Architecture de réseau de neurones
-
Le Transformer

Guillaume Bourmaud

PLAN

I. Histoire du “Transformer”

II. Couche d’attention à softmax

III. Équivariance par permutation et encodage de la position

IV. Application à des images

V. Limites et tendances actuelles

I) Histoire du “Transformer”

“Attention is All You Need”, NIPS 2017

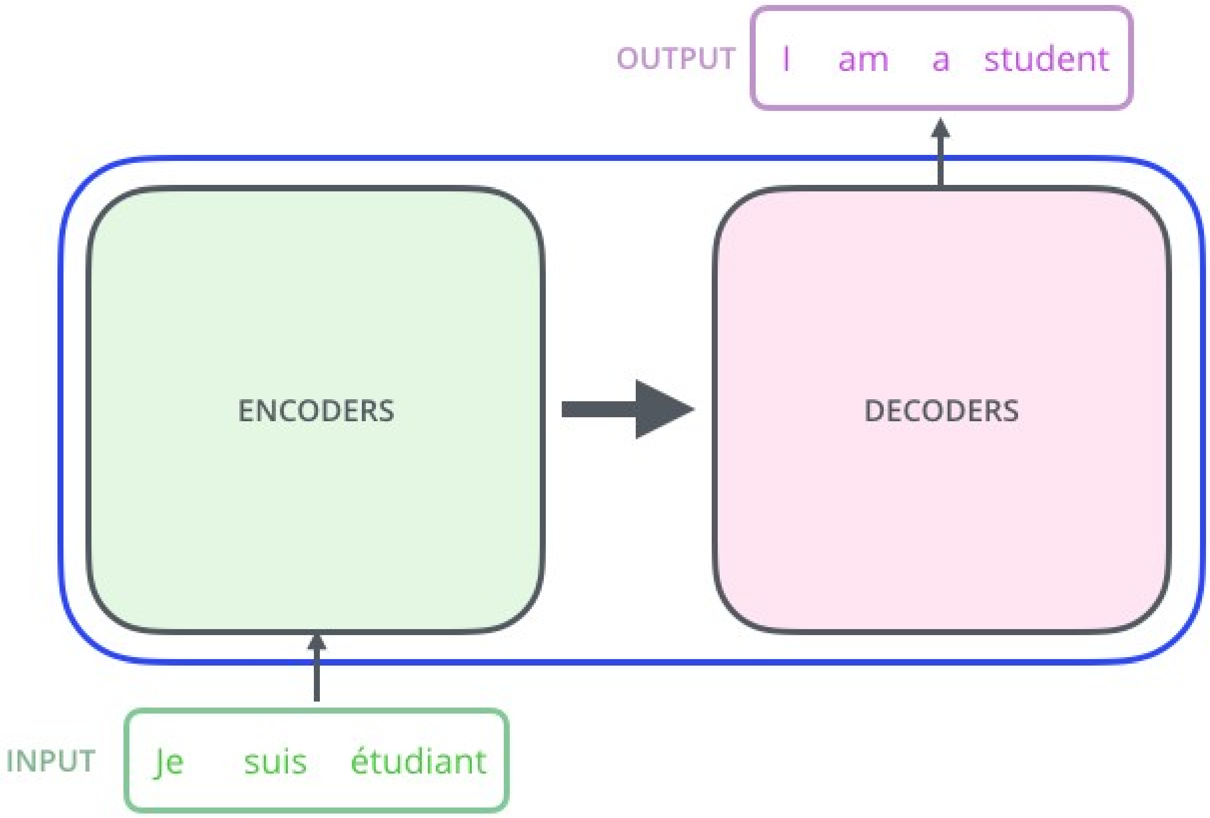
Sequence-to-sequence


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

I)

Architecture encodeur-décodeur

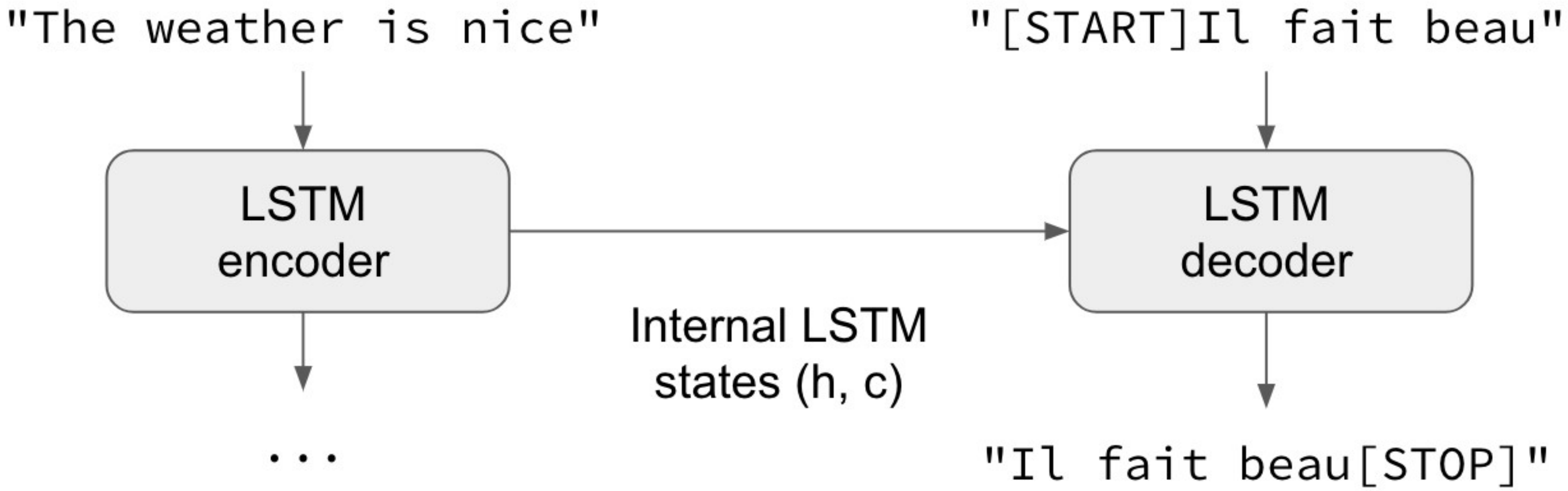
Encodeur :
- RNN
- ou CNN
- ou Transformer



Décodeur :
- RNN
- ou CNN
- ou Transformer

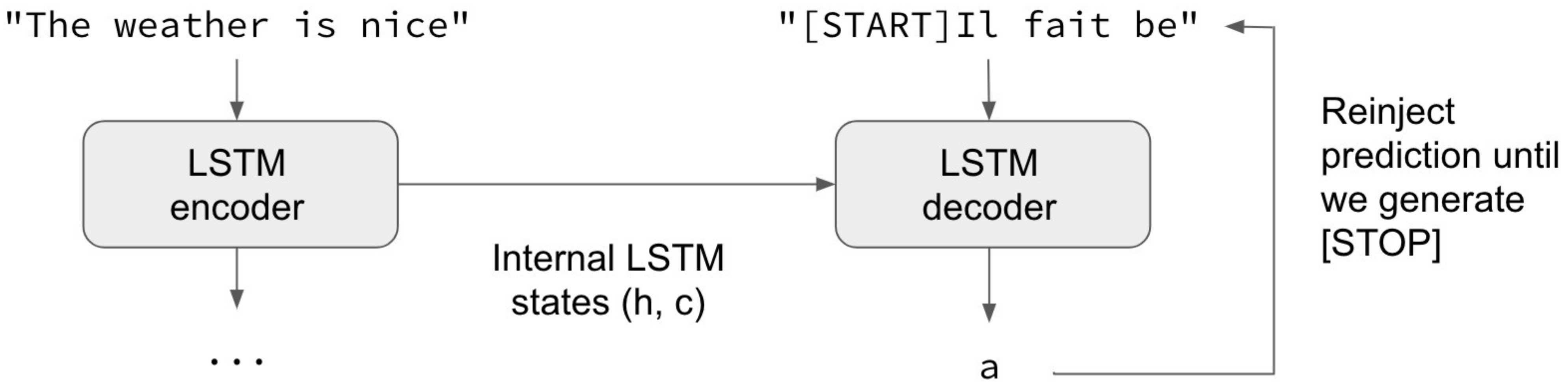
Exemple avec un RNN

Apprentissage (en “Teacher-Forcing”)



Exemple avec un RNN (suite)

Inférence



I)

