A Stochastic Approach to Bi-Level Optimization for Hyperparameter Optimization and Meta Learning

Minyoung Kim[1], **Timothy Hospedales**[1,2]
[1]Samsung AI Center Cambridge, UK
[2]University of Edinburgh, UK
mikim21@gmail.com, t.hospedales@ed.ac.uk

Abstract

We tackle the general differentiable meta learning problem that is ubiquitous in modern deep learning, including hyperparameter optimization, loss function learning, few-shot learning, invariance learning and more. These problems are often formalized as Bi-Level optimizations (BLO). We introduce a novel perspective by turning a given BLO problem into a stochastic optimization, where the inner loss function becomes a smooth probability distribution, and the outer loss becomes an expected loss over the inner distribution. To solve this stochastic optimization, we adopt Stochastic Gradient Langevin Dynamics (SGLD) MCMC to sample inner distribution, and propose a recurrent algorithm to compute the MC-estimated hypergradient. Our derivation is similar to forward-mode differentiation, but we introduce a new first-order approximation that makes it feasible for large models without needing to store huge Jacobian matrices. The main benefits are two-fold: i) Our stochastic formulation takes into account uncertainty, which makes the method robust to suboptimal inner optimization or non-unique multiple inner minima due to overparametrization; ii) Compared to existing methods that often exhibit unstable behavior and hyperparameter sensitivity in practice, our method leads to considerably more reliable solutions. We demonstrate that the new approach achieves promising results on diverse meta learning problems and easily scales to learning 87M hyperparameters in the case of Vision Transformers.

Extended version — https://arxiv.org/pdf/2410.10417

1 Introduction

We consider general differentiable meta learning, which includes bi-level optimization (BLO) problems such as hyperparameter optimization (HPO) and few-shot meta-learning. We introduce our problem setting, terminology and notation through a representative BLO problem of learning neural network regularizer hyperparameters (aka weight decay, or *L1/2*-regularization). Expressed as a BLO, this is

$$\min_{\lambda} \mathbb{E}_{(x,y)\sim V}[l_V(x,y;\theta^*(\lambda))] \quad (1)$$
$$\text{s.t. } \theta^*(\lambda) = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim T}[l_T(x,y;\theta)] + R(\lambda,\theta),$$

where θ are neural network parameters to learn, λ are all hyperparameters to learn (often denoted as outer variables or hyperparameters, depending on the context), T and V indicate training and validation data sets, respectively, l_T and l_V indicate training and validation losses, and R is the regularization or weight decay term. In the parameter-wise regularization case, $R(\lambda,\theta) = \sum_j \lambda_j \theta_j^2$ or $\sum_j \lambda_j |\theta_j|$ in which λ has the same structure as θ.

The main challenge in differentiable BLO is to efficiently and accurately compute the hypergradient $dl_V/d\lambda$ to update the hyperparameters. In the literature there exist some well-known approaches. Some aim to unroll the inner-loop optimization (i.e., forward- or reverse-mode differentiation (Franceschi et al. 2017), denoted by FMD and RMD, respectively), while others aim to compute it via the implicit function theorem (IFT) to circumvent the infeasible memory cost of saving intermediate Hessian matrices (in FMD) or computation graphs (in RMD) from unrolled inner optimization steps. Despite the theoretical promise of IFT-based meta-gradient (Rajeswaran et al. 2019; Lorraine, Vicol, and Duvenaud 2020) for memory-efficiency, it still suffers from a difficult and potentially unstable Hessian inverse, and it is highly reliant on the *gradient-equal-to-zero* (i.e., perfectly converged) condition for the inner optimization. In practice these drawbacks mean that it is unreliable and hard to tune.

A fundamental issue for all these existing BLO solutions is that the inner optimization is deterministic, and they are unable to account for the uncertainty in the inner problem and/or its solution. When applied to contemporary deep learning there are at least two highly practical examples of inner optimization uncertainty: When implemented in practice by minibatch SGD with a finite number of steps, the inner loop is not perfectly converged (violating IFT's assumption); and when applied to non-convex neural networks, the inner optimization has many local minima due to overparametrization (Vicol et al. 2022), leading to uncertainty over which inner minima is returned.

To address these issues, we introduce a new stochastic gradient approach to hyper-gradient calculation for BLO. We generalize the standard deterministic BLO problem to a stochastic optimization where the inner optimization produces a smooth probability distribution, and the outer optimization takes an expectation with respect to the inner posterior. This stochastic generalization allows us to deal with uncertainty in the inner optimization arising from noise, the use of minibatch SGD, or from multiple local minima. More

specifically, we exploit SGLD to obtain posterior samples of the inner optimization posterior, and derive a new efficient hypergradient estimation algorithm by taking derivatives of the SGLD step equations.

2 Proposed Algorithm

We first introduce *stochastic optimization*, and show that it generalizes the BLO problem. We argue that solving the stochastic optimization is preferable to solving the deterministic BLO due to better handling of inner optimization uncertainty, which we illustrate via a failure case of BLO in Sec. 2.2. Our proposed algorithm for solving the stochastic optimization is described in Sec. 2.3, in which new recursions for meta gradients are derived from stochastic gradient MCMC (Langevin dynamics).

2.1 Stochastic optimization (SO)

We aim to solve the *stochastic optimization* problem:

$$\min_{\lambda} \mathbb{E}_{p(\theta|\lambda)}[f(\lambda,\theta)] \text{ s.t. } p(\theta|\lambda) = \frac{\exp(-E(\lambda,\theta))}{Z(\lambda)}, \quad (2)$$

where $f(\lambda,\theta)$ and $E(\lambda,\theta)$ are the given scalar functions for the problem, and $Z(\lambda)$ is the normalizer that ensures $p(\theta|\lambda)$ to be a proper distribution. We have no restriction or constraints on the problem instance functions $f()$ and $E()$ except for the minimal assumption that evaluating these functions and taking their derivatives are straightforward and easy. We will see that this stochastic optimization formulation is quite flexible and general enough to encompass the general Bi-Level optimization (BLO) problem, that often arises in deep learning, as a special case. We will see this connection in detail in what follows.

