AdaLomo: Low-memory Optimization with Adaptive Learning Rate

Kai Lv, Hang Yan, Qipeng Guo, Haijun Lv, Xipeng Giu

Eline POT & Seonjae KANG, Sorbonne Université

Introduction

Les grands modèles de langage (LLMs) ont été sujets de grande attention depuis leurs premières apparitions, et ont continué de montrer leurs applications dans diverses tâches de traitement des langues automatiques. Cependant, à l'amélioration de leur performance s'imposent simultanément des contraintes d'optimisation. Dans les domaines d'application où l'efficacité est cruciale, mitiger les difficultés engendrées par la taille et complexité des modèles devient d'autant plus important.

Dans cet article, les auteurs introduisent **AdaLomo (Low-memory optimization with Adaptive Learning Rate)**, un algorithme d'optimisation pour les grands modèles de langage, qui vise à égaler la performance de la baseline tout en augmentant l'efficacité de la gestion de mémoire.

Motivation

La conception d'AdaLomo repose sur une analyse des distinctions entre deux algorithmes d'optimisation, LOMO et Adam.

LOMO (LOw-Memory Optimization)

LOMO a été proposé pour rendre le fine-tuning des paramètres de LLMs plus efficace en terme de mémoire. Pendant la backpropagation, le calcul des gradients et la mise à jour des paramètres sont simultanés, dans le but de réduire l'utilisation de mémoire. Cette procédure est appelée le « fused backward ».

Conséquence du fused backward pour la mitigation du phénomène de « vanishing gradient » par normalisation de gradient : nécessité d'avoir deux passes backward (un pass pour calculer la norme du gradient et un autre pour mettre à jour les paramètres).

LOMO est efficace en terme d'usage de mémoire, mais est sensible au choix d'hyper-paramètres et peut se retrouver bloqué dans des minimas locaux. Les modifications souvent proposées pour résoudre ces limites augmentent souvent la consommation de mémoire.

Adam (Adaptive Moment Estimation)

Adam est un optimiseur combinant les avantages du SGD (Stochastic Gradient Descent) avec une estimation adaptative des moments pour mettre à jour les taux d'apprentissage de chaque paramètre du modèle. L'estimation du second moment (ou variance) a un impact particulier sur la convergence globale d'Adam, qui performe bien mieux que LOMO sous cette métrique.

Les mises à jour de cette estimation de la variance s'écrivent

$$\begin{cases} \boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \\ \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t} \\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{\boldsymbol{g}_t}{\sqrt{\hat{\boldsymbol{v}}_t}+\epsilon} \end{cases},$$

où $\boldsymbol{\theta}_t$ les paramètres du modèle au temps t, $\boldsymbol{g}_t$ le gradient de $\boldsymbol{\theta}_t$, $\boldsymbol{v}_t$ le second moment (variance) au temps t, et α, β_2 des hyperparamètres représentant respectivement le taux d'apprentissage et le taux d'exponential decay du second moment.

Il se trouve que l'estimation du second moment est particulièrement efficace pour traiter les données sparses, et permet aux paramètres qui sont rarement mis à jour de recevoir des pas d'updates plus importants. Le second moment peut être décomposé ou compressé afin de réduire l'utilisation de la mémoire.

AdaLomo

Le but est donc de combiner l'efficacité en terme de mémoire de LOMO avec la performance d'Adam. A partir de ces analyses, les auteurs décident d'incorporer une estimation du second moment et d'abandonner celle du premier moment.L'article reprend pour ce faire l'idée de factorisation en matrices non négatives (NMF) : c'est la décomposition de la variance de l'algorithme **Adafactor**, qui ne garde en mémoire qu'une représentation factorisée des gradients précédents lors de l'apprentissage : v_t est factorisé en r_t, c_t, et ce sont ces deux états d'optimiseur qui sont stockés en mémoire plutôt que v_t.

On utilise le principe de Grouped Update Normalization qui permet des ajustements adaptatifs pour chaque paramètre afin de garantir la stabilité du modèle. Cette méthode assure des mises à jour pertinentes pour chaque paramètre, contrairement à la normalisation globale qui peut ralentir ou accélérer à tort les mise à jour de certains paramètres. C'est la raison pour laquelle on applique la normalisation Root Mean Square avec : $RMS(U_t) = \sqrt{\text{Mean}(U_t^2)}$.