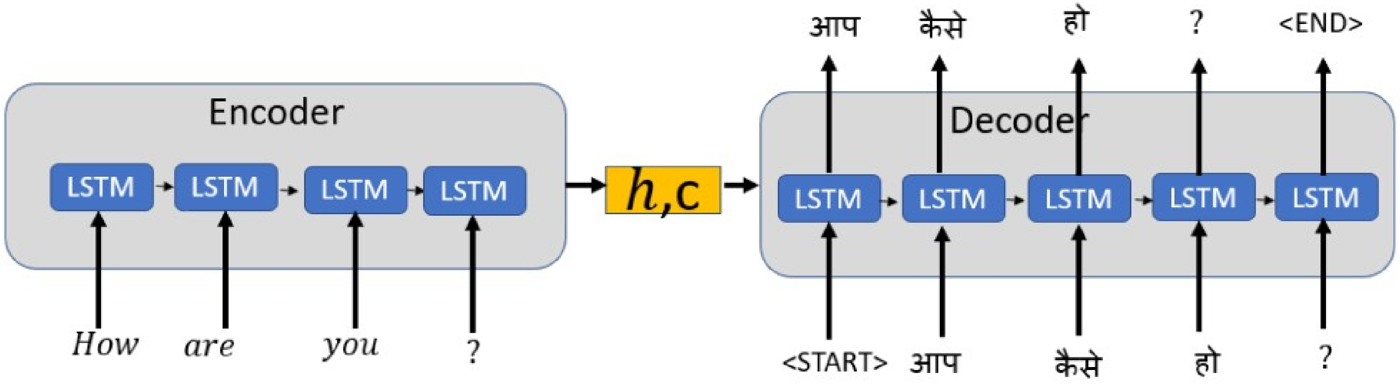
Exemple avec un RNN (suite)

Limites

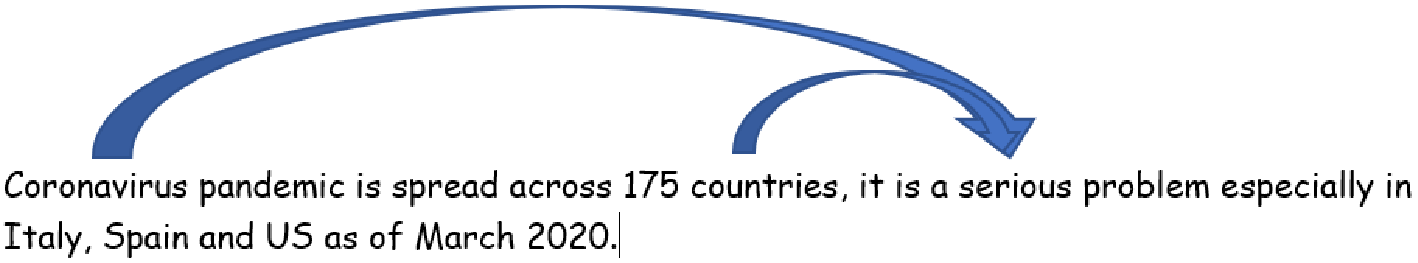
Calcul séquentiel

Apprentissage lent

Comment paralléliser ?



Difficile d’apprendre de longues dépendances avec un RNN.

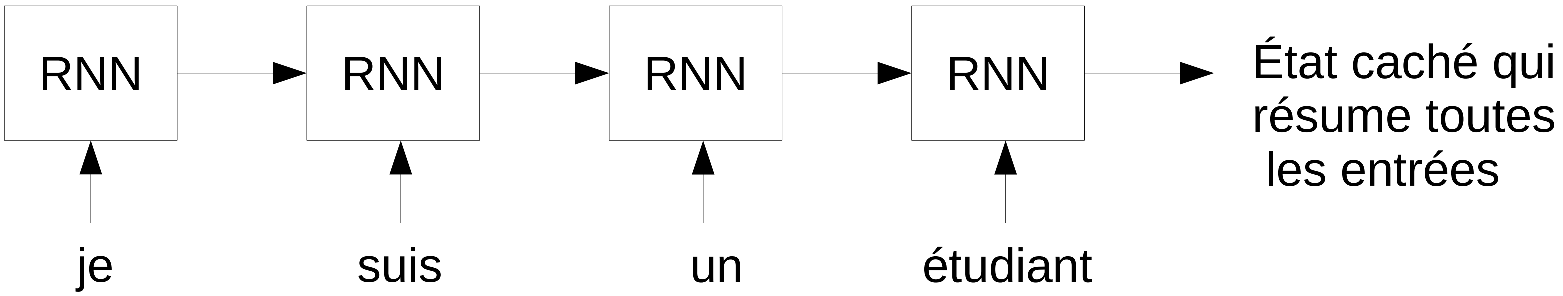


I)

Transformer vs RNN

Encodeur RNN

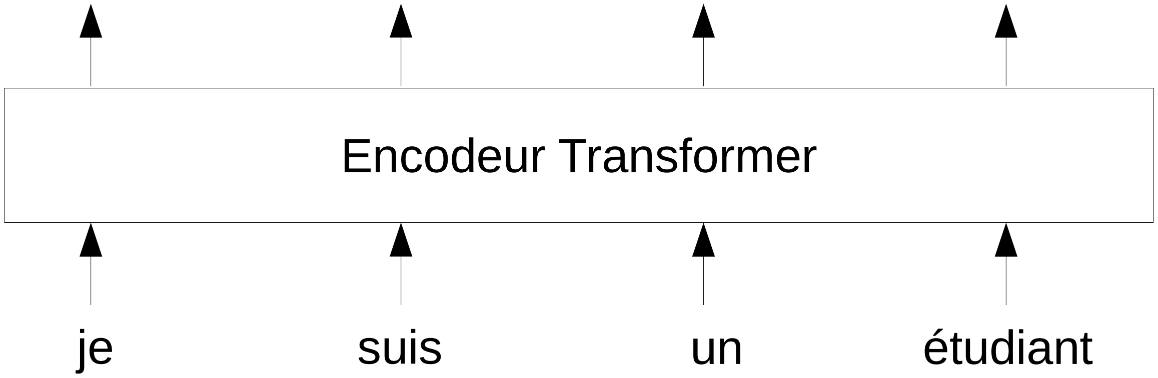
Calcul séquentiel
+
Chemin de la longueur de la phrase d’entrée



Encodeur Transformer

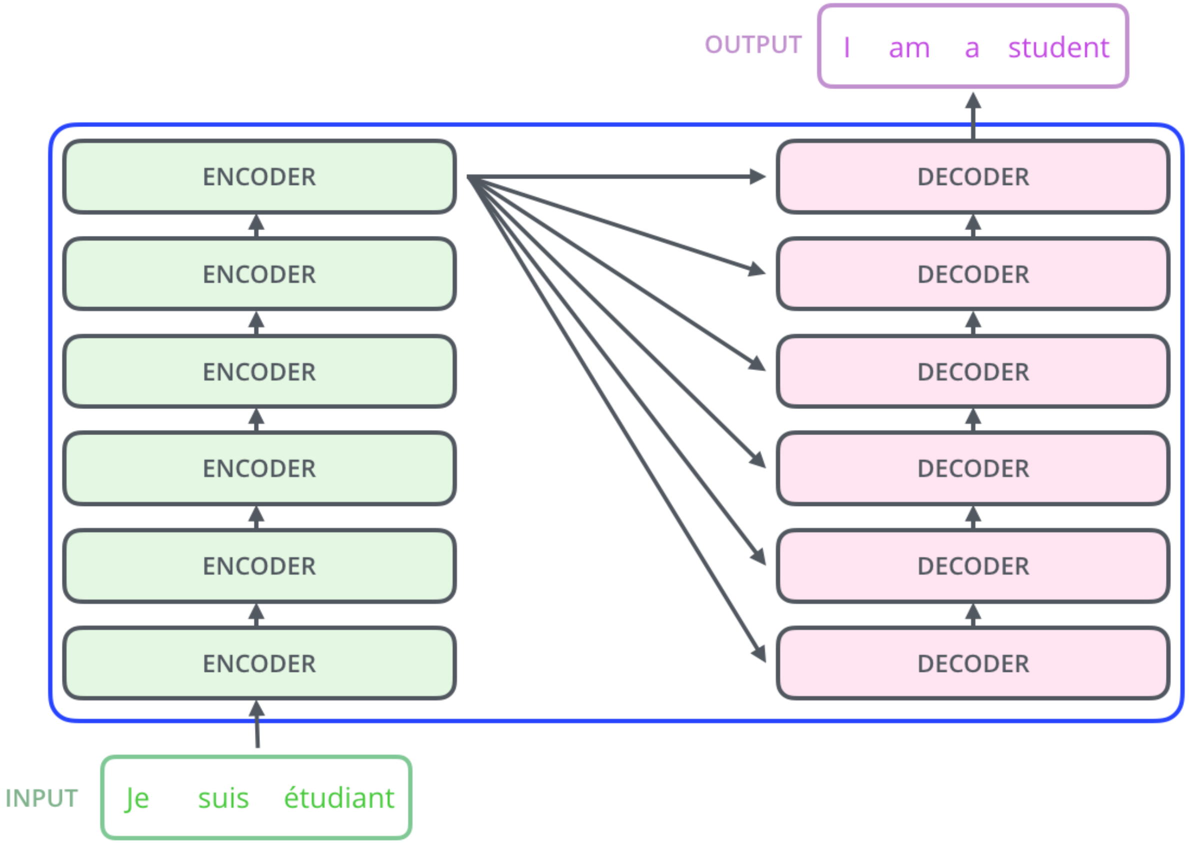
Calcul en parallèle
+
Chemin de longueur constante

N entrées, N sorties. Chaque sortie dépend de **toutes** les entrées.