♣ BLO as a special case of Stochastic optimization. The Bi-Level optimization (BLO) problem, despite its several different forms, can be formally and concisely expressed as:

$$\min_{\lambda} \mathcal{L}_V(\lambda,\theta^*(\lambda)) \text{ s.t. } \theta^*(\lambda) \underbrace{=}_{\text{or } \in} \arg\min_{\theta} \mathcal{L}_T(\lambda,\theta). \quad (3)$$

We often call $\mathcal{L}_V()$ the *outer objective* and $\mathcal{L}_T()$ the *inner objective*. In typical *hyperparameter optimization* (HPO) problems, $\mathcal{L}_V()$ and $\mathcal{L}_T()$ correspond to validation and training losses respectively. Meanwhile λ is the set of hyperparameters to learn and θ are the main parameters (weights of the neural network). We will show that this (deterministic) BLO problem in (3) is a special case of the stochastic optimization in (2). First we let the objective function $f(\lambda,\theta)$ in (2) be equal to $\mathcal{L}_V(\lambda,\theta)$, that is,

$$f(\lambda,\theta) := \mathcal{L}_V(\lambda,\theta), \quad (4)$$

and define the distribution $p(\theta|\lambda)$ as:

$$p(\theta|\lambda) := \frac{e^{-\mathcal{L}_T(\lambda,\theta)/\tau}}{Z_\tau(\lambda)}, \quad Z_\tau(\lambda) = \int e^{-\mathcal{L}_T(\lambda,\theta)/\tau}\, d\theta, \quad (5)$$

and $\tau > 0$ is a hyperparameter known as the *temperature*. In other words, we set the energy function of $p(\theta|\lambda)$ to be:

$$E(\lambda,\theta) := \mathcal{L}_T(\lambda,\theta)/\tau \quad (6)$$

Through the hyperparameter τ we can control *how certain (or uncertain)* the impact of the inner optimization is on the outer optimization. As an extreme case, it is obvious that as we tend $\tau \to 0$, we make $p(\theta|\lambda)$ converge to the delta function[1] $\delta(\theta - \theta^*(\lambda))$, and thus the stochastic problem (2) coincides with (3).

2.2 Why is SO better than deterministic BLO?

BLO's deterministic nature inner optimization might be problematic in certain situations. In this section we aim to highlight this issue via an illustrative example where SO succeeds and BLO fails.

Mainstream BLO algorithms used in deep learning rely on *one single* optimal solution $\theta^*(\lambda)$ for the inner optimization given λ, even though the inner optimization problem can be noisy (e.g., noise in training data) and/or can have multiple different local/global optima. Note that the latter situation almost always arises for models with a high degree of redundancy (i.e., overparametrization) such as deep neural networks. The worst scenario is that a (deterministic) BLO algorithm unfortunately selectings an optimum $\theta^*(\lambda)$ that leads to a poor (high) outer loss at each outer iteration, which can eventually lead us to select a poor outer variable λ as the final solution of the BLO.

On the other hand, our stochastic optimization formulation in (2) essentially takes into account *all* θ as possible solutions to the inner optimization problem through a probability distribution $p(\theta|\lambda)$. It is also beneficial in that the solutions become more robust to potential noise that may reside in the inner problem such as noisy training data, or solution by minibatch-SGD, since we deal with the *average* outer loss over all possible θ's instead of a single-outcome loss.

To illustrate this benefit clearly and explicitly, we present a toy scenario of the polynomial function regression hyperparameter learning problem. First we consider the quadratic $f_{true}(x) = -0.4x^2 - 0.2x - 0.3$ as the true data generating function on $x \in [-1,1]$, from which we sample four points: $T = \{(-0.75,-0.375),(0.75,-0.675)\}$ constituting the training data, and $V = \{(-0.5,-0.3),(0.5,-0.5)\}$ the validation data (See Fig. 1(a)).

Our model is assumed to have a cubic form, $f(x;\lambda,\theta) = \lambda x^3 + \theta_2 x^2 + \theta_1 x + \theta_0$ where $\lambda \in [-1,1]$ is the hyperparameter that decides the degree/smoothness of the polynomial, and $\theta = [\theta_0,\theta_1,\theta_2]^\top \in [-1,1]^3$ are the main parameters. We define the losses as: $\mathcal{L}_V(\lambda,\theta) = \sum_{(x,y)\in V}(f(x;\lambda,\theta)-y)^2$ and $\mathcal{L}_T(\lambda,\theta) = \sum_{(x,y)\in T}(f(x;\lambda,\theta)-y)^2$ for the validation and train dataset V and T, respectively.

With the BLO formulation, we can solve it by an exhaustive *tabular* method. Namely, for each candidate λ we find *all* possible inner optimal solutions $\theta^*(\lambda)$, which are infinitely many, but can be enumerated for some grid of values. Then for each candidate $(\lambda,\theta^*(\lambda))$ we evaluate the outer loss value $\mathcal{L}_V(\lambda,\theta^*(\lambda))$. This is depicted in Fig. 1(c). We can see that for each row (given λ), there are many different $\theta^*(\lambda)$ that yield different outer losses $\mathcal{L}_V$ (cooler colors for lower $\mathcal{L}_V$ than warmer). Note however that they all attain perfect inner loss $\mathcal{L}_T = 0$ (i.e., *all are global inner optima*). Meanwhile, by inspection, can see that the optimal λ has to be 0 (thus quadratic), and $\theta^*(0)$ (one of the fourth column in Fig 1(c))

[1] Or a mixture of delta functions if there are multiple global minima $\theta^*(\lambda)$.

