Guarantees for Tuning the Step Size using a Learning-to-Learn Approach

Xiang Wang* Shuai Yuan† Chenwei Wu‡ Rong Ge§

July 1, 2020

Abstract

Learning-to-learn – using optimization algorithms to learn a new optimizer – has successfully trained efficient optimizers in practice. This approach relies on meta-gradient descent on a meta-objective based on the trajectory that the optimizer generates. However, there were few theoretical guarantees on how to avoid meta-gradient explosion/vanishing problems, or how to train an optimizer with good generalization performance. In this paper we study the learning-to-learn approach on a simple problem of tuning the step size for quadratic loss. Our results show that although there is a way to design the meta-objective so that the meta-gradient remain polynomially bounded, computing the meta-gradient directly using backpropagation leads to numerical issues that look similar to gradient explosion/vanishing problems. We also characterize when it is necessary to compute the meta-objective on a separate validation set instead of the original training set. Finally, we verify our results empirically and show that a similar phenomenon appears even for more complicated learned optimizers parametrized by neural networks.

1 Introduction

Choosing the right optimization algorithm and related hyper-parameters is important for training a deep neural network. Recently, a series of works (e.g., Andrychowicz et al. (2016); Wichrowska et al. (2017)) proposed to use learning algorithms to find a better optimizer. These papers use a learning-to-learn approach: they design a class of possible optimizers (often parametrized by a neural network), and then optimize the parameters of the optimizer (later referred to as meta-parameters) to achieve better performance. We refer to the optimization of the optimizer as the meta optimization problem, and the application of the learned optimizer as the inner optimization problem. The learning-to-learn approach solves the meta optimization problem by defining a meta-objective function based on the trajectory that the inner-optimizer generates, and then using back-propagation to compute the meta-gradient.

Although the learning-to-learn approach has shown empirical success, there are very few theoretical guarantees for learned optimizers. In particular, since the optimization for meta-parameters is usually a nonconvex problem, does it have bad local optimal solutions? Current ways of optimizing meta-parameters rely on unrolling the trajectory of the inner-optimizer, which is very expensive and often lead to exploding/vanishing gradient problems, is there a way to alleviate these problems? Can we have a provable way of designing meta-objective to make sure that the inner optimizers can achieve good generalization performance?

In this paper we answer some of these problems in a simple setting, where we use the learning-to-learn approach to tune the step size of the standard gradient descent/stochastic gradient descent algorithm. We will see that even in this simple setting, many of the challenges still remain and we can get better learned optimizers by choosing the right meta-objective function. Though our results are proved only in the simple setting, we empirically verify the results using complicated learned optimizers with neural network parametrizations.

*Duke University. Email: xwang@cs.duke.edu
†Duke University. Email: shuai@cs.duke.edu
‡Duke University. Email: chenwei.wu592@duke.edu
§Duke University. Email: rongge@cs.duke.edu

1.1 Challenges of learning-to-learn approach and our results

Metz et al. (2019) highlighted several challenges in the meta-optimization for learning-to-learn approach. First, they observed that the optimal parameters for the learned optimizer (or even just the step size for gradient descent) can depend on the number of training steps t of the inner-optimization problem. This was also separately proved theoretically in a least-squares setting in Ge et al. (2019). Because of this, one needs to do the meta-training for an optimizer that runs for enough number of steps (similar to the number of steps that it would take when we apply the learned optimizer). However, when the number of steps is large, the meta-gradient can often explode or vanish, which makes it difficult to solve the meta-optimization problem.

Our first result shows that this is still true in the case of tuning step size for gradient descent on a simple quadratic objective. In this setting, we show that there is a unique local and global minimizer for the step size, and we also give a simple way to get rid of the gradient explosion/vanishing problem.

Theorem 1 (Informal)**.** *For tuning the step size of gradient descent on a quadratic objective, if the meta-objective is the loss of the last iteration, then the meta-gradient can explode/vanish. If the meta-objective is the* $\log$ *of the loss of the last iteration, then the meta-gradient is polynomially bounded. Further, doing meta-gradient descent with a step size of* $1/\sqrt{k}$ *(where* k *is the number of meta-gradient steps) provably converges to the optimal step size.*

Surprisingly, even though taking the $\log$ of the objective solves the gradient explosion/vanishing problem, one cannot simply implement such an algorithm using auto-differentiation tools such as those used in TensorFlow (Abadi et al., 2016). The reason is that even though the meta-gradient is polynomially bounded, if we compute the meta-gradient using the standard back-propagation algorithm, the meta-gradient will be the ratio of two exponentially large/small numbers, which causes numerical issues. Detailed discussion for the first result appears in Section 3.