I)

Transformer : vue globale



I)

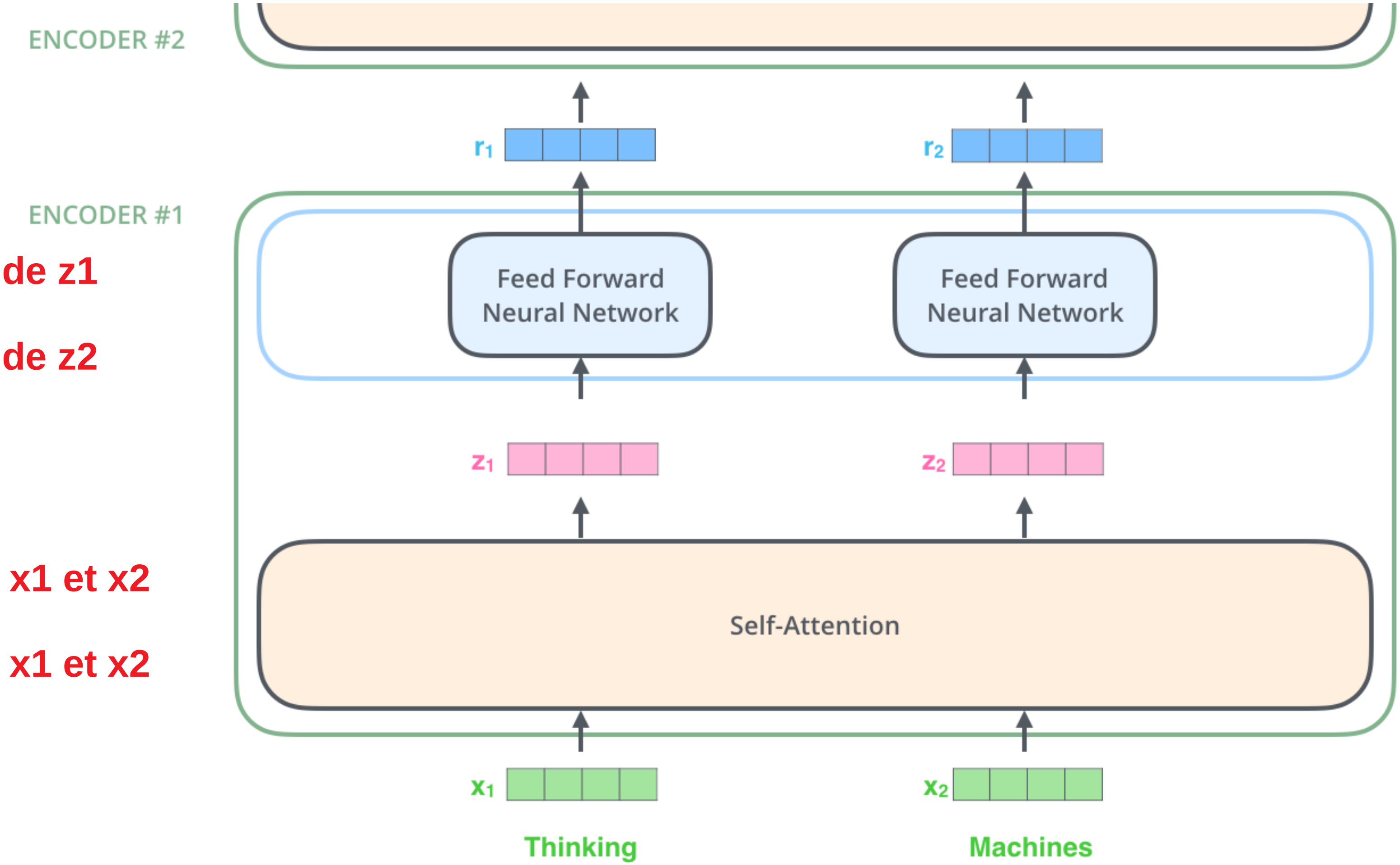
Transformer : encodeur

r1 dépend de z1

r2 dépend de z2

z1 dépend de x1 et x2

z2 dépend de x1 et x2



II) Couche d’attention à 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 $1 \times D$

II)

Attention utilisant la fonction softmax

Entrées

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

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

Fonction

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

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 $\mathbf{y}_j$

Attention utilisant la fonction softmax

Entrées

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

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

Fonction

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

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 $1 \times D$

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

Fonction

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

Softmax

Attention utilisant la fonction softmax

Entrées

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

$\{\mathbf{y}_i\}_{i=1\ldots N_y}$: 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 $\{\mathbf{y}_i\}_{i=1\ldots N_y}$

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

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

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

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

Paramètres

$\mathtt{Q}$: matrice “query” de taille $D \times L$

$\mathtt{K}$: matrice “key” de taille $D \times L$

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

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

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

Paramètres

$\mathtt{Q}$: matrice “query” de taille $D \times L$

$\mathtt{K}$: matrice “key” de taille $D \times L$

$\mathtt{V}$: matrice “value” de taille $D \times D$

Attention utilisant la fonction softmax

Entrées

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

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

Fonction

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

Paramètres

$\mathtt{Q}$: matrice “query” de taille $D \times L$

$\mathtt{K}$: matrice “key” de taille $D \times L$

$\mathtt{V}$: matrice “value” de taille $D \times D$

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

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

Entrées

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

$\{\mathbf{y}_i\}_{i=1\ldots N_y}$: vecteurs de dimension D $\longrightarrow$ $\mathtt{Y}$: matrice de taille $N_y \times D$

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

Entrées

$\{\mathbf{x}_i\}_{i=1\ldots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

$\{\mathbf{y}_i\}_{i=1\ldots N_y}$: vecteurs de dimension D ⟶ $\mathtt{Y}$: matrice de taille $N_y \times D$

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

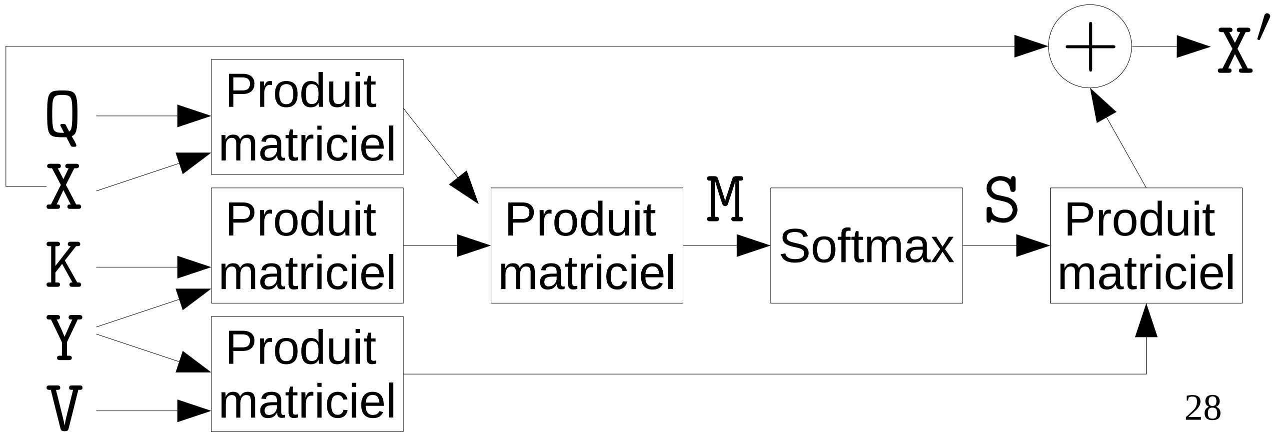
$\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D $\longrightarrow$ $\mathtt{Y}$: matrice de taille $N_y \times D$

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

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

$$\mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1)$$

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



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

II)

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

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

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

Calcul

- Nombre d'opérations potentiellement très élevé

+ Parallélisable

Stockage

- Mémoire requise pour stocker M et S potentiellement très élevée

II)