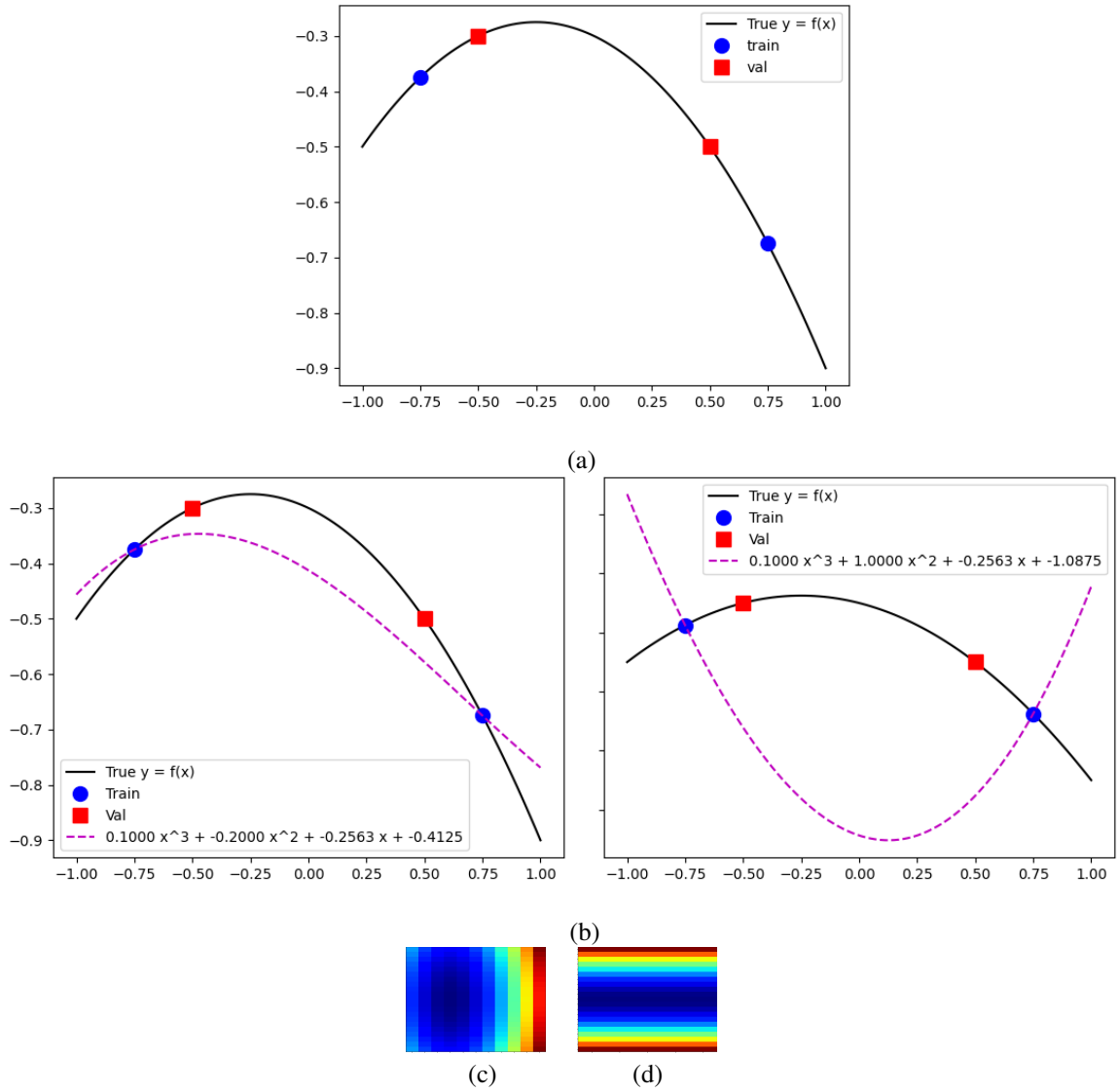

Figure 1: Illustrative toy problem. (a) Training and validation data. (b) Two $\theta^*(\lambda)$ solutions at $\lambda = 0.1$; (Left) Good $\theta^*(\lambda)$ (val loss 0.0083), (Right) Poor $\theta^*(\lambda)$ (val loss 0.3833). (c) $\mathcal{L}_V(\lambda, \theta^*(\lambda))$. Each row $= \lambda$, each column $=$ one of $\theta^*(\lambda)$. Blue (red) indicates low (high) loss value. (d) Row-wise average validation losses $\mathbb{E}_{p(\theta|\lambda)}[\mathcal{L}_V(\lambda, \theta)]$ employed in our proposed stochastic optimization (SO).

identifies the true quadratic f_{true}. Examples of good and poor $\theta^*(\lambda)$ at $\lambda = 0.1$ are shown in Fig. 1(b).

Now, consider typical deterministic BLO solutions, which select just *one of the inner optima* $\theta^*(\lambda)$ for each λ. In Fig. 1(c), we can think of them as *randomly* selecting a column for each row λ. Then the final solution λ is chosen as the one with the smallest validation loss among those randomly selected. So, there is a high chance that the final solution is not the optimal one $\lambda = 0$ (e.g., when a good $\theta^*(\lambda)$ is chosen for some suboptimal $\lambda \neq 0$, but a poor $\theta^*(\lambda)$ is chosen for $\lambda = 0$). On the other hand, the stochastic optimization considers the average $\mathbb{E}_{p(\theta|\lambda)}[\mathcal{L}_V(\lambda, \theta)]$ for each λ, and selects the λ with the smallest average. This is visualized in Fig. 1(d), where each row (λ) has an averaged validation loss. Clearly it is guaranteed to select the optimal $\lambda = 0$ as the final solution.

When we ran these two approaches on the tabularized BLO problem with 21 (11) equally spaced λ ($\theta^*(\lambda)$) grid point values, the deterministic BLO solution via the random column selection was able to find the optimal $\lambda = 0$ for only three out of 20 different runs[2], whereas stochastic optimization always found the optimal solution.

[2]We emphasize that this result has nothing to do with the quality of hypergradients, because it is a grid search. So previous variance-reduction techniques (Yang and Kwok 2022) for hypergradients

2.3 Proposed Solution: SGLD Gradient Recursion

We propose a novel method to solve the stochastic optimization (2). Using the Monte-Carlo approximation,

$$\mathbb{E}_{p(\theta|\lambda)}[f(\lambda,\theta)] \approx \frac{1}{M}\sum_{m=1}^{M} f(\lambda,\theta^m),\ \theta^m \sim p(\theta|\lambda), \quad (7)$$

where M is the number of MC samples. The (approximate) hypergradient of the objective can be written as:

$$\frac{d}{d\lambda}\mathbb{E}_{p(\theta|\lambda)}[f(\lambda,\theta)] \approx \frac{1}{M}\sum_{m=1}^{M} \frac{df(\lambda,\theta^m)}{d\lambda}, \quad (8)$$

and using the chain rule (d for total, ∂ for partial gradients),

$$\frac{df(\lambda,\theta^m)}{d\lambda} = \frac{\partial f(\lambda,\theta^m)}{\partial\lambda} + \frac{\partial f(\lambda,\theta^m)}{\partial\theta}\cdot\frac{d\theta^m}{d\lambda}. \quad (9)$$