Another challenge is about the generalization performance of the learned optimizer. If one just tries to optimize the performance of the learned optimizer on the training set (we refer to this as the train-by-train approach), then the learned optimizer might overfit. Metz et al. (2019) proposed to use a train-by-validation approach instead, where the meta-objective is defined to be the performance of the learned optimizer on a separate validation set.

Our second result considers a simple least squares setting where $y = \langle w^*, x\rangle + \xi$ and $\xi \sim \mathcal{N}(0, \sigma^2)$. We show that when the number of samples is small and the noise is large, it is important to use train-by-validation; while when the number of samples is much larger train-by-train can also learn a good optimizer.

Theorem 2 (Informal)**.** *For a simple least squares problem in* d *dimensions, if the number of samples* n *is a constant fraction of* d *(e.g.,* $d/2$*), and the samples have large noise, then the train-by-train approach performs much worse than train-by-validation. On the other hand, when number of samples* n *is large, train-by-train can get close to error* $d\sigma^2/n$*, which is optimal.*

We discuss the details in Section 4. In Section 5 we show that such observations also hold empirically for more complicated learned optimizers – for an optimizer parametrized by neural network, the generalization performance of train-by-validation is better when there is more noise or when there are fewer training data.

1.2 Related work

Learning-to-learn for supervised learning The idea of using a neural network to parametrize an optimizer started in Andrychowicz et al. (2016), which used an LSTM to directly learn the update rule. Before that, the idea of using optimization to tune parameters for optimzers also appeared in Maclaurin et al. (2015). Later, Li and Malik (2016); Bello et al. (2017) applied techniques from reinforcement learning to learn an optimizer. Wichrowska et al. (2017) used a hierarchical RNN as the optimizer. Metz et al. (2019) adopted a small MLP as the optimizer and used dynamic weighting of two gradient estimators to stabilize and speedup the meta-training process.

Learning-to-learn in other settings Ravi and Larochelle (2016) used LSTM as a meta-learner to learn the update rule for training neural networks in the few-shot learning setting, Wang et al. (2016) learned an RL algorithm by another meta-learning RL algorithm, and Duan et al. (2016) learned a general-purpose RNN that can adapt to different RL tasks.

Gradient-based meta-learning Finn et al. (2017) proposed Model-Agnostic Meta-Learning (MAML) where they parameterize the update rule (optimizer) for network parameters and learn a shared initialization for the optimizer using the tasks sampled from some distribution. Subsequent works generalized or improved MAML, e.g., Rusu et al. (2018) learned a low-dimensional latent representation for gradient-based meta-learning, and Li et al. (2017) generalized MAML and enabled the concurrent learning of learning rate and update direction.

Learning assisted algorithms design Similar ideas can also be extended to applications of learning in algorithms design, where one tries to develop a meta-algorithm selecting an algorithm from a family of parametrized algorithms. For theoretical guarantees on these meta-algorithms, Gupta and Roughgarden (2017) first established this framework which models the algorithm-selection process as a statistical learning problem. In particular, their framework can bound the number of tasks it takes to tune a step size for gradient descent. However, they didn't consider the meta-optimization problem. Based on Gupta and Roughgarden (2017), people have developed and analyzed the meta-algorithms for partitioning and clustering (Balcan et al., 2016), tree search (Balcan et al., 2018a), pruning (Alabi et al., 2019), machanism design (Balcan et al., 2018c), ridge regression (Denevi et al., 2018), stochastic gradient descent (Denevi et al., 2019), and private optimization (Balcan et al., 2018b).

Tuning step size/step size schedule for SGD Shamir and Zhang (2013) showed that SGD with polynomial step size scheduling can almost match the minimax rate in convex non-smooth settings, which was later tightened by Harvey et al. (2018) for standard step size scheduling. Assuming that the horizon T is known to the algorithm, the information-theoretically optimal bound in convex non-smooth setting was later achieved by Jain et al. (2019) which used another step size schedule, and Ge et al. (2019) showed that exponentially decaying step size scheduling can achieve near optimal rate for least squares regression.

2 Preliminaries

In this section, we first introduce some notations, then formulate the learning-to-learn framework.

