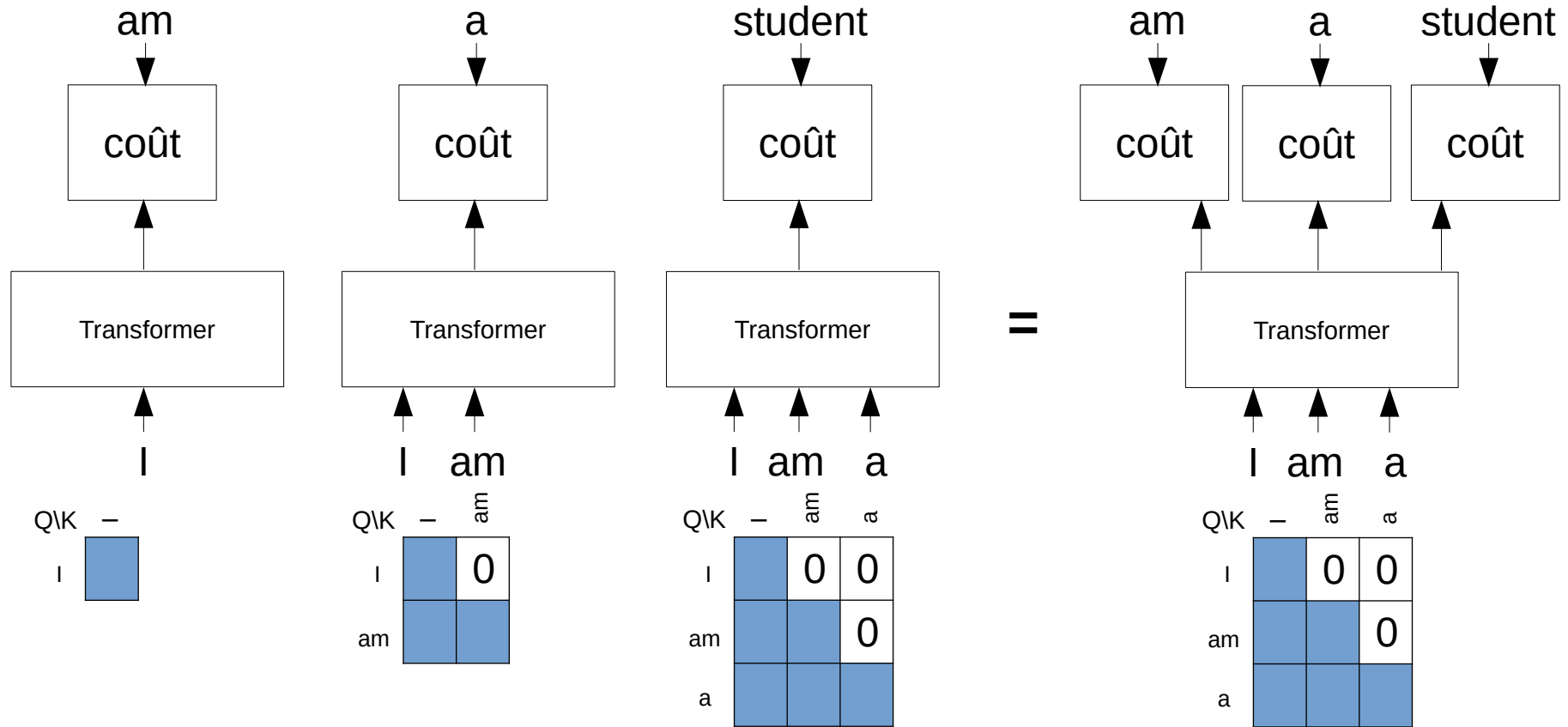
Paramètres $\{\mathbf{Q}_h\}_{h=1 \dots H}$: matrices "query" de taille $D \times L$
 $\{\mathbf{K}_h\}_{h=1 \dots H}$: matrices "key" de taille $D \times L$
 $\{\mathbf{V}_h\}_{h=1 \dots H}$: matrices "value" de taille $D \times L$
 $\{\mathbf{W}_h\}_{h=1 \dots H}$: matrices "output" de taille $L \times D$

“Masked self-attention”



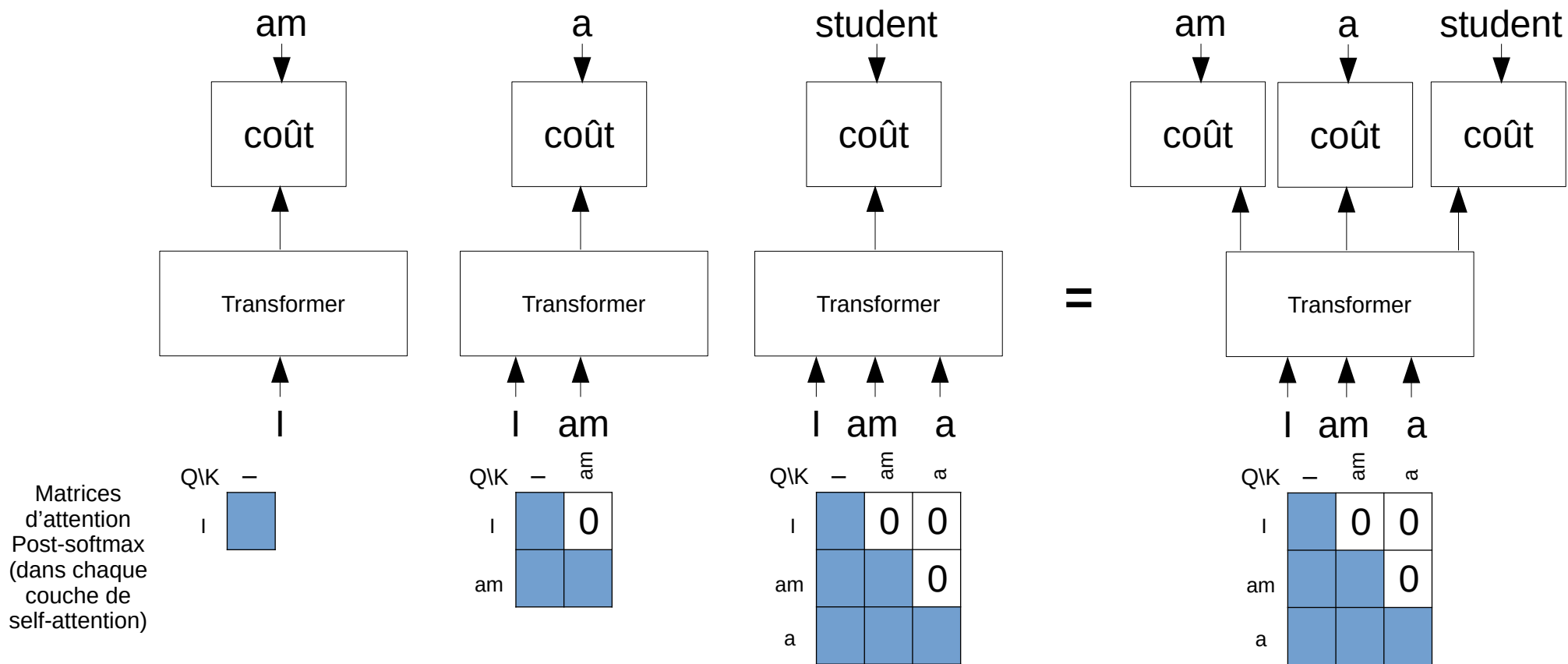
II)

“Masked self-attention”



II)

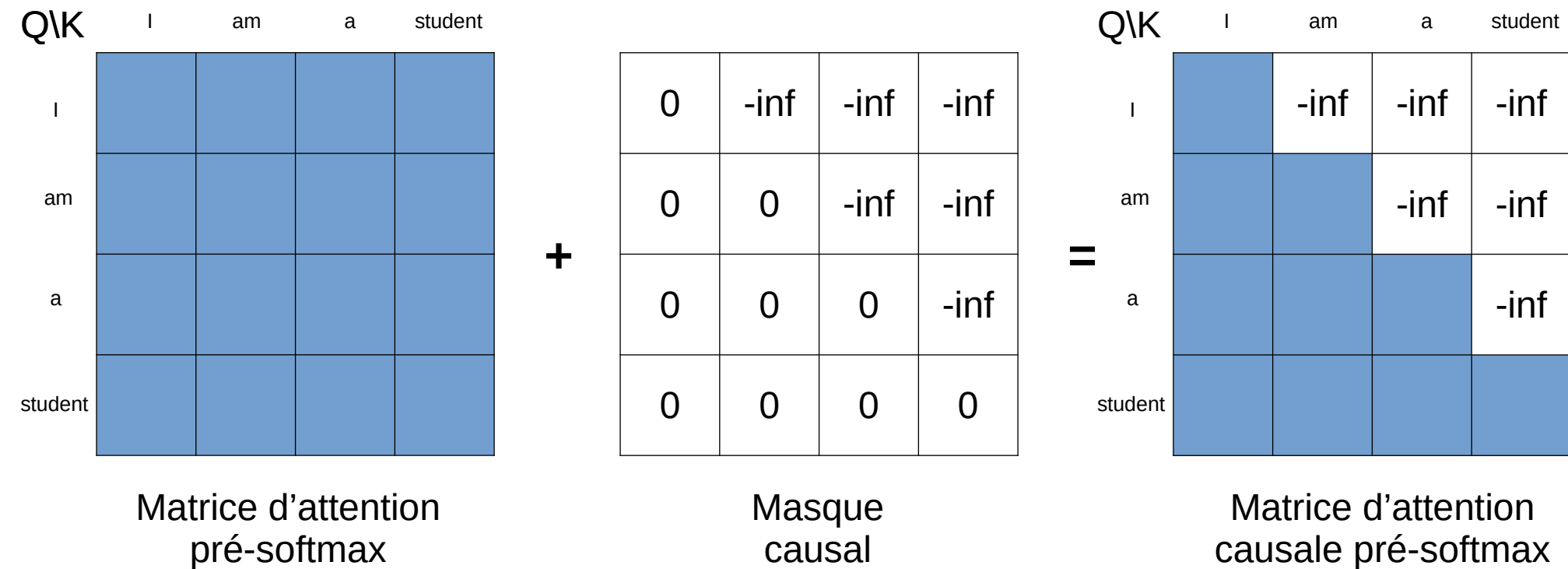
“Masked self-attention”