Algoritme 1 AdaLomo

Données: $f(\cdot)$ modèle avec paramètre $\boldsymbol{\theta}$
α taux d'apprentissage
T nombre de pas max
$\mathcal{D}$ dataset d'apprentissage
$\mathcal{L}$ fonction de perte
β coefficient de décroissance des poids (weight decay)
ϵ constante de régularisation

pour $t = 1$ à T **faire**
  Échantillon $\mathcal{B} = (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
  $\hat{\boldsymbol{y}} \leftarrow f(\boldsymbol{x}, \boldsymbol{\theta})$
  $\ell \leftarrow \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$
  pour chaque θ_i dans l'ordre de la backpropagation **faire**
    $g_{t,i} = \nabla_{\theta_{t-1,i}} \ell$
    $r_{t,i} = \beta r_{t-1,i} + (1-\beta) g_{t,i}^2 \mathbf{1}_n$
    $c_{t,i} = \beta c_{t-1,i} + (1-\beta) \mathbf{1}_m^T g_{t,i}^2$
    $v_{t,i} = r_{t,i} c_{t,i}$
    $u_{t,i} = g_{t,i} / v_{t,i}$
    $\hat{u}_{t,i} = u_{t,i} / max(1, RMS(u_{t,i})) \times max(\epsilon, RMS(\theta_{t-1,i}))$
    $\theta_{t,i} = \theta_{t-1,i} - \alpha_t \hat{u}_{t,i}$
    $g_{t,i-1} \leftarrow$ None
  fin pour
fin pour

Experiences

Nous avons recodé l'optimiseur AdaLomo en reprenant les idées du code des auteurs afin de pouvoir l'utiliser aisément. Nous avons testé la performance d'AdaLomo sur deux tâches de traitement de langues automatique: la **génération de texte** et la **traduction**.Pour ces deux tâches, on compare AdaLomo avec les optimiseurs Adam et SGD.

- **Génération de texte :** Génération de texte avec un LSTM, sur le jeu de données wikitext-2. On utilise la perplexité comme métrique d'évaluation (l'exponentielle de la moyenne du logarithme négatif de vraisemblance de l'ensemble de test).
- **Traduction :** Implémentation sur le jeu de données de traduction anglais/français (170 651 paires de phrases) sur un modèle Seq2Seq. On utilise la Cross Entropy Loss comme métrique. On évalue aussi l'utilisation de la mémoire du système.

Résultats

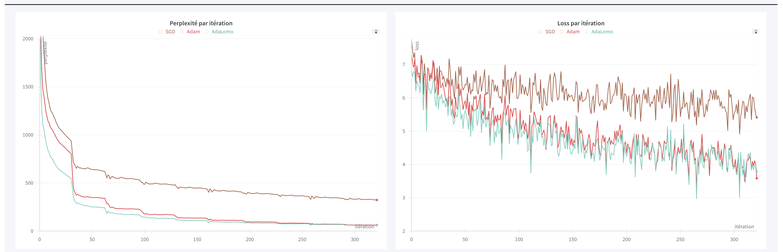

Figure 1. Résultats pour la Génération de texte

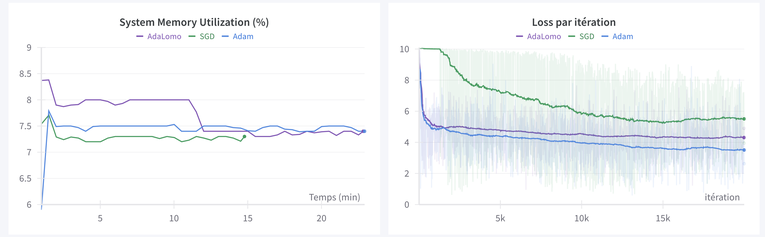

Figure 2. Résultats lors de l'apprentissage pour la Traduction anglais-français

Conclusion

- Article assez clair dans les idées générales mais certaines parties et choix pour l'algorithme manquent d'explication/ justification.
- **Performance** de AdaLomo **similaire à Adam** dans nos expériences en terme de convergence, **apport en efficacité de la gestion de mémoire moins évident**.
- Difficle de voir le véritable apport de cet article puisque Adafactor propose déjà Adam avec la factorisation du moment d'ordre 2 ainsi que la normalisation RMS.
- L'article montre des courbes d'expériences montrant que AdaLomo obtient de meilleures performances que AdamW, Adafactor, LoRA et LOMO. Mais les auteurs ne fournissent pas de preuves ou d'explications théoriques qui démontrent la supériorité de AdaLomo. Ils ne fournissent pas non plus de garantie théoriqe que AdaLomo converge vers un minimiseur local ou global.

References

[1] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources, 2023.

[2] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost, 2018.

SORBONNE UNIVERSITÉ