Improved Penalty Method via Doubly Stochastic Gradients for Bilevel Hyperparameter Optimization

Wanli Shi[1], **Bin Gu**[1,2,3*]

[1] Nanjing University of Information Science & Technology, P.R.China
[2]MBZUAI, United Arab Emirates
[3]JD Finance America Corporation, Mountain View, CA, USA
wanlishi@nuist.edu.cn, jsgubin@gmail.com

Abstract

Hyperparameter optimization (HO) is an important problem in machine learning which is normally formulated as a bilevel optimization problem. Gradient-based methods are dominant in bilevel optimization due to their high scalability to the number of hyperparameters, especially in a deep learning problem. However, traditional gradient-based bilevel optimization methods need intermediate steps to obtain the exact or approximate gradient of hyperparameters, namely hypergradient, for the upper-level objective, whose complexity is high especially for high dimensional datasets. Recently, a penalty method has been proposed to avoid the computation of the hypergradient, which speeds up the gradient-based BHO methods. However, the penalty method may result in a very large number of constraints, which greatly limits the efficiency of this method, especially for high dimensional data problems. To address this limitation, in this paper, we propose a doubly stochastic gradient descent algorithm (DSGPHO) to improve the efficiency of the penalty method. Importantly, we not only prove the proposed method can converge to the KKT condition of the original problem in a convex setting, but also provide the convergence rate of DSGPHO which is the first result in the references of gradient-based bilevel optimization as far as we know. We compare our method with three state-of-the-art gradient-based methods in three tasks, i.e., data denoising, few-shot learning, and training data poisoning, using several large-scale benchmark datasets. All the results demonstrate that our method outperforms or is comparable to the existing methods in terms of accuracy and efficiency.

Introduction

Hyperparameter optimization problems (HO) are at the center of several important machine learning problems, which are usually formulated as bilevel optimization problems (Feurer and Hutter 2019). Any machine learning algorithms crucially depend on the choice of their hyperparameters. However, manually tunning hyperparameters is often time-consuming, and depends heavily on human's prior knowledge, which makes it difficult to find the optimal hyperparameters. Therefore, to choose the set hyperparameters automatically has attracted great attention, and large amounts of HO methods have emerged, such as grid search, random search (Bergstra et al. 2011; Bergstra and Bengio 2012), solution path (Gu and Sheng 2017; Gu, Liu, and Huang 2017; Bao, Gu, and Huang 2019; Gu and Ling 2015), and several Bayesian methods (Thornton et al. 2013; Brochu, Cora, and De Freitas 2010; Swersky, Snoek, and Adams 2014; Wu et al. 2019).

In real-world applications, the number of hyperparameters increases shapely, especially in a deep learning problem. In these problems, gradient-based methods are dominant in bilevel problems with continuous hyperparameter set due to their high scalability to the amount of hyperparameters. The main idea of the gradient-based methods is to first approximate the solution on the training set and then compute the gradient descent direction for hyperparameters, namely hypergradient. However, calculating the exact hypergradient needs the computation of the inverse Hessian of the lower-level problem, which makes it impractical for high dimensional data problem. To solve this problem, many methods have been proposed to approximate the hypergradient by using some intermediate steps, such as solving a linear system (the approximate methods (Domke 2012; Pedregosa 2016; Rajeswaran et al. 2019)) or using reverse/forward mode differentiation (e.g. (Maclaurin, Duvenaud, and Adams 2015; Domke 2012; Franceschi et al. 2017; Swersky, Snoek, and Adams 2014)). Although these methods can avoid the computation of inverse Hessian, the intermediate steps usually need extra loops which will affect the efficiency of these gradient-based methods.

Recently, (Mehra and Hamm 2019) pointed out that the intermediate steps are not necessary. They formulate the bilevel problem as a single level constrained problem, by replacing the lower-level objective with its first-order optimal necessary condition. Then the penalty framework can be used to solve this constrained problem. With updating the model parameters and hyperparameters alternately, the penalty method can finally obtain the exact hypergradient. It speeds up the gradient-based methods but brings a new problem in the meanwhile. Since the constraints are calculated from the first order necessary condition of the lower-level objective, it may lead to the optimization problem with a large number of constraints, especially for high dimensional datasets or deep learning methods. Calculating the gradient of the penalty term in such a heavily constrained problem

*Corresponding Authors

Method	Problem	Update v	Intermediate steps	Time	Space	Convergence Rate
FMD	Bilevel	$v \leftarrow v - \eta \nabla_v g$	$P = P(\mathbf{I} - \eta \nabla^2_{vv} g) - \eta \nabla^2_{uv} g$	$O(dmT)$	$O(dm)$	–
RMD	Bilevel	$v \leftarrow v - \eta \nabla_v g$	$p = p - \eta \nabla^2_{uv} g \cdot q$ $q = (\mathbf{I} - \eta \nabla^2_{vv} g) q$	$O(dT)$	$O(dT + m)$	–
Approx	Bilevel	$v \leftarrow v - \eta \nabla_v g$	Solve $\min_q \|\nabla^2_{vv} g \cdot q - \nabla_v f\|_2^2$	$O(dT)$	$O(d + m)$	–
Penalty	Single level	$v \leftarrow v - \eta(\nabla_v f + \mu \nabla^2_{vv} g \cdot \nabla_v g)$	Not reqired	$O(dT)$	$O(d + m)$	–
DSGPHO	Single level	$v \leftarrow v - \eta \tilde{\nabla}_v \mathcal{L}$	Not reqired	$O(T)$	$O(d + m)$	$O(1/\sqrt{K})$

Table 1: The complexity of various gradient methods for update the hyperparameters one time. Here d is the size of model parameters, m denotes the size of hyperparameters and T is the number of model parameters update per hypergradient computation. $\tilde{\nabla}_v \mathcal{L} = \tilde{\nabla}_v f + [\mu c_j + z_j] \nabla_v c_j$ where z_j is the jth element of dual parameter, $\mu > 0$ is the penalty parameter and c_j is jth element of $\nabla_v g$. K denotes the total number of hyperparameter updates.

will greatly limits its efficiency.