Masquer le “futur” permet un apprentissage très efficace (et l'utilisation du “KV cache” à l'inférence).

II)

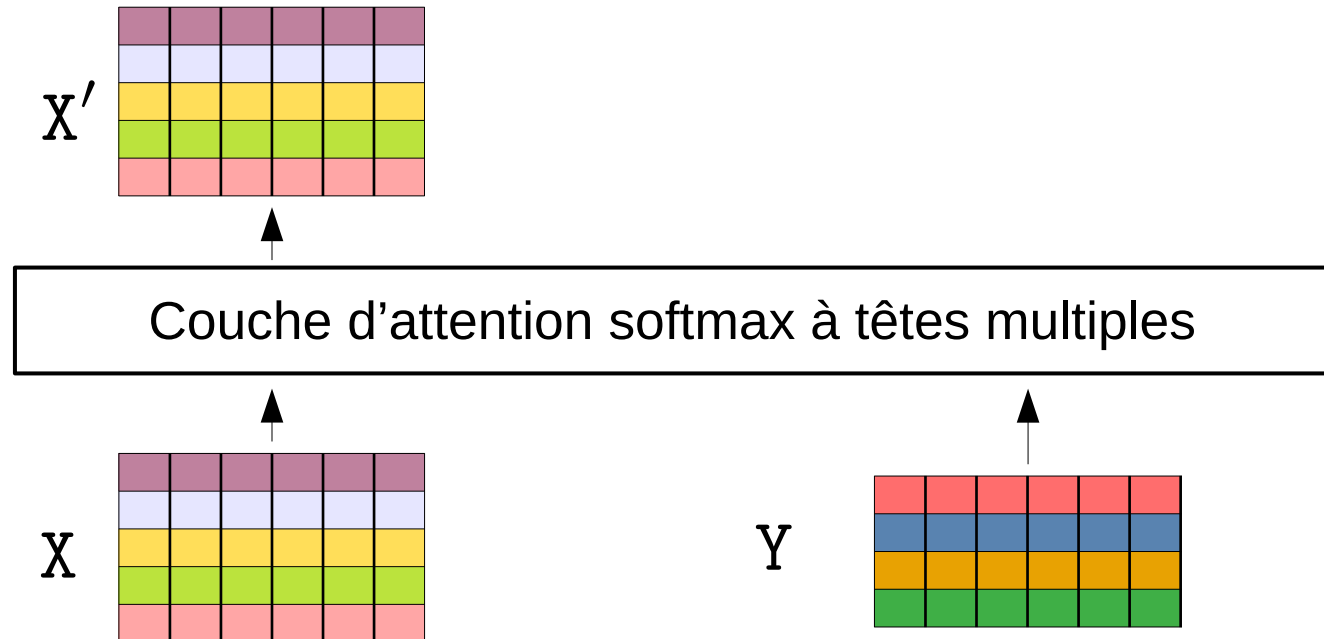
“Masked self-attention” (suite)



II)

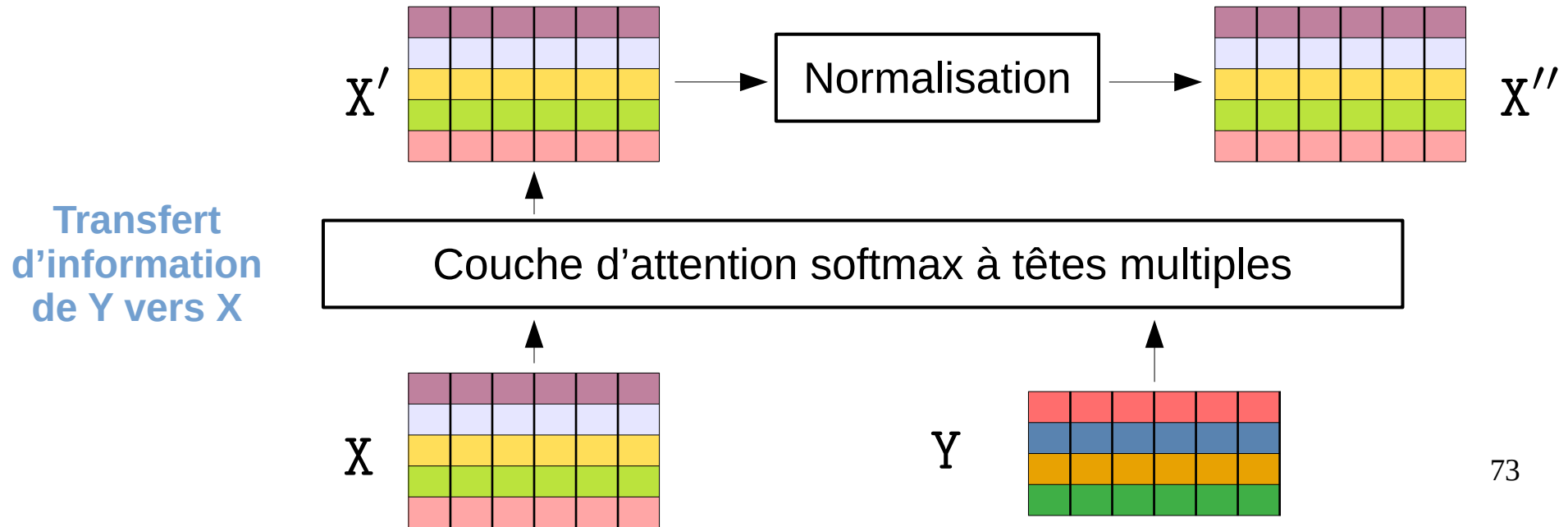
Bloc d'attention "classique"

Transfert
d'information
de Y vers X



II)

Bloc d'attention "classique"