2.1 Notations

For any integer n, we use $[n]$ to denote $\{1, 2, \cdots, n\}$. We use $\|\cdot\|$ to denote the ℓ_2 norm for a vector and the spectral norm for a matrix. We use $\langle \cdot, \cdot \rangle$ to denote the inner product of two vectors. For a symmetric matrix $A \in \mathbb{R}^{d\times d}$, we denote its eigenvalues as $\lambda_1(A) \geq \cdots \geq \lambda_d(A)$. We denote the d-dimensional identity matrix as I_d. We also denote the identity matrix simply as I when the dimension is clear from the context. We use $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ to hide constant factor dependencies. We use $\text{poly}(\cdot)$ to represent a polynomial on the relevant parameters with constant degree.

2.2 Learning-to-learn framework

We consider the learning-to-learn approach applied to training a distribution of learning tasks. Each task is specified by a tuple $(\mathcal{D}, S_{\text{train}}, S_{\text{valid}}, \ell)$. Here $\mathcal{D}$ is a distribution of samples in $X \times Y$, where X is the domain for the sample and Y is the domain for the label/value. The sets S_{train} and S_{valid} are samples generated independently from $\mathcal{D}$, which serve as the training and validation set (the validation set is optional). The learning task looks to find a parameter $w \in W$ that minimizes the loss function $\ell(w, x, y) : W \times X \times Y \to \mathbb{R}$, which gives the loss of the parameter w for sample (x, y). The training loss for this task is $\hat{f}(w) := \frac{1}{|S_{\text{train}}|} \sum_{(x,y)\in S_{\text{train}}} \ell(w, x, y)$, while the population loss is $f(w) := \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(w, x, y)]$.

The goal of inner-optimization is to minimize the population loss $f(w)$. For the learned optimizer, we consider it as an update rule $u(\cdot)$ on weight w. The update rule is a parameterized function that maps the weight at step τ and its history to the step $\tau + 1 : w_{\tau+1} = u(w_\tau, \nabla\hat{f}(w_\tau), \nabla\hat{f}(w_{\tau-1}), \cdots; \theta)$. In most parts of this paper, we consider the update rule u as gradient descent mapping with step size as the trainable parameter (here $\theta = \eta$ which is the step size for gradient descent). That is, $u_\eta(w) = w - \eta\nabla\hat{f}(w)$ for gradient descent and $u_\eta(w) = w - \eta\nabla_w\ell(w, x, y)$ for stochastic gradient descent where (x, y) is a sample randomly chosen from the training set S_{train}.

In the outer (meta) level, we consider a distribution $\mathcal{T}$ of tasks. For each task $P \sim \mathcal{T}$, we can define a meta-loss function $\Delta(\theta, P)$. The meta-loss function measures the performance of the optimizer on this learning task. The meta objective, for example, can be chosen as the target training loss $\hat{f}$ at the last iteration (this is the train-by-train approach), or the loss on the validation set (train-by-validation).

The training loss for the meta-level is the average of the meta-loss across m different specific tasks $P_1, P_2, ..., P_m$, that is, $\hat{F}(\theta) = \frac{1}{m}\sum_{i=1}^{m} \Delta(\theta, P_k)$. The population loss for the meta-level is the expectation over all the possible specific tasks $F(\theta) = \mathbb{E}_{P\sim\mathcal{T}}[\Delta(\theta, P)]$.

In order to train an optimizer by gradient descent, we need to compute the gradient of meta-objective $\hat{F}$ in terms of meta parameters θ. The meta parameter is updated once after applying the optimizer on the inner objective t times to generate the trajectory $w_0, w_1, ..., w_t$. The meta-gradient is then computed by unrolling the optimization process and back-propagating through the t applications of the optimizer. As we will see later, this unroll procedure is costly and can introduce meta-gradient explosion/vanishing problems.

3 Alleviating gradient explosion/vanishing problem for quadratic objective

First we consider the meta-gradient explosion/vanishing problem. More precisely, we say the meta-gradient explodes/vanishes if it is exponentially large/small with respect to the number of steps t of the inner-optimizer.

In this section, we consider a very simple instance of the learning-to-learn approach, where the distribution $\mathcal{T}$ only contains a single task P, and the task also just defines a single loss function f[1]. Therefore, in this section $\hat{F}(\eta) = F(\eta) = \Delta(\eta, P)$. We will simplify notation and only use $\hat{F}(\eta)$.

