SmolGrad-rs

Synopsis du projet S4 2026

Alexandre Joaquim Lima Salgueiro `alexandre-joaquim.lima-salgueiro`
Willian Huang Hong `william.huang-hong`
Romain Bailly `romain.bailly`
Tom Huynh `tom.huynh`

January 16, 2026

1 Présentation du projet

L'objectif de **SmolGrad-rs** est double : développer un moteur bas niveau de différenciation automatique (*Autograd Engine*) et construire par-dessus une bibliothèque de haut niveau permettant l'assemblage et l'entraînement de modèles plus ou moins complexes.
Ce projet vise à produire un framework modulaire capable d'entraîner des architectures standards (MLP, CNN, GPT) sur des jeux de données réels (ex: MNIST, TinyStories).

Le projet s'articule autour de deux piliers :

1. **Le moteur (Backend) :** Gestion des tenseurs, construction du graphe dynamique et exécution des passes avant/arrière.
2. **L'interface de haut niveau (Frontend) :** Abstractions de haut niveau (Couches, Optimiseurs, Fonctions de perte) offrant une interface utilisateur intuitive.

2 Intérêt Algorithmique

2.1 Graphes de Calcul : Tri Topologique et Élagage

La représentation interne des calculs est un Graphe Orienté Acyclique (DAG).

- **Tri Topologique :** Pour que la rétropropagation soit mathématiquement valide, le graphe doit être linéarisé. Un algorithme de tri (DFS en ordre suffixe) garantit que les gradients d'un nœud ne sont calculés qu'une fois ses dépendances résolues.
- **Élagage :** Avant l'exécution, une passe algorithmique détecte et supprime les branches du graphe qui ne contribuent pas au calcul de la fonction de coût, optimisant ainsi le temps de calcul.

2.2 Opérations Tensorielles et Fusion

L'implémentation des couches nécessite des algorithmes efficaces de manipulation matricielle.

- **Broadcasting & Striding (Zero-Copy) :** Gestion des tenseurs via des vues. L'utilisation de strides permet de manipuler, transposer ou redimensionner des tenseurs sans déplacer de données en mémoire. Il est possible de privilégier une approche "Zero-Copy", où plusieurs tenseurs peuvent pointer vers le même buffer physique, optimisant drastiquement l'usage de la bande passante mémoire.
- **Fusion des noyaux :** Pour limiter les goulots d'étranglement mémoire, la détéction des séquences d'opérations élémentaires (ex: $Mul \rightarrow Add \rightarrow ReLU$) pour les compiler en une seule super-opération maximise l'usage des registres CPU/GPU.

2.3 Programmation Dynamique

Le mode "Reverse Accumulation" de l'autodiff est une forme de programmation dynamique. L'enjeu est de gérer une "Wengert List" : une structure de données linéaire enregistrant chronologiquement chaque opération effectuée lors de la passe avant. Cette liste stocke les pointeurs vers les opérandes et les résultats intermédiaires, transformant le graphe de calcul en une séquence d'instructions impératives. Lors de la rétropropagation, le moteur parcourt cette liste en sens inverse pour accumuler les gradients, minimisant ainsi l'empreinte mémoire par une gestion contiguë et prédictible des buffers.

3 Calcul Scientifique

3.1 Différenciation Matricielle et VJP

L'autodifférenciation en mode inverse (Backpropagation) repose sur l'application de la règle de la chaîne généralisée aux tenseurs.

- **Produit Vecteur-Jacobienne (VJP) :** Plutôt que de calculer la matrice Jacobienne complète J_f (dont la taille serait trop grande pour des réseaux profonds), il existe le *Vector-Jacobian Product*.

  $$\nabla_x L = (\nabla_y L)^T \cdot J_f$$

  Cela permet de propager le gradient de la perte scalaire L à travers le graphe sans jamais instancier de matrices $n \times m$ intermédiaires, optimisant ainsi la complexité spatiale.

3.2 Stabilité et Analyse Numérique

La manipulation de nombres flottants introduit des erreurs d'arrondi qui peuvent diverger dans des réseaux profonds.

- **Gestion de la Précision :** Le moteur devra garantir la stabilité numérique des opérations sensibles (ex: Softmax, LogSumExp) pour éviter les problèmes d'*underflow* ou d'*overflow*.

3.3 Optimisation Non-Linéaire

L'entraînement des modèles repose sur la résolution de problèmes d'optimisation non-convexes.

- **Algorithmes du Premier Ordre :** Implémentation de solveurs numériques tels que SGD (Stochastic Gradient Descent) avec Momentum et Adam (Adaptive Moment Estimation).
- **Initialisation Statistique :** L'implémentation des méthodes d'initialisation de *Kaiming* (He) et *Xavier* (Glorot) nécessitera une analyse de la variance des activations pour maintenir une distribution stable du signal à travers les couches (préservation de la variance).