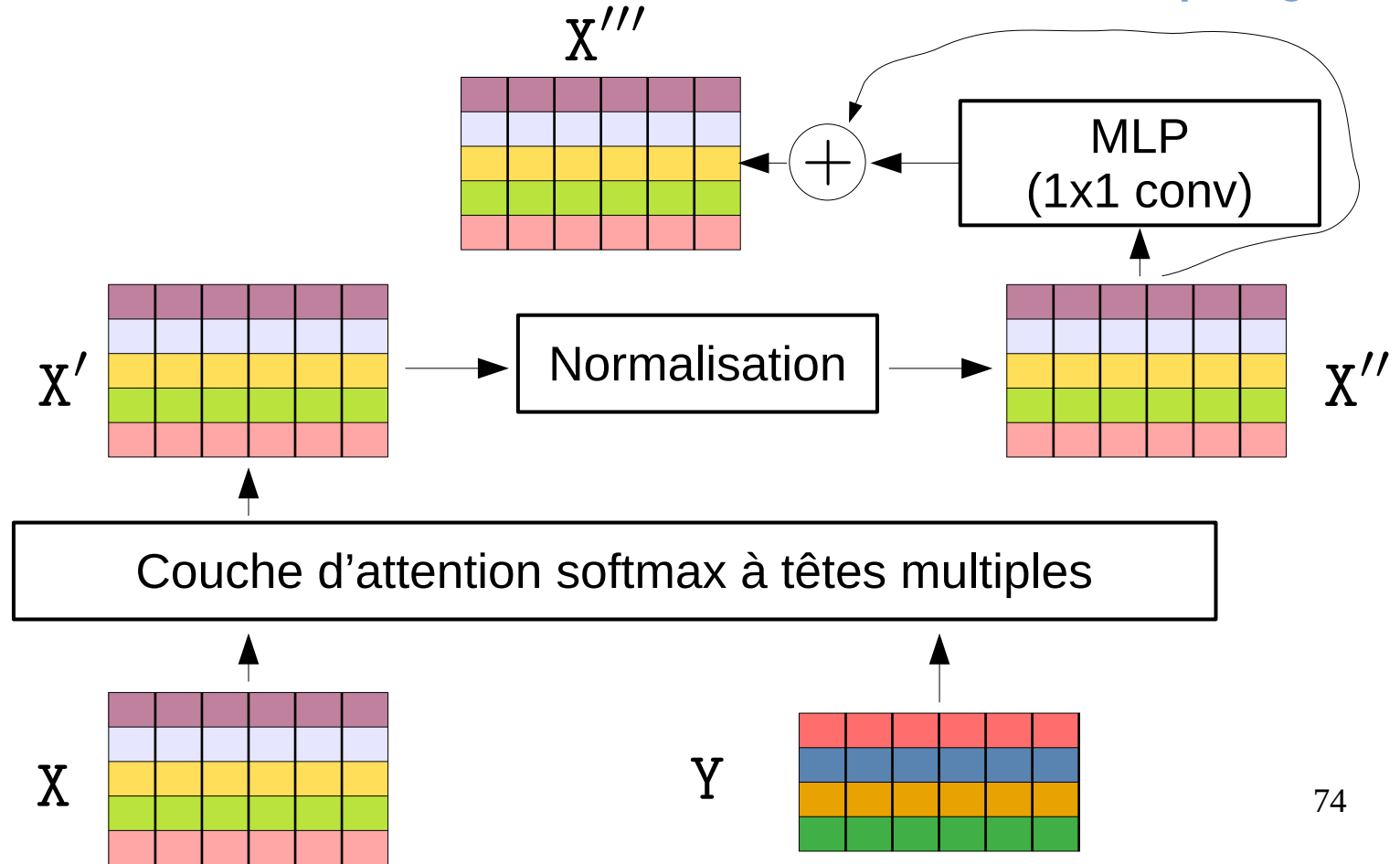


II)

Bloc d'attention "classique"

Transformation
non-linéaire de
chaque ligne

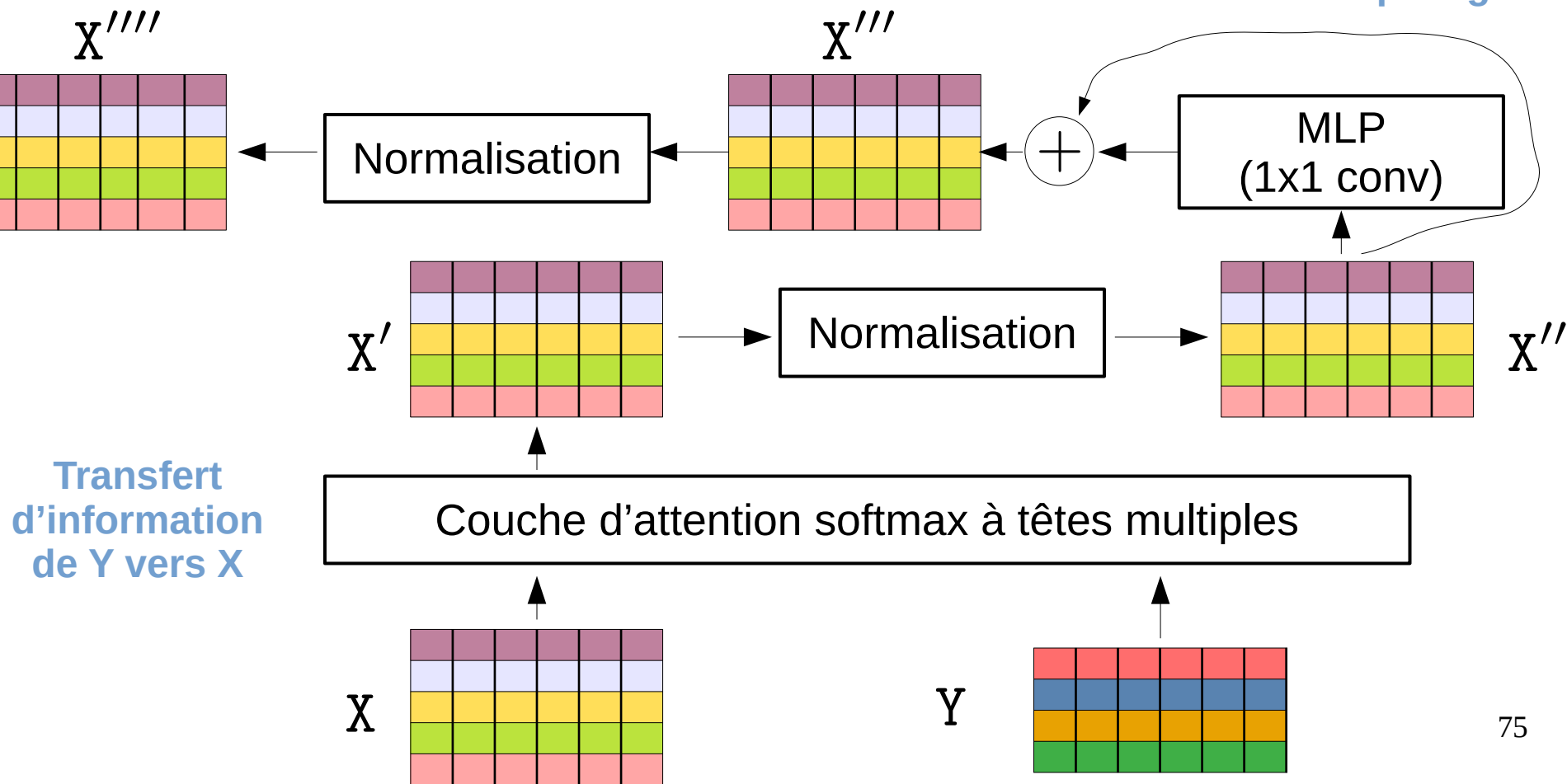
Transfert
d'information
de Y vers X



II)

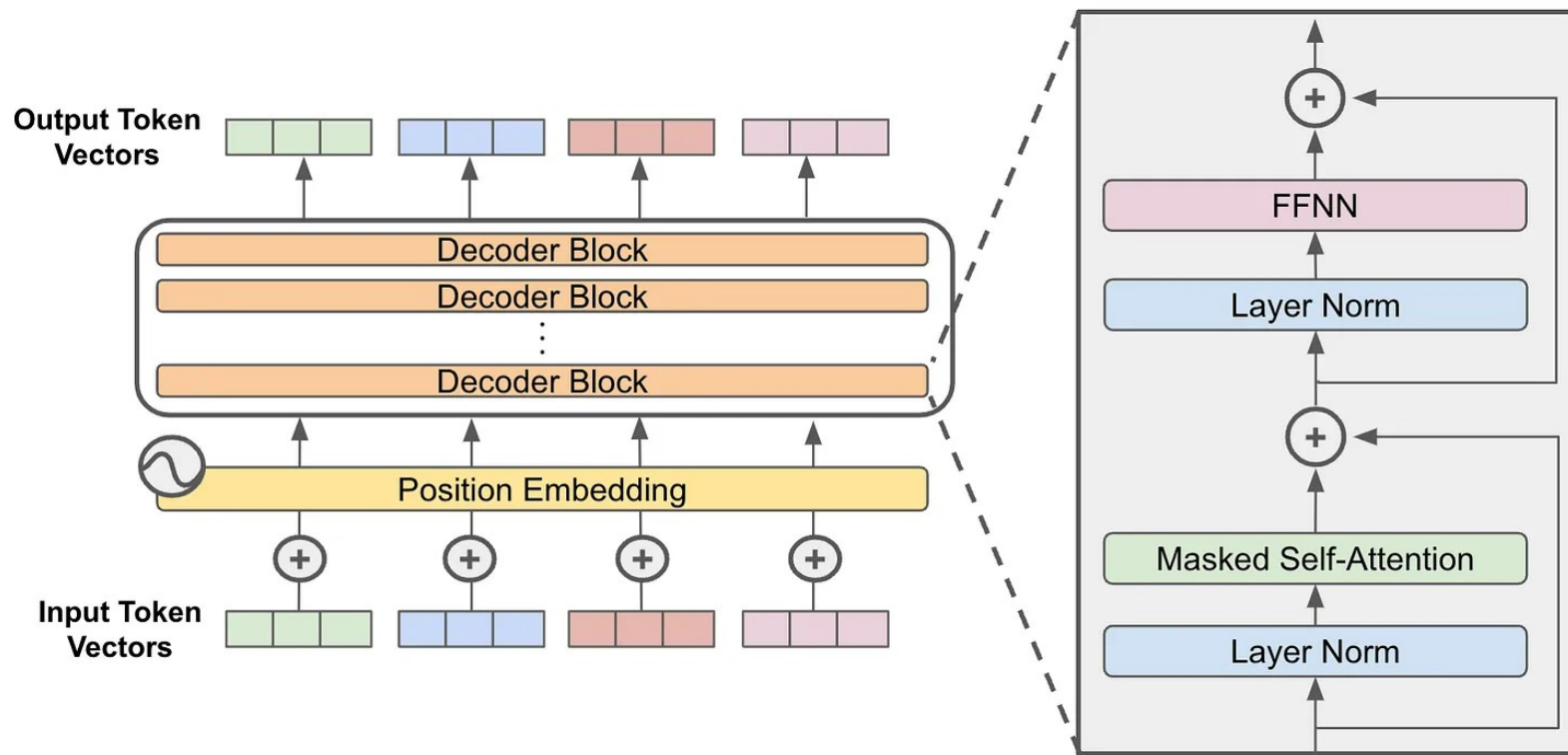
Bloc d'attention "classique"