Note that each sample θ^m is a function of λ. However, in general it may be difficult to express θ^m explicitly in terms of λ. We instead adopt the stochastic-gradient Langevin dynamic SGLD (Welling and Teh 2011), which repeats the recurrence:

$$\theta \ \leftarrow\ \theta + \frac{\epsilon}{2}\frac{\partial \log p(\theta|\lambda)}{\partial\theta} + \sqrt{\epsilon}z, \quad (10)$$

where $z\sim\mathcal{N}(0,\kappa^2 I)$ and ϵ is a small (ideally infinitesimal) step size. We assume that $\epsilon^{1+\alpha}$ is virtually 0 for $\alpha\geq 0.5$ (e.g., $\epsilon^2\simeq 0$ and $\epsilon^{1.5}\simeq 0$), which is reasonable as ϵ is considered to be very small. A nice property is that (10) only requires the *gradient* of $\log p(\theta|\lambda)$, not the log-density itself, so even though the normalizer $Z(\lambda)$ might be complicated, we can only deal with the energy function, that is,

$$\frac{\partial \log p(\theta|\lambda)}{\partial\theta} = -\frac{\partial E(\lambda,\theta)}{\partial\theta}. \quad (11)$$

It is known that after some burn-in period, the iterates of (10) converge to the samples of the stationary distribution $p(\theta|\lambda)$. Also according to the Markov chain theorems (Gamerman and Lopes 2006), we can start from *any* initial iterate θ to make the chain converge to the target stationary distribution after a burn-in period. We let $\theta^{(0)}$ be the first iterate in the SGLD recurrence: $\theta^{(0)}$ can be either independent of λ (e.g., chosen randomly and independently of λ) or a function of λ (e.g., λ is the initial network parameters to be meta-learned in the MAML-type problems)[3]. In the former $\frac{d\theta^{(0)}}{d\lambda}=0$, while for the latter $\frac{d\theta^{(0)}}{d\lambda}$ is non-zero, but we assume that it is *(sub)linearly sparse*[4], and can be efficiently computed (e.g., in the MAML-type problems $\theta^{(0)}=\lambda$, and $\frac{d\theta^{(0)}}{d\lambda}=I$). We also let B be the the number of burn-in iterations before convergence. Therefore, we run the SGLD recurrence for $(B+M)$ iterations in total, throwing away the first B iterates, and collecting the last M iterates. Following the notation in (7), we set: $\theta^1:=\theta^{(B+1)}, \theta^2:=\theta^{(B+2)}, \ldots, \theta^M:=\theta^{(B+M)}$.

would not address the issue.

[3]Even though we can choose $\theta^{(0)}$ independently of λ, since we are only able to run the SGLD recurrences a *finite* number of times, it might be useful to have the first iterate $\theta^{(0)}$ dependent on λ. A good example is the MAML-type problems where we set $\theta^{(0)}=\lambda$.

[4]It means that for any $\dim(\theta)$-dim vector v, the product $v^\top\frac{d\theta^{(0)}}{d\lambda}$ can be computed in $O(\dim(\theta)+\dim(\lambda))$ time and space.

Now, for the first recurrence $\theta^{(0)}\rightarrow\theta^{(1)}$, that is,

$$\theta^{(1)} = \theta^{(0)} + \frac{\epsilon}{2}\frac{\partial \log p(\theta^{(0)}|\lambda)}{\partial\theta} + \sqrt{\epsilon}z^{(0)}, \quad (12)$$

by taking the gradient of both sides with respect to λ,

$$\frac{d\theta^{(1)}}{d\lambda} = \frac{d\theta^{(0)}}{d\lambda} + \frac{\epsilon}{2}\frac{d}{d\lambda}\frac{\partial \log p(\theta^{(0)}|\lambda)}{\partial\theta} + \frac{d}{d\lambda}\sqrt{\epsilon}z^{(0)} \quad (13)$$

$$= \frac{d\theta^{(0)}}{d\lambda} + \frac{\epsilon}{2}\left(\frac{\partial^2 \log p(\theta^{(0)}|\lambda)}{\partial\lambda\partial\theta} + \frac{\partial^2 \log p(\theta^{(0)}|\lambda)}{\partial\theta^2}\cdot\frac{d\theta^{(0)}}{d\lambda}\right)$$

$$= \left(I + \frac{\epsilon}{2}A(\lambda,\theta^{(0)})\right)\cdot\frac{d\theta^{(0)}}{d\lambda} + \frac{\epsilon}{2}B(\lambda,\theta^{(0)}), \quad (14)$$

where in (14) we let

$$A(\lambda,\theta) := \frac{\partial^2 \log p(\theta|\lambda)}{\partial\theta^2},\ B(\lambda,\theta) := \frac{\partial^2 \log p(\theta|\lambda)}{\partial\lambda\partial\theta}. \quad (15)$$

We move on to the next recurrence $\theta^{(1)}\rightarrow\theta^{(2)}$ and derive its derivatives similarly. Continuing this up to $\theta^{(m)}$, we have:

$$\frac{d\theta^{(m)}}{d\lambda} \simeq \left(I + \frac{\epsilon}{2}\Big(A(\lambda,\theta^{(0)}) + \cdots + A(\lambda,\theta^{(m-1)})\Big)\right)\cdot\frac{d\theta^{(0)}}{d\lambda}$$
$$+\ \frac{\epsilon}{2}\Big(B(\lambda,\theta^{(0)}) + \cdots + B(\lambda,\theta^{(m-1)})\Big). \quad (16)$$

The symbol $\simeq$ originates from exploitation of $\epsilon^2\simeq 0$, and the detailed derivations for (16) can be found in the extended version of the paper. However, storing even a single $A(\lambda,\theta)$ or $B(\lambda,\theta)$ is computationally infeasible for large-scale scenarios due to its huge matrix dimension $(\dim(\theta)\times\dim(\theta))$ or $(\dim(\theta)\times\dim(\lambda))$.

