Entre performance et frugalité en TAL : Approches pour la réduction de la taille des (L)LMs

Xavier Pillet, Anastasia Volkova, Nicolas Greffard, Richard Dufour

ENTRE PERFORMANCE ET FRUGALITÉ EN TAL : APPROCHES POUR LA RÉDUCTION DE LA TAILLE DES (L)LMS

Xavier Pillet
Nantes Université, LS2N & Valeuriad
pillet.xavier@valeuriad.fr

Anastasia Volkova
Inria Lyon, CITI
anastasia.volkova@inria.fr

Nicolas Greffard
Valeuriad
greffard.nicolas@valeuriad.fr

Richard Dufour
Nantes Université, LS2N
richard.dufour@ls2n.fr

25 Janvier 2024

Introduction

Ces dernières années, les performances des tâches de traitement automatique du langage (TAL) ont largement progressé notamment au travers des modèles de langues pré-entraînés. Ces avancées se sont traduites par une augmentation importante de la taille de ces modèles [1], les rendant de plus en plus coûteux en matériel et en temps de traitement, que ce soit au niveau de leur entraînement et/ou de leur utilisation en inférence. Pour pallier ces problèmes, de nombreux travaux ont émergé autour de l'optimisation des modèles de langue pré-entraînés. Cet article résume les principales approches actuelles sur la réduction du coût mémoire des paramètres (*e.g.* les poids et les biais), considérant le fait que la RAM des GPU est un goulot d'étranglement majeur pour les modèles les plus larges [2].

Méthodes

Plusieurs approches existent, notamment celles qui consistent à diminuer le nombre de paramètres (élagage) et celles consistant à en réduire la précision des formats numériques (quantification).

Élagage

L'élagage (*pruning* en anglais) consiste généralement à forcer certains paramètres à 0. Ces nombreux 0 ne sont vraiment intéressants que s'ils exploitent avantageusement les possibilités logicielles (*e.g.* la structure de données du tenseur parcimonieux) et les accélérations matérielles, ce qui est plus facile si l'emplacement des 0 est connue à l'avance. Nous distinguons alors l'élagage non-structuré (l'emplacement des 0 est *a priori* aléatoire) du structuré.

Une des premières techniques non-structurées consiste à utiliser l'information du second ordre. Après entraînement, nous pouvons supprimer les petits paramètres qui se situent dans les directions propres de la *hessienne* à petites valeurs propres (*i.e.* ses directions propres plates) [3]. Comme le gradient ne change pas beaucoup le long de ces directions et qu'il est presque nul à la fin de l'entraînement, cela signifie que la fonction de coût y est aussi stable et que les paramètres peuvent être légèrement modifiés le long de ces directions, sans impacter la performance du modèle.

Une autre méthode d'élagage est celle du *ticket gagnant*. Un ticket gagnant est une combinaison de paramètres qui contient un grand pourcentage de 0, à la fin de l'entraînement. Cela peut être obtenu au prix de nombreux réentraînements coûteux [4]. Cette parcimonie est non-structurée, mais elle peut atteindre jusqu'à 90 % de 0 pour les modèles TAL [4]. Ceci suggère une sur-paramétrisation initiale importante et que l'on devrait pouvoir, dès l'entraînement, utiliser moins de paramètres. Cela est en effet possible avec l'heuristique d'entraînement des *tickets truqués*, où de nombreux paramètres sont initialisés et maintenus à 0 au cours de l'entraînement, sauf ceux à la dérivée (*i.e.* l'erreur) la plus forte (donc au meilleur potentiel pour l'apprentissage) qui sont insérés [5].