Transformation
non-linéaire de
chaque ligne



II)

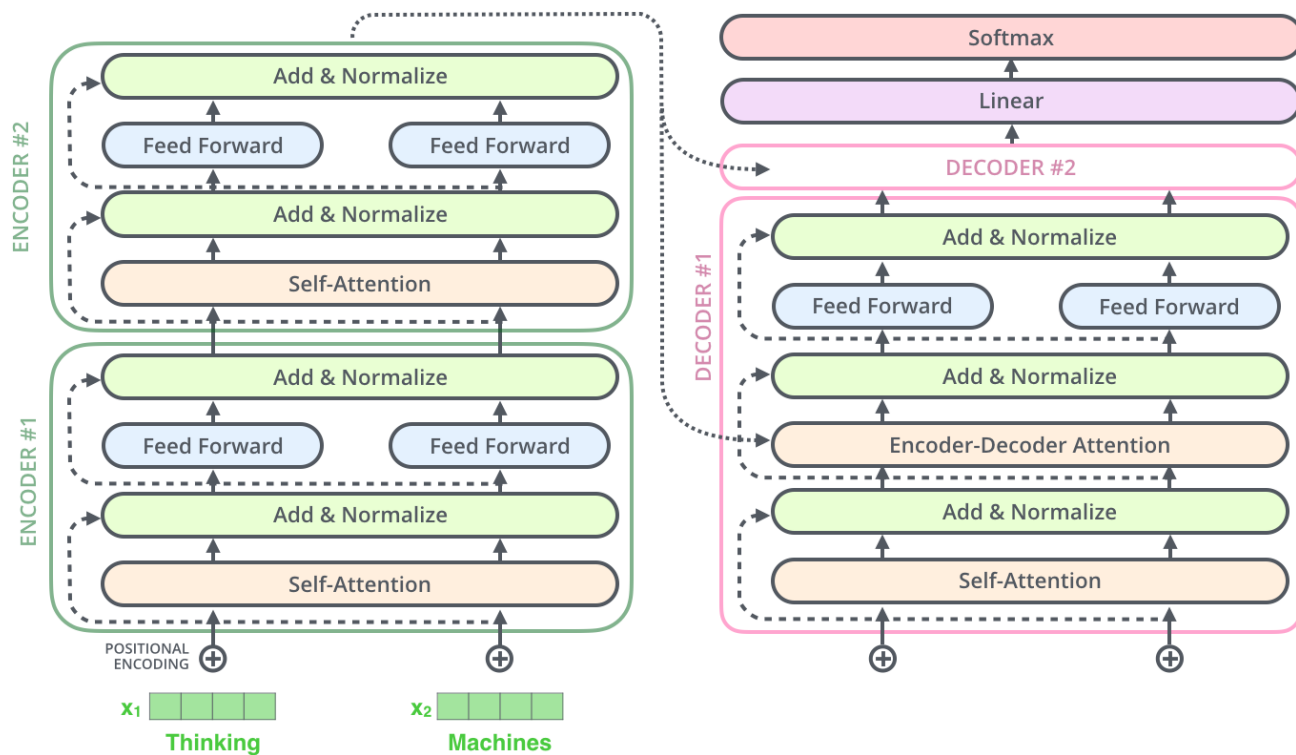
Vue détaillée d'un Transformer “Decoder-Only”



Remarque : pas de “cross-attention”, seulement des (masked) “self-attention”

II)

Vue détaillée d'un Transformer “Encoder-Decoder”

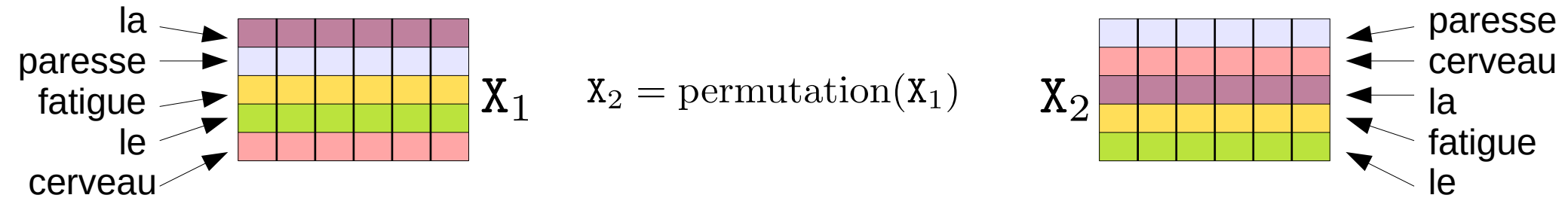


Remarque : “Encoder-Decoder Attention” = “cross-attention”

III) Équivariance par permutation et encodage de la position

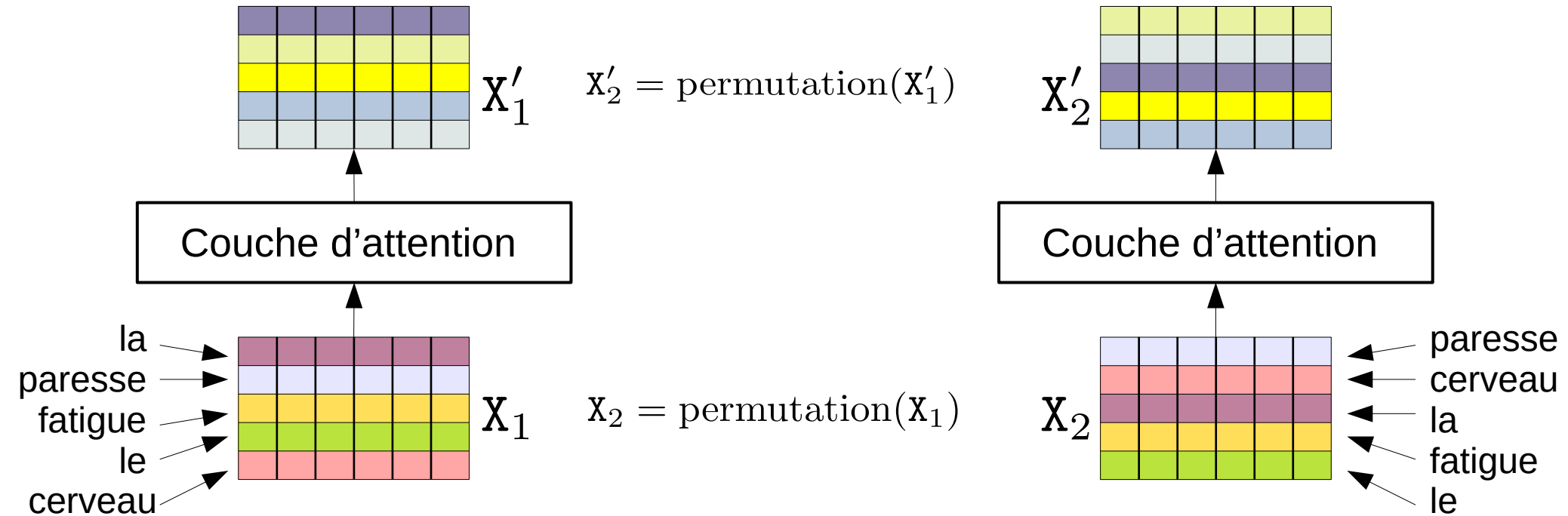
III)