To address this limitation, in this paper, we propose a doubly stochastic gradient descent method to improve the efficiency of the penalty method (Mehra and Hamm 2019). Similar to (Mehra and Hamm 2019), we first transform the bilevel problem to a single level constrained problem and give its corresponding augmented Lagrangian function. We first randomly sample a validation data instance to calculate the stochastic gradient of the upper-level objective. Then, we randomly sample a constraint and calculate its corresponding gradient. By combining these terms together, we obtain the stochastic gradient of the augmented Lagrangian function, and update the model parameters and hyperparameters, alternately, by using this stochastic gradient. Since in our method, we have two sources of randomness, i.e., the validation set and the constraints set, we can denote our method as a doubly stochastic gradient descent method for the penalty hyperparameter optimization (DSGPHO). We prove that our method can achieve the unbiased estimation of the exact hypergradient asymptotically and it finally converges to the KKT point of the original problem. Importantly, we also provide the convergence rate of DSGPHO which is the first result in the references of gradient-based bilevel optimization as far as we know. We compare our method with three state-of-the-art gradient-based methods in three tasks, i.e., data denoising, few-shot learning and training data poisoning, using several large scale benchmark datasets. All the results demonstrate that our method outperforms or is comparable to the existing methods in terms of accuracy and efficiency.

Contributions We summarized the contributions as follows,

1. In this paper, we propose a doubly stochastic gradient descent method to improve the efficiency of the penalty method. Within each iteration, we randomly sample a validation instance and a constraint to calculate the stochastic gradient to update model parameters and hyperparameters alternately.
2. We prove that DSGPHO has a rate of $O(1/\sqrt{K})$ to converge to the KKT point of the original problem. In addition, we also prove that our method can obtain an unbiased estimation of the hypergradient asymptotically.
3. We apply our method to solve various problems. The experimental results outperform other methods in terms of accuracy and training-time which demonstrate the superiority of our method.

Related Works

In this section, we give a brief review of the gradient-based hyperparameter optimization methods.

As we mentioned in the previous section, many gradient method use some intermediate steps to approximate the hypergradient. Specifically, (Franceschi et al. 2017; Maclaurin, Duvenaud, and Adams 2015; Shaban et al. 2019) use the forward/reverse mode differentiation (RMD/FMD) to approximate the hypergradient. Both of them view the training procedure as dynamical system with a state $v_{t+1} = \Phi(v_t, u) := v_t - \eta \nabla_v g(u, v_t)$, for $t = 1, \cdots, T$, where v denote the model parameters, u denotes the hyperparameters, η is the learning rate and g is the training objective on training set. According to (Feurer and Hutter 2019), RMD needs another T iterations to calculate $p = p - \eta \nabla^2_{uv} g \cdot q$ and $q = (\mathbf{I} - \eta \nabla^2_{vv} g) q$ and then outputs the hypergradient p, where I is the identity matrix. In each p update step we need $O(d)$ time and space to calculate the Jacobian vector product, where d is the size of model parameters. Totally, it needs $O(dT)$ time and $O(m+Td)$ space to get the hypergradient, where m is the size of hyperparameters. FMD needs to calculate $P = P(\mathbf{I} - \eta \nabla^2_{vv} g) - \eta \nabla^2_{uv} g$ and update v at the same time. Then the hypergradient can be computed by using P. (Feurer and Hutter 2019) pointed out that it needs $O(dm)$ time to get each P. Hence the time complexity for calculate one hypergradient is $O(dmT)$, and the space complexity is $O(dm)$ for P is a $d \times m$ matrix. (Pedregosa 2016) solve a linear problem $\min_q \|\nabla^2_{vv} g \cdot q - \nabla_v f\|_2^2$ and use the solution to approximate the hypergradient. When using gradient method to solve this problem, we require $O(d)$ time to obtain the Hessian-vector product in each iteration according to (Pearlmutter 1994). Assume the total gradient iteration number is T, then we need $O(dT)$ time to calculate the hypergradient one time. Obviously, the space complexity is $O(d+m)$. (Mehra and Hamm 2019) propose a penalty method which do not need the intermediate steps to compute the hypergradient. Thus the computational complexity for per hyperparameter update is mainly from the steps of updating v by using $v \leftarrow v - \eta(\nabla_v f + \mu \nabla^2_{vv} g \cdot \nabla_v g)$. Since the Hessian-vector product complexity is $O(d)$, we need $O(dT)$ time and $O(d + m)$ space to update v for T iterations. We summarized all these complexity in Table 1.

According to this table, we have that if d is very large, these methods become impractical.

Preliminaries

In this section, we give a brief introduction to bilevel hyperparameter optimization.

Bilevel Hyperparamter Optimization

The hyperparameter optimization problems are at the center of several important machine learning problems, where once the first party (hyperparameters) makes its choice affecting the optimal choice for the second party (model parameters). The hyperparameter optimization problems are commonly formulated as the following bilevel optimization problems,

$$\min_{u\in U} f(u,v),\ s.t.\ v=\arg\min_{v\in V} g(u,v), \tag{1}$$

where u denotes the hyperparameters, v denotes the model parameters, and U and V are the convex set in $\mathbb{R}^m$ and $\mathbb{R}^d$. In this paper, we assume that the upper- and lower-level objectives f and g are twice continuously differentiable in both u and v. In addition, we assume that the lower-level cost g is convex in v for each given u.