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

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

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

Calcul

- Nombre d'opérations potentiellement très élevé

+ Parallélisable

Stockage

- Mémoire requise pour stocker M et 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) $\longrightarrow$ $360\mathrm{Go}$

II)

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

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

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

$$\mathtt{M} = \mathtt{XQ}(\mathtt{XK})^\top \;:\; N_x \times N_x$$
$$\mathtt{S} = \mathrm{softmax}(\mathtt{M},\mathrm{dim}{=}1) \;:\; N_x \times N_x$$
$$\mathtt{X}' = \mathtt{X} + \mathtt{SXV} \;:\; N_x \times D$$

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

II)

Attention softmax à têtes multiples
”Multi-head dot-product attention”

Entrées

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

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

---

1 tête = 1 façon
“d’attirer l’attention”

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

H têtes = H façons
“d’attirer l’attention”

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

Attention softmax à têtes multiples (suite) ”Multi-head dot-product attention”

Entrées

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

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

Fonction

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

Paramètres

$\{\mathtt{Q}_h\}_{h=1 \ldots H}$: matrices “query” de taille $D \times L$

$\{\mathtt{K}_h\}_{h=1 \ldots H}$: matrices “key” de taille $D \times L$

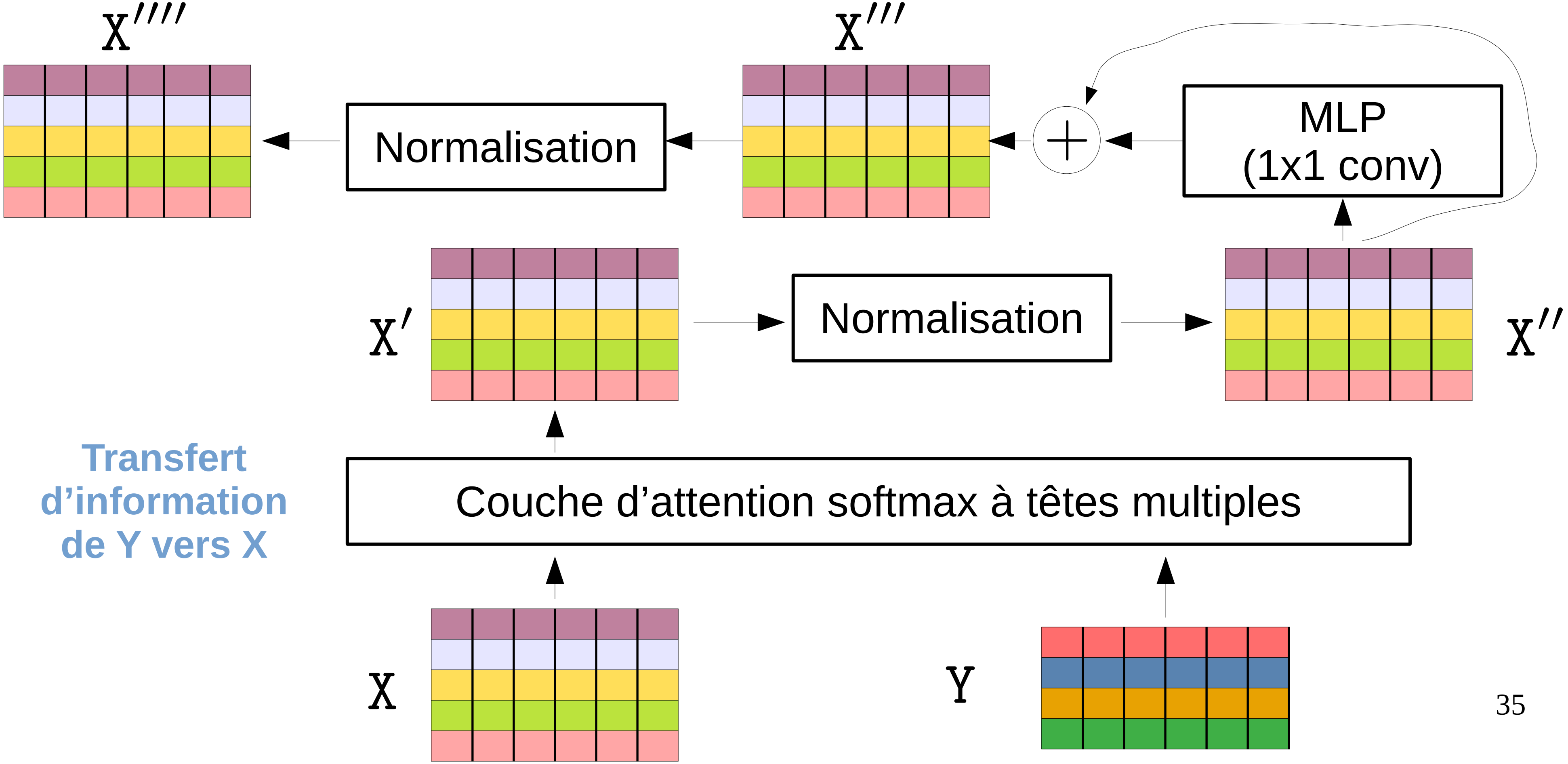
$\{\mathtt{V}_h\}_{h=1 \ldots H}$: matrices “value” de taille $D \times L$

$\{\mathtt{W}_h\}_{h=1 \ldots H}$: matrices “output” de taille $L \times D$

II)

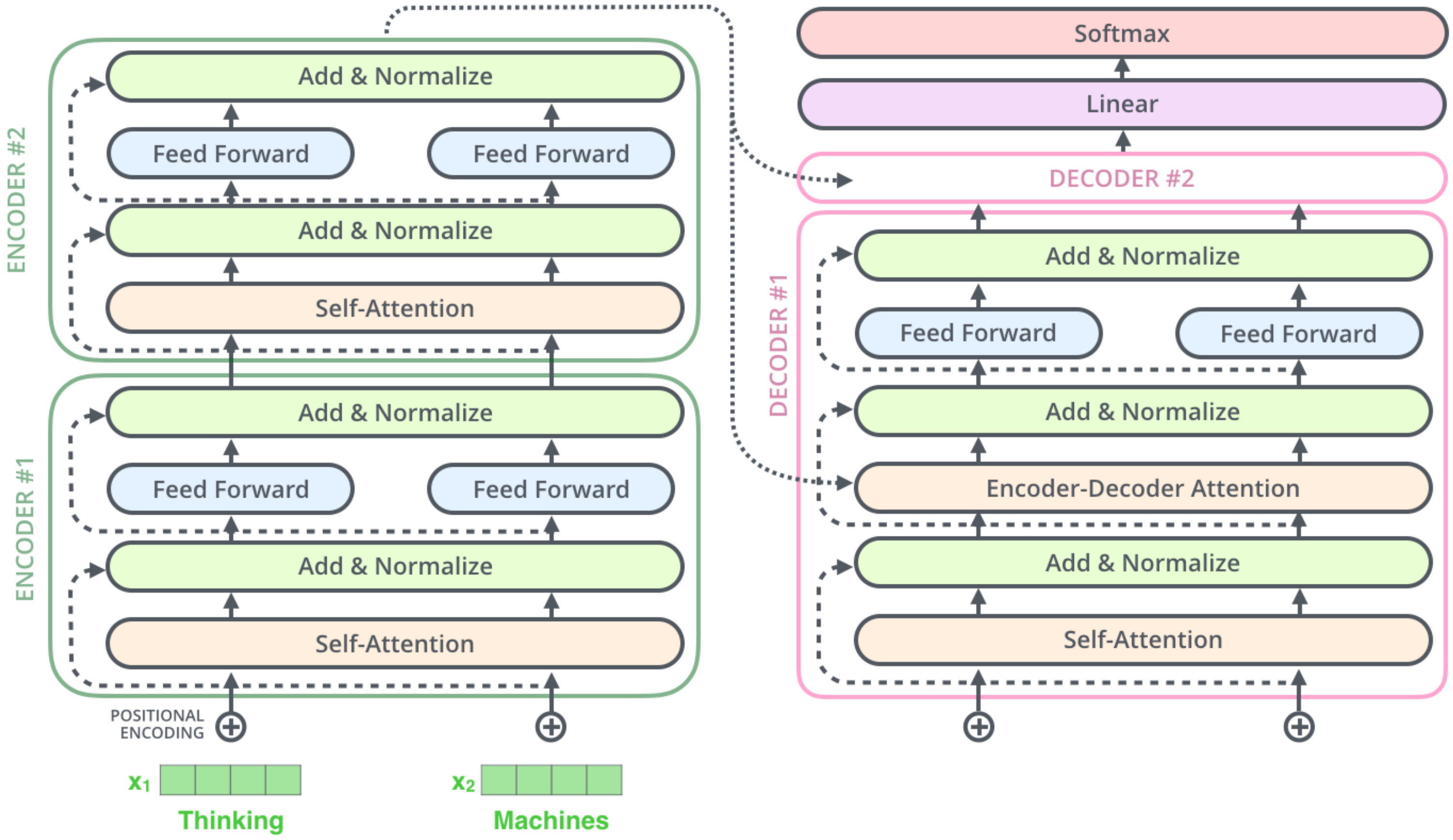
Bloc d’attention “classique”