Équivariance par permutation



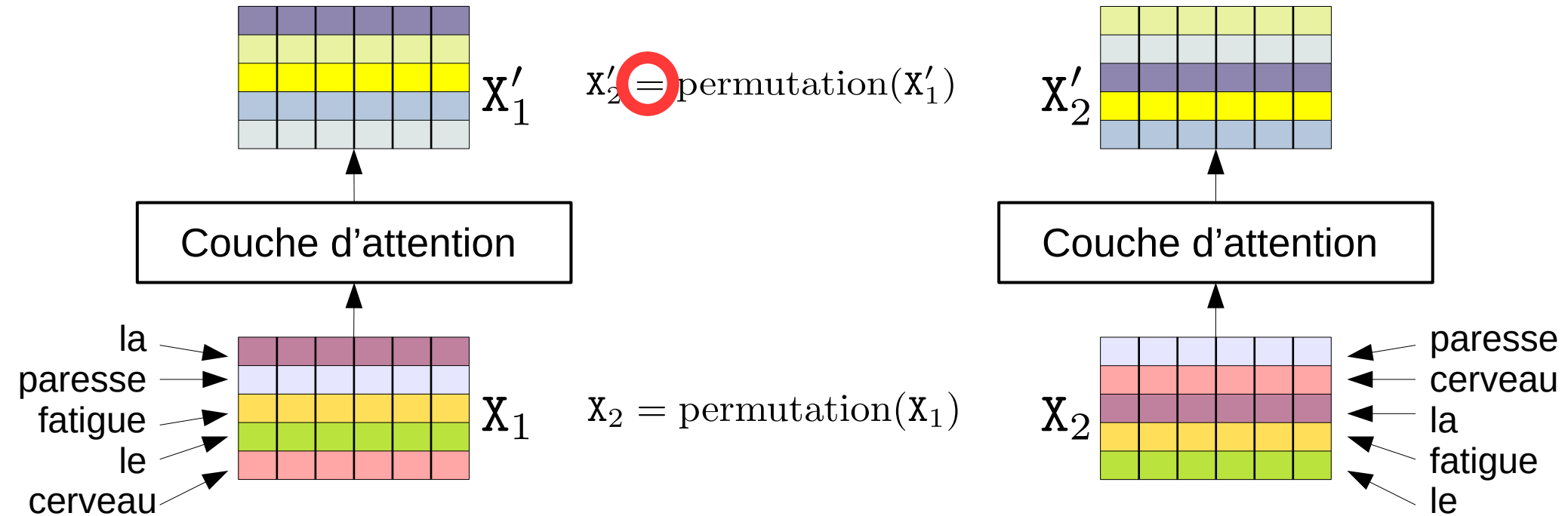
III)

Équivariance par permutation



III)

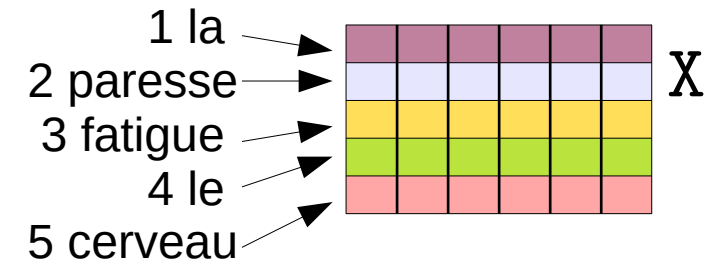
Équivariance par permutation



**Pour un ensemble ordonné (phrase, signal, image, etc.)
→ nécessité d'encoder la position**

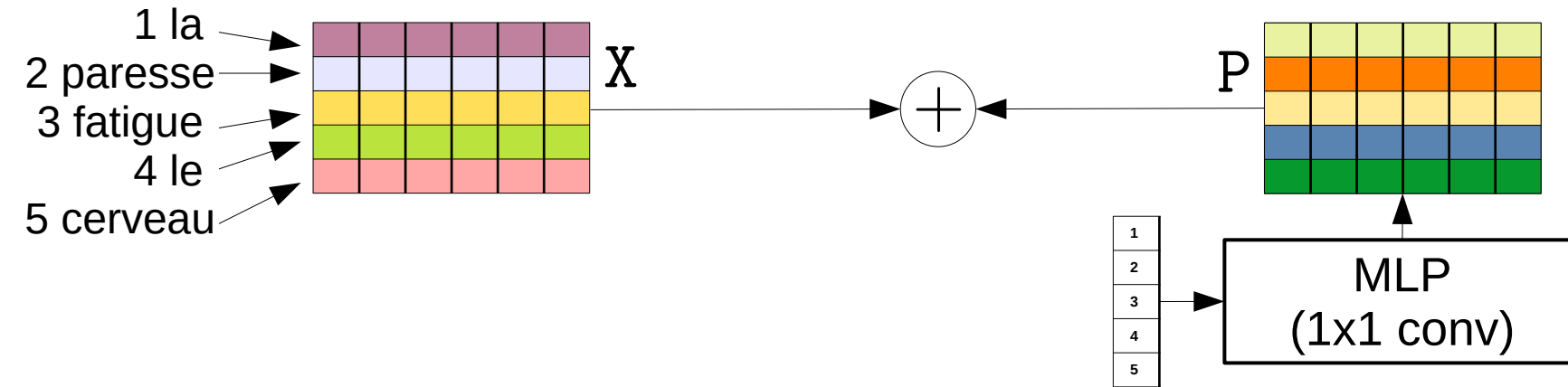
III)

Encodage de la position



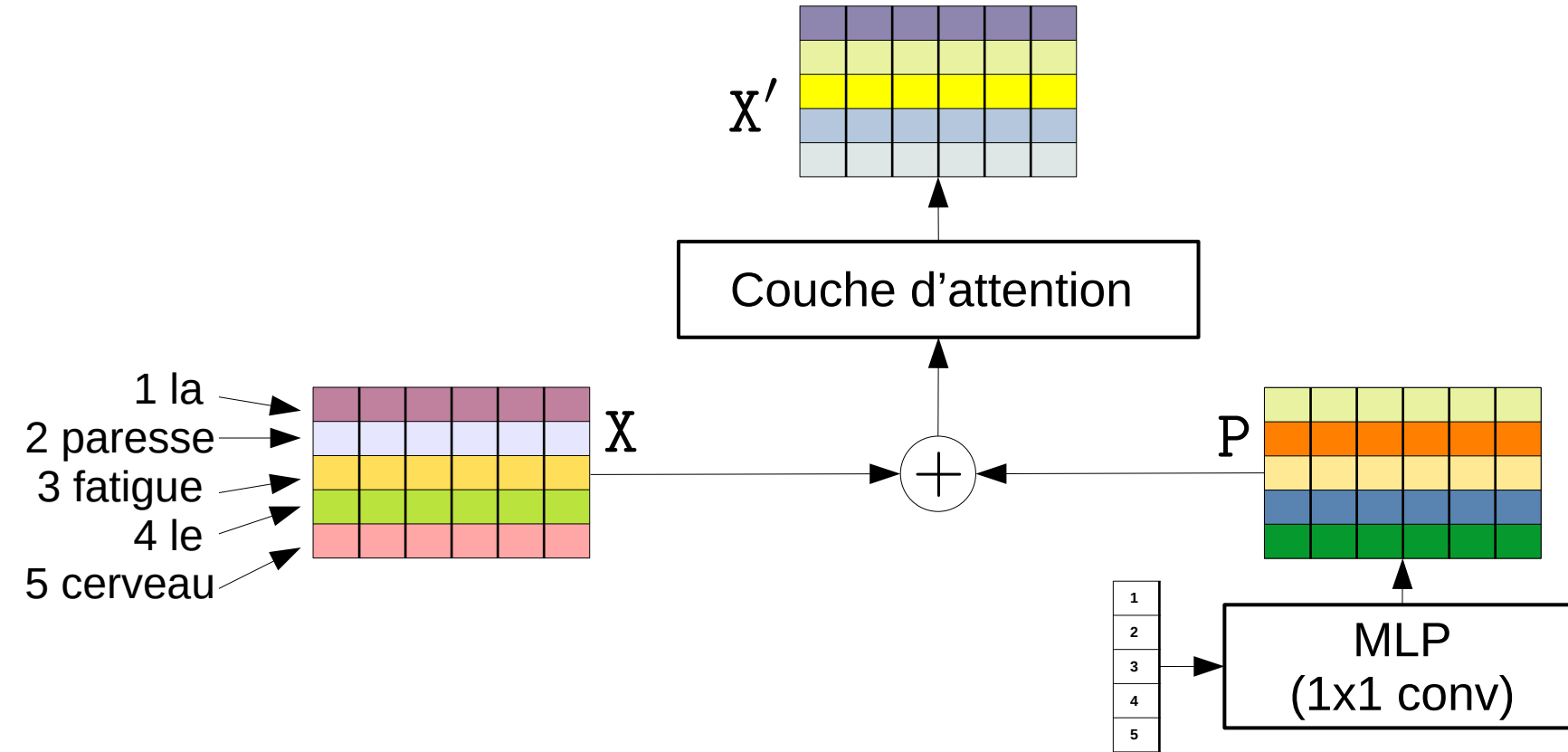
III)