The inner task P is a simple quadratic problem, where the starting point is fixed at w_0, and the loss function is $f(w) = \frac{1}{2}w^\top H w$ for some fixed positive definite matrix H. Without loss of generality, assume w_0 has unit ℓ_2 norm. Suppose the eigenvalue decomposition of H is $\sum_{i=1}^{d} \lambda_i u_i u_i^\top$. Throughout this section we assume $L = \lambda_1(H)$ and $\alpha = \lambda_d(H)$ are the largest and smallest eigenvalues of H with $L > \alpha$. For each $i \in [d]$, let c_i be $\langle w_0, u_i\rangle$ and let $c_{\min} = \min(|c_1|, |c_d|)$. We assume $c_{\min} > 0$ for simplicity. If w_0 is randomly and uniformly sampled from the unit sphere, with 0.99 probability $c_{\min}$ is $\Theta(1/\sqrt{d})$. Let $\{w_{\tau,\eta}\}$ be the GD sequence running on $f(w)$ starting from w_0 with step size η. We consider several ways of defining meta-objective, including using the loss of the last point directly, or using the $\log$ of this value. We first show that although choosing $\hat{F}(\eta) = f(w_{t,\eta})$ does not have any bad local optimal solution, it has the gradient explosion/vanishing problem.

Theorem 3. *Let the meta objective be $\hat{F}(\eta) = f(w_{t,\eta}) = \frac{1}{2}w_{t,\eta}^\top H w_{t,\eta}$. We know $\hat{F}(\eta)$ is a strictly convex function in η with an unique minimizer. However, for any step size $\eta < 2/L$, $|\hat{F}'(\eta)| \le t\sum_{i=1}^{d} c_i^2\lambda_i^2|1-\eta\lambda_i|^{2t-1}$; for any step size $\eta > 2/L$, $|\hat{F}'(\eta)| \ge c_1^2L^2t(\eta L-1)^{2t-1} - L^2t$.*

Note that in Theorem 3, when $\eta < 2/L$, $|\hat{F}'(\eta)|$ is exponentially small because $|1-\eta\lambda_i| < 1$ for all $i \in [d]$; when $\eta > 2/L$, $|\hat{F}'(\eta)|$ is exponentially large because $\eta L - 1 > 1$. Intuitively, gradient explosion/vanishing happens because the meta-loss function becomes too small or too large. A natural idea to fix the problem is to take the $\log$ of the meta-loss function to reduce its range. We show that this indeed works. More precisely, if we choose $\hat{F}(\eta) = \frac{1}{t}\log f(w_{t,\eta})$, then we have

Theorem 4. *Let the meta objective be $\hat{F}(\eta) = \frac{1}{t}\log f(w_{t,\eta})$. We know $\hat{F}(\eta)$ has a unique minimizer η^* and $\hat{F}'(\eta) = O\left(\frac{L^3}{c_{\min}^2\alpha(L-\alpha)}\right)$ for all $\eta \ge 0$. Let $\{\eta_k\}$ be the GD sequence running on $\hat{F}$ with meta step size $\mu_k = 1/\sqrt{k}$. Suppose the starting step size $\eta_0 \le M$. Given any $1/L > \epsilon > 0$, there exists $k' = \frac{M^6}{\epsilon^2}poly(\frac{1}{c_{\min}}, L, \frac{1}{\alpha}, \frac{1}{L-\alpha})$ such that for all $k \ge k'$, $|\eta_k - \eta^*| \le \epsilon$.*

For convenience, in the above algorithmic result, we reset η to zero once η goes negative. Note that although we show the gradient is bounded and there is a unique optimizer, the problem of optimizing η is still not convex because the meta-gradient is not monotone. We use ideas from quasi-convex optimization to show that meta-gradient descent can find the unique optimal step size for this problem.

[1] In the notation of Section 2, one can think that $\mathcal{D}$ contains a single point $(0, 0)$ and the loss function $f(w) = \ell(w, 0, 0)$.

Surprisingly, even though we showed that the meta-gradient is bounded, it cannot be effectively computed by doing back-propagation due to numerical issues. More precisely:

Corollary 1. *If we choose the meta-objective as $\hat{F}(\eta) = \frac{1}{t}\log f(w_{t,\eta})$, when computing the meta-gradient using back-propagation, there are intermediate results that are exponentially large/small in number of inner-steps t.*

Indeed, in Section 5 we empirically verify that standard auto-differentiation tools can still fail in this setting. This suggests that one should be more careful about using standard back-propagation in the learning-to-learn approach. The proofs of the results in this section are deferred into Appendix A.