On peut aussi concevoir des couches nativement parcimonieuses, ce qui permet d'obtenir en plus une parcimonie structurée. Par exemple, utiliser des matrices de poids creuses dans les couches linéaires. Si l'on souhaite rendre un transformer complètement parcimonieux, nous devons concevoir une version parcimonieuse à la fois des têtes d'attention et du réseau direct en sortie. Pour les têtes d'attention, il y a le *longformer* [6], qui utilise aussi des matrices creuses. Pour le réseau direct, une version parcimonieuse est la *mixture d'experts* [7] fondée sur une softmax parcimonieuse. Combinées, ces deux couches forment un transformer complètement parcimonieux, pouvant bénéficier d'accélération matérielle et traiter de longs documents, comme dans le cas du *terraformer* [8].

Quantification

La quantification (*quantization* en anglais) est une approche de réduction de la taille des modèles qui consiste à réduire la précision numérique des paramètres pour optimiser la consommation mémoire et le temps de calcul. La paradigme existant pour l'entraînement est de stocker les paramètres en format à virgule flottante (*floating point* (FP) en anglais) de taille 32 bits (FP32), mais de faire les multiplications sur GPU en FP16 [9]. Cette précision mixte FP32/FP16 a permis l'émergence des gigamodèles de langue (*Large Language Models* en anglais, ou LLMs) [1]. Pour l'inférence on a pas besoin de la virgule flottante et on peut tirer partie de la vitesse de calcul plus rapide des entiers, notamment à 8 bits (INT8) [10]. En revanche, le passage des flottants vers les entiers après entraînement (*post-training quantization* en anglais) n'est pas trivial et influe sur les performances. Cela peut-être fait en utilisant l'information de la hessienne, de façon similaire à l'élagage [10]. Cependant, si l'on dispose des données d'entraînement, on peut faire du ré-entraînement en quantifiant (*quantization aware training* en anglais), ce qui est souvent plus performant [11].

L'apparition de nouvelles architectures GPU avec un support matériel d'opérations au format FP8 [12] rend possible l'entraînement directement en précision mixte FP16/FP8. La quantification en INT8 n'apparaît donc plus nécessaire [2]. Cependant, les entiers sont meilleurs pour représenter des nombres uniformément distribués et sont plus rapides en calcul, alors que les flottants représentent mieux les nombres entre 0 et 1 par exemple [13]. Or, un effet de seuil semble apparaître pour les modèles suffisamment larges (de l'ordre la dizaine de milliards de paramètres) qui voient apparaître une forte asymétrie dans la distribution des valeurs des poids des couches d'attention. Cette asymétrie est corrélée à une amélioration importante des performances et rend les poids mieux représentables par des FP8 [2]. Donc pour les modèles en dessous du milliard de paramètres, quantifier en INT8 reste pertinent, si l'on lisse la distribution des poids des attentions, grâce à la régularisation L_2 par exemple [13]. Au delà, mieux veut utiliser les FP8 pour les LLMs, pour conserver le phénomène d'émergence [13]. Des travaux en cours étudient un nouveau format numérique, à base de micro-exposants (MX), qui factorise en commun les exposants des tenseurs et sous-tenseurs, ce qui permettrait de bien représenter ces distributions asymétriques, tout en étant efficace en termes de stockage [14].

Matrices à bas rang

S'inspirant de l'algèbre linéaire pour le calcul haute performance, afin d'économiser de la RAM, on peut approcher les grandes matrices avec de beaucoup plus petites, grâce à des approximations à bas rang. Cette nouvelle représentation est efficace en termes de consommation de mémoire et on peut aussi entraîner directement les décompositions de tenseurs. Cela est notamment fait dans les smartphones avec les MobileNets [15].

Adaptateurs

Pour faire face au coût important du ré-entraînement des LLMs, la solution des *adaptateurs* a été proposée : elle consiste à intercaler des couches adaptatrices en leur sein [16]. On peut alors ré-entraîner uniquement les matrices de paramètres de ces couches adaptatrices, sans toucher au reste du modèle, ce qui permet de pouvoir n'utiliser qu'un LLM générique et de l'adapter à plusieurs domaines différents juste en changeant les adaptateurs. Comme ces adaptateurs rajoutent des sous-couches, ils ralentissent un peu les calculs et rajoutent des paramètres. L'utilisation de matrices adaptatrices à bas rang permet de minimiser cet impact, ce qui donne les méthodes LoRA (de l'anglais *Low Rank Adapter*) [17], ou QLoRA pour les modèles quantifiés [18].