Obviously, in the bilevel optimization problem, the upper-level problem $\min_u f(u,v)$ is a usual minimization problem (minimizing the loss on the validation set) where v is constrained to be the solution to the lower-level problem $\min_v g(u,v)$ depended on u (usually minimizing the loss on training set). Thus, we can replace the lower-level problem by its first-order necessary condition for optimality, resulting in the following constrained optimization problem.

$$\min_{u,v} f(u,v),\ s.t.\ c(u,v)=\nabla_v g(u,v)=0 \tag{2}$$

where each element of the equality constraint $c(u,v)$ is $c_j(u,v),\ j=1,\cdots,d$. According to (Mehra and Hamm 2019), we known that solving the single level problem (2) is equivalent to the bilevel optimization problem (1).

Penalty Method

In this subsection, we give an introduction to the penalty method. The penalty method is one of the most common methods to solve the constrained problems. The main idea of the penalty methods is to optimize the original cost function pulsing a penalty term of the constraints multiplied a positive parameter. By making the penalty parameter larger, we can penalize constraint violation more severely, thereby forcing the minimizer of the penalty function closer to the feasible region of the constrained problem.

Thus, based on the penalty method, the constrained problem (2) can be reformulated as $L(u,v;\mu)=f(u,v)+\frac{\mu}{2d}\sum_{j=1}^{d}c_j^2(u,v)$ where $\mu>0$ is the penalty parameter. It makes good intuitive sense to consider a sequence of values $\{\mu^k\}$ with $\mu^k\uparrow\infty$ as $k\to\infty$, and to seek the approximate minimizer u^k and v^k of $(u^k,v^k)=\arg\min_{u,v} f(u,v)+\frac{\mu^k}{2d}\sum_{j=1}^{d}c_j^2(u,v)$ for each given μ_k. According to (Bard 2013), we have that for each given μ_k, the sequence $\{u^k,v^k\}$ has limit points any one of which is a solution of the original problem (1).

Recently, (Mehra and Hamm 2019) proposed a gradient method based on the penalty framework. For each given μ_k, they update the model parameters and hyperparameters alternately by using the full gradient of the penalty function until the norm of the gradient converges to a small tolerance. However, in real-world applications, the number of constraints d is usually very large. Specifically, when using the linear model for high dimension data (e.g. more than $10,000$), the size of constraints is the same as the data dimension. What's worse, when using deep model, the size of constraints becomes further larger, usually more than $100,000$. It would be computationally expensive, if at every update, we inquire about the value and gradient of all constraints. In addition, if the size of validation set is big, it also takes a lot of time to calculate the gradient of the upper-level objective. Meanwhile, to solve the quadratic penalty function, the penalty parameter μ^k needs to be large enough, which makes it impossible to get the solution in a limited time.

Proposed Method

In this section, to address the above limitations, we propose our stochastic method based on the penalty framework.

To get the solution in a limited time, instead of using the quadratic penalty function, we add Lagrangian multipliers and obtain the following augment Lagrangian function (Bertsekas 1976),

$$\mathcal{L}(u,v,z;\mu)=f(u,v)+\Psi_\mu(u,v,z) \tag{3}$$

where z denotes the Lagrangian multiplier and $\mu>0$ is the penalty parameter, $\Psi_\mu(u,v,z)=\frac{1}{d}\sum_{j=1}^{d}\psi_\mu(c_j(u,v),z_j)$ and $\psi_\mu(c_j(u,v),z_j)=z_jc_j(u,v)+\frac{\mu}{2}c_j^2(u,v)$.

Then, we use the stochastic manner to solve the augment Lagrangian function. First, we give the update rule of model parameters. In t-th v update iteration, we randomly sample a constraint $c_j(u,v)$ and calculate its gradient $\nabla_v c_j(u,v)$ w.r.t. v. Let $h_v^t=\left[\mu_k c_j(u,v^t)+z_j^t\right]\nabla_v c_j(u,v^t)$. If d is large enough, h^k is an unbiased estimation of the gradient of Ψ_μ and $\nabla_v\Psi_\mu=\mathbb{E}[h_v^t]$. In addition, since the upper-level cost f is usually formulated as the expectation on the validation set, we can also obtain the stochastic gradient $\tilde{\nabla}_v f(u,v^t)$ of $f(u,v^t)$ at v^t by randomly sample a validation instance and we have $\nabla_v f(u,v^t)=\mathbb{E}[\tilde{\nabla}_v f(u,v^t)]$. Combining the above two terms, we can obtain the stochastic gradient of augmented Lagrangian function (3) $\tilde{\nabla}_v\mathcal{L}=h_v^t+\tilde{\nabla}_v f(u,v^t)$ and $\mathbb{E}\left[\mathbb{E}[\tilde{\nabla}_v\mathcal{L}]\right]=\nabla_v\mathcal{L}$, where $\nabla_v\mathcal{L}$ denotes the full gradient of $\mathcal{L}$ w.r.t v. Then we can apply the projected stochastic gradient to update the model parameters $v^{t+1}=\mathrm{Proj}_V(v^t-\eta_v^t\tilde{\nabla}_v\mathcal{L})$, where $\mathrm{Proj}_V(\cdot)$ denotes the projection on to the set V. Since V is the original set of v, the projection operation can be omitted.