4 Train-by-train vs. train-by-validation

Next we consider the generalization ability of simple optimizers. In this section we consider a simple family of least squares problems. Let $\mathcal{T}$ be a distribution of tasks where every task $(\mathcal{D}(w^*), S_{\text{train}}, S_{\text{valid}}, \ell)$ is determined by a parameter $w^* \in \mathbb{R}^d$ which is chosen uniformly at random on the unit sphere. For each individual task, $(x, y) \sim \mathcal{D}(w^*)$ is generated by first choosing $x \sim \mathcal{N}(0, I_d)$ and then computing $y = \langle w^*, x\rangle + \xi$ where $\xi \sim \mathcal{N}(0, \sigma^2)$ with $\sigma \geq 1$. The loss function $\ell(w, x, y)$ is just the squared loss $\ell(w, x, y) = \frac{1}{2}(y - \langle w, x\rangle)^2$. That is, the tasks are just standard least-squares problems with ground-truth equal to w^* and noise level σ^2.

For the meta-loss function, we consider two different settings. In the train-by-train setting, the training set S_{train} contains n independent samples, and the meta-loss function is chosen to be the training loss. That is, in each task P, we first choose w^* uniformly at random, then generate $(x_1, y_1), ..., (x_n, y_n)$ as the training set S_{train}. The meta-loss function $\Delta_{TbT(n)}(\eta, P)$ is defined to be

$$\Delta_{TbT(n)}(\eta, P) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \langle w_{t,\eta}, x_i\rangle)^2.$$

Here $w_{t,\eta}$ is the result of running t iterations of gradient descent starting from point 0 with step size η. Note we truncate a sequence and declare the meta loss is high once the wight norm exceeds certain threshold. Specifically, if at the τ-th step, $\|w_{\tau,\eta}\| \geq 40\sigma$, we freeze the training on this task and set $w^{(k)}_{\tau',\eta} = 40\sigma u$ for all $\tau \leq \tau' \leq t$, for some arbitrary vector u with unit norm. As before, the empirical meta objective in train-by-train setting is the average of the meta-loss across m different specific tasks $P_1, P_2, ..., P_m$, that is,

$$\hat{F}_{TbT(n)}(\eta) = \frac{1}{m}\sum_{k=1}^{m}\Delta_{TbT(n)}(\eta, P_k). \tag{1}$$

In the train-by-validation setting, the specific tasks are generated by sampling n_1 training samples and n_2 validation samples for each task, and the meta-loss function is chosen to be the validation loss. That is, in each specific task P, we first choose w^* uniformly at random, then generate $(x_1, y_1), ..., (x_{n_1}, y_{n_1})$ as the training set S_{train} and $(x'_1, y'_1), ..., (x'_{n_2}, y'_{n_2})$ as the validation set S_{valid}. The meta-loss function $\Delta_{TbV(n_1,n_2)}(\eta, P)$ is defined to be

$$\Delta_{TbV(n_1,n_2)}(\eta, P) = \frac{1}{2n_2}\sum_{i=1}^{n_2}(y'_i - \langle w_{t,\eta}, x'_i\rangle)^2.$$

Here again $w_{t,\eta}$ is the result of running t iterations of the gradient descent on the training set starting from point 0, and we use the same truncation as before. The empirical meta objective is defined as

$$\hat{F}_{TbV(n_1,n_2)}(\eta) = \frac{1}{m}\sum_{k=1}^{m}\Delta_{TbV(n_1,n_2)}(\eta, P_k), \tag{2}$$

where each P_k is independently sampled according to the described procedure.

We first show that when the number of samples is small (in particular $n < d$) and the noise is a large enough constant, train-by-train can be much worse than train-by-validation, even when $n_1 + n_2 = n$ (the total number of samples used in train-by-validation is the same as train-by-train)

Theorem 5. *Let $\hat{F}_{TbT(n)}(\eta)$ and $\hat{F}_{TbV(n_1,n_2)}(\eta)$ be as defined in Equation* (1) *and Equation* (2) *respectively. Assume $n, n_1, n_2 \in [d/4, 3d/4]$. Assume noise level σ is a large constant c_1. Assume unroll length $t \geq c_2$, number of training tasks $m \geq c_3 \log(mt)$ and dimension $d \geq c_4 \log(mt)$ for certain constants c_2, c_3, c_4. With probability at least* 0.99 *in the sampling of training tasks, we have*

$$\eta^*_{train} = \Theta(1) \text{ and } \mathbb{E}\left\|w_{t,\eta^*_{train}} - w^*\right\|^2 = \Omega(1)\sigma^2,$$

for all $\eta^*_{train} \in \arg\min_{\eta \geq 0} \hat{F}_{TbT(n)}(\eta)$;

$$\eta^*_{valid} = \Theta(1/t) \text{ and } \mathbb{E}\left\|w_{t,\eta^*_{valid}} - w^*\right\|^2 = \|w^*\|^2 - \Omega(1)$$

for all $\eta^*_{valid} \in \arg\min_{\eta \geq 0} \hat{F}_{TbV(n_1,n_2)}(\eta)$. In both equations the expectation is taken over new tasks.