Encodage de la position



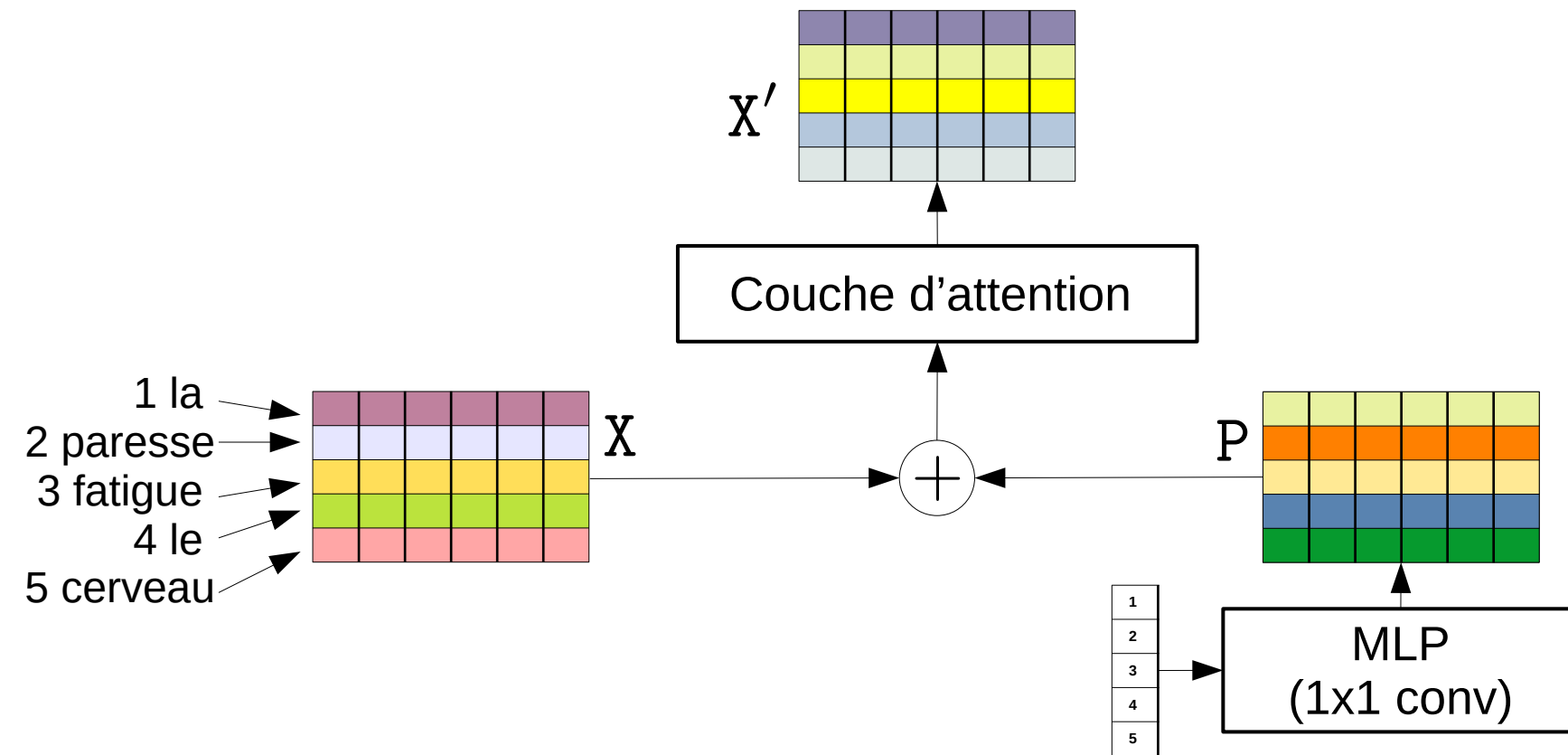
III)

Encodage de la position



III)

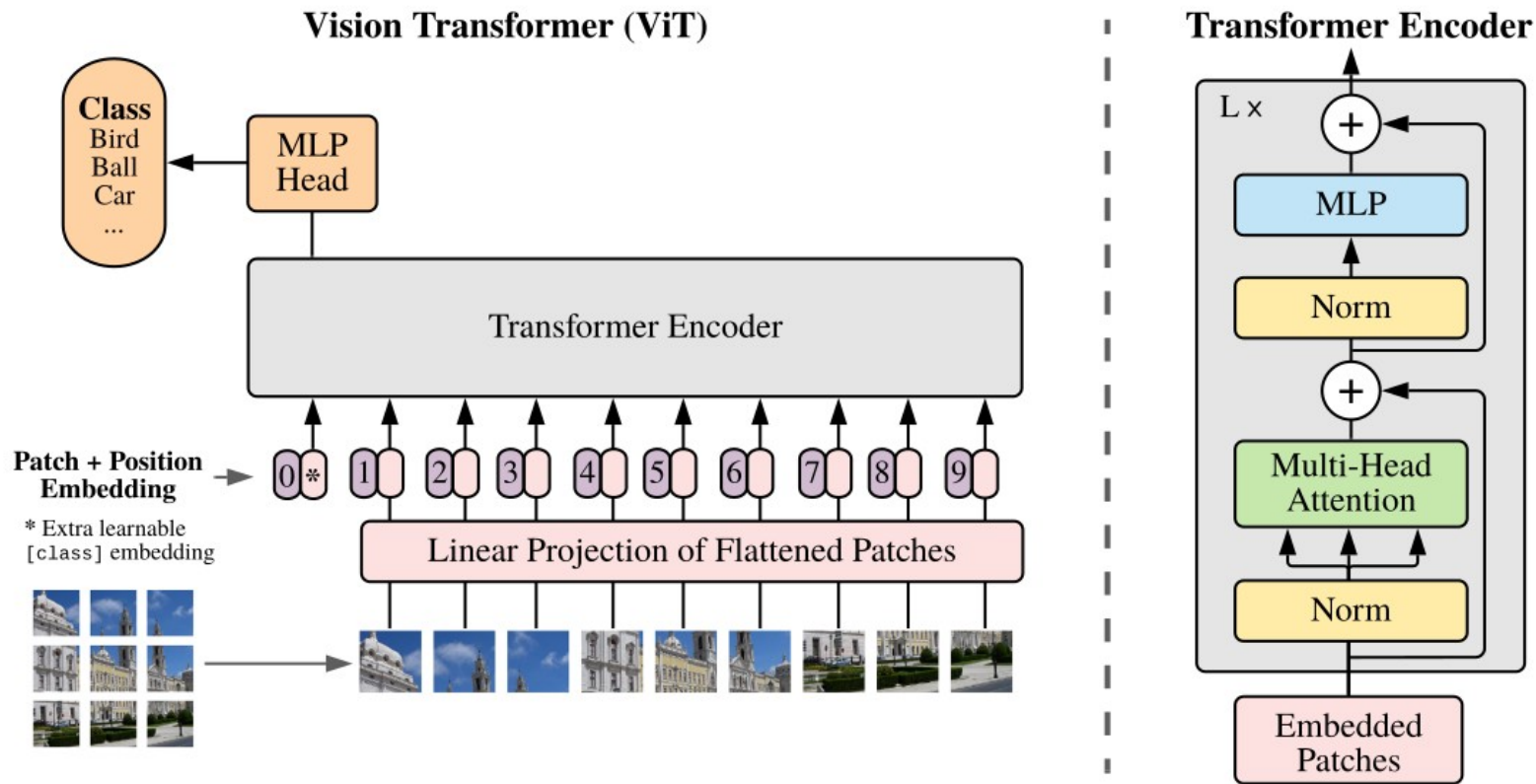
Encodage de la position



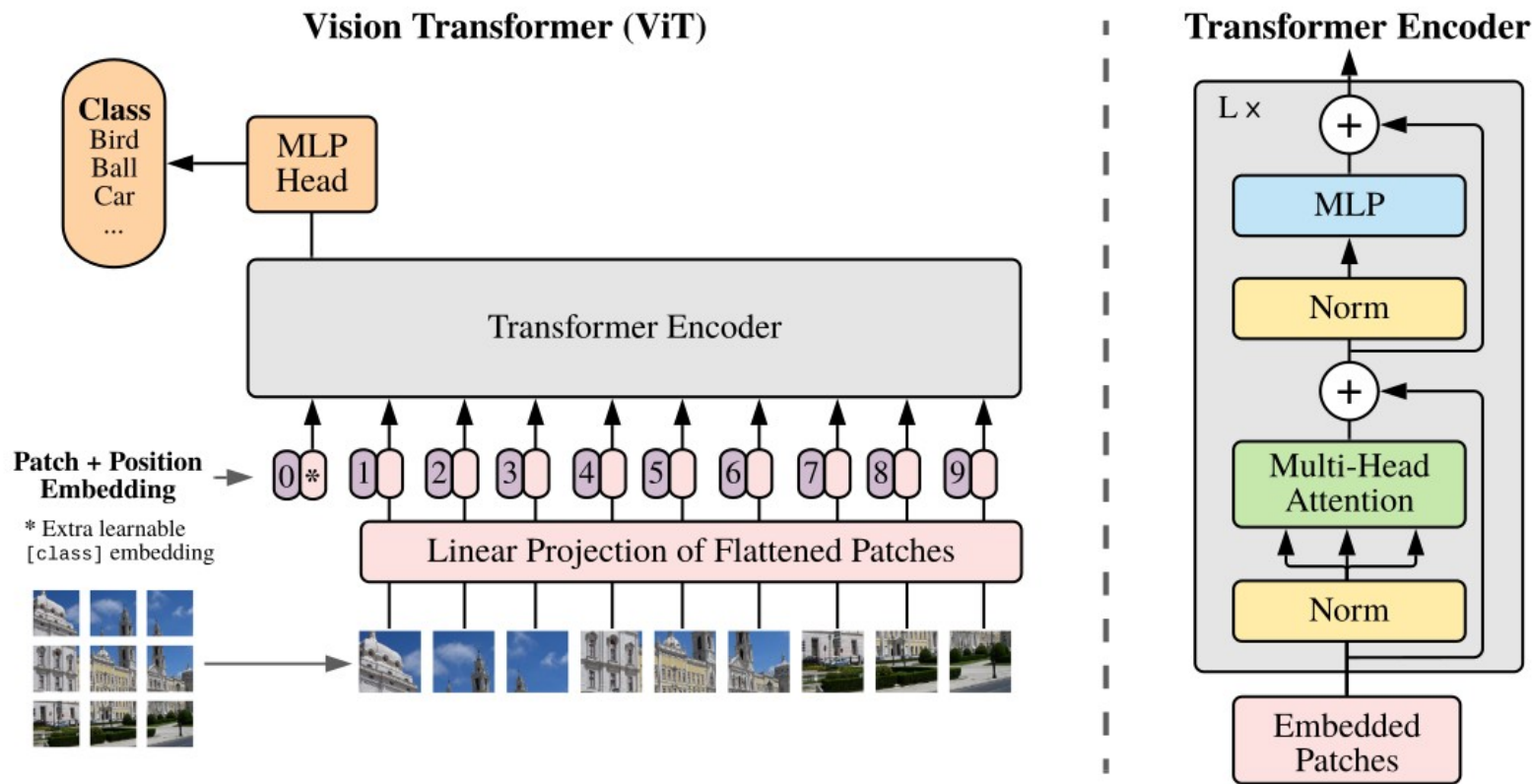
Remarque : Il existe d'autres façons d'encoder la position (ex: "Fourier Features" ou RoPE).⁹¹