By using the same approach, we can obtain the stochastic gradient of function (3) with respect to u as $\tilde{\nabla}_u\mathcal{L}=h_u^k+$

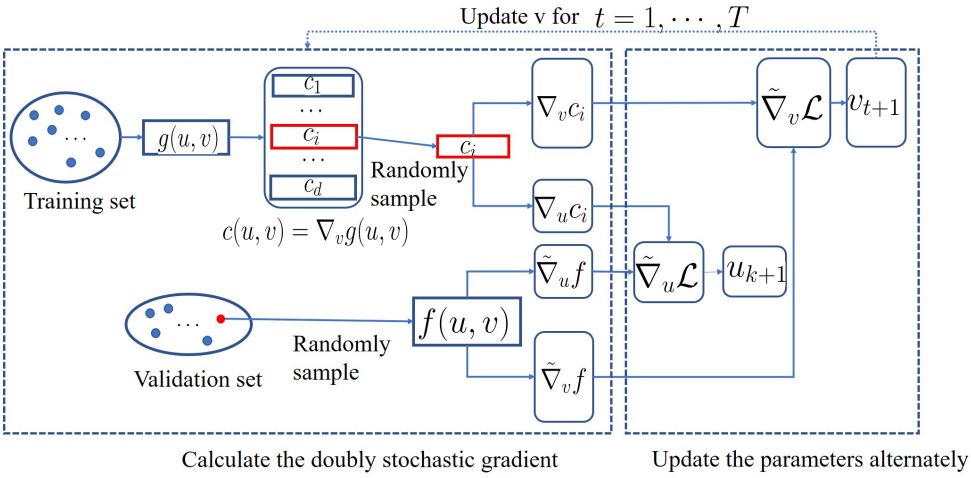


Figure 1: Illustration of a single step to update hyperparameters in Algorithm 1

$\tilde{\nabla}_u f(u^k, v)$ where $h_u^k = \left[\mu_k c_j(u^k, v) + z_j^k\right] \nabla_u c_j(u^k, v)$ and we have $\mathbb{E}\left[\mathbb{E}[\tilde{\nabla}_u \mathcal{L}]\right] = \nabla_u \mathcal{L}$. Similarly, we can perform the projected stochastic gradient to the hyperparameters $u^{k+1} = \text{Proj}_U(u^k - \eta_u^k \tilde{\nabla}_u \mathcal{L})$ and the projection operation can also be omitted.

The overall algorithm is shown in Algorithm 1. We also illustrate a single step of updating the hyperparameter of our method in Figure 1. Note instead of updating u and v simultaneously, we first use the stochastic gradient $\tilde{\nabla}_v \mathcal{L}$ to update v for T iterations and then use the stochastic gradient $\tilde{\nabla}_u \mathcal{L}$ to update u for a single time. Since it contains two sources of randomness, i.e., the random validation instance and random constraint, we denote our method as doubly stochastic gradient method for penalty hyperparameter optimization, namely DSGPHO. For each given μ_k, we find the ϵ_k-optimal solution (u^k, v^k) of problem (3). Once the tolerance ϵ_k is satisfied, we update the Lagrangian multipliers by using $z^{k+1} = z^k + \mu^k c(u^k, v^{tk})$ and then enlarge the penalty parameter μ^k and reduce the tolerance ϵ_k. We show that it can finally converge to a KKT point in Theorem 1.

Theorem 1. *Suppose the tolerance ϵ_k is a positive and convergent ($\epsilon_k \to 0$) sequence and μ^k is a positive non-decreasing sequence. Let $\{u^k, v^k, z^k\}$ be the sequence of approximate solutions to the augmented Lagrangian function with penalty parameter μ^k and tolerance $\|\nabla_v \mathcal{L}\|_2^2 + \|\nabla_u \mathcal{L}\|_2^2 < \epsilon_k^2$ for all $k = 1, 2, \cdots$. Then any limit point of $\{u^k, v^k, z^k\}$ satisfies the KKT conditions of the constraint problem (2).*

In our method, we do not need any intermediate steps to approximate the hypergradient. This is because that if we find the minimum v^* of the augmented Lagrangian function, our method can get the unbiased estimation of the hypergradient asymptotically when Algorithm 1converges. We give this result in Lemma 1.

Lemma 1. *Give u, let v^* be the optimal solution $v^* := \arg\min_v \mathcal{L}(u, v, z; \mu)$. Then we have $\mathbb{E}\left[\mathbb{E}[\tilde{\nabla}_u \mathcal{L}(u, v, z; \mu)]\right] = \frac{\partial f}{\partial u}(u, v^*)$, where $\frac{\partial f}{\partial u}(u, v^*)$ denotes the hypergradient.*

Furthermore, instead of sampling one constraint function every time, we can sample a small set of b of constraint function. Then, the stochastic gradient of Ψ_μ can be calculated by $h = \frac{1}{b}\sum_{j=1}^b \left[\mu_k c_j(u, v) + z_j\right] \nabla_v c_j(u, v)$. In addition, we can also sample a small batch of validation instances to calculate the stochastic gradient of the upper-level objective.

Obviously, in our method, the main computational complexity is from updating the model parameters. Assume that we only sample one data instance and one constraint. Then in each inner iteration, we only need $O(1)$ computational complexity to calculate stochastic gradient $\tilde{\nabla}_v \mathcal{L}$. Thus the total complexity of a single step to update the hyperparameters is $O(T)$. In addition, we only need to save the model parameters and hyperparameters with the space complexity $O(d + m)$. Compare with the complexity summarized in Table 1, our method obviously has lower computational complexity and do not need intermediate steps to obtain the hypergradient.