In the lower bound of $\mathbb{E}\left\|w_{t,\eta^*_{\text{train}}} - w^*\right\|^2$, $\Omega(1)$ hides no dependency on σ. Note that in this case, the number of samples n is smaller than d, so the least square problem is under-determined and the optimal training loss would go to 0 (there is always a way to simultaneously satisfy all n equations). This is exactly what train-by-train would do – it will choose a large constant learning rate which guarantees the optimizer converges exponentially to the empirical risk minimizer (ERM). However, when the noise is large making the training loss go to 0 will overfit to the noise and hurt the generalization performance. Train-by-validation on the other hand will choose a smaller learning rate which allows it to leverage the information in the training samples without overfitting to noise. Theorem 5 is proved in Appendix B. We also prove similar results for SGD in Appendix D

We emphasize that neural networks are often over-parameterized, which corresponds to the case when $d > n$. Indeed Liu and Belkin (2018) showed that variants of stochastic gradient descent can converge to the empirical risk minimizer with exponential rate in this case. Therefore in order to train neural networks, it is better to use train-by-validation. On the other hand, we show when the number of samples is large ($n \gg d$), train-by-train can also perform well.

Theorem 6. *Let $\hat{F}_{TbT(n)}(\eta)$ be as defined in Equation 1. Assume noise level is a constant c_1. Given any $1 > \epsilon > 0$, assume training set size $n \geq \frac{cd}{\epsilon^2}\log(\frac{nm}{\epsilon d})$, unroll length $t \geq c_2 \log(\frac{n}{\epsilon d})$, number of training tasks $m \geq \frac{c_3 n^2}{\epsilon^4 d^2}\log(\frac{tnm}{\epsilon d})$ and dimension $d \geq c_4$ for certain constants c, c_2, c_3, c_4. With probability at least* 0.99 *in the sampling of training tasks, we have*

$$\mathbb{E}\left\|w_{t,\eta^*_{train}} - w^*\right\|^2 \leq (1+\epsilon)\frac{d\sigma^2}{n},$$

for all $\eta^*_{train} \in \arg\min_{\eta \geq 0} \hat{F}_{TbT(n)}(\eta)$, where the expectation is taken over new tasks.

Therefore if the learning-to-learn approach is applied to a traditional optimization problem that is not over-parameterized, it is OK to use train-by-train. In this case, the empirical risk minimizer (ERM) already has good generalization performance, and train-by-train optimizes the convergence towards the ERM. We defer the proof of Theorem 6 into Appendix C.

5 Experiments

Optimizing step size for quadratic objective We first validate the results in Section 3. We fixed a 20-dimensional quadratic objective as the inner problem and vary the number of inner steps t and initial value η_0. We compute the meta-gradient directly using a formula which we derive in supplementary material. We use the algorithm suggested in Theorem 4, except we choose the meta-step size to be $1/(100\sqrt{k})$ as the constants in the Theorem were not optimized.

An example training curve of η for $t = 80$ and $\eta_0 = 0.1$ is shown in Figure 1, and we can see that η converges quickly within 300 steps. Similar convergence also holds for larger t or much larger initial η_0. Figure 2 shows that as observed in Metz et al. (2019), the optimal step size depends on the number of inner-training steps.

In contrast, we also implemented the meta-training with Tensorflow, where the code was adapted from the previous work of Wichrowska et al. (2017). Experiments show that in many settings (especially with large t and large η_0) the implementation does not converge.

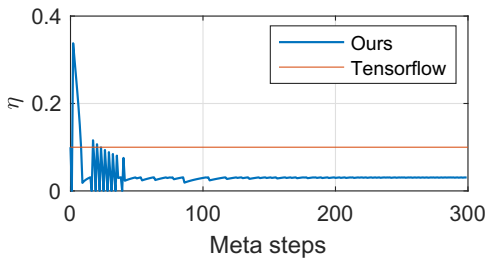

Figure 1: Training η ($t = 80$, $\eta_0 = 0.1$)

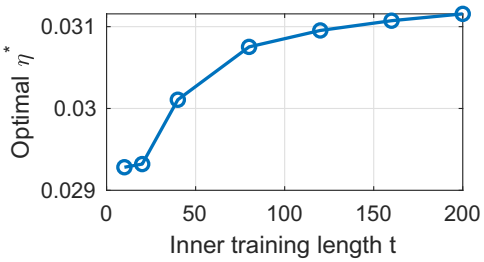

Figure 2: Optimal η^* for different t

Train-by-train vs. train-by-validation, synthetic data Here we validate our theoretical results in Section 4 using the least-squares model defined there. In all experiments we fix the input dimension d to be 1000.