Instead, we derive a recursion directly from (9). Specifically, we focus on the second term of (9), denoted as $g_m(\lambda)$,

$$\frac{df(\lambda,\theta^{(m)})}{d\lambda} = \frac{\partial f(\lambda,\theta^{(m)})}{\partial\lambda} + \underbrace{\frac{\partial f(\lambda,\theta^{(m)})}{\partial\theta}\cdot\frac{d\theta^{(m)}}{d\lambda}}_{=:g_m(\lambda)}. \quad (17)$$

Note that $g_m(\lambda)$ is a $\dim(\lambda)$-dimensional vector, thus easy to deal with computationally. Now we derive the recursion for $g_m(\lambda)$ as follows:

$$g_m(\lambda) = \frac{\partial f(\lambda,\theta^{(m)})}{\partial\theta}\cdot\frac{d\theta^{(m)}}{d\lambda} \quad (18)$$

$$= \frac{\partial f(\lambda,\theta^{(m)})}{\partial\theta}\cdot\left(\frac{d\theta^{(m-1)}}{d\lambda} + \frac{\epsilon}{2}\left(B(\lambda,\theta^{(m-1)}) + A(\lambda,\theta^{(m-1)})\cdot\frac{d\theta^{(m-1)}}{d\lambda}\right)\right) \quad (19)$$

$$\simeq \frac{\partial f(\lambda,\theta^{(m)})}{\partial\theta}\cdot\left(\frac{d\theta^{(m-1)}}{d\lambda} + \frac{\epsilon}{2}B(\lambda,\theta^{(m-1)}) + \frac{\epsilon}{2}A(\lambda,\theta^{(m-1)})\cdot\frac{d\theta^{(0)}}{d\lambda}\right) \quad (20)$$

$$= \underbrace{\frac{\partial f(\lambda,\theta^{(m)})}{\partial\theta}\cdot\frac{d\theta^{(m-1)}}{d\lambda}}_{\approx\ g_{m-1}(\lambda)\ +\ (\theta^{(m)}-\theta^{(m-1)})\cdot\frac{\partial^2 f(\lambda,\theta^{(m-1)})}{\partial\theta^2}\cdot\frac{d\theta^{(0)}}{d\lambda}} + \quad (21)$$

$$\frac{\epsilon}{2}\frac{\partial f(\lambda,\theta^{(m)})}{\partial\theta}\cdot\left(B(\lambda,\theta^{(m-1)}) + A(\lambda,\theta^{(m-1)})\cdot\frac{d\theta^{(0)}}{d\lambda}\right)$$

To have (20) from (19) we again exploit $\epsilon^2\simeq 0$ using the

expression (16). In the under-brace of (21) we use a first-order approximation where the details and justification can be found in the extended version. The quality of our first-order approximation is also empirically demonstrated in Sec. 4.1.

In summary, we have the recursion for $g_m(\lambda)$:

$$g_m(\lambda) \;=\; g_{m-1}(\lambda) + \tag{22}$$
$$(\theta^{(m)} - \theta^{(m-1)}) \cdot \frac{\partial^2 f(\lambda, \theta^{(m-1)})}{\partial \theta^2} \cdot \frac{d\theta^{(0)}}{d\lambda} +$$
$$\frac{\epsilon}{2} \frac{\partial f(\lambda, \theta^{(m)})}{\partial \theta} \cdot \left(B(\lambda, \theta^{(m-1)}) + A(\lambda, \theta^{(m-1)}) \cdot \frac{d\theta^{(0)}}{d\lambda} \right).$$

In (22), we can see that it is not needed to store the large matrices A and B at all, since we have the vector-Hessian product forms. They can be easily computed in most modern deep learning libraries with the auto-differentiation capability. In `PyTorch`, for instance, $\frac{\partial f}{\partial \theta} \cdot B$ in (22) can be obtained by calling `autograd.grad()` with $t_1 = \frac{\partial}{\partial \theta} \log p(\theta^{(m-1)}|\lambda)$ and $t_2 = \frac{\partial}{\partial \theta} f(\lambda, \theta^{(m)})$ as:

$$\texttt{grad}\big(t_1,\ \lambda,\ \texttt{grad\_outputs} = t_2\big), \tag{23}$$

recalling that $B(\lambda, \theta) := \frac{\partial^2}{\partial \lambda \partial \theta} \log p(\theta|\lambda)$. We can compute $\frac{\partial f}{\partial \theta} \cdot A$ and $(\theta^{(m)} - \theta^{(m-1)}) \cdot \frac{\partial^2 f(\lambda, \theta^{(m-1)})}{\partial \theta^2}$ in a similar manner. Overall the computation for (22) requires only linear time and memory complexity in $\dim(\theta)$ and $\dim(\lambda)$ as we prove in Sec. 2.4. The initial $g_0(\lambda)$ can be easily computed[5]. Once $g_m(\lambda)$ is computed, we can plug it back into (17) and (8) to compute the ultimate hypergradient.

♣ **Summary.** Our hypergradient $h := \frac{d}{d\lambda} \mathbb{E}_{p(\theta|\lambda)}[f(\lambda, \theta)]$ can be approximately computed by the recursion: Initially we have $g_0(\lambda)$ and $h = 0$, and for $m = 1, \ldots, B+M$,

- $\theta^{(m)} = \theta^{(m-1)} + \frac{\epsilon}{2} \frac{\partial \log p(\theta^{(m-1)}|\lambda)}{\partial \theta} + \sqrt{\epsilon} z^{(m-1)}$ (24)
- Apply $g_{m-1}(\lambda) \to g_m(\lambda)$ recursion step in (22)
- If $m > B,\ h \leftarrow h + \frac{1}{M} \left(\frac{\partial f(\lambda, \theta^{(m)})}{\partial \lambda} + g_m(\lambda) \right)$ (25)

Our proposed method is summarized as pseudocode in Alg. 1. Unlike the IFT methods (Lorraine, Vicol, and Duvenaud 2020; Rajeswaran et al. 2019), our approach does not require Hessian inversion (e.g., the inverse of $\frac{\partial^2}{\partial \theta^2} \log p(\theta|\lambda)$). Furthermore, the IFT methods are all reliant to the zero inner gradient condition, thus can be sensitive (in an unpredictable way) to the quality of the inner optimization.

Besides, when the inner optimization can only be performed with stochastic minibatch-based updates (true for most deep learning scenarios), then this further deteriorates IFT's performance since the gradient vanishing condition would not hold for the minibatch version almost surely. On the other hand, in our approach, note that the gradient $\frac{\partial \log p(\theta|\lambda)}{\partial \theta}$ can be safely replaced by the minibatch stochastic gradient, as we resort to the stochastic-gradient MCMC (Welling and Teh 2011). This can lead to the same guarantee of the convergence to the samples from $p(\theta|\lambda)$.