Distillation

La *distillation* consiste à entraîner un petit réseau à imiter le comportement d'un plus large [19]. Elle est coûteuse en termes de ré-entraînement mais parvient à comprimer efficacement certains modèles [20] et permet de tirer indirectement partie de leurs corpus d'entraînement parfois privés et onéreux. Certains fournisseurs s'en protègent en l'interdisant *via* des licences, telle que celle de LLaMA 2. La distillation peut être mixée avec de la quantification [21].

Défis méthodologiques

L'étude des différentes méthodes de réduction de (L)LM pose certains défis méthodologiques. Premièrement, la plupart utilisent un ré-entraînement ou affinage final. Hors, beaucoup sont sous-entraînés [22]. Cela pose alors la question de l'origine de la restauration de la performance après réduction : est-t-elle due à l'efficacité de la méthode, ou au fait que l'entraînement était poursuivi ? Deuxièmement, le coût parfois prohibitif pour la recherche académique de leur entraînement est un autre obstacle à la reproductibilité scientifique des résultats, ce qui limite leur pertinence. Finalement, de plus en plus de métriques sur des tâches différentes sont utilisées, dont la pertinence et la qualité est discutable. Par exemple l'une des performances émergeantes est sur une tâche d'addition à 8 bits, ce qui est ridicule compte tenu de la taille du modèle : pour 8 bits, il y a $2^8 \times 2^8 = 65536$ cas d'additions possibles. Donc avec plus de 7 milliards de paramètres, le modèle peut avoir tout appris par cœur, surappris ces additions [2].

Conclusion et perspectives

Diverses approches de réduction de la taille des grands modèles de langues ont été étudiées. De nombreuses questions restent en suspens, telles que : Si l'on arrive à élaguer à ce point les modèles, a-t-on réellement besoin d'autant de paramètres ? Quelle est la performance maximale que l'on peut atteindre avec les modèles frugaux ? Comment la quantification, par exemple, influence-t-elle spécifiquement certaines tâches ou compétences langagières des modèles ?

Références

[1] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama : Open and efficient foundation language models. *arXiv preprint arXiv :2302.13971*, 2023.

[2] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 () : 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv :2208.07339*, 2022.

[3] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[4] Sai Prasanna, Anna Rogers, and Anna Rumshisky. When bert plays the lottery, all tickets are winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3208–3229, 2020.

[5] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery : Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

[6] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer : The long-document transformer. *arXiv preprint arXiv :2004.05150*, 2020.

[7] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks : The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv :1701.06538*, 2017.

[8] Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is enough in scaling transformers. *Advances in Neural Information Processing Systems*, 34 :9895–9907, 2021.

[9] Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. Mixed precision training with 8-bit floating point. *arXiv preprint arXiv :1905.12334*, 2019.

[10] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert : Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

[11] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert : Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

[12] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv :2209.05433*, 2022.

[13] Mart van Baalen, Andrey Kuzmin, Suparna S Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, et al. Fp8 versus int8 for efficient deep learning inference. *arXiv preprint arXiv :2303.17951*, 2023.

[14] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv :2310.10537*, 2023.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets : Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv :1704.04861*, 2017.

[16] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[17] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora : Low-rank adaptation of large language models. *arXiv preprint arXiv :2106.09685*, 2021.

[18] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora : Efficient finetuning of quantized llms. *arXiv preprint arXiv :2305.14314*, 2023.

[19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert : smaller, faster, cheaper and lighter. *arXiv preprint arXiv :1910.01108*, 2019.

[20] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep ? *Advances in neural information processing systems*, 27, 2014.

[21] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv :1802.05668*, 2018.

[22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv :2203.15556*, 2022.