Lecture de *Attention Is All You Need*

Serge Rosmorduc
serge.rosmorduc@lecnam.net
Conservatoire National des Arts et Métiers

2 mars 2018

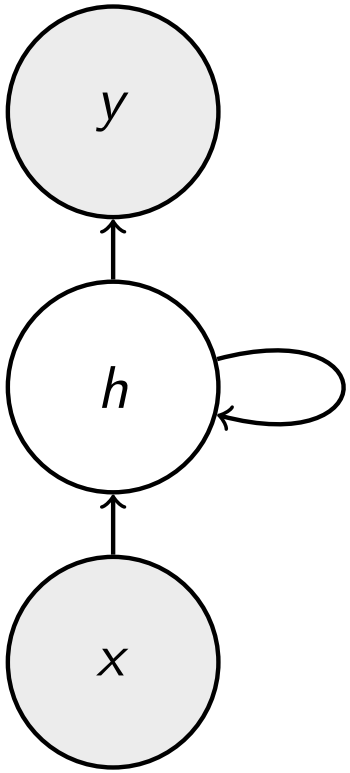
Contexte

- Tâche : traduction automatique ;
- réécriture de séquence ;
- on a une séquence $x_1, x_2, .., x_n$ en entrée, on veut produire une séquence $y_1, ..., y_m$ en sortie ;
- en pratique, on veut calculer

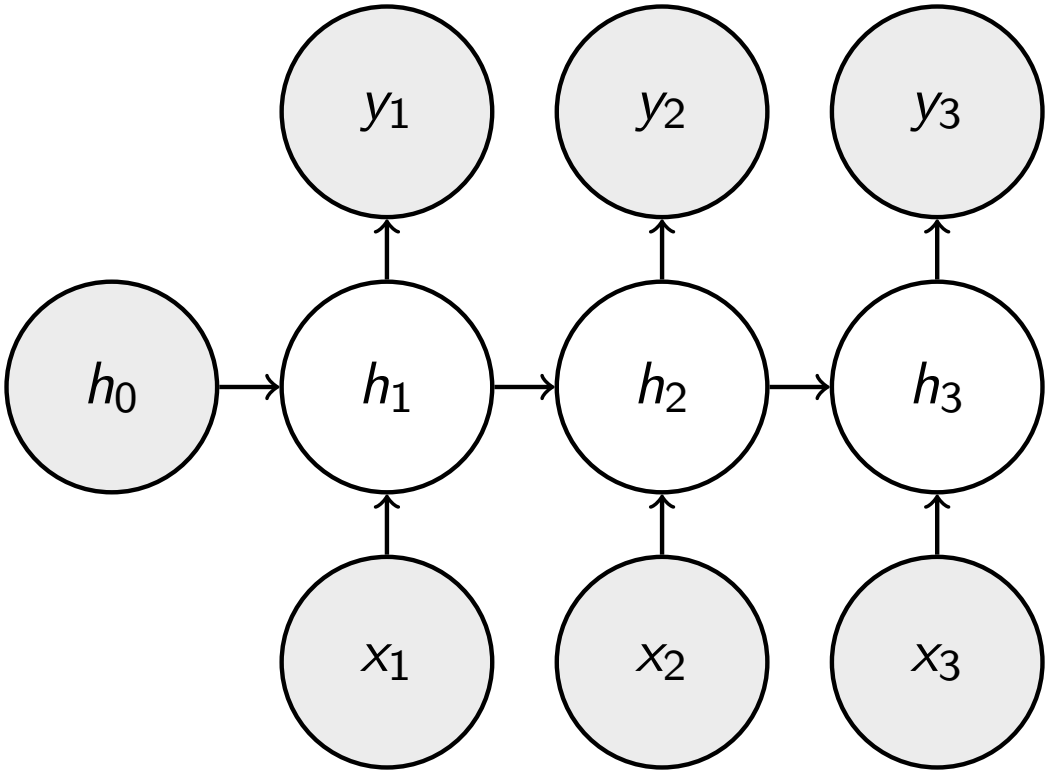
$$\text{argmax}(P(Y_k = y | x_1, \cdots, x_n, y_1, \cdots, y_{k-1}))$$

- applications très nombreuses : traduction automatique ; résumé automatique ; analyse syntaxique (balisage)...