Transformation non-linéaire de chaque ligne



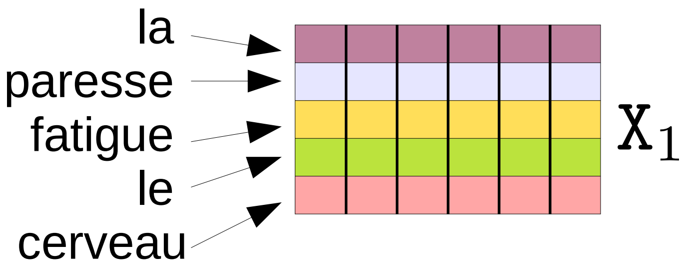
II)

Vue détaillée du Transformer

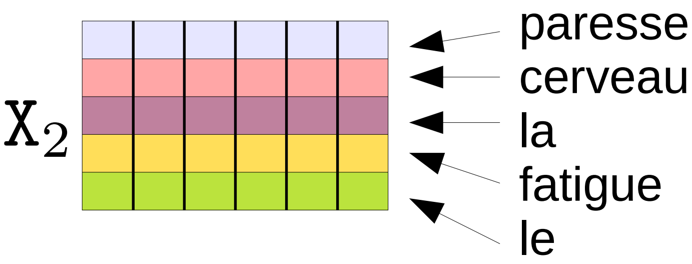


III) Équivariance par permutation et encodage de la position

Non-équivariance par permutation

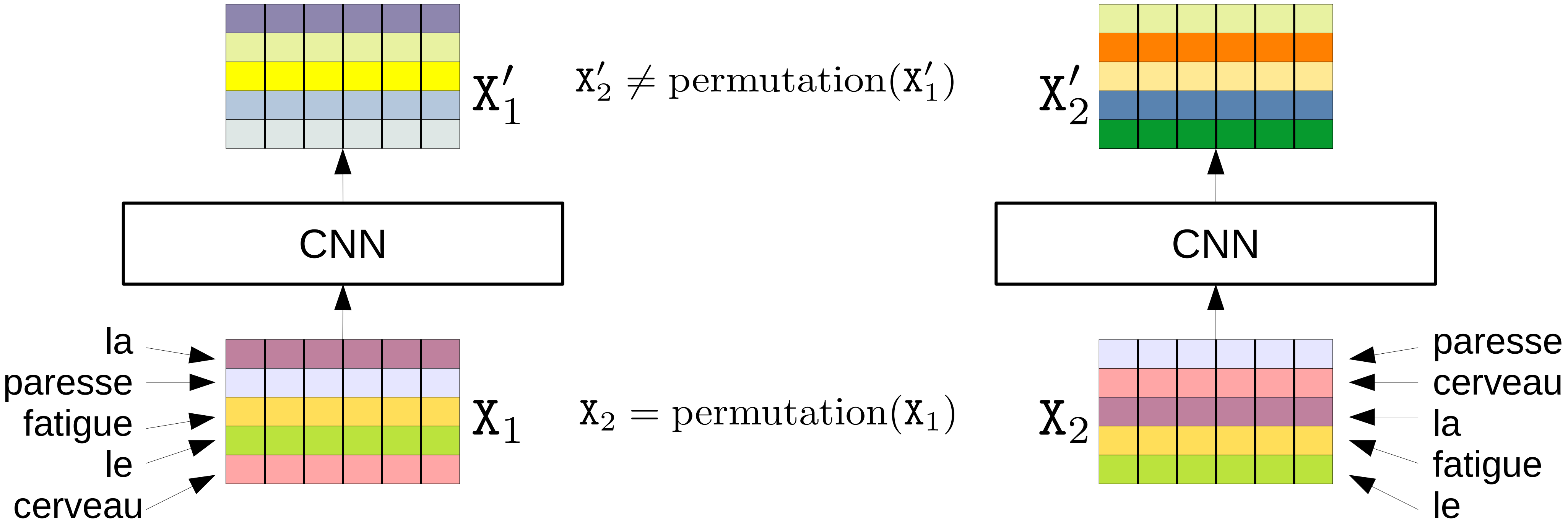


$$X_2 = \mathrm{permutation}(X_1)$$



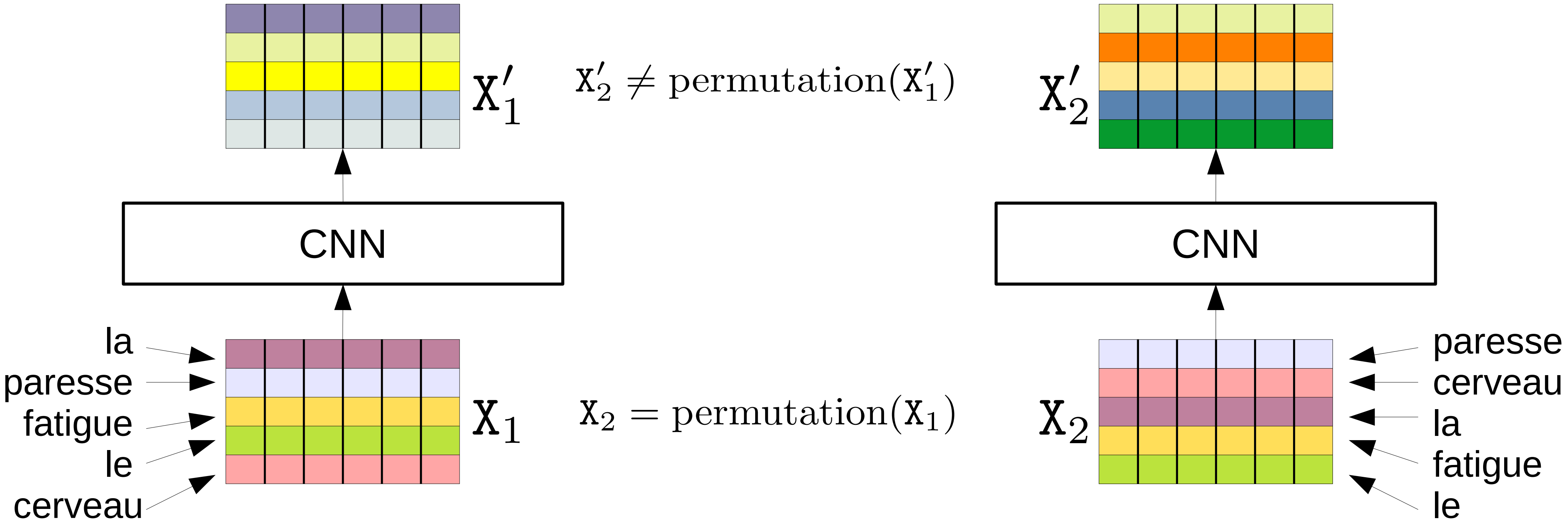
III)

Non-équivariance par permutation



III)

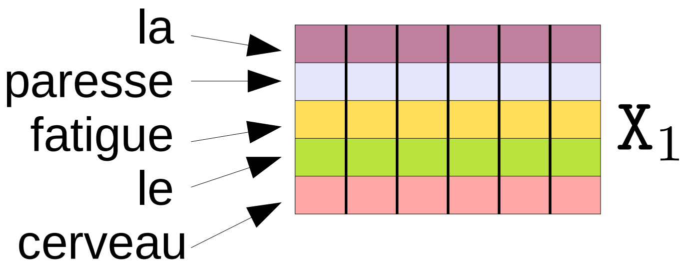
Non-équivariance par permutation



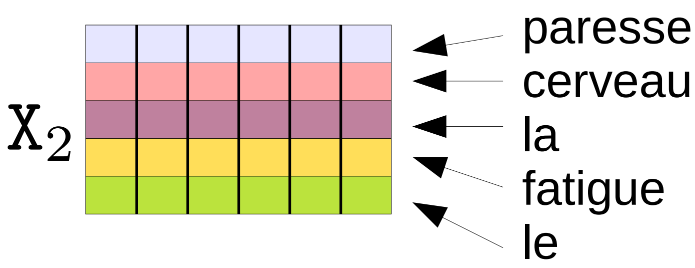
CNN adapté aux ensembles ordonnés (phrase, signal, image, etc.)
→ ses filtres s’appliquent sur un voisinage

III)

