AS - TP 10
Mécanisme d'attention propre

Nicolas Baskiotis - Benjamin Piwowarski

2019-2020

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 $x_1^{(0)}, \ldots, x_n^{(0)}$, mais une séquence *contextualisée* $x_1^{(L)}, \ldots, x_n^{(L)}$ où $x_i^{(L)}$ est une représentation 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.

De manière formelle, la représentation du $i$ème élément à l'étape l est donnée par la fonction résiduelle :

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

On voit ici deux nouveautés par rapport au mécanisme d'attention vu au TP 9 qui sont essentielles pour pouvoir apprendre de manière efficace les L couches du modèle.

- L'utilisation d'une fonction résiduelle qui modifie la représentation ;
- Une normalisation de l'entrée : $\tilde{x}_i^{(l-1)}$ est normalisé afin que la moyenne et la déviation standard soit proche de zéro sur le corpus d'apprentissage (*BatchNorm* vu en TP 8).
- L'utilisation d'une fonction g_θ permettant de modifier la représentation avant la couche d'attention propre suivante (utilisez un module linéaire suivi d'une non linéarité, ou d'autres fonctions plus complexes en fonction du sur-apprentissage observé ou non).

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

$$f_\theta(x_i^{(l-1)}; x_1^{(l-1)}, \ldots, x_n^{(l-1)}) = \sum_{j=1}^{n} p(a_i^l) v_\theta^{(l)}(\tilde{x}_j^{(l-1)}) \tag{1}$$

avec

$$\log p(a_i^l) = \text{constante} + \frac{1}{\sqrt{k}} q_\theta^{(l)}(\tilde{x}_i^{(l-1)}) \cdot k_\theta^{(l)}(\tilde{x}_j^{(l-1)})$$

où k est la dimension des représentations (le $k^{-1/2}$ permet d'avoir toujours les mêmes magnitudes de gradient quelque soit la dimension). Un dropout peut également être utilisé sur l'attention.

Préparation des données et du modèle

Reprenez le code du TP9 que vous allez étendre pour inclure des modèles basés sur l'attention propre : comme dans le TP9, 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.

Exo 1 : Modèle de base

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).

Exo 2 : 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 à ce problème, les modèles d'attention propre utilisent des plongements de position, i.e. on associe à chaque position i une représentation pe_i.

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

Dans le code fourni, vous trouverez une classe `PositionalEncoding` qui permet d'ajouter à des séquences de représentations (batch $\times$ length $\times$ k). Afin de comprendre ce que fait cette fonction, calculez le produit scalaire entre pe_i et pe_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. les $x_i^{(0)}$).

Exo 3 : 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.

Liens et références

- Le papier originel [**VaswaniAttentionAllYou2017**]
- Transformers from scratch (Août 2019) : une entrée de blog très bien faites sur les transformers.