[5]Specifically, $g_0 = 0$ if $\theta^{(0)}$ is independent of λ. If $\theta^{(0)} = \lambda$ in the MAML-type initial parameter meta-learning problems, then $g_0 = \frac{\partial f(\lambda, \theta^{(0)})}{\partial \theta}$, and can be easily computed.

Algorithm 1: Proposed algorithm (HPO-SGLD) to solve (2).

Input: Problem functions $f()$ and $E()$ in (2).
B (# burn-in steps) and M (# MC samples).
Initialize: λ.
For each outer iteration (λ):
Choose $\theta^{(0)}$, compute $g_0(\lambda)$ and set $h = 0$.
For $m = 1, \ldots, B+M$:
Perform (24), (22), and (25) in this order.
(Now we have hypergradient $h = \frac{d}{d\lambda} \mathbb{E}_{p(\theta|\lambda)}[f(\lambda, \theta)]$)
Update $\lambda \leftarrow \lambda - \eta \cdot h$ for some step size η.

2.4 Convergence Analysis & Complexity Analysis

Convergence Analysis. If $\theta^{(0)}$ is independent of λ, we show that: i) (Theorem 1) our first-order approximation is *exact* in the infinitesimal sense, and from which we prove that ii) (Theorem 2) our HPO-SGLD algorithm converges to an optimal solution at linear convergence rate. That is, to achieve ϵ-accuracy, we need $O(1/\epsilon)$ number of iterations. The details and proofs can be found in the extended version of the paper.

Theorem 1 (Exactness of First-order Approximation)**.** *If (A1) and (A2) in Supplement B in the extended version hold, then our first-order approximation used for $g_m(\lambda)$ recursion becomes exact (i.e., approximation error $\simeq 0$).*

Theorem 2 (Convergence of HPO-SGLD)**.** *If (A1-A3) in Supplement B in the extended version hold, and if the target function $f(\lambda, \theta)$ in Eq.(2) is Lipschitz continuous in λ for each θ, then for a sufficiently large number (M) of the Monte Carlo samples, our HPO-SGLD algorithm (Alg. 1) converges to an optimal solution of the stochastic optimization Eq.(2) at linear rate. More specifically, with constant step size η that satisfies $\eta \leq \frac{1}{L}$ (L is a Lipschitz constant of f), the following two cases hold: (a) For general (non-convex) f, the gradient vanishes as:*

$$\mathbb{E}\Big\| \nabla_\lambda \mathbb{E}_{p(\theta|\lambda^{(k)})}\big[f(\lambda^{(k)}, \theta)\big] \Big\|^2 \;\leq$$
$$\frac{\mathbb{E}_{p(\theta|\lambda^{(0)})}\big[f(\lambda^{(0)}, \theta)\big] - \mathbb{E}_{p(\theta|\lambda^*)}\big[f(\lambda^*, \theta)\big]}{\eta k} = O\Big(\frac{1}{k}\Big) \tag{26}$$

hence converging to a local optimum; (b) If $f(\lambda, \theta)$ is convex in λ for each θ, then

$$\mathbb{E}\Big| \mathbb{E}_{p(\theta|\lambda^{(k)})}\big[f(\lambda^{(k)}, \theta)\big] - \mathbb{E}_{p(\theta|\lambda^*)}\big[f(\lambda^*, \theta)\big] \Big| \;\leq$$
$$\frac{||\lambda^{(0)} - \lambda^*||_2^2}{2\eta k} \;=\; O\Big(\frac{1}{k}\Big) \tag{27}$$

where $\lambda^{(k)}$ is the iterate at the k-th outer iteration, and λ^* is an optimal solution.

Deterministic Counterpart. Our derivation in a *deterministic* setting, namely if we removed all stochastic components by dropping SGLD noise terms $\sqrt{\epsilon} z$ in (24), looks very similar to forward-mode differentiation (FMD). The main difference is that we proposed the recurrence on g_m with the first-order approximation in (21), allowing us to circumvent saving the Hessian matrices A and B. The FMD, on the other hand, without such approximation, needs to save these matrices, making it infeasible especially for high-dim λ cases. This is the crucial point that enables our algorithm to be practical in real-world deep learning situations. In Supplement C.1 in

	Time	Space
FMD	$O(T \cdot \dim(\lambda) \cdot (\dim(\theta)+\dim(\lambda)))$	$O(\dim(\theta)+\dim(\lambda))$
RMD	$O(T \cdot (\dim(\theta)+\dim(\lambda)))$	$O(T \cdot (\dim(\theta)+\dim(\lambda)))$
Ours	$O(T \cdot (\dim(\theta)+\dim(\lambda)))$	$O(\dim(\theta)+\dim(\lambda))$

Table 1: Time/space complexity. $T = \#$ inner iterations.

the extended version of the paper, we provide more detailed views on this claim, scrutinizing our deterministic version, the FMD algorithms, and the similarity between the two.

Computational Complexity. We analyze the time and space complexity of the proposed algorithm and contrast it to FMD algorithms as well as the reverse-mode differentiation (RMD). The analysis is based on the techniques from algorithmic differentiation (Griewank and Walther 2008; Baydin et al. 2015; Franceschi et al. 2017). The complexity of the competing algorithms is summarized in Tab. 1, and we leave details of our analysis in Supplement C.2 in the extended version of the paper.

3 Related Work

Existing methods for hypergradient estimation in BLO can fall into two categories: *IFT-based* and *Unroll-based*. We highlight the key ideas and known issues of these approaches here, and for more comprehensive summaries we refer the readers to survey papers (Chen et al. 2022; Liu et al. 2021). IFT hinges on the stationary condition of the inner optimal solution, and computes the hypergradient from this implicit definition. Two popular variants differ in how to approximate the Hessian inversion required in IFT – by Neumann series approximation (Lorraine, Vicol, and Duvenaud 2020) and conjugate gradient (Rajeswaran et al. 2019) respectively. Since both variants are reliant on the gradient-equal-to-0 stationarity condition and involve Hessian inversion, their overall performance is sensitive to the quality of the inner optimization in an unpredictable, highly nonlinear way.