Convergence Analysis

In this section, we give the convergence analysis of our proposed method in convex case. Our analysis follows the analysis in (Xu 2020). Let $w := (u, v)$ refer to the pair. Then, in our analysis, we consider the following update rules to update u and v, simultaneously, instead of the exact rules in Algorithm 1 as $w^{k+1} = \text{Proj}_W(w^k - \mathbf{D}_k^{-1} \nabla_w \mathcal{L}(w^k, z^k; \mu))$ where W denotes the feasible region of w, $\mathbf{D}_k^{-1} \succ 0$ is the diagonal mtrix for each iteration and we have $\nabla_w \mathcal{L}(w^k, z^k; \mu) = \frac{1}{b}\sum_{j=1}^b \left[\mu c_j(w^k) + z_j^k\right] \nabla_w c_j(w^k) + \tilde{\nabla}_w f(w^k)$ where $\tilde{\nabla}_w f(w^k)$ denotes the stochastic gradient of $f(w)$ w.r.t. w. Note the update rule can be viewed as an approximation of the rules in Algorithm 1 if $T = 1$. In addition, we assume that μ is large enough such that we can

Algorithm 1 DSGPHO

Input: $K, T, \eta_v, \eta_u, \mu_0, \lambda_0, \epsilon_0, c_\mu > 1, 0 < c_\epsilon < 1$.
Output: u^k, v^k.

for $k = 1, ..., K$ **do**
  for $t = 0, \cdots, T$ **do**
    Randomly sample a validation training data instance .
    Randomly sample a constraint.
    Calculate the doubly stochastic gradient $\tilde{\nabla}_v \mathcal{L}$.
    Update model parameters $v^{t+1} = \text{Proj}_V(v^t - \eta_v^t \tilde{\nabla}_v \mathcal{L})$.
  end for
  Randomly sample a validation training data instance .
  Randomly sample a constraint.
  Calculate the doubly stochastic gradient $\tilde{\nabla}_u \mathcal{L}$.
  Update hyperparameters $u^{k+1} = \text{Proj}_U(u^k - \eta_u^k \tilde{\nabla}_u \mathcal{L})$.
  if $\|\nabla_u \mathcal{L}\|_2^2 + \|\nabla_v \mathcal{L}\|_2^2 \leq \epsilon_k^2$ **then**
    $\mu^{k+1} = c_\mu \mu^k$.
    $\epsilon_{k+1} = c_\epsilon \epsilon_k$.
    $z^{k+1} = z^k + \mu^k c(u^k, v^{tk})$
  end if
end for

reduce the complexity without considering the increasing of μ.

Then, we give the following assumptions which are widely used in theoretical analysis.

Assumption 1. *The stochastic gradient $\tilde{\nabla}_w f(w)$ is unbiased and bounded, i.e., there is a constant σ such that we have $\mathbb{E}[\tilde{\nabla}_w f(w)] = \nabla_w f(w)$ and $\mathbb{E}\left[\|\tilde{\nabla}_w f\|\right] \leq \sigma^2$ In addtion, there exist constants F and G such that we have $|c(w)| \leq F, \|\nabla_w c_j(w)\|_2 \leq G$.*

Assumption 2. *All the constraints $c_j(w)$ and the upper-level objective $f(w)$ are convex functions.*

The following lemma is used to build the inequality in our analysis for running one iteration of hyperparameter update.

Lemma 2. *For any deterministic or stochastic z, it holds that*

$$
\begin{aligned}
&-\Psi(w^k, z^k) + \frac{1}{d}\sum_{j=1}^{d} z_j f_j(w^k) + \frac{1}{2}\mathbb{E}[\|z^{k+1} - z\|_2^2] \\
=&\mathbb{E}[\langle z^k - z, de_{jk} \odot \nabla_z \Psi(w^k, z^k) - \nabla_z \Psi(w^k, z^k)\rangle] \\
&+ 1/2\|z^k - z\|_2^2 - 1/2(\mu - 1)\mathbb{E}[\|z^{k+1} - z\|_2^2]
\end{aligned}
$$

By the previous lemma, we establish the important result for running one iteration update w and then use it to show the convergence rate.

Theorem 2. *Under the Assumptions 1 and 2, and assume that $\boldsymbol{D}_k \succeq \frac{\boldsymbol{I}}{\alpha^k}$, $\forall k$, for a positive number sequence $\{\alpha^k\}_{k\geq 1}$, where $\boldsymbol{I}$ is the identity matrix. Let (w, z) be any deterministic or stochastic vector. Then*

$$
\begin{aligned}
&\mathbb{E}[f(w) + \frac{1}{d}\sum_{j=1}^{d} z_j \nabla_v g_j(w)] + \frac{1}{2}\mathbb{E}\|w^{k+1} - w\|_{\boldsymbol{D}_k}^2 \\
&+ \frac{1}{2}\mathbb{E}\|z^{k+1} - z\|_2^2 \\
\leq&\mathbb{E}[f(w) + \Psi(w, z^k)] + \frac{1}{2}\mathbb{E}[\|w^k - w\|_{\boldsymbol{D}_k}^2] \\
&+ \frac{1}{2}\mathbb{E}[\|z^k - z\|_2^2] \\
&+ \alpha^k(\sigma^2 + 2\mu^2 F^2 G^2 + \frac{2G^2}{d}\mathbb{E}[\|z^k\|_2^2]) \\
&- \frac{1}{2}(\mu - 1)\mathbb{E}[\|z^{k+1} - z^k\|_2^2] \\
&- \mathbb{E}[\langle w^k - w, \tilde{\nabla}_w f^k - \nabla_w f^k\rangle] \\
&- \mathbb{E}[\langle w^k - w, h^k - \nabla_w \Psi(w^k, z^k)\rangle] \\
&+ \mathbb{E}[\langle z^k - z, de_{jk} \odot \nabla_z \Psi(w^k, z^k) - \nabla_z (w^k, z^k)\rangle]
\end{aligned}
$$

Then, based on the above results, here we directly give the convergence result with constant step size $\mathbf{D}_k$. We give the detailed proof in our appendix.

