DEEPL - TP 9
Mécanisme d'attention propre

Nicolas Baskiotis - Stéphane Rivaud - Benjamin Piwowarski - Laure Soulier

2025-2026

Introduction (brève, cf cours)

Dans ce TP, nous allons nous intéresser aux mécanismes d'attention *propre* (*self-attention*).

Contrairement au TP précédent, le but de l'attention n'est pas de construire une représentation de taille fixe d'une séquence $x_1^{(0)}, \dots, x_n^{(0)}$, mais une séquence de représentations *contextualisée* $x_1^{(L)}, \dots, x_n^{(L)}$ où $x_i^{(L)}$ est la ième entrée contextualisée : chaque élément de la séquence d'entrée calcule une représentation qui lui est propre, en utilisant un mécanisme d'attention dont la question est donnée par la représentation de l'élément. Le processus est répété L fois afin d'obtenir une meilleure représentation finale.

Concrètement, chaque élément $x_i^{(l)}$ de la séquence après l couches est dérivée en trois représentations en utilisant trois couches linéaires : une représentation *query* $q_{\theta^{(l)}}(x_i^{(l)})$, une représentation *key* $k_{\theta^{(l)}}(x_i^{(l)})$ et une représentation *value* $v_{\theta^{(l)}}(x_i^{(l)})$. Les représentations query et key vont permettre de calculer l'attention $\alpha_{ij}^{(l)}$ que l'élément i porte à l'élément j :

$$\log p(a_{ij}^{(l)}) = \text{constante} + \frac{1}{\sqrt{d}} q_{\theta^{(l)}}(x_i^{(l)}) \cdot k_{\theta^{(l)}}(x_j^{(l)})$$

où d est la dimension des représentations (le $d^{-1/2}$ permet d'avoir toujours les mêmes magnitudes de gradient quelque soit la dimension).

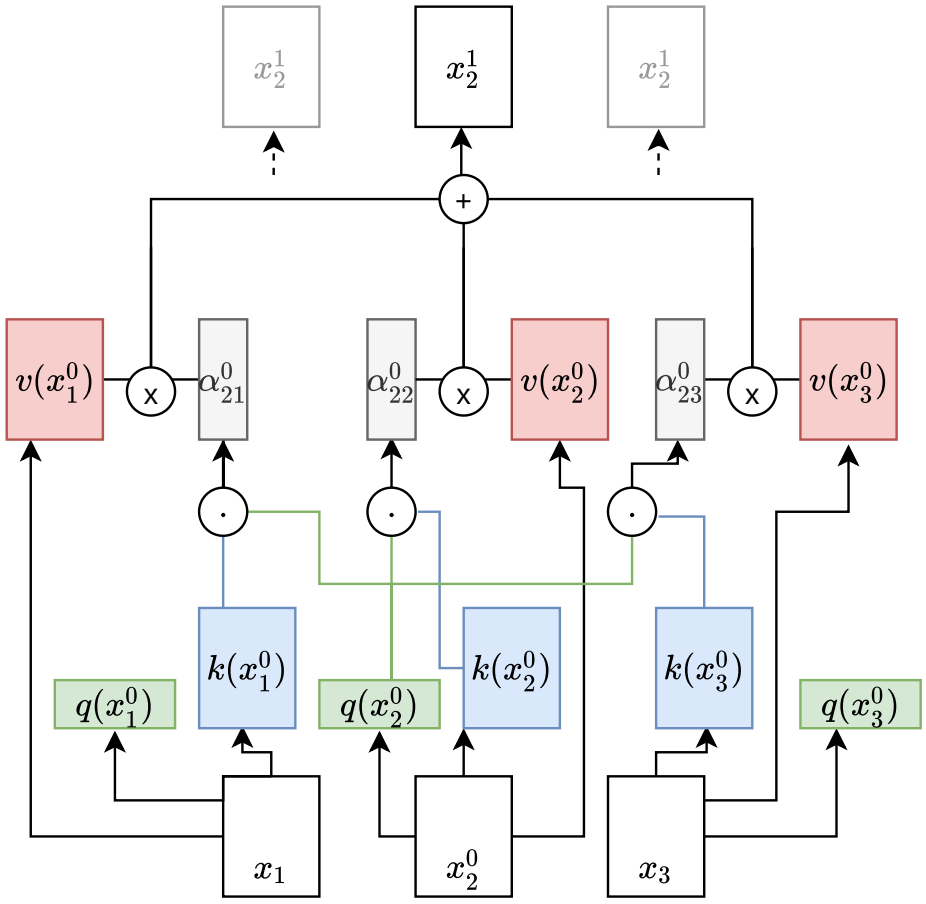
La représentation $x_i^{(l+1)}$ de l'élément à la couche d'après est calculée en faisant la moyenne des *value* des éléments de la couche l pondérée par l'attention ainsi calculée :

$$f_{\theta^{(l)}}(x_i^{(l)}; x_1^{(l)}, \dots, x_n^{(l)}) = \sum_{j=1}^{n} p(a_{ij}^{(l)}) v_{\theta^{(l)}}(x_j^{(l)}) \tag{1}$$