IV) Application à des images

“An image is worth 16x16 words”, ICLR 2021



“An image is worth 16x16 words”, ICLR 2021



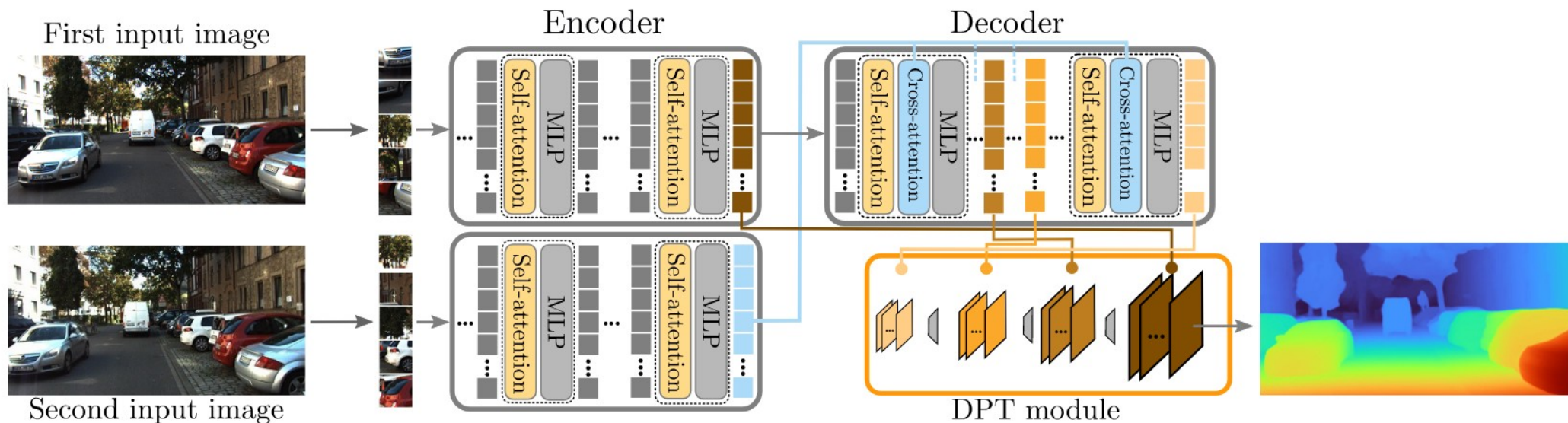
Architecture ViT très utilisée de nos jours.

Architectures DINOv2 et DINOv3

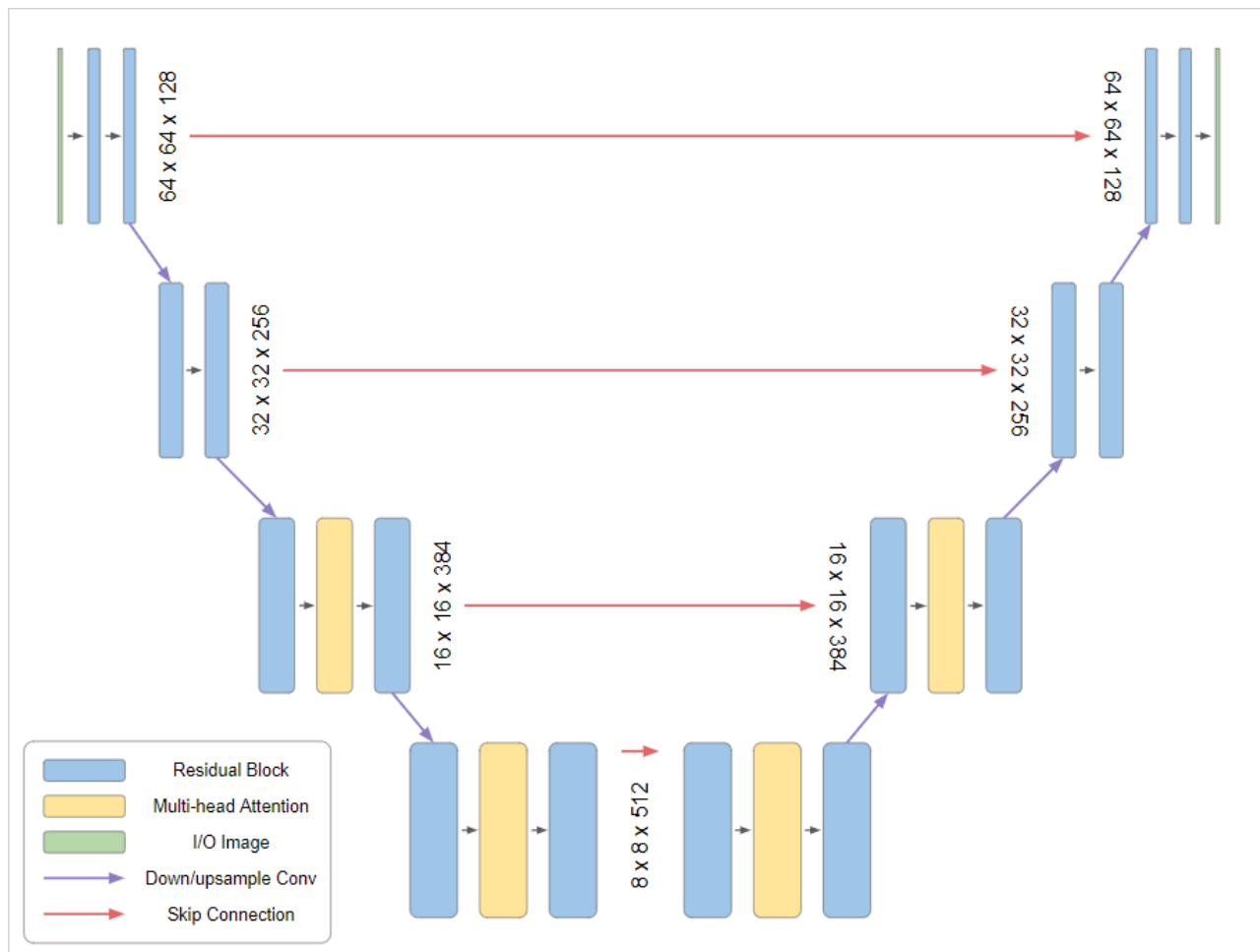
| | DINOv2 | DINOv3 |
|------------------|-----------|--------|
| Backbone | ViT-giant | ViT-7B |
| #Params | 1.1B | 6.7B |
| #Blocks | 40 | 40 |
| Patch Size | 14 | 16 |
| Pos. Embeddings | Learnable | RoPE |
| Registers | 4 | 4 |
| Embed. Dim. | 1536 | 4096 |
| FFN Type | SwiGLU | SwiGLU |
| FFN Hidden Dim. | 4096 | 8192 |
| Attn. Heads | 24 | 32 |
| Attn. Heads Dim. | 64 | 128 |

IV)

“CroCo v2: Improved Cross-view Completion Pre-training for Stereo Matching and Optical Flow”, CVPR 2023



U-Net avec des blocs d'attention



Architecture très utilisée par les méthodes de diffusion (prochain cours)

V) Limites

v)

Limites des couches d'attention à softmax

$$M = XQ(YK)^{\top} : N_x \times N_y \quad S = \text{softmax}(M, \text{dim}=1) : N_x \times N_y$$

Problème : Inapplicable pour des ensembles de grandes tailles.

v)

Limites des couches d'attention à softmax

$$M = XQ(YK)^{\top} : N_x \times N_y \quad S = \text{softmax}(M, \text{dim}=1) : N_x \times N_y$$

Problème : Inapplicable pour des ensembles de grandes tailles.