Theorem 3. *Under the Assumption 1 and 2, let $\{(w^k, z^k)\}$ be the sequence generated from our Algorithm. For total iteration number K, let $\bar{w} = \frac{1}{K}\sum_{j=1}^{K} w^k$ and $\bar{w} = \frac{1}{K}\sum_{j=1}^{K} z^k$, and define $\phi_1(w) = \frac{3}{2\alpha}\|x^1 - x\|_2^2 + \alpha\left(\frac{3}{2}\sigma^2 + 3\mu^2 F^2 G^2 + \frac{3G^2}{d}\frac{C_1}{1 - \frac{8\alpha G^2}{d}} + \frac{F^2}{2}\right)$. Then we have*

$$
\begin{aligned}
&\mathbb{E}[f(\bar{w}) - f(w^*)] \leq \frac{1}{\sqrt{K}}\left(2\phi_1(w^*) + \frac{9(\alpha+1)}{2\alpha}\|z^*\|_2^2\right) \\
&\mathbb{E}[\|c(w)\|_2^2] \leq \frac{1}{\sqrt{K}}\left(\phi_1(w^*) + \frac{\alpha+1}{2\alpha}\|1 + z^*\|_2^2\right)
\end{aligned}
$$

where $C_1 = \frac{2}{\alpha}\|w^1 - w^*\|_2^2 + 4\|z^*\| + 4\alpha(\sigma^2 + 2\mu^2 F^2 G^2)$

Remark 1. *Theorem 3 shows that our method has an approximate convergence rate of $O(1/\sqrt{K})$, where K is the total iteration number of hyperparameter updating.*

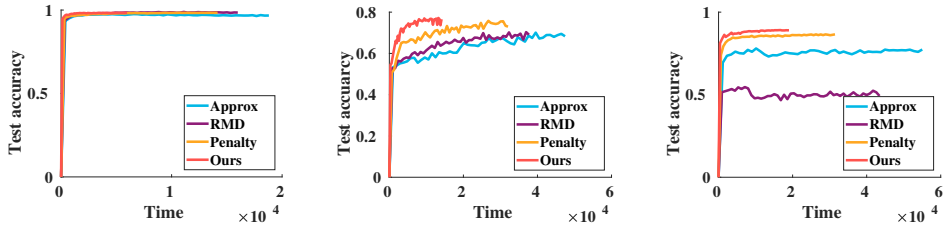
Experiments

In this section, we compare our method with several state-of-the-art gradient-based methods to show the superiority of our method in terms of accuracy and efficiency.

Baselines

We summarized the methods used in our experiments as follows

1. **Penalty**. The method proposed in (Mehra and Hamm 2019). It formulates the bilevel optimization problem as a one-level problem with equality constraints, and then the gradient method can be used to solve the new problem.



(a) MNIST (b) CIFAR10 (c) SVHN

Figure 2: The training time of Penalty, Approx, RMD and our proposed method used in denoisy task

Dataset	Noise	Approx	Penalty	RMD	Ours
CIFAR10	25%	68.7 ± 0.6	74.4 ± 1.2	68.9 ± 1.4	**75.6** ± 1.7
MNIST	25%	96.7 ± 0.2	97.2 ± 1.4	**98.4** ± 0.1	**98.4** ± 0.1
SVHN	25%	77.4 ± 1.5	86.3 ± 0.2	50.9 ± 1.2	**88.9** ± 0.1

Table 2: Test accuracy (%) for data denoising task with 25% noisy.(Mean±std).The best value is shown in bold type.

2. **Approx**. The method proposed in (Pedregosa 2016). It solves an additional linear problem to approximate the hypergradient to update the hyperparameters.
3. **RMD**. The method proposed in (Franceschi et al. 2017). An additional loop is used to calculate the hypergradient.

Applications

We evaluate the performance of the proposed method in the following machine learning tasks.

Data denoising by importance learning In many real-world applications, we need to learn models from the datasets with corrupted labels. In such case, we need to re-weight the importance of each training instance to reduce the impact of noise (Liu and Tao 2015; Yu et al. 2017; Ren et al. 2018). To find the correct weight of training instance can be formulated as the following bilevel hyperparameter optimization problem,

$$\min_u \mathbb{E}_{D_{vl}} l(h(x_i; v^*, u), y_i),$$
$$s.t.\ v^* = \arg\min_v \mathbb{E}_{D_{tr}} u_i l(h(x_i; v, u), y_i)$$

where h denotes the classifier parameterized by v, l denotes the loss, u_i denotes the weight of each data instance, and D_{val} and D_{tr} denote the validation set and training set, respectively.