$f_{\theta^{(l)}}$ est une fonction d'attention (propre) où la question, clef et valeur sont des fonctions linéaires de chaque entrée $x_i^{(l)}$. Les fonctions sont les mêmes pour toutes les entrées mais sont distinctes d'une couche l à une autre (i.e. chaque module calculant la question q, la clef k ou la valeur v a les mêmes paramètres pour un l donné).

On introduit souvent une dernière couche linéaire suivi d'une non-linéarité (une fonction ReLU par exemple) pour obtenir la représentation de $x_i^{(l+1)}$:

$$x_i^{(l+1)} = g_{\theta^{(l)}}(f_{\theta^{(l)}}(x_i^{(l)}; x_1^{(l)}, \ldots, x_n^{(l)}))$$



Préparation des données et du modèle

Reprenez le code du TP8 que vous allez étendre pour inclure des modèles basés sur l'attention propre : comme dans le TP8, vous utiliserez les données IMDB qui contient 50,000 commentaires venant de Internet Movie Database (IMDb), et des plongements de mots Glove (voir le code fourni pour le TP). Le but est de classifier un commentaire comme étant "positif" (pos) ou "négatif" (neg) ; la mesure de performance est le taux de bonne classification.

1 Modèle de base

Question 1

Implémentez un modèle de base basé sur l'attention propre avec $L = 3$. Afin de classifier, vous utiliserez comme représentation finale la moyenne. Comparez les performances avec celles du TP 9 (en particulier le module mean).

2 Modèle résiduel

Lorsque le nombre de couches augmente sensiblement, l'apprentissage est très dégradé. On utilise pour palier ce problème une architecture résiduelle :

$$x_i^{(l)} = g_\theta \left(\tilde{x}_i^{(l-1)} + f_\theta(\tilde{x}_i^{(l-1)}; \tilde{x}_1^{(l-1)}, \ldots, \tilde{x}_n^{(l-1)}) \right)$$

Une normalisation de l'entrée est souvent nécessaire afin de stabiliser le réseau : une Layer Normalization est employée à cette fin pour obtenir les entrées normalisées $\tilde{x}_i^{(l-1)}$.

Question 2

Implémentez le modèle résiduel et comparez les résultats au modèle précédent.

3 Ajout des plongements de position

Le modèle décrit plus haut a un problème important : il n'y a aucune notion de séquence prise en compte, ce qui pose problème pour des tâches telles que la détection de sentiment (ex. pour bien prendre en compte la négation, "I did not like" vs "I did not know that I would enjoy").

Afin de palier ce problème, les modèles d'attention propre utilisent des plongements de position, i.e. on associe à chaque position i un vector pe_i dont la h-ème composante est définie comme suit (d est la dimension de l'espace latent) :

$$pe_{ih} = \begin{cases} \sin\left(i/10000^{h/d}\right) & \text{si h est pair} \\ \cos\left(i/10000^{(h-1)/d}\right) & \text{sinon} \end{cases}$$

Question 3

Dans le code fourni, vous trouverez une classe `PositionalEncoding` qui permet d'ajouter à des séquences de représentations (batch × length × k). Il faut spécifier au départ la longueur maximum de tout batch pour que les vecteurs soient pré-calculés.
Afin de comprendre ce que fait cette fonction, calculez le produit scalaire entre pe_i et pe_j (pour tout i et j) sous forme d'une carte heatmap.
Modifiez le modèle de l'exercice 1 en ajoutant des *postional embeddings* à la représentation initiale (i.e. remplacer les $x_i^{(0)}$ par $x_i^{(0)} + pe_i$).

Il est également possible d'apprendre une représentation des positions plutôt que de les pré-calculer analytiquement. Les modèles plus récents (type BERT) utilisent une représentation des positions relatives plutôt que absolues.

4 Ajout d'un token CLS

Les modèles de type transformer utilisent une autre technique pour calculer une représentation de taille fixe permettant de prédire la classe, en introduisant un token spécial `CLS` au début de la séquence à classifier, i.e. $x_1^{(0)} = x_{CLS}$. L'embedding de ce token est appris, et la représentation contextualisée $x_1^{(L)}$ est utilisée pour prédire la classe.

Question 4

Ajouter le pseudo-token `CLS` et comparez les performances avec le modèle précédent.

Liens et références

- Le papier originel "Vaswani et al., Attention is All You Need, 2017"
- Transformers from scratch (Août 2019) : une entrée de blog très bien faite sur les transformers.