Séparer la sémantique de la position dans les transformers: Approche

L'idée est de réserver certaines dimensions de l'embedding pour être positionnelles, exclusivement. C'est un peu comme l'embedding absolu, mais au lieu d'ajouter la représentation positionnelle à celle sémantique, on lui dédie des dimensions!

$$d_{\text{model}} = d_{\text{pos}} + d_{\text{sem}},$$

L'embedding de chaque token peut donc être écrit comme $\mathbf{e}_i = \left[\mathbf{p}_i \parallel \mathbf{s}_i \right] \in \mathbb{R}^{d_{\text{model}}}$, avec $\mathbf{p}_i \in \mathbb{R}^{d_{\text{pos}}}$ (partie positionnelles du token) et $\mathbf{s}_i \in \mathbb{R}^{d_{\text{sem}}}$ (partie sémantique du token).

La taille de p_i (nombre de dimensions positionnelles) est un hyperparamètre. On a d'abord testé $d_{\text{pos}} = d_{\text{head}} = \frac{d_{\text{model}}}{\text{nb\_head}}$.

Afin que le sémantique n'influe pas sur les dimensions positionnelles, nous devons prendre quelques précautions, notamment modifier W_V, W_O ainsi que le MLP.

Au niveau de l'attention

Nous devons forcer W_V à être diagonale blocks:

$$W_V = \begin{pmatrix} W_{\text{pos}} & 0 \\ 0 & W_{\text{sem}} \end{pmatrix}$$

Pour avoir:

$$V = \left[\mathbf{p} \parallel \mathbf{s} \right] \times \begin{pmatrix} W_{\text{pos}} & 0 \\ 0 & W_{\text{sem}} \end{pmatrix} = \left[\mathbf{p}W_{\text{pos}} \parallel \mathbf{s}W_{\text{sem}} \right]$$

Ainsi, en multipliant par l'attention:

$$A = \text{softmax}(\frac{\mathbf{Q} \cdot \mathbf{K}}{\sqrt{d_{\text{head}}}}) \in \mathbb{R}^{n \cdot n}$$

On a: $pW_{\text{pos}} \in \mathbb{R}^{(n \times d_p)}$ $sW_{\text{sem}} \in \mathbb{R}^{(n \times d_s)}$

Et plus précisément, la somme dans: $z_{i,k} = \sum_{j=1}^{d_{\text{model}}} A_{i,j} V_{i,k}$ se passe sur la ligne j et la colonne k fixes. Ainsi, d_{pos} et d_{sem} ne se mélangent pas.

Ensuite, pour le MHA (multi-head), on concatène la sortie de toutes les têtes:

$$\text{MHA} = \left[\mathbf{A}^{h_0} V^{h_0} \parallel \mathbf{A}^{h_1} V^{h_1} \parallel \cdots \parallel \mathbf{A}^{h_H} V^{h_H} \right] \in \mathbb{R}^{n \times (H \times d_{\text{head}})}$$

Au niveau de l'output

Comme pour la Value, on doit s'assurer que cette transformation linéaire ne mélange pas le sémantique et positionnel. On fixe donc W_O diagonale blocks, également:

$$W_O = \begin{pmatrix} W_{O_{\text{pos}}} & 0 \\ 0 & W_{O_{\text{sem}}} \end{pmatrix}$$

Ainsi:

$$\text{attn\_out} = \left[\mathbf{A}^{h_0} V^{h_0} \parallel \mathbf{A}^{h_1} V^{h_1} \parallel \cdots \parallel \mathbf{A}^{h_H} V^{h_H} \right] W_O \in \mathbb{R}^{(n \times d_{\text{model}})}$$

Au niveau du MLP: un peu plus compliqué

Nous devons réaliser 2 MLP différents pour s'assurer ici que les dimensions ne se mélangent pas: un MLP sur les dimensions positionnelles, et un autre sur les dimensions sémantiques. Le résultat final sera tout simplement une concaténation des 2.
On applique d'abord une projection linéaire pour obtenir le mlp_pre. À cette étape, on transforme la sortie de l'attention (AttnOutput) vers une taille intermédiaire, généralement $4 \times d_{\text{model}}$. Il est donc important de veiller à ne pas recombiner les dimensions positionnelles et sémantiques :

$$\text{mlp\_pre} = [\text{AttnOutputPos}\, W_I^{\text{pos}}; \text{AttnOutputSem}\, W_I^{\text{sem}}]$$

Avec :

$$\begin{cases} \text{AttnOutputPos} = \text{AttnOutput}[\cdot, : \text{pos}] \\ \text{AttnOutputSem} = \text{AttnOutput}[\cdot, \text{pos} :] \\ W_I^{\text{pos}} \in \mathbb{R}^{d_{\text{pos}} \times (4 \times d_{\text{pos}})} \\ W_I^{\text{sem}} \in \mathbb{R}^{d_{\text{sem}} \times (4 \times d_{\text{sem}})} \end{cases}$$

Ensuite, on applique les fonctions d'activation (séparemment sur la partie positionnelle et sémantique), donc on obtient:

$$\text{mlp\_post} = [\text{ACT}(\text{AttnOutputPos} W_I^{\text{pos}}); \text{ACT}(\text{AttnOutputSem} W_I^{\text{sem}})]$$

$$\text{mlp\_out} = [\text{mlp\_post\_pos} W_P^{\text{pos}}; \text{mlp\_post\_sem} W_P^{\text{sem}}]$$