In the first experiment, we fix the size of the data ($n = 500$ for train-by-train, $n_1 = n_2 = 250$ for train-by-validation). Under different noise levels, we find the optimal η^* by a grid search on its meta-objective for train-by-train and train-by-validation settings respectively. We then use the optimal η^* found in each of these two settings to test on 10 new least-squares problem. The mean RMSE, as well as its range over the 10 test cases, are shown in Figure 3. We can see that for all of these cases, the train-by-train model overfits easily, while the train-by-validation model performs much better and does not overfit. Also, when the noise becomes larger, the difference between these two settings becomes more significant.

Figure 3: Training and testing RMSE for different σ values (500 samples)

In the next experiment, we fix $\sigma = 1$ and change the sample size. For train-by-validation, we always split the samples evenly into training and validation set. The results are shown in Figure 4. We can see that the gap between these two settings is decreasing as we use more data, as expected by Theorem 6.

Figure 4: Training and testing RMSE for different samples sizes ($\sigma = 1$)

Train-by-train vs. train-by-validation, MLP optimizer on MNIST Finally we consider a more complicated multi-layer perceptron (MLP) optimizer on MNIST data set. We use the same MLP optimizer as in Metz et al. (2019), details of this optimizer is discussed in supplementary material.

As the inner problem, we use a two-layer fully-connected network of 100 and 20 hidden units with ReLU activations. The inner objective is the classic 10-class cross entropy loss, and we use mini-batches of 32 samples at inner training.

To see whether the comparison between train-by-train and train-by-validation behaves similarly to our theoretical results, we consider different number of samples and different levels of label noise. First, consider optimizing the MNIST dataset with small number samples. In this case, the train-by-train setting uses 1,000 samples (denoted as "TbT1000"), and we use another 1,000 samples as the validation set for the train-by-validation case (denoted as "TbV1000+1000"). To be fair to train-by-train we also consider TbT2000 where the train-by-train algorithm has access to 2000 data points. Figure 5 shows the results – all the models have training accuracy close to 1, but both TbT1000 and TbT2000 overfits the data significantly, whereas TbV1000+1000 performs well.

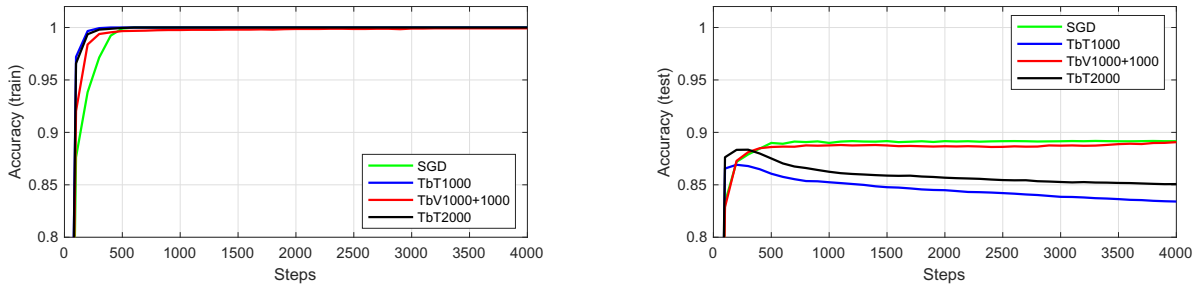

Figure 5: Training and testing accuracy for different models (1000 samples, no noise)

To show that when the noise is higher, the advantage of train-by-validation increases, we keep the same sample size and consider a "noisier" version of MNIST, where we randomly change the label of a sample with probability 0.2 (the new label is chosen uniformly at random, including the original label). The results are shown in Figure 6. We can see that both train-by-train models, as well as SGD, overfit easily with training accuracy close to 1 and their test performances are low. The train-by-validation model performs much better.

