HippoTrainer

Gradient-Based Hyperparameter Optimization for PyTorch

Daniil Dorin, Igor Ignashin, Nikita Kiselev, Andrey Veprikov

Intelligent Systems, MIPT, 2025

Project description

Motivation

Hyperparameter tuning is time-consuming and computationally expensive, often requiring extensive trial and error to find optimal configurations.

Used algorithms

1. T1 – T2
2. Implicit Function Theorem (IFT)
3. Hyperparameter optimization with approximate gradient (HOAG)
4. Distilling Reverse-Mode Automatic Differentiation (DrMAD)

Solution

`HippoTrainer` is a flexible and scalable library for gradient-based hyperparameter optimization built on `PyTorch`.

Brief algorithms description

Hyperparameter Optimization Problem

Given a vector of model parameters $\mathbf{w} \in \mathbb{R}^P$ and a vector of hyperparameters $\boldsymbol{\lambda} \in \mathbb{R}^H$. One aim to find optimal hyperparameters $\boldsymbol{\lambda}^*$:

$$\begin{aligned} \boldsymbol{\lambda}^* &= \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_{\text{val}}(\mathbf{w}^*, \boldsymbol{\lambda}), \\ \text{s.t.} \;\; \mathbf{w}^* &= \arg\min_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}, \boldsymbol{\lambda}) \end{aligned}$$

Often $\mathbf{w}$ are optimized with gradient descent, so unrolled optimization is typically used:

$$\mathbf{w}_{t+1} = \boldsymbol{\Phi}(\mathbf{w}_t, \boldsymbol{\lambda}), \quad t = 0, \ldots, T-1.$$

Hypergradient Calculation

Chain rule gives us a hypergradient $d_{\boldsymbol{\lambda}}\mathcal{L}_{\text{val}}(\mathbf{w}_T, \boldsymbol{\lambda})$, viewing $\mathbf{w}_T$ as a function of $\boldsymbol{\lambda}$:

$$\underbrace{d_{\boldsymbol{\lambda}}\mathcal{L}_{\text{val}}(\mathbf{w}_T, \boldsymbol{\lambda})}_{\text{hypergradient}} = \underbrace{\nabla_{\boldsymbol{\lambda}}\mathcal{L}_{\text{val}}(\mathbf{w}_T, \boldsymbol{\lambda})}_{\text{hyperparam direct grad.}} + \underbrace{\nabla_{\mathbf{w}}\mathcal{L}_{\text{val}}(\mathbf{w}_T, \boldsymbol{\lambda})}_{\text{parameter direct grad.}} \times \underbrace{\frac{d\mathbf{w}_T}{d\boldsymbol{\lambda}}}_{\text{best-response Jacobian}}$$

- Here best-response Jacobian is hard to compute!

Typical Solution — Implicit Function Theorem

$$\frac{d\mathbf{w}_T}{d\boldsymbol{\lambda}} = -\underbrace{\left[\nabla^2_{\mathbf{w}}\mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{-1}}_{\text{inversed training Hessian}} \times \underbrace{\nabla_{\mathbf{w}}\nabla_{\boldsymbol{\lambda}}\mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})}_{\text{training mixed partials}}.$$

- Hessian inversion is a cornerstone of many algorithms.

Leveraging Neumann series

To exactly invert a $P \times P$ Hessian, we require $\mathcal{O}(P^3)$ operations, which is intractable for modern NNs. We can efficiently approximate the inverse with the Neumann series:

$$\left[\nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{-1} = \lim_{i \to \infty} \sum_{j=0}^{i} \left[\mathbf{I} - \nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{j}.$$

T1 – T2 (Igor)

$$\left[\nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{-1} \approx \mathbf{I}, \qquad i = 0, T = 1$$

IFT (Nikita)

$$\left[\nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{-1} \approx \sum_{j=0}^{i} \left[\mathbf{I} - \nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{j}$$

$$+ \text{ Efficiently compute } \nabla_{\boldsymbol{\lambda}} \mathcal{L}_{\text{val}}(\mathbf{w}_T, \boldsymbol{\lambda}) \times \left[\nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda})\right]^{-1}$$

Approximate Gradient

HOAG (Daniil)

Use conjugate gradient (CG) to invert the Hessian approximately: solve system

$$\nabla^2_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}_T, \boldsymbol{\lambda}) \cdot \mathbf{z} = \nabla_{\boldsymbol{\lambda}} \mathcal{L}_{\text{val}}(\mathbf{w}_T, \boldsymbol{\lambda}).$$

Linear Trajectory Approximation

DrMAD (Andrey)

Instead of storing all intermediate weights $\mathbf{w}_0, \ldots, \mathbf{w}_T$, DrMAD approximates the training trajectory as a linear combination of the initial $\mathbf{w}_0$ and final $\mathbf{w}_T$ weights:

$$\mathbf{w}(\beta) = (1 - \beta)\mathbf{w}_0 + \beta \mathbf{w}_T, \quad 0 < \beta < 1.$$

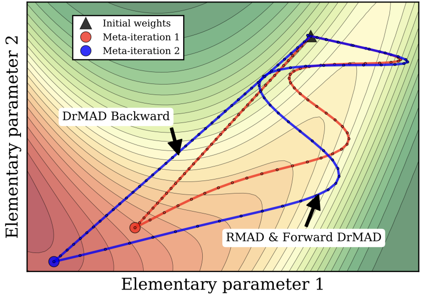
Algorithm 1 DrMAD Algorithm

1: Initialize $\mathbf{w}_0$

2: Train model to obtain $\mathbf{w}_T$

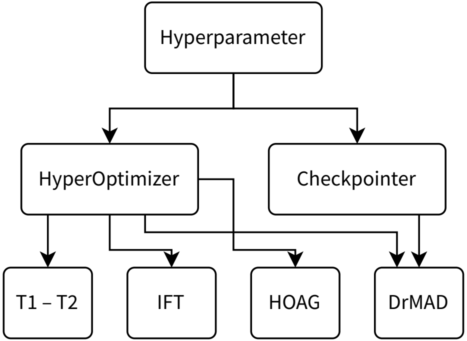
3: Approximate trajectory using $\mathbf{w}(\beta)$

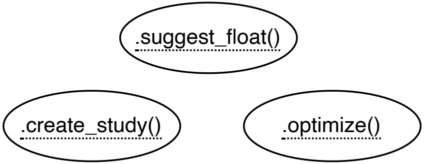
4: Compute hypergradients using the approximated trajectory



Scheme of the project

Project scheme





Proof of concept

Proof of concept idea

- Optimize regularization hyperparameters (L2)
- Implement Random Search as the simplest method