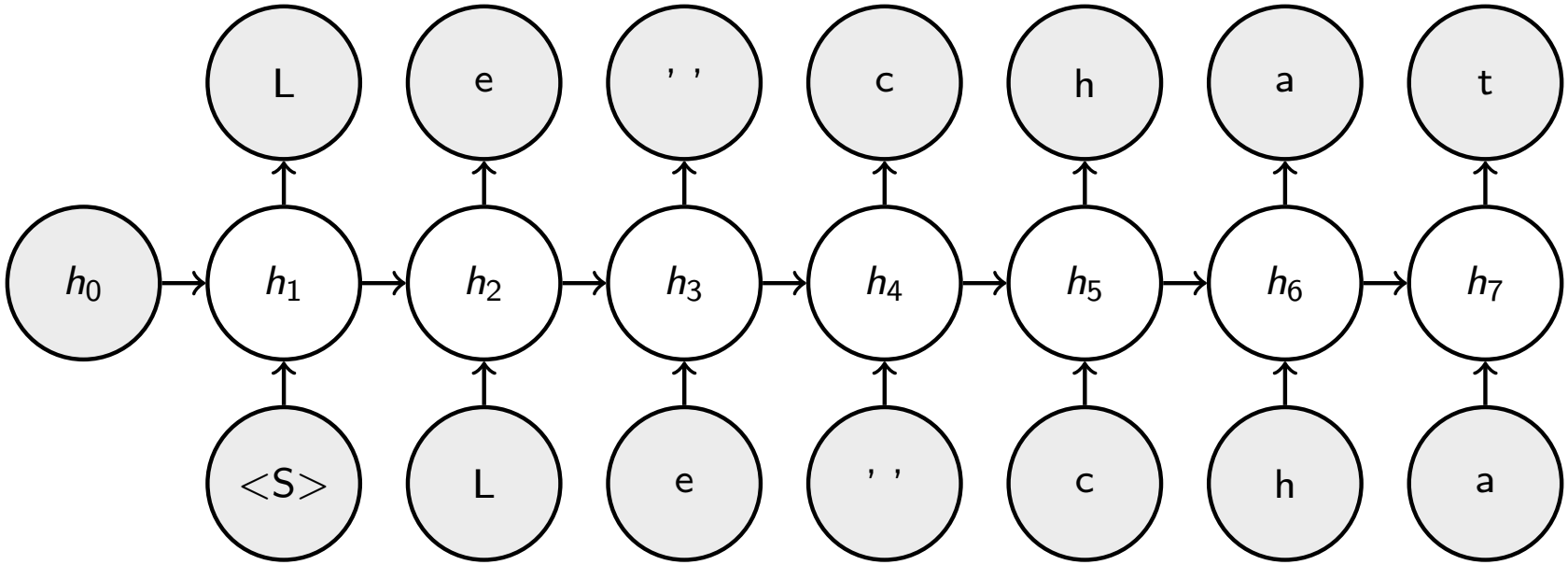
Rappel : RNN



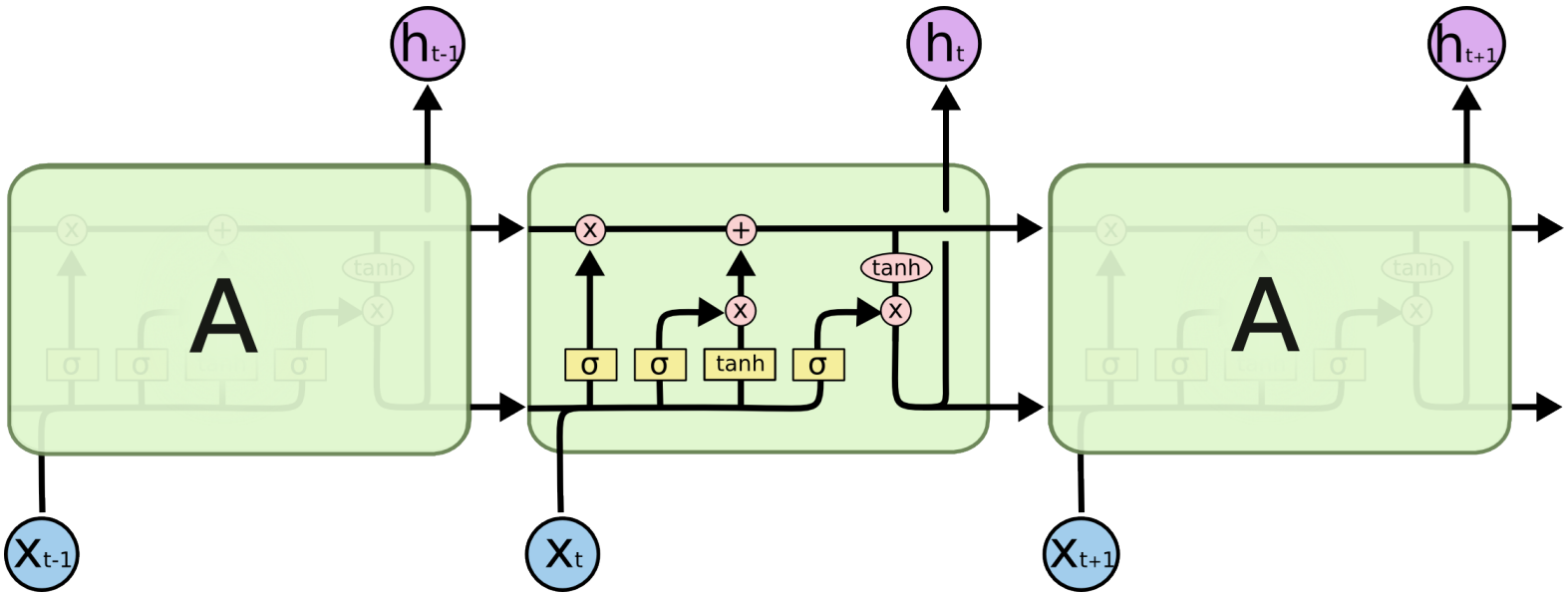
Rappel : RNN



RNN : prédiction de lettre...

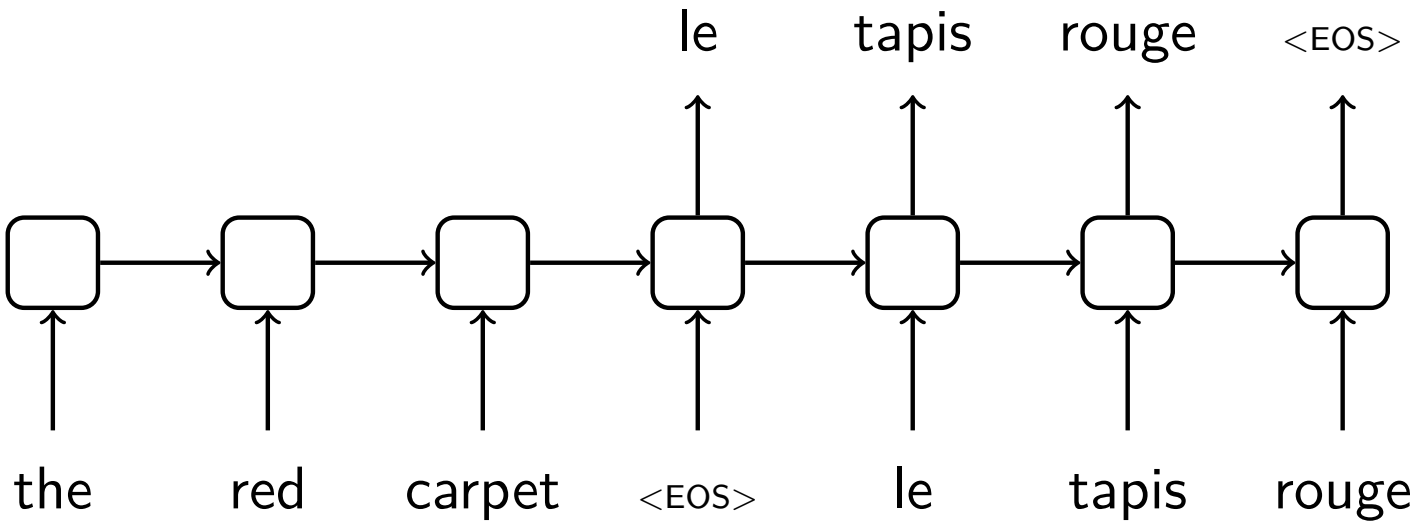


LSTM



Pour la traduction : architecture encodeur/décodeur

Sutskever, I., Vinyals, O., et Le, Q. (2014). « Sequence to sequence learning with neural networks » . *In Advances in Neural Information Processing Systems* (NIPS 2014).