Unroll-based approaches basically approximate the inner optimal solution by an SGD iterate at some final (finite) step. So we have a chain of dependency constraints, on which we can compute the hypergradients by the chain rule. The two different ways of applying the chain rule correspond to FMD and RMD (Franceschi et al. 2017). Despite its simplicity, RMD suffers from excessive memory usage to maintain a large computation graph built for the unrolled inner optimization, which in turn leads to truncated approximations (Luketina et al. 2016) that bring problems of their own (Metz et al. 2019). FMD also requires large computational resources to store huge Jacobian matrices and does not scale to high-dimensional λ (Micaelli and Storkey 2021). Although our derivation looks similar to FMD in some respects, our new approximate recurrence circumvents FMD's inherent overhead. Other gradient-free strategies for solving BLOs such as evolution (Gonzalez and Miikkulainen 2021) also do not scale to high-dimensional λ.

In our stochastic optimization we transform the inner loss function into a probability distribution via energy-based modeling (EBM). Previously, there were attempts to apply SGLD to EBM sampling (Du and Mordatch 2019; Grathwohl et al. 2020). However, there approaches have nothing to do with *meta learning*, and our novelty is to bring an EBM perspective to meta-learning and hyperparameter optimization.

	λ^* (Error)	θ^* (Error)
True from line search	0.7487 (-)	0.6629 (-)
HPO-SGLD (Ours)	0.7488 (0.0001)	0.6629 (0.0000)
IFT-Neumann	0.7491 (0.0004)	0.6625 (0.0004)
IFT-CG	0.7488 (0.0001)	0.6628 (0.0001)
RMD	0.7360 (0.0127)	0.6770 (0.0141)
RMD-FO	0.6916 (0.0572)	0.7228 (0.0599)
FMD	0.7360 (0.0127)	0.6770 (0.0141)
RMD (1000 inner iterations)	0.7489 (0.0002)	0.6627 (0.0002)
RMD-FO (1000 inner iterations)	0.6916 (0.0572)	0.7226 (0.0597)
FMD (1000 inner iterations)	0.7489 (0.0002)	0.6627 (0.0002)

Table 2: Synthetic 1D problem. Test errors.

4 Experiments

4.1 Synthetic 1D Problem

We consider a simple BLO problem where in (3) we define $\mathcal{L}_V(\lambda,\theta) = (\lambda-\theta)^2 + \left(\theta-\frac{1}{2}\right)^2$, $\mathcal{L}_T(\lambda,\theta) = \frac{\theta^3}{3} - (1-\lambda^2)\theta$ in $\lambda, \theta \in [0,1]$. The inner loss function admits a closed-form global minimum $\theta^*(\lambda) = \sqrt{1-\lambda^2}$. And by plugging this in the outer loss function and with some 1D line search, we have the optimal solution $\lambda^* = 0.7487$ at $\theta^* = 0.6629$. We tackle this problem using the hypergradient methods, and the results summarized in Table 2. For all competing methods we run up to 100 inner iterations while in our HPO-SGLD, we split the inner iterations into $B = 50$ burn-in steps and $M = 50$ MC sample accumulation steps, to be fair. Other experimental details are summarized in the extended version of the paper. We see that both IFT-Neumann/CG and our HPO-SGLD are equally good, accurately identifying the true optimal values. On the other hand, RMD and FMD require more inner iterations as they only reach comparable errors at 1000 inner iterations. We also vary the number of inner loop iterations, and the results are shown in Fig. 2.

Noisy inner loss case. Now to make the problem more realistic and difficult, we modify the problem in a way that the inner loss function is randomly perturbed at every call. More specifically, the new inner loss function is defined as:

$$\mathcal{L}_T(\lambda,\theta) = (1/3+\epsilon_1)\cdot\theta^3 - (1-\lambda^2+\epsilon_2)\cdot\theta \qquad (28)$$

where $\epsilon_1, \epsilon_2 \sim \text{Uniform}(-0.3, 0.3)$. This modification makes the problem more realistic by mimicking the real-world situations where we often observe noise from various sources in the problem data (e.g., stochastic minibatch formation or noise in inputs and/or labels) in the inner loss function. In this case our stochastic optimization formulation (Sec. 2.1) is expected to be particularly useful. The results summarized in Table 3 show that deterministic approaches like IFT-Neumann/CG are very sensitive to the noisy inner loss function, failing to attain the optimal values. This is mainly due to violation of the strict gradient-equal-to-0 condition for the implicit function theorem. On the other hand, our stochastic optimization treatment, for different degrees of stochasticity

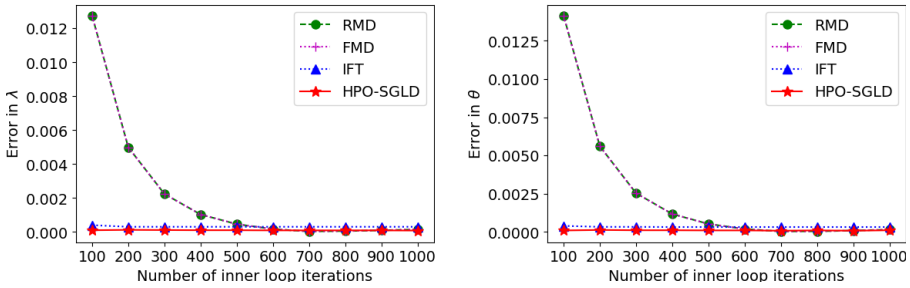

Figure 2: (Synthetic 1D) The number of inner iterations vs. the errors of the learned solutions, (Left) $|\lambda - \lambda^*|$ and (Right) $|\theta - \theta^*|$.

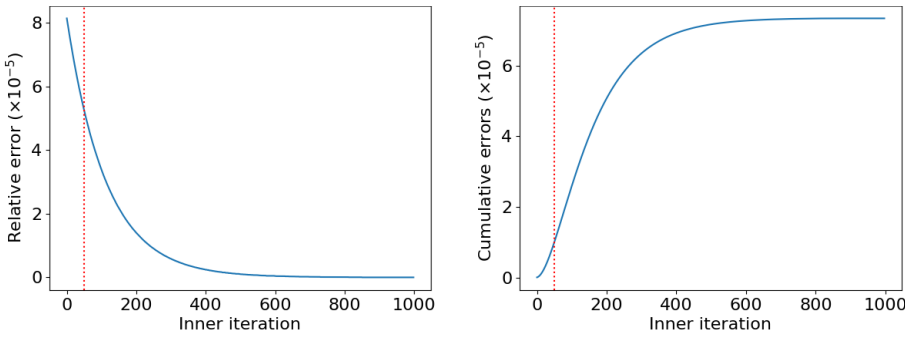

Figure 3: (Left) Relative errors between the true products of gradients vs. our first-order approximates. (Right) Cumulative errors between true hypergradients and our estimates. Red/dotted lines indicate the end of the burn-in period.