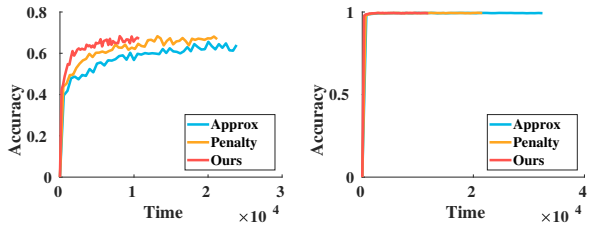
In this task, we conduct the experiments on the dataset **MNIST**, **SVHN**, **CIFAR10**. For each dataset, we split it into three subsets, i.e., training set, test set, and validation set and we introduce 25% noise into the training set. The validation set of each dataset contains 1000, 10000, and 1000 points, respectively. For each hyperparameters update step, we update the model parameters for 50 times. For our method, we use 500 validation instances and 2048 constraints to calculate the doubly stochastic gradient. We fix the step size of hyperparameters at 0.01 and tune the step size of model parameters in $0.001, 0.0001, 0.00001$ for all methods. In addition, for the architectures of convolutional neural networks, we follow the setting in (Mehra and Hamm 2019). We show the test accuracy in Table 2 for 5 runs and the time-accuracy curve in Figure 2. Obviously, as shown in Table 2, our method has the best test accuracy in this task. Figure 2 demonstrates that our method converges faster than other state-of-the-art methods. In addition, our method uses the least time to train the model when we fix the u update times at 5000. This is because that compared with Approx and RMD, our method does not need additional steps to approximate or calculate the hypergradient. With the continuous training, DSGPHO can finally obtain the unbiased estimation of hypergradient. Although Penalty has the same characteristic, it may lead to a large amount of constraints. This means that in each model parameters update step, Penalty needs to evaluate the value and gradient of all constraints which greatly limits the efficiency. However, our method only needs the information of a subset constraints and a subset of validation instances which can greatly reduce the time complexity.

Few-shot learning Then, we compare the performance of these methods in the Few-shot learning task (Snell, Swersky, and Zemel 2017; Sung et al. 2018; Santoro et al. 2016). Few-shot learning trains a model on several related tasks and then generalizes to unseen tasks with just a few examples. We can learn a common representation for various tasks and then train the task specific layers. The few-shot learning can be formulated as following bilevel optimization problem,

$$\min_u \mathbb{E}_{D_{val}} l(h_i(M(x_i, u), v_i^*), y_i),$$
$$s.t.\ v_i^* = \arg\min_{v_i} \mathbb{E}_{D_{tr}} l(h_i(M(x_i, u), v_i), y_i)$$

where $M(\cdot, u)$ are the maps for all tasks parameterized by u, h_i denotes ith task's classifier parameterized by w_i.

In this task, we use the **Mini-ImageNet** (Vinyals et al. 2016) and **Omniglot** (Lake, Salakhutdinov, and Tenenbaum 2015) datasets. We generate meta-training set and meta-testing set using images from disjoint classes. For Omniglot, we use the first 1200 classes to build meta-training set and the rest to build meta-testing set (Santoro et al. 2016). For Mini-Imagenet, we split 64 classes in meta-training and 16 classes in meta-validation and 20 classes in meta-testing. We fix the step size of u at 0.1 and search v step size from $\{0.01, 0.001, 0.0001\}$. We randomly sample 2048 con-



(a) Mini-ImageNet (b) Omniglot

Figure 3: The training time of Penalty, Approx and DSGPHO used in few-shot learning task

Dataset	Problem	Approx	Penalty	Ours
MiniImageNet	5-way 5-shot	**68.3** $\pm$ 1.8	67.2 $\pm$ 0.5	67.3 $\pm$ 0.5
Omiglot	5-way 5-shot	99.3 $\pm$ 0.1	**99.4** $\pm$ 0.1	99.3 $\pm$ 0.1

Table 3: Accuracy (%) for few-shot task.(Mean$\pm$std). The best value is shown in bold type.

straints in each iteration. We show the results in Table 3 and Figure 3. From Table 3, we can find that our method has the comparable result to Penalty and Approx. From Figure 3, we can find that our method can converge to the solution faster than Penalty and Approx. This is because that, in each v update step, our method only samples a batch of constraints and does not need to approximate the hypergradient. This leads to a lower computational complexity in our method.

Training data poisoning Finally, we compare these methods on the training-data poisoning task (Muñoz-González et al. 2017; Shafahi et al. 2018; Mei and Zhu 2015; Koh and Liang 2017). In this task, we modify the training data such that we make the model learned from these data performs poorly than that learned from the original data. The problem of finding the poisoned data, of which the labels are not correct, can be formulated as follows,

$$\min_u -\frac{1}{N_{vl}} \sum_{D_{vl}} l(h(x_i, u, v^*), y_i)$$

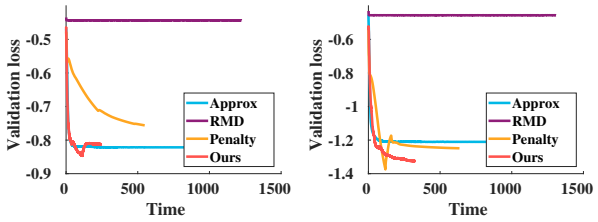
$$s.t.\ v^* = \arg\min_v \frac{1}{N} \sum_{D_{or} \cup P} l(h(x_i, u, w), y_i)$$

where D_{or} denotes the original dataset and P denotes the set of poisoned instances.

In our experiments, we follow the setting in (Muñoz-González et al. 2017) and test the untargeted MNIST using data augmentation technique. We use logistic regression to train the classifiers. Our goal is to make the performance lower on the test set. We split 1000 instances into training set, 1000 into validation set and 8000 into testing set. We randomly sample 10 and 30 instances from the training set and give them random incorrect labels. For our method, we randomly sample 2048 constraints. We fix the learning rate of u at 0.1 and choose learning rate of v form $\{0.01, 0.001, 0.0001\}$. We set the total number of update u at 5000. From Table 4, we can find that results are comparable to other methods when we have 10 poisoned points and

Poisoned	Approx	Penalty	RMD	Ours
10	**81.5** $\pm$ 0.5	82.9 $\pm$ 0.7	86.1 $\pm$ 0.7	82.7 $\pm$ 0.7
30	76.4 $\pm$ 0.3	74.9 $\pm$ 0.1	84.5 $\pm$ 0.1	**74.7** $\pm$ 0.6

Table 4: Test accuracy (%) for untargeted poisoned attack.(The lower accuracy is better) .