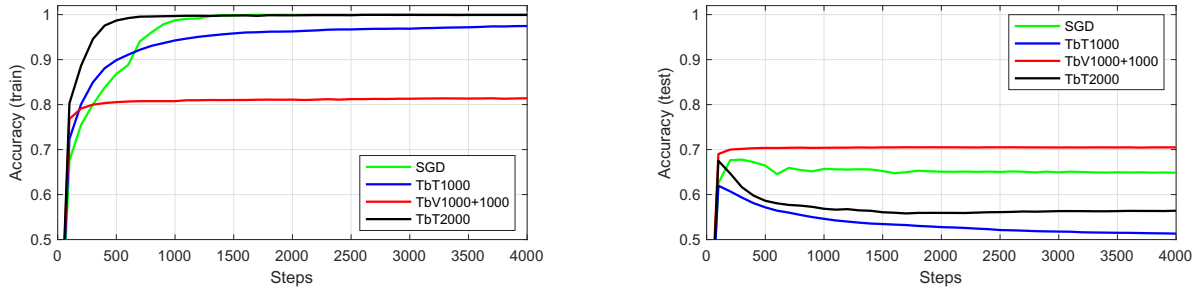

Figure 6: Training and testing accuracy for different models (1000 samples, 20% noise)

Finally we run experiments on the complete MNIST data set (without label noise). For the train-by-validation setting, we split the data set to 50,000 training samples and 10,000 validation samples. As shown in Figure 7, in this case train-by-train and train-by-validation performs similarly (in fact both are slightly weaker than the tuned SGD baseline). This shows that when the sample size is sufficiently large, train-by-train can get comparable results as train-by-validation.

Acknowledgements

Rong Ge, Xiang Wang and Chenwei Wu are supported in part by NSF Award CCF-1704656, CCF-1845171 (CAREER), CCF-1934964 (Tripods), a Sloan Research Fellowship, and a Google Faculty Research Award. Part of the

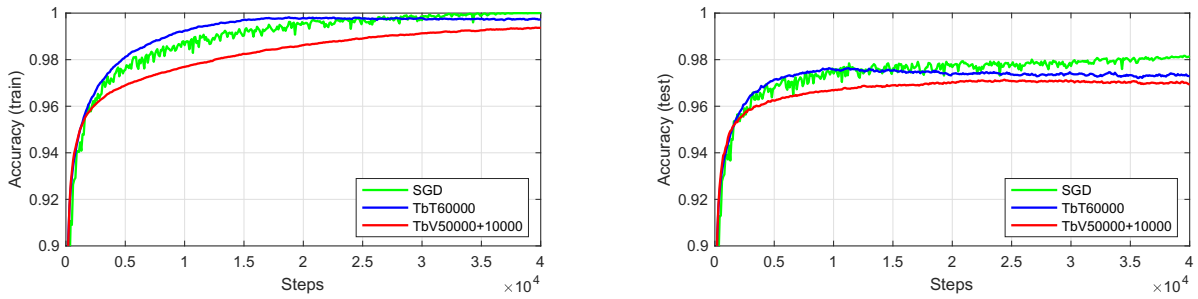

Figure 7: Training and testing accuracy for different models (all samples, no noise)

work was done when Rong Ge and Xiang Wang were visiting Instituted for Advanced Studies for "Special Year on Optimization, Statistics, and Theoretical Machine Learning" program. We acknowledge the valuable early discussions with Yatharth Dubey.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

Alabi, D., Kalai, A. T., Ligett, K., Musco, C., Tzamos, C., and Vitercik, E. (2019). Learning to prune: Speeding up repeated computations. *arXiv preprint arXiv:1904.11875*.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989.

Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. (2018a). Learning to branch. *arXiv preprint arXiv:1803.10150*.

Balcan, M.-F., Dick, T., and Vitercik, E. (2018b). Dispersion for data-driven algorithm design, online learning, and private optimization. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 603–614. IEEE.

Balcan, M.-F., Nagarajan, V., Vitercik, E., and White, C. (2016). Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. *arXiv preprint arXiv:1611.04535*.

Balcan, M.-F., Sandholm, T., and Vitercik, E. (2018c). A general theory of sample complexity for multi-item profit maximization. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 173–174.

Bello, I., Zoph, B., Vasudevan, V., and Le, Q. V. (2017). Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 459–468. JMLR. org.

Denevi, G., Ciliberto, C., Grazzi, R., and Pontil, M. (2019). Learning-to-learn stochastic gradient descent with biased regularization. *arXiv preprint arXiv:1903.10399*.

Denevi, G., Ciliberto, C., Stamos, D., and Pontil, M. (2018). Incremental learning-to-learn with statistical guarantees. *arXiv preprint arXiv:1803.08089*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Ge, R., Kakade, S. M., Kidambi, R., and Netrapalli, P. (2019). The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares. In *Advances in Neural Information