Extrapolation à une taille de contexte plus grande - 1

L'architecture proposée sépare explicitement l'espace de représentation $\mathbb{R}^{d_{model}}$ en deux sous-espaces distincts : un espace positionnel $\mathbb{R}^{d_{pos}}$ et un espace sémantique $\mathbb{R}^{d_{sem}}$, où $d_{pos} + d_{sem} = d_{model}$. Cette décomposition présente un avantage fondamental pour l'extrapolation à des longueurs de séquence non vues pendant l'entraînement. Dans notre implémentation, seuls les embeddings positionnels $\mathbf{p}_i \in \mathbb{R}^{d_{pos}}$ dépendent de l'indice de position i, tandis que les représentations sémantiques restent invariantes par rapport à la position. Formellement, pour une position i et un token t, la représentation finale s'écrit :

$$\mathbf{h}_{i,t} = [\mathbf{p}_i \oplus \mathbf{s}_t] \in \mathbb{R}^{d_{model}} \tag{1}$$

où $\oplus$ dénote la concaténation et $\mathbf{s}_t$ est la représentation sémantique du token t. Cette architecture permet de limiter le problème d'extrapolation uniquement aux d_{pos} dimensions positionnelles, préservant ainsi l'intégrité des représentations sémantiques apprises. Lorsque le modèle rencontre une position $i > L_{max}$ (où L_{max} est la longueur maximale vue pendant l'entraînement), seul $\mathbf{p}_i$ nécessite une extrapolation, réduisant considérablement la complexité du problème et améliorant la stabilité des prédictions sur des séquences longues.

Extrapolation à une taille de contexte plus grande - 2

Nous proposons plusieurs approches pour extrapoler les embeddings positionnels au-delà de la fenêtre d'entraînement $[0, L_{max} - 1]$:

Extrapolation sinusoïdale

Inspirée des embeddings positionnels originaux des Transformers, cette méthode utilise des fonctions sinusoïdales pour générer des représentations positionnelles :

$$\mathbf{p}_i^{(2k)} = \sin\left(\frac{i}{10000^{2k/d_{pos}}}\right), \quad \mathbf{p}_i^{(2k+1)} = \cos\left(\frac{i}{10000^{2k/d_{pos}}}\right) \tag{2}$$

où $k \in \{0, 1, \ldots, \lfloor d_{pos}/2 \rfloor\}$. Les embeddings sont ensuite mis à l'échelle par $\sigma = \text{std}(\{\mathbf{p}_j\}_{j=0}^{L_{max}-1})$ pour correspondre à l'amplitude des embeddings appris.

Extrapolation à une taille de contexte plus grande - 3

Extrapolation par analyse de Fourier
Cette méthode décompose les embeddings appris en composantes fréquentielles et reconstruit les positions extrapolées :

$$\mathbf{p}_i = \frac{2}{L_{max}} \sum_{k=1}^{K} \text{Re}\left[\hat{\mathbf{p}}_k \cdot e^{2\pi jki/L_{max}}\right] \quad (3)$$

où $\hat{\mathbf{p}}_k$ sont les K coefficients de Fourier dominants obtenus par FFT des embeddings appris, avec $K \ll L_{max}$ pour éviter le sur-apprentissage.

Extrapolation à une taille de contexte plus grande - 4

Extrapolation cyclique linéaire
Cette approche combine une répétition cyclique avec un décalage linéaire :

$$\mathbf{p}_i = \mathbf{p}_{i \bmod L_{max}} + \left\lfloor \frac{i}{L_{max}} \right\rfloor \cdot \boldsymbol{\delta} \tag{4}$$

où $\boldsymbol{\delta} = \frac{1}{L_{max}-1} \sum_{j=1}^{L_{max}-1} (\mathbf{p}_j - \mathbf{p}_{j-1})$ représente la dérive positionnelle moyenne.

Extrapolation à une taille de contexte plus grande - 5

Extrapolation apprise
Un réseau de neurones apprend une transformation continue des positions normalisées :

$$\mathbf{p}_i = (1 - \alpha_i) \cdot \mathbf{p}_{i \bmod L_{max}} + \alpha_i \cdot f_\theta \left(\frac{i}{L_{max}} \right) \tag{5}$$

où $f_\theta : \mathbb{R} \to \mathbb{R}^{d_{pos}}$ est un MLP et $\alpha_i = \sigma \left(5 \cdot \left(\frac{i}{L_{max}} - 1 \right) \right)$ assure une transition douce.

Extrapolation à une taille de contexte plus grande - 6

Extrapolation de type ALiBi
Inspirée d'ALiBi, cette méthode applique une décroissance logarithmique aux embeddings :

$$\mathbf{p}_i = \mathbf{p}_{\min(i, L_{max}-1)} \cdot \frac{1}{1 + \log(1 + \max(0, i - L_{max} + 1))} \tag{6}$$

Cette approche maintient la structure des embeddings tout en atténuant progressivement leur magnitude pour les positions lointaines.