	λ^* (Error)	θ^* (Error)
True from line search	0.7487 (-)	0.6629 (-)
HPO-SGLD $\tau = 10^{-2}$ (Ours)	0.7495 (0.0008)	0.6516 (0.0113)
HPO-SGLD $\tau = 10^{-5}$ (Ours)	0.7473 (0.0014)	0.6383 (0.0246)
HPO-SGLD $\tau = 10^{-10}$ (Ours)	0.7468 (0.0019)	0.6411 (0.0218)
IFT-Neumann	0.7640 (0.0153)	0.6203 (0.0426)
IFT-CG	0.7544 (0.0057)	0.6318 (0.0311)
RMD	0.7363 (0.0124)	0.6779 (0.0150)
RMD-FO	0.6923 (0.0564)	0.7235 (0.0606)
FMD	0.7364 (0.0123)	0.6778 (0.0149)
RMD (1000 inner iterations)	0.7517 (0.0030)	0.6484 (0.0145)
RMD-FO (1000 inner iterations)	0.6926 (0.0561)	0.7075 (0.0446)
FMD (1000 inner iterations)	0.7517 (0.0030)	0.6484 (0.0145)

Table 3: Noisy synthetic 1D problem.

considered ($\tau = 10^{\{-2,-5,-10\}}$), leads to more robust estimation of the optimal values. Considering the highly stochastic nature of the problem, we see that incorporating more stochasticity (i.e., larger τ) in our HPO-SGLD model, leads to more accurate estimation.

Goodness of First-Order Approximation To demonstrate the quality of the first-order approximation that we introduced in (21), that is, $\frac{\partial f(\lambda,\theta^{(m)})}{\partial \theta} \cdot \frac{d\theta^{(m-1)}}{d\lambda} \approx g_{m-1}(\lambda) + (\theta^{(m)} - \theta^{(m-1)}) \cdot \frac{\partial^2 f(\lambda,\theta^{(m-1)})}{\partial \theta^2} \cdot \frac{d\theta^{(0)}}{d\lambda}$, we record and compare the true product of gradient terms (left hand side) and our approximate values (right hand side). The results are visualized in Fig. 3, and we clearly see that the approximation quality is very good. We measure the errors from the beginning of our recurrences. Hence the final accumulated meta-gradient estimate would be even more accurate since we drop the meta-gradient estimates at the early iterations (relatively larger errors) as burn-in steps.

4.2 Additional Experimental Results

We leave additional experimental results in the extended version. They include: (a) Experiments on other interesting meta learning problems including L1 regularizer HPO in ERM learning, optimal loss function learning, few-shot meta learning, meta learning of implicit neural representation, and invariance learning; (b) Ablation study; (c) Wall clock running time comparison; and (d) Comparison with other recent stochastic methods for BLO and the evolutionary search methods.

5 Conclusion

We have introduced a new hypergradient estimation method for meta learning. Our stochastic optimization formulation takes into account uncertainty in inner optimization, rendering solutions to robust to noise and non-unique inner optima. Our proposed forward recursion method enables computationally tractable solutions even in large scale scenarios (e.g., 87M parameters and 87M hyperparameters for VIT-B-16).

References

Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M. 2015. Automatic differentiation in machine learning: A survey. *arXiv preprint arXiv:1502.05767*.

Chen, C.; Chen, X.; Ma, C.; Liu, Z.; and Liu, X. 2022. Gradient-based Bi-level Optimization for Deep Learning: A Survey. *arXiv preprint arXiv:2207.11719*.

Du, T.; and Mordatch, I. 2019. Implicit Generation and Modeling with Energy Based Models. *NeurIPS*.

Franceschi, L.; Donini, M.; Frasconi, P.; and Pontil, M. 2017. Forward and Reverse Gradient-Based Hyperparameter Optimization. In *ICML*.

Gamerman, D.; and Lopes, H. F. 2006. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC Texts in Statistical Science.

Gonzalez, S.; and Miikkulainen, R. 2021. Optimizing Loss Functions Through Multivariate Taylor Polynomial Parameterization. *GECCO-2021*.

Grathwohl, W.; Wang, K.; Jacobsen, J.; Duvenaud, D.; Norouzi, M.; and Swersky, K. 2020. Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One. *ICLR*.

Griewank, A.; and Walther, A. 2008. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. In *Society for Industrial and Applied Mathematics, Second Edition*.

Liu, R.; Gao, J.; Zhang, J.; Meng, D.; and Lin, Z. 2021. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *TPAMI*, 44(12): 10045–10067.

Lorraine, J.; Vicol, P.; and Duvenaud, D. 2020. Optimizing Millions of Hyperparameters by Implicit Differentiation. AISTATS.

Luketina, J.; Berglund, M.; Greff, K.; and Raiko, T. 2016. Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters. In *ICML*.

Metz, L.; Maheswaranathan, N.; Nixon, J.; Freeman, D.; and Sohl-Dickstein, J. 2019. Understanding and correcting pathologies in the training of learned optimizers. In *ICML*.

Micaelli, P.; and Storkey, A. J. 2021. Gradient-based hyperparameter optimization over long horizons. *NeurIPS*.

Rajeswaran, A.; Finn, C.; Kakade, S.; and Levine, S. 2019. Meta-Learning with Implicit Gradients. In *NeurIPS*.

Vicol, P.; Lorraine, J. P.; Pedregosa, F.; Duvenaud, D.; and Grosse, R. B. 2022. On Implicit Bias in Overparameterized Bilevel Optimization. In *ICML*.

Welling, M.; and Teh, Y. W. 2011. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *ICML*.

Yang, H.; and Kwok, J. 2022. Efficient Variance Reduction for Meta-learning. In *ICML*.