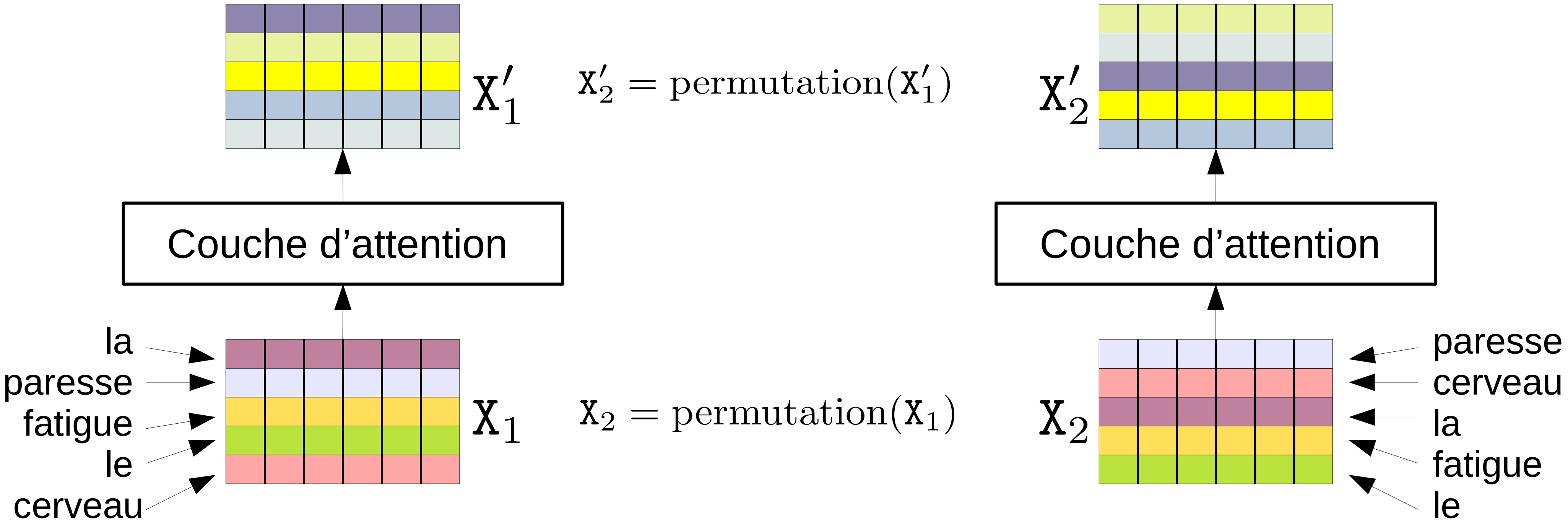
Équivariance par permutation



$$\mathrm{X}_2 = \mathrm{permutation}(\mathrm{X}_1)$$



Équivariance par permutation

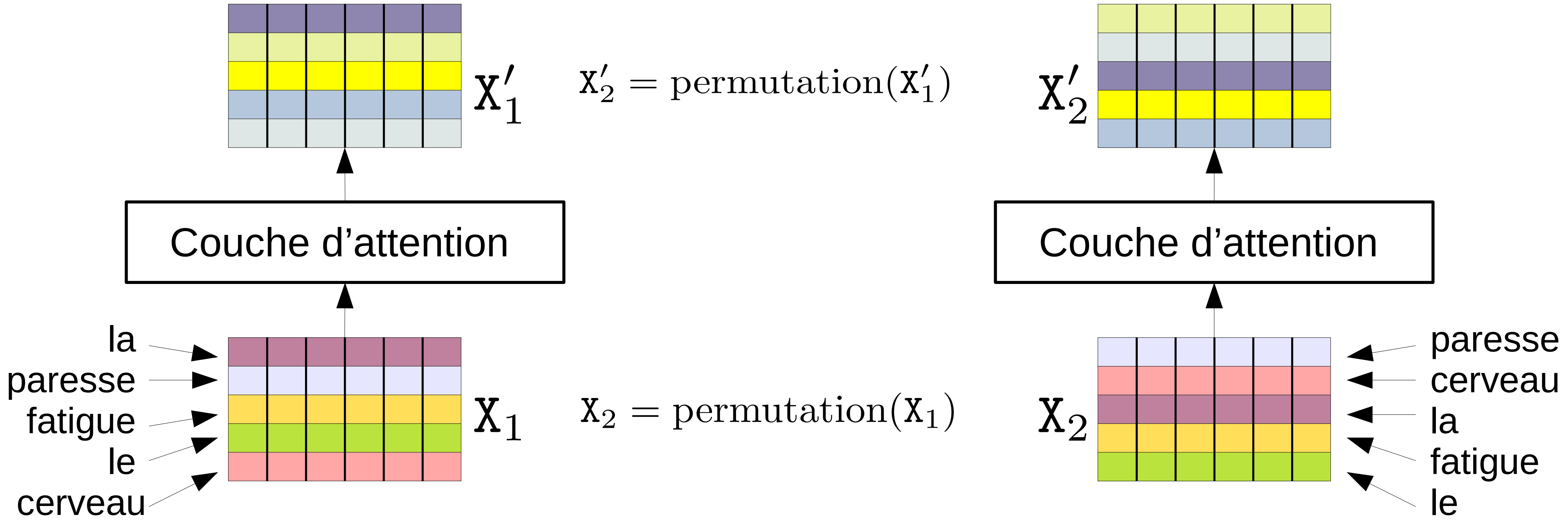


$$\mathtt{X}_2' = \mathrm{permutation}(\mathtt{X}_1')$$

$$\mathtt{X}_2 = \mathrm{permutation}(\mathtt{X}_1)$$

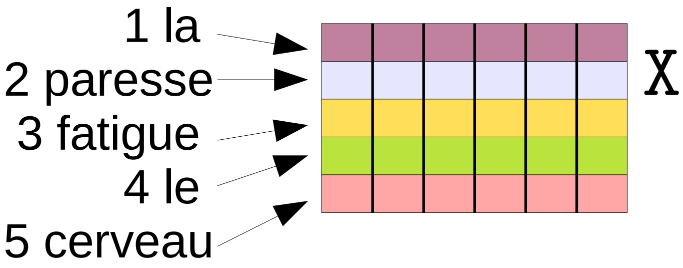
III)

Équivariance par permutation



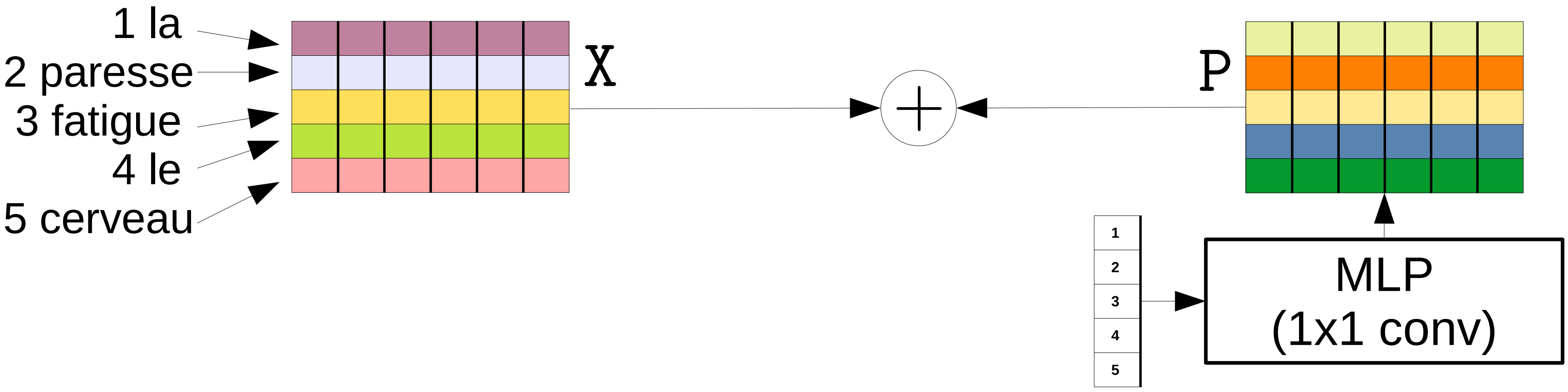
Pour un ensemble ordonné (phrase, signal, image, etc.)
→ necessité d'encoder la position

Encodage de la position

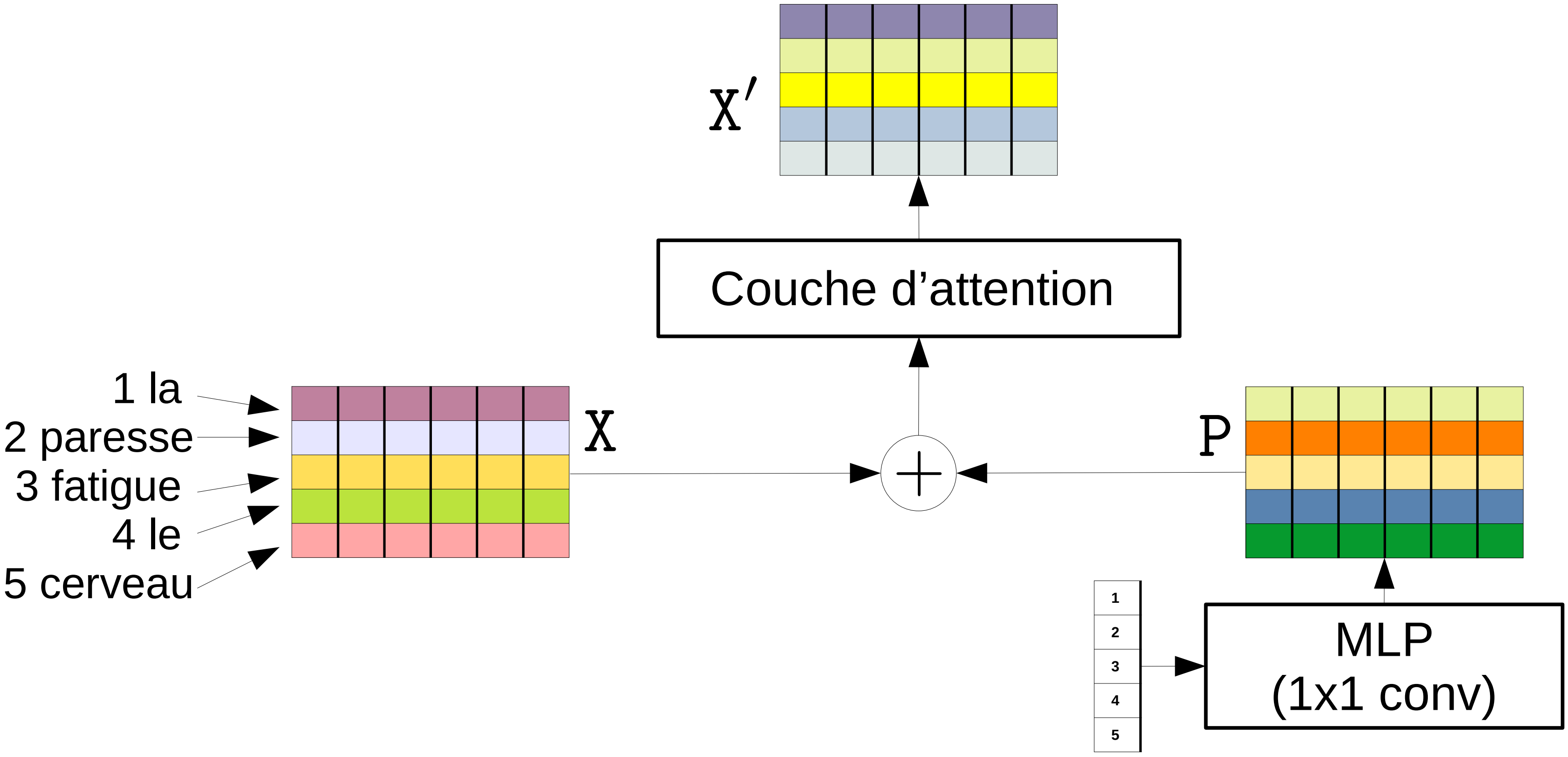


III)

Encodage de la position



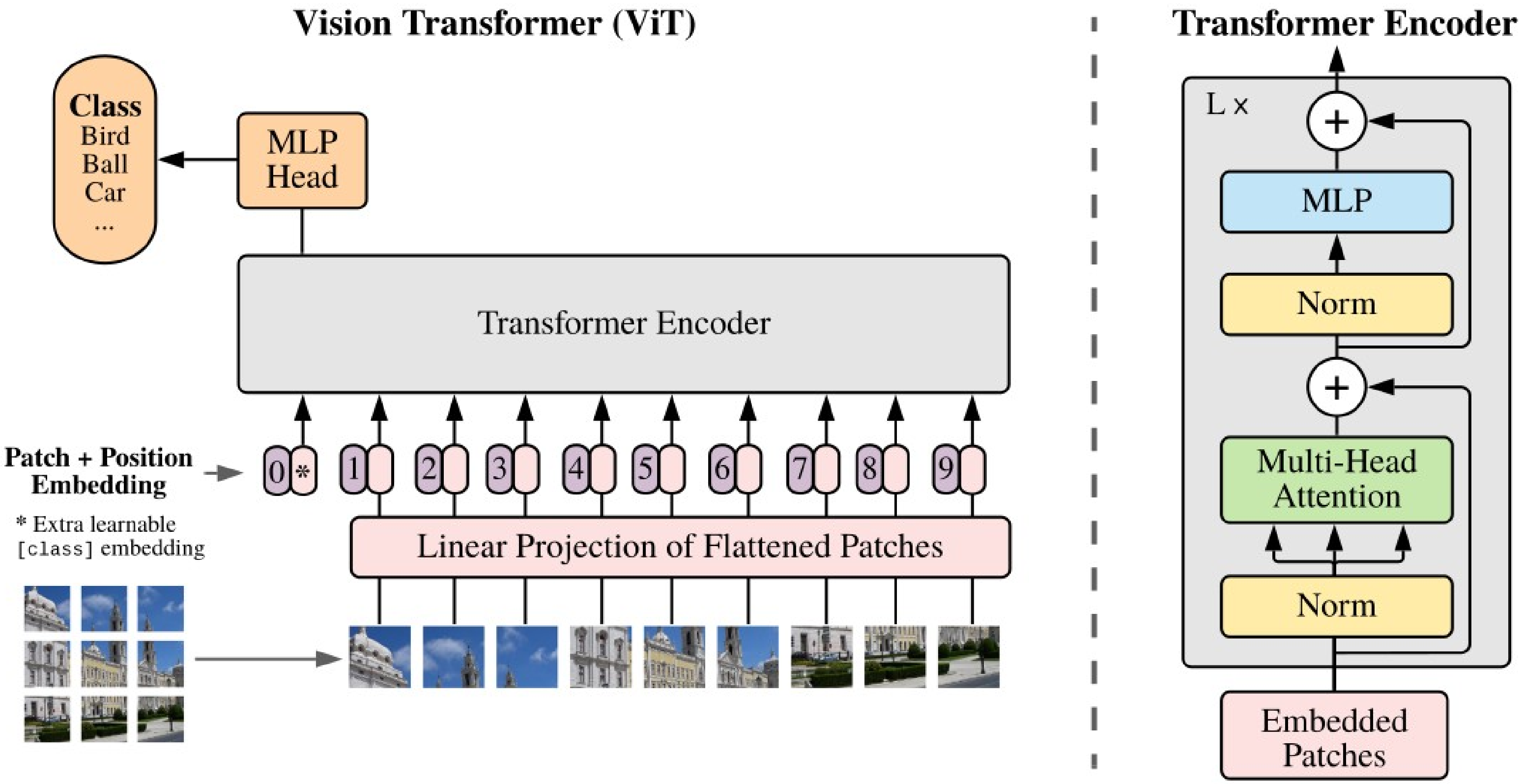
Encodage de la position



IV) Application à des images

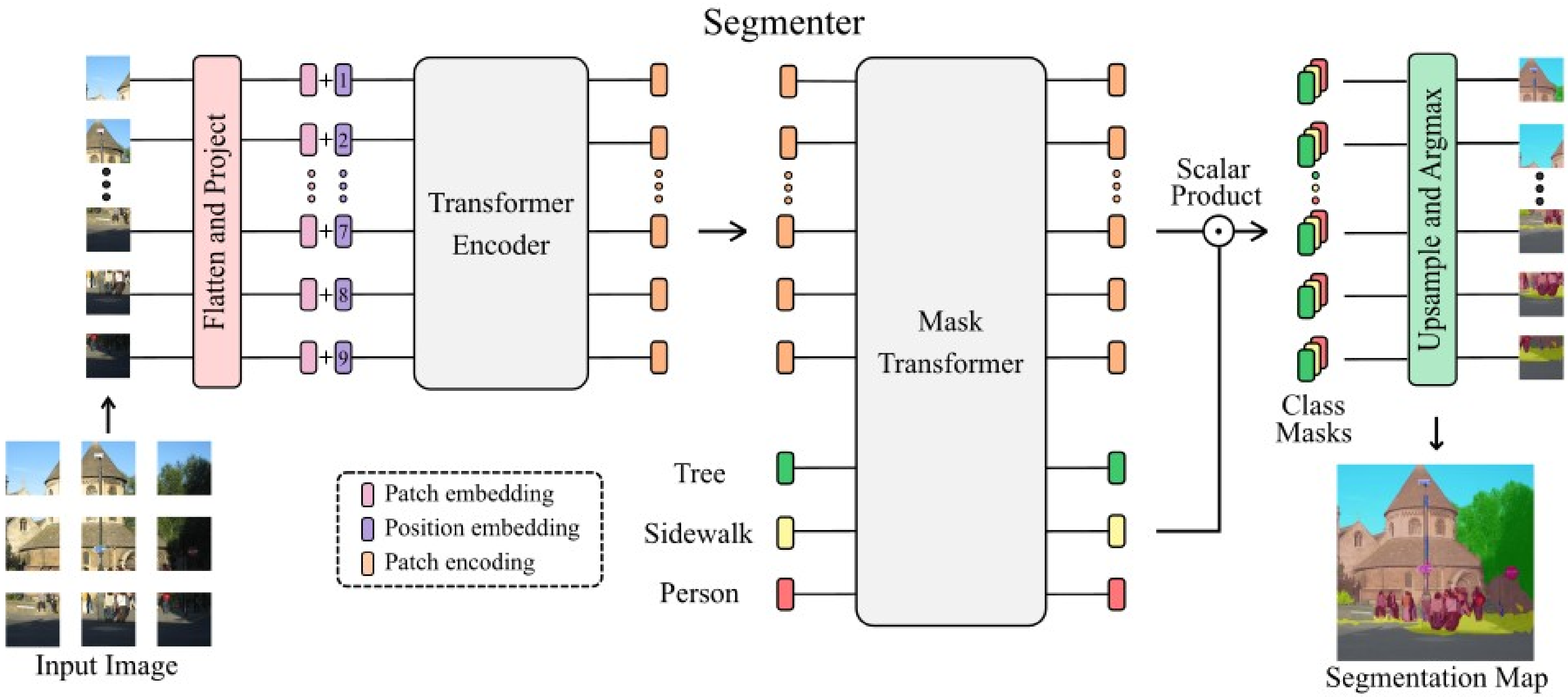
IV)

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



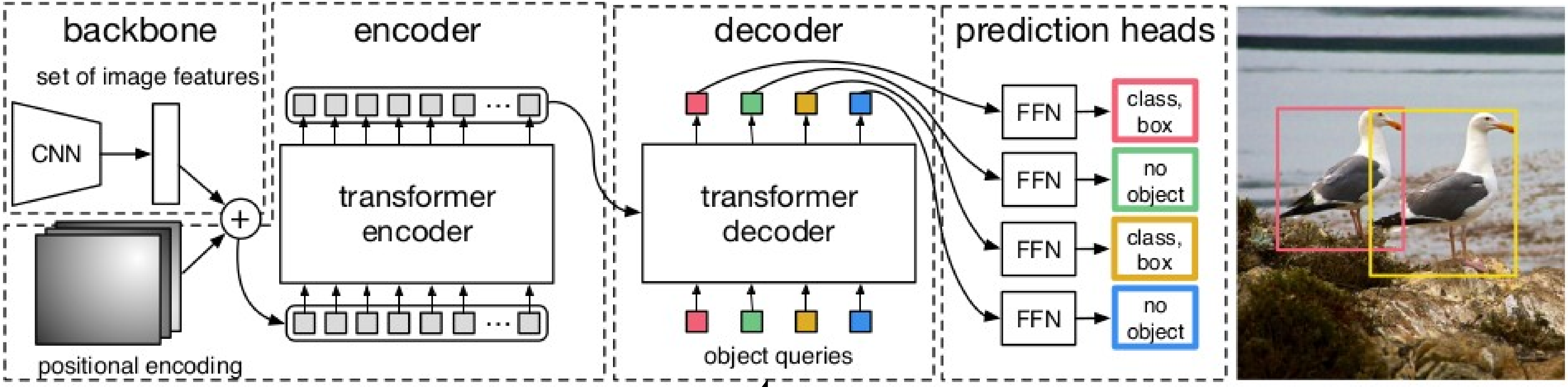
IV)

“Segmenter: Transformer for Semantic Segmentation”, ICCV 2021



IV)

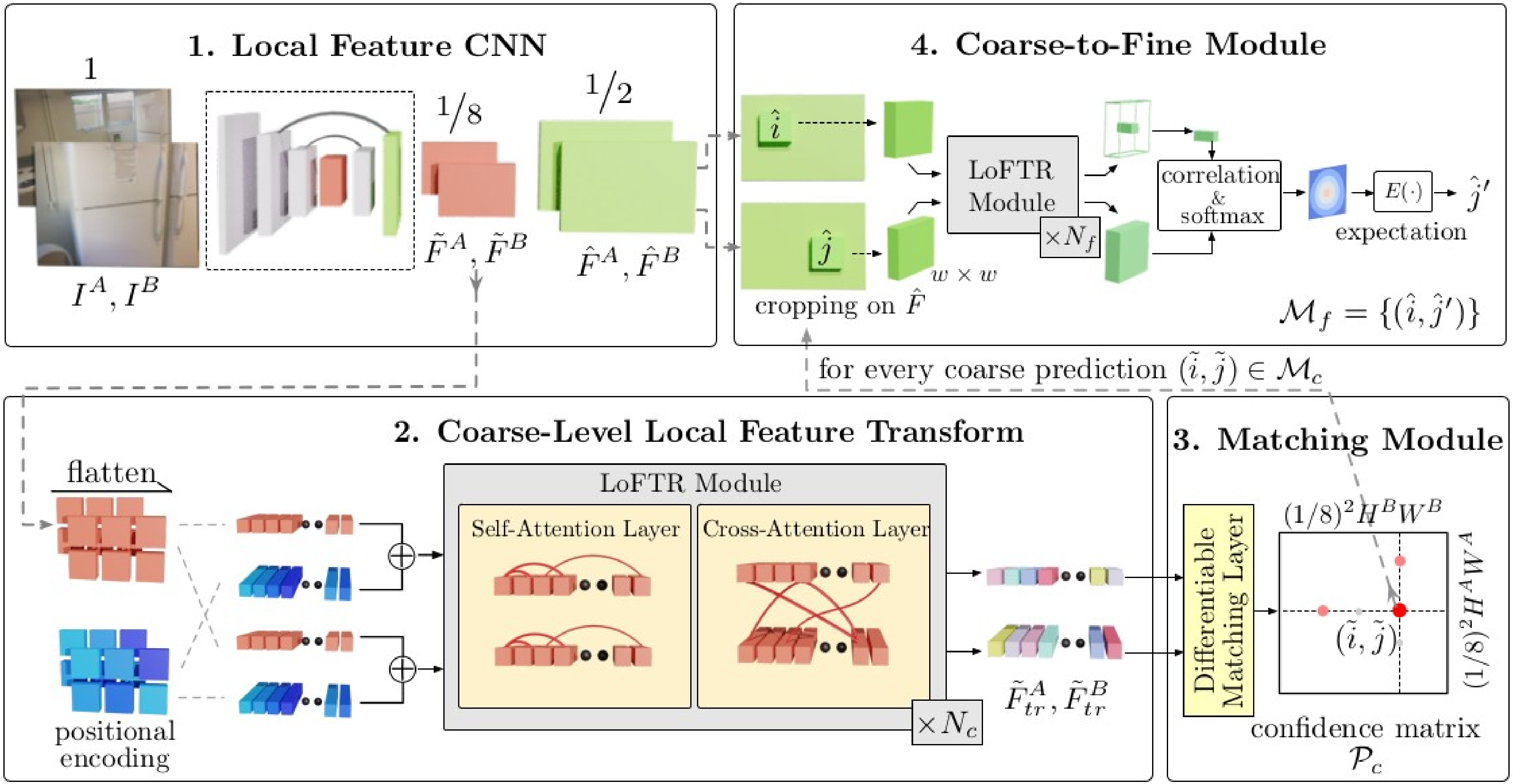
“DETR: End-to-End Object Detection With Transformers”, ECCV 2020



En pratique, il y a 100 “object queries”, donc 100 boîtes englobantes prédites.

IV)

"LoFTR: Detector-Free Local Feature Matching with Transformers", CVPR 2021



V) Limites et tendances actuelles

Limites des couches d'attention à softmax

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

Problème : Inapplicable pour des ensembles de grandes tailles.

Solutions :

- Appliquer des couches d'attention à softmax en cherchant à réduire Nx et/ou Ny
- 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

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

Remplacement du noyau exponentiel par un noyau linéaire

Se simplifie en

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

Indépendant de x, plus besoin de calculer ni stocker explicitement les matrices M et S

Exemple de réduction de Nx et/ou Ny : PerceiverIO

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