Solutions :

- Appliquer des couches d'attention à softmax en cherchant à réduire N_x et/ou N_y
→ **ViT** ou **U-Net avec attention**
- Modifier la couche d'attention softmax

v)

Exemple de modification de la couche d'attention softmax : couche d'attention linéaire

“Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention”, ICML 2020

$$\mathbf{x}' = \mathbf{x} + \left(\sum_{j=1}^{N_y} \frac{\phi(\mathbf{xQ})\phi(\mathbf{y}_j\mathbf{K})^\top}{\sum_{k=1}^{N_y} \phi(\mathbf{xQ})\phi(\mathbf{y}_k\mathbf{K})^\top} \mathbf{y}_j \right) \mathbf{V}$$

Remplacement du
noyau exponentiel par
un noyau linéaire

Se simplifie en

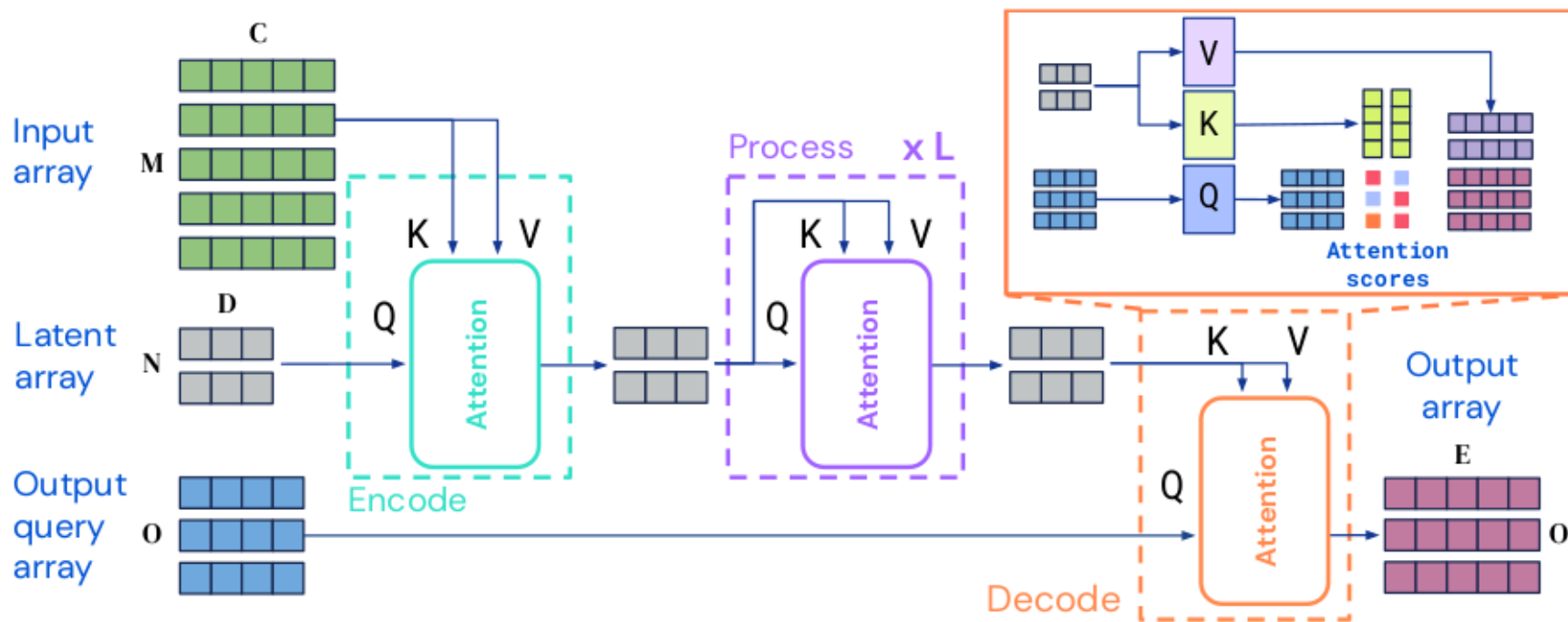


$$\mathbf{x}' = \mathbf{x} + \phi(\mathbf{xQ}) \frac{\sum_{j=1}^{N_y} \phi(\mathbf{y}_j\mathbf{K})^\top \mathbf{y}_j \mathbf{V}}{\phi(\mathbf{xQ}) \sum_{k=1}^{N_y} \phi(\mathbf{y}_k\mathbf{K})^\top}$$

Indépendant de \mathbf{x} , plus
besoin de calculer ni
stocker explicitement
les matrices \mathbf{M} et \mathbf{S}

Exemple de réduction de N_x et/ou N_y : PerceiverIO

“Perceiver IO: A General Architecture for Structured Inputs & Outputs.” arXiv, 2021



Annexe : Hallucinations

Prompt : “What was the title of Adam Kalai’s dissertation?”

ChatGPT: Adam Tauman Kalai’s Ph.D. dissertation (completed in 2002 at CMU) is entitled:
(GPT-4o) “Boosting, Online Algorithms, and Other Topics in Machine Learning.”

DeepSeek: “Algebraic Methods in Interactive Machine Learning”... at Harvard University in 2005.

Llama: “Efficient Algorithms for Learning and Playing Games”... in 2007 at MIT.

Le vrai titre est “Probabilistic and on-line methods in machine learning.”

→ Les LLM ne sont pas fiables ! Il faut vérifier les réponses.

Annexe : In-Context Learning

Prompt

English: Hello

French: Bonjour

English: Thank you

French: Merci

English: How are you?

French:

→ Les couches d'attention permettent “d'apprendre à faire une tâche”, ici traduire, sans faire un apprentissage (c'est-à-dire sans modifier les paramètres du réseau), simplement à partir du prompt.

Annexe : Chain-of-thought

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅

Annexe : Scaling laws

En pratique :

- Le nombre de GPU disponible est connu
 - La date à laquelle on voudrait déployer un modèle est fixée
- La quantité de calculs (FLOPs) qu'on va pouvoir faire pendant l'entraînement est connue et on voudrait savoir s'il vaut mieux :
- 1) Agrandir le réseau et donc voir moins de données pendant l'entraînement
 - 2) Garder un réseau petit et donc voir plus de données pendant l'entraînement

Annexe : Scaling laws (suite)

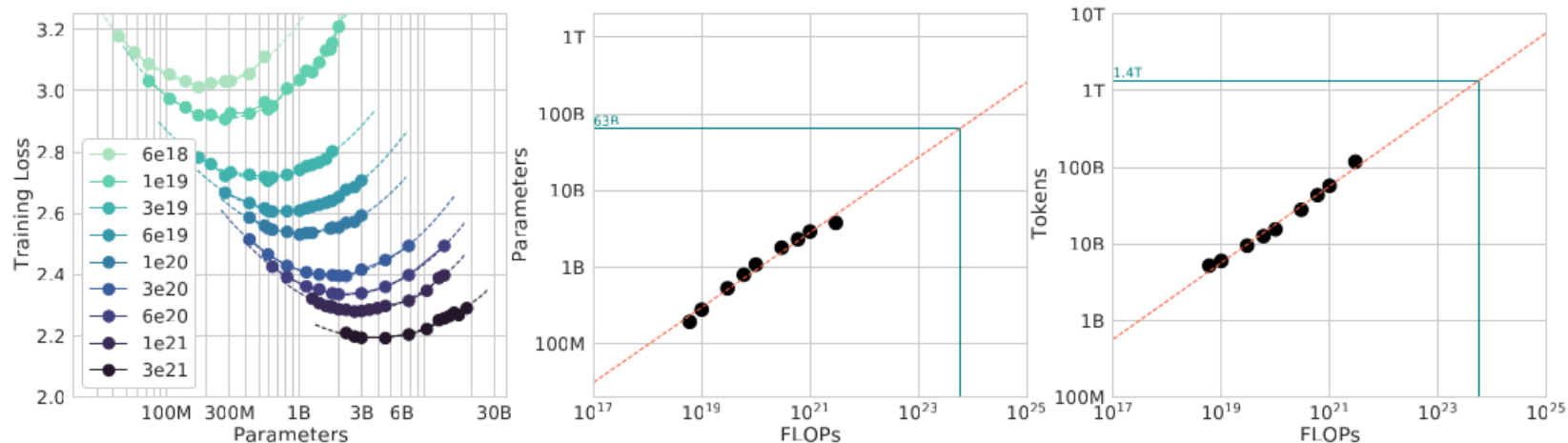


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

Annexe : Tête DPT (“Dense Prediction Transformer”)