(a) 10 poisoned points (b) 30 poisoned points

Figure 4: The validation loss vs. time of Penalty, Approx, RMD and DSGPHO in untargeted poisoned task

we have the best result when we use 30 poisoned points. We show the validation loss and train time in Figure 4. From this figure, we can find that our method can obtain the lower validation loss in a less time. It is reasonable that the performance in terms of training-time are not as good as that in the previous tasks. This is because that in this task, we use a linear model, which makes the constraints are not as much as those in the previous tasks.

Based on all above results, we can conclude that our method is superior to the state-of-the-art gradient-based methods in terms of efficiency and accuracy.

Conclusion

In this paper, we proposed a doubly stochastic gradient method to improve the performance of penalty method for bilevel hyperparameter optimization. Within each update step, we randomly sample a validation point and a single constraint to calculate the stochastic gradient. We also prove that our method can obtain the hypergradient and converge to saddle point at the rate of $O(1/\sqrt{K})$. The experimental results also demonstrate the superiority our method in terms of training time and accuracy. Our method needs f and g to be twice continuously differentiable in both u and v, which limits the usage of our method. In the feature, we will explore the bilevel problem with the nonsmooth lower-level problems or the discrete hyperparameters.

Acknowledgments

B. Gu was partially supported by National Natural Science Foundation of China (No: 62076138), the Qing Lan Project (No.R2020Q04), the National Natural Science Foundation of China (No.62076138), the Six talent peaks project (No.XYDXX-042) and the 333 Project (No. BRA2017455) in Jiangsu Province.

References

Bao, R.; Gu, B.; and Huang, H. 2019. Efficient Approximate Solution Path Algorithm for Order Weight L_1-Norm with Accuracy Guarantee. In *2019 IEEE International Conference on Data Mining (ICDM)*, 958–963. IEEE.

Bard, J. F. 2013. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media.

Bergstra, J.; and Bengio, Y. 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research* 13(1): 281–305.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, 2546–2554.

Bertsekas, D. P. 1976. On penalty and multiplier methods for constrained minimization. *SIAM Journal on Control and Optimization* 14(2): 216–235.

Brochu, E.; Cora, V. M.; and De Freitas, N. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* .

Domke, J. 2012. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, 318–326.

Feurer, M.; and Hutter, F. 2019. Hyperparameter optimization. In *Automated Machine Learning*, 3–33. Springer, Cham.

Franceschi, L.; Donini, M.; Frasconi, P.; and Pontil, M. 2017. Forward and reverse gradient-based hyperparameter optimization. *arXiv preprint arXiv:1703.01785* .

Gu, B.; and Ling, C. 2015. A new generalized error path algorithm for model selection. In *International Conference on Machine Learning*, 2549–2558.

Gu, B.; Liu, G.; and Huang, H. 2017. Groups-keeping solution path algorithm for sparse regression with automatic feature grouping. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 185–193.

Gu, B.; and Sheng, V. S. 2017. A solution path algorithm for general parametric quadratic programming problem. *IEEE Transactions on Neural Networks and Learning Systems* 29(9): 4462–4472.

Koh, P. W.; and Liang, P. 2017. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730* .

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science* 350(6266): 1332–1338.

Liu, T.; and Tao, D. 2015. Classification with noisy labels by importance reweighting. *IEEE Transactions on pattern analysis and machine intelligence* 38(3): 447–461.

Maclaurin, D.; Duvenaud, D.; and Adams, R. 2015. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, 2113–2122.

Mehra, A.; and Hamm, J. 2019. Penalty Method for Inversion-Free Deep Bilevel Optimization. *arXiv preprint arXiv:1911.03432* .

Mei, S.; and Zhu, X. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *AAAI*, 2871–2877.

Muñoz-González, L.; Biggio, B.; Demontis, A.; Paudice, A.; Wongrassamee, V.; Lupu, E. C.; and Roli, F. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 27–38.

Pearlmutter, B. A. 1994. Fast exact multiplication by the Hessian. *Neural Computation* 6(1): 147–160.

Pedregosa, F. 2016. Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355* .

Rajeswaran, A.; Finn, C.; Kakade, S. M.; and Levine, S. 2019. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, 113–124.

Ren, M.; Zeng, W.; Yang, B.; and Urtasun, R. 2018. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050* .

Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065* .

Shaban, A.; Cheng, C.-A.; Hatch, N.; and Boots, B. 2019. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 1723–1732.

Shafahi, A.; Huang, W. R.; Najibi, M.; Suciu, O.; Studer, C.; Dumitras, T.; and Goldstein, T. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, 6103–6113.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, 4077–4087.

Sung, F.; Yang, Y.; Zhang, L.; Xiang, T.; Torr, P. H.; and Hospedales, T. M. 2018. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1199–1208.

Swersky, K.; Snoek, J.; and Adams, R. P. 2014. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896* .

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847–855.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*, 3630–3638.

Wu, J.; Chen, X.-Y.; Zhang, H.; Xiong, L.-D.; Lei, H.; and Deng, S.-H. 2019. Hyperparameter optimization for

machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology* 17(1): 26–40.

Xu, Y. 2020. Primal-dual stochastic gradient method for convex programs with many functional constraints. *SIAM Journal on Optimization* 30(2): 1664–1692.

Yu, X.; Liu, T.; Gong, M.; Zhang, K.; Batmanghelich, K.; and Tao, D. 2017. Transfer learning with label noise. *arXiv preprint arXiv:1707.09724* .