Forward and Reverse Gradient-Based Hyperparameter Optimization

Maksim Tyurikov

MIPT, 2023

December 5, 2023

Motivation

Main idea

There are many approaches to selecting hyperparameters of models. The increasing complexity of machine learning algorithms has driven a large amount of research in the area of hyperparameter optimization (HO). Early approaches based on grid search quickly become impractical as the number of hyperparameters grows and are even outperformed by random search. In this paper, was followed an alternative direction, where gradient-based algorithms are used to optimize the performance on a validation set with respect to the hyperparameters.

Hyperparameter Optimization

We focus on training procedures based on the optimization of an objective function $J(v)$ with respect to v as a dynamical system with a state $s_t \in w$ that collects weights and possibly accessory variables such as velocities and accumulated squared gradients.

$$s_t = \Phi_t(s_{t-1}, \lambda) \text{ for t} = 1, \ldots, \text{T} \tag{1}$$

where T is the number of iterations, s_0 contains initial weights and initial accessory variables, and, for every $t \in \{1, ..., T\}$

$$\Phi_t : (\mathbb{R}^d \times \mathbb{R}^m) \to \mathbb{R}^d \tag{2}$$

Where $\lambda \in \mathbb{R}^m$ is the vector of hyperparameters that we wish to tune.

Example

As simple example of these dynamics occurs when training a neural network by gradient descent with momentum (GDM), in which case $s_t = (v_t, w_t)$

$$v_t = \mu v_{t-1} + \nabla J_t(w_{t-1}) \tag{3}$$

$$w_t = w_{t-1} - \eta v_t \tag{4}$$

where $\mathcal{J}_t$ is the objective associated with the t-th minibatch, μ is the rate and η is the momentum. In this example, $\lambda = (\mu, \eta)$. Our goal is to optimize the hyperparameters according to a certain error function E evaluated at the last iterate s_T

$$f(\lambda) = E(s_T(\lambda)) \tag{5}$$

$$\min_{\lambda \in \Lambda} f(\lambda) \tag{6}$$

Hypergradient Computation

Forward-Mode

$$\nabla f(\lambda) = \nabla E(s_T)\frac{ds_T}{d\lambda} \tag{7}$$

$$s_t = \Phi_t(s_{t-1}, \lambda) \tag{8}$$

$$\frac{ds_t}{d\lambda} = \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial s_{t-1}}\frac{ds_{t-1}}{d\lambda} + \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial \lambda} \tag{9}$$

Defining $Z_t = \frac{ds_t}{d\lambda}$, $A_t = \frac{\partial \Phi_t(s_{t-1},\lambda)}{\partial s_{t-1}}$, $B_t = \frac{\partial \Phi_t(s_{t-1},\lambda)}{\partial \lambda}$

$$Z_t = Z_{t-1}A_t + B_t \tag{10}$$

Hypergradient Computation

Reverse-Mode

Defining $A_t = \frac{\partial \Phi_t(s_{t-1},\lambda)}{\partial s_{t-1}}$, $B_t = \frac{\partial \Phi_t(s_{t-1},\lambda)}{\partial \lambda}$

$$\alpha_t = \begin{cases} \nabla E(s_T), \text{ if t} = \text{T} \\ \nabla E(s_T) A_T ... A_{t+1}, \text{ if t} \in 1, \ . \ . \ . \ , \text{T-1} \end{cases} \tag{11}$$

$$\nabla f(\lambda) = \sum_{t=1}^{T} \alpha_t B_t \tag{12}$$

Algorithm

Forward and Reverse Gradient-Based Hyperparameter Optimization

Algorithm 1 REVERSE-HG

Input: λ current values of the hyperparameters, s_0 initial optimization state
Output: Gradient of validation error w.r.t. λ
for $t = 1$ **to** T **do**
  $s_t = \Phi_t(s_{t-1}, \lambda)$
end for
$\alpha_T = \nabla E(s_T)$
$g = 0$
for $t = T - 1$ **downto** 1 **do**
  $g = g + \alpha_{t+1} B_{t+1}$
  $\alpha_t = \alpha_{t+1} A_{t+1}$
end for
return g

Algorithm 2 FORWARD-HG

Input: λ current values of the hyperparameters, s_0 initial optimization state
Output: Gradient of validation error w.r.t. λ
$Z_0 = 0$
for $t = 1$ **to** T **do**
  $s_t = \Phi_t(s_{t-1}, \lambda)$
  $Z_t = A_t Z_{t-1} + B_t$
end for
return $\nabla E(s) Z_T$

Data Hyper-cleaning (MNIST)

- Oracle: the accuracy of the minimizer of E_{tr} trained on clean examples only, i.e. $(\mathcal{D}_{\text{tr}} \setminus \mathcal{D}_f) \cup \mathcal{V}$; this setting is effectively taking advantage of an oracle that tells which examples have a wrong label;
- Baseline: the accuracy of the minimizer of E_{tr} trained on all available data $\mathcal{D} \cup \mathcal{V}$;
- DH-R: the accuracy of the data hyper-cleaner with a given value of the $L1$ radius, R. In this case, we first optimized hyperparameters and then constructed a cleaned training set $\mathcal{D}_c \subset \mathcal{D}_{\text{tr}}$ (keeping examples with $\lambda_i > 0$); we finally trained on $\mathcal{D}_c \cup \mathcal{V}$.

	Accuracy %	F_1
Oracle	90.46	1.0000
Baseline	87.74	-
DH-1000	90.07	0.9137
DH-1500	90.06	0.9244
DH-2000	90.00	0.9211
DH-2500	90.09	0.9217

Phone Classification (TIMIT)

1. Vanilla: the secondary target is ignored ($\rho = 0$); η and μ are set to 0.075 and 0.5 respectively as in (Badino, 2016).

2. RS: random search with $\rho \sim \mathcal{U}(0, 4)$, $\eta \sim \mathcal{E}(0.1)$ (exponential distribution with scale parameter 0.1) and $\mu \sim \mathcal{U}(0, 1)$ (Bergstra & Bengio, 2012).

3. RTHO: real-time hyperparameter optimization with initial learning rate and momentum factor as in Vanilla and initial ρ set to 1.6 (best value obtained by grid-search in Badino (2016)).

4. RTHO-NT: RTHO with "null teacher," i.e. when the initial values of ρ, η and μ are set to 0. We regard this experiment as particularly interesting: this initial setting, while clearly not optimal, does not require any background knowledge on the task at hand.

	Accuracy %	Time (min)
Vanilla	59.81	12
RS	60.36	300
RTHO	61.97	164
RTHO-NT	61.38	289

Literature

1. **Main article** Forward and Reverse Gradient-Based Hyperparameter Optimization.