Optimisation d'architecture appliquée aux réseaux de neurones convolutifs (CNN)

Guillaume Lacharme[1], Hubert Cardot[1], Christophe Lenté[1], Nicolas Monmarché[1]

Univ. Tours, LIFAT, 37000 Tours, France

guillaume.lacharme@univ-tours.fr

Mots-clés : *Optimisation bi-level, optimisation continue, réseaux de neurones convolutifs*

1 Introduction

Il est possible de synthétiser le problème d'optimisation d'architecture à l'aide de la formulation suivante :

$$\min_{\lambda} \quad \mathcal{L}_{valid}(w^*(\lambda), \lambda) \tag{1}$$

$$s.t. \quad w^*(\lambda) = \text{argmin}_w \mathcal{L}_{train}(w, \lambda), \tag{2}$$

avec $\boldsymbol{\lambda} \in \mathbb{N}^m \rightarrow$ les hyperparamètres architecturaux du réseau de neurones,
$\boldsymbol{w} \in \mathbb{R}^n \rightarrow$ les poids du réseau de neurones,
$\boldsymbol{\mathcal{L}_{valid}|\mathcal{L}_{train}} \rightarrow$ respectivement l'erreur de validation et d'entraînement

On constate qu'il s'agit d'un problème à deux niveaux. (1) correspond au problème d'optimisation "extérieur" et (2) correspond au problème d'optimisation "intérieur". Nous avons choisi de nous focaliser sur les hyperparamètres architecturaux des réseaux de neurones convolutifs (CNN).

2 Recherche d'architecture différentielle (DARTS)

Un réseau de neurones convolutif peut être caractérisé à l'aide d'un graphe acyclique direct (DAG). On définit les noeuds et les arcs du DAG comme suit :

— noeuds $x^{(j)}$: Addition ($\sum$) ou concaténation ($\bigoplus$) des arcs
— arcs $o^{(i,j)}$: Opérations

$x^{(0)}$ correspond au noeud d'entrée du graphe. Dans notre cas, il s'agit d'une image que l'on représente sous la forme d'un tenseur à 3 dimensions. Chaque arc applique un traitement spécifique sur le tenseur qu'il reçoit en entrée.

$$o^{(i,j)} : \mathbb{R}^{h_i \times l_i \times p_i} \rightarrow \mathbb{R}^{h_j \times l_j \times p_j}, \quad x_i \mapsto y^{(i,j)} = o^{(i,j)}(x_i) \qquad x^{(j)} = \sum_i y^{(i,j)} \text{ ou } x^{(j)} = \bigoplus_i y^{(i,j)} \tag{3}$$

On définit 8 choix d'opérations possibles sur les arcs. Pour les 8 choix, il a 4 opérations qui ne possèdent pas de poids à entraîner. L'objectif est de trouver pour chaque arc l'opération qui minimise l'erreur de validation.

Le problème (1) peut être défini comme un problème d'optimisation de boîte noire discrète. Il existe de nombreux algorithmes conçus à cet effet tels que les algorithmes évolutionnaires ou les algorithmes d'apprentissage par renforcement. Ces méthodes permettent d'obtenir de bons résultats cependant elles sont très coûteuses à mettre en place. Il faut 2000 GPU jours pour

l'apprentissage par renforcement (RL)[2] (sur CIFAR10) ou 3150 GPU jours (sur CIFAR10) pour les algorithmes génétiques [3].

Afin d'éviter ce processus coûteux, Liu et al (2019)[1] a proposé de transformer le problème d'optimisation discret en un problème d'optimisation continu. Pour ce faire, on remplace chaque arc par un mélange d'arc. Le mélange d'arc correspond à une somme pondérée des 8 opérations possibles.

$$o^{(i,j)}(x^{(i)}) \quad = \quad \sum_{o \in O} \frac{e^{\lambda_o^{(i,j)}}}{\sum_{o' \in O} e^{\lambda_{o'}^{(i,j)}}} \times o(x^{(i)}) \tag{4}$$

O correspond à l'ensemble des opérations possibles ($Card(O) = 8$) sur l'arc reliant le sommet i au sommet j. Le graphe qui contient des mélanges d'arcs est appelé hypergraphe. Le réseau de neurones associé à cet hypergraphe est appelé hypernetwork.

Afin de ne pas avoir à réentraîner l'hypernetwork pour chaque nouvelle valeur λ, Liu et al (2019)[1] optimise le problème (1) en alternant l'actualisation de w et λ :

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \eta_w . \nabla_w \mathcal{L}_{train}(w_t, \lambda_t) \\ \lambda_{t+1} &\leftarrow \lambda_t - \eta_\lambda . \nabla_\lambda \mathcal{L}_{valid}(w_{t+1}, \lambda_t) \end{aligned} \tag{5}$$

Cette approche permet de trouver une solution en 1 GPU jour cependant elle a tendance à converger vers des topologies de réseau peu profondes. Il existe plusieurs explications possibles à de tels résultats :

1. Les opérations sans poids permettent au réseau de neurones de converger plus rapidement. Par conséquent, les gradients sur ce type d'opérations seront plus forts.
2. L'entraînement de l'hypernetwork permet d'éviter d'avoir à entraîner indépendamment chaque architecture. Cependant, on remarque un phénomène de coadaptation entre les différentes opérations. On se retrouve confronté à un problème de sélections de variables corrélées.
3. La corrélation entre les performances du réseau discrétisé et du réseau continu n'est pas forcément établie.
4. L'approximation du gradient sur les hyperparamètres (hypergradient) entraîne un biais d'horizon court terme.

L'objectif de cette communication porte sur l'étude de ces limites et des améliorations possibles.

Références

[1] Hanxiao Liu, Karen Simonyan, and Yiming Yang DARTS : Differentiable Architecture Search *ICLR* , 2019.

[2] Barret Zoph, Vijay Vasudevan, Jonathon Shlens and Quoc V. Le Learning Transferable Architectures for Scalable Image Recognition *CVPR* , 2018.

[3] Esteban Real, Alok Aggarwal, Yanping Huang and Quoc V. Le Regularized Evolution for Image Classifier Architecture Search *AAAI* , 2019.

[4] Luca Franceschi, Michele Donini, Paolo Frasconi, Massimiliano Pontil Forward and Reverse Gradient-Based Hyperparameter Optimization *ICML*, 2017.

[5] Matthew MacKay, Paul Vicol, Jonathan Lorraine, David Duvenaud and Roger B. Grosse Self-Tuning Networks : Bilevel Optimization of Hyperparameters using Structured Best-Response Functions *ICLR*, 2019.

[6] Jonathan Lorraine, Paul Vicol and David Duvenaud Optimizing Millions of Hyperparameters by Implicit Differentiation *AISTATS*, 2020.