En pratique :

- un LSTM pour lire le texte en entrée : l'encodeur ;
- un LSTM pour produire la sortie : le décodeur ;
- La mémoire du décodeur est initialisée au départ avec le dernier état de l'encodeur.

Limitations

- Le dernier état de l'encodeur (un vecteur de taille fixe) représente la *totalité* du texte en entrée ;
- gère difficilement de longues phrases.

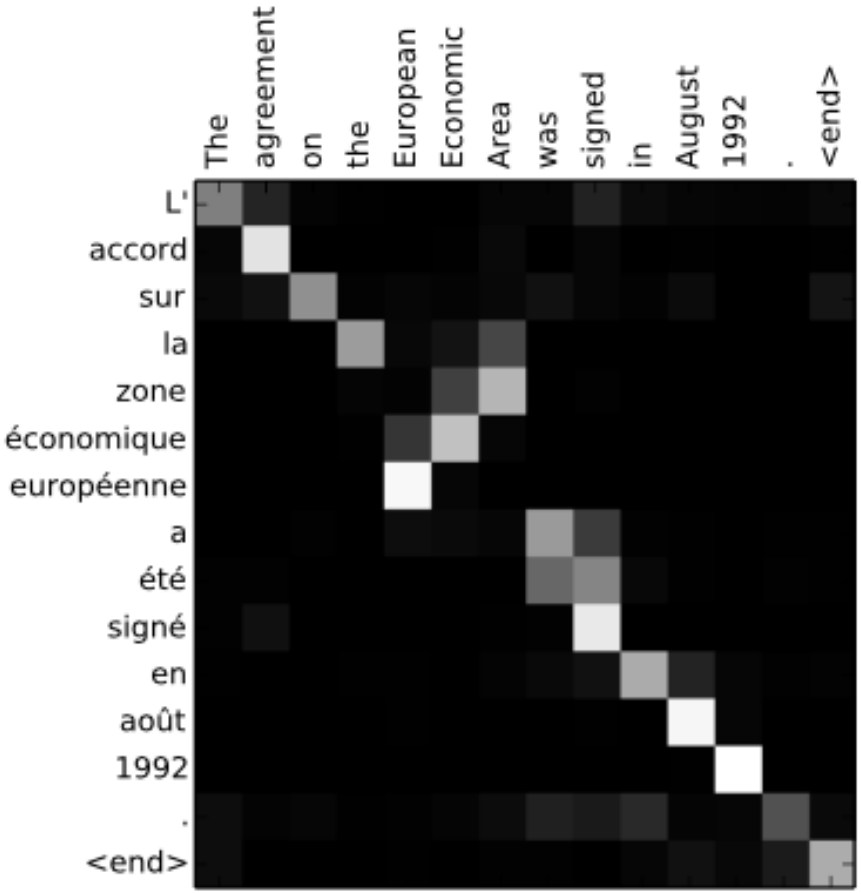
Ajout de l'attention

Bahdanau S., Cho K et Bengio Y « Neural machine translation by jointly learning to align and translate » CoRR, abs/1409.0473, 2014.
Luong M-T, Pham H., et Manning C. D.. « Effective approaches to attention-based neural machine translation » arXiv preprint arXiv :1508.04025, 2015.

- Le décodeur construit la traduction en utilisant la totalité des états intermédiaires de l'encodeur ;
- au lieu d'utiliser le dernier état h_n, on utilise une combinaison linéaire de tous les états, $\sum_{i=1}^{n} a_i h_i$
- le vecteur unitaire a est *l'attention* ;
- il est calculé à chaque étape de la production de la sortie

Implication des mécanismes d'attention

L'attention fournit une matrice a_{ij} qui relie chaque mot i de la sortie avec les mots j de l'entrée.



(Bahdanau et al.)

Attention is all you need ?

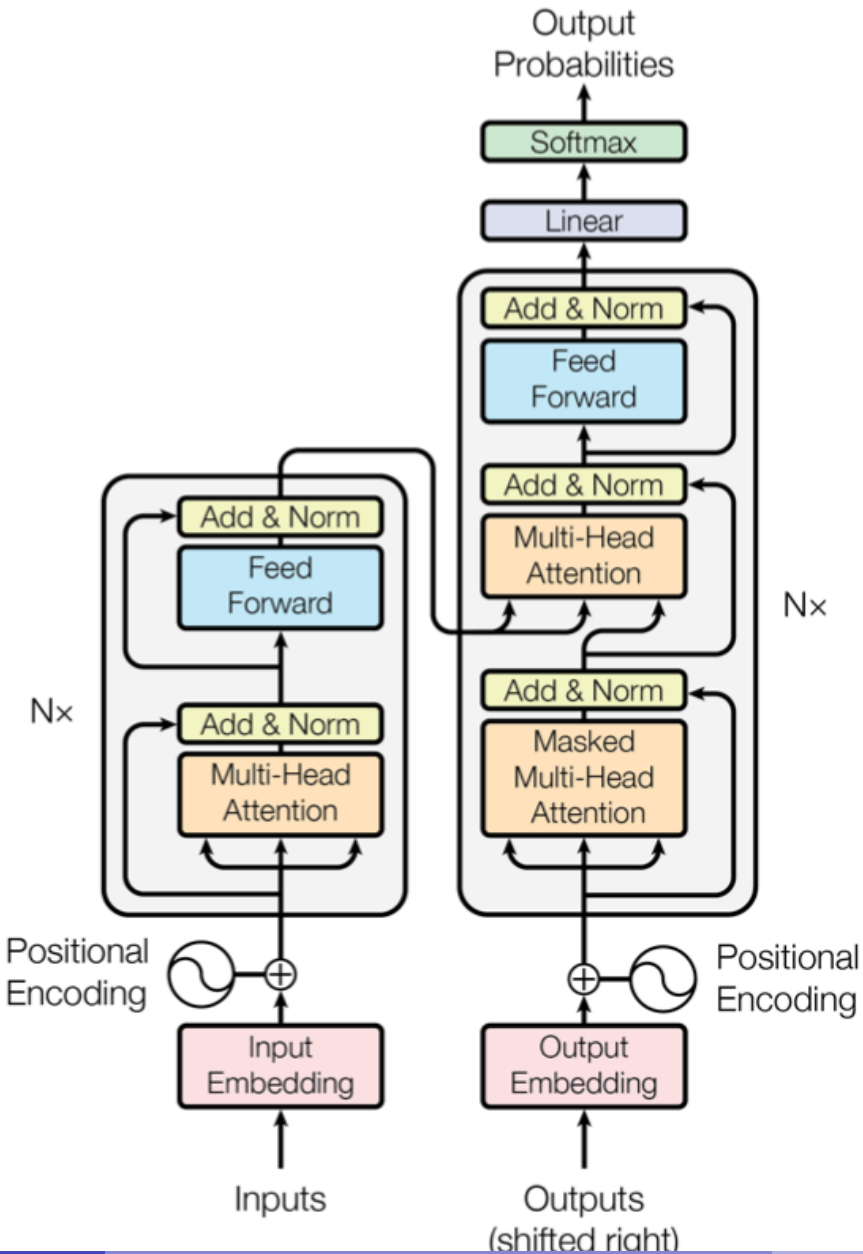
Motivation

- Les mécanismes à base de réseaux récurrents sont lents à entraîner ;
- la rétropropagation du gradient se fait sur toute la séquence
- ils se parallélisent assez mal ;
- en pratique, ils sont à la peine sur les longues phrases.

Idée

Les mécanismes d'attention permettent de relier des séquences de mots ; Pourquoi ne pas se passer des autres mécanismes, et n'utiliser que l'attention ?

Attention is all you need



Entrée et couche d'embedding

- l'entrée est un vecteur d'index (entiers) de dimension variable n, chaque index correspondant à un mot du vocabulaire d'entrée ;
- l'embedding passe à une *matrice* de dimensions (n, e) ;

Différence avec les systèmes récurrents : pas tant dans la représentation initiale, que dans le fait que *tous les mots* de l'entrée seront traités en parallèle.

Mécanisme d'attention

Utilisé deux fois :

- entre l'encodeur et le décodeur, relie les mots de la traduction à ceux du texte initial (voir exemple de Bahdanau et al.) ;
- à l'intérieur de l'encodeur : *self-attention.* Relie les mots du texte d'origine entre eux (idem pour le décodeur).
- la *self-attention* est **cruciale** :
  - pour deux mots w_i et w_j de l'entrée, l'attention a_{ij} sera le seul endroit où les informations sur les deux mots sont combinées entre elles ;
  - dans un réseau récurrent, l'état caché h_j (j $>$ i) est produit à partir de l'état h_i, et permet donc de coder que les deux mots sont présents.

Modélisation de l'attention : Query/Key/Value

Dans les systèmes qui utilisent l'attention (par exemple pour de la traduction)
Notre but : étant donné une séquence d'entrée w_i et une sortie de w', calculer une distribution de probabilité a_i « attention portée par w' à w_i, » et appliquer cette distribution à une famille de vecteurs v_i représentant l'entrée.

$$v' = \sum_i a_i v_i$$

On conceptualise ça en disant que l'attention va être calculée à partir d'une *clef* (provenant de la séquence d'entrée) et d'une *requête* fournie par la sortie, l'attention obtenue étant utilisée pour pondérer des *valeurs* (provenant à nouveau de la séquence d'entrée)
En pratique :

- on n'a pas une, mais plusieurs requêtes ;
- dans notre cas, clef et valeurs ont généralement la même source.

Détails et dimensions pour l'attention

On définit l'attention :

$$\text{Attention}(Q, K, V) = \left(\text{softmax}(\frac{QK^T}{\sqrt{d_k}})\right) V$$

- Q de dimensions $(n' \times d_k)$
- K de dimensions $(n \times d_k)$
- V de dimensions $(n \times d_v)$
- résultat de dimensions $n' \times d_v$

elle n'a pas de paramètres. Ceux-ci seront introduits en amont, en calculant en fait $\text{Attention}(Q_0 W^Q, K_0 W^K, V_0 W^V)$

Attention multi-tête

Idée

Permettre au système d'apprendre plusieurs facteurs de décision indépendants, au lieu de tout moyenner dans *une* attention.

En pratique : on calcule plusieurs fois l'attention :

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots \text{head}_h)$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

On introduit une pénalisation dans la fonction de coût pour éviter que les W_i^Q (resp. K,V) soient égaux pour deux valeurs de i différentes, et ainsi représentent plusieurs phénomènes d'attention.

Intérêt du mécanisme d'attention

- complexité $O(n^2\dot{d})$, contre $O(n\dot{d}^2)$ pour les réseaux récurrents ou convolutifs ;
- très bien parallélisables (pour l'apprentissage - en exploitation, c'est différent) ;
- pour tout couple de mots de l'entrée, quelle que soit leur distances, la profondeur des calculs qui les relie est *fixe* : le système gère plus facilement les dépendances à distance ;
- la visualisation des attentions donne (un peu) d'information sur ce que fait le modèle.

Codage des positions

- Supposons qu'un même mot se retrouve plusieurs fois dans l'entrée ;
- avec le mécanisme d'attention, les lignes qui correspondent à ce mot produiront deux lignes identiques dans la sortie ;
- le mécanisme lui-même est indifférent aux positions.
- il faut donc les coder explicitement dans les données elles-mêmes.

Codage des positions

On ajoute à chaque vecteur représentant un mot un vecteur (de même dimension) codant ses coordonnées.
Soit d la dimension du vecteur PE en question, et p la position du mot ($1 \leq p \leq n$).
On prend :

$$\mathrm{PE}[pos, 2i] = \sin(\frac{p}{1000^{2i/d}})$$
$$\mathrm{PE}[pos, 2i+1] = \cos(\frac{p}{1000^{2i/d}})$$

Intérêt :

- fonctionne quelle que soit la taille du texte ;
- $\exists M \in L(\mathbb{R}^d, \mathbb{R}^d) \mid \mathrm{PE}(pos + k) = M\mathrm{PE}(pos)$

Le batch

- En pratique, on ajoute une dimension de plus, et on travaille sur des tenseurs de dimension 3 ;
- classiquement, on utilise du padding pour chaque étape de mini-batch.

Quelques questions

- Pourquoi ce codage-là des positions ?
- En particulier, pourquoi le sommer avec l'embedding du mot ?
- choix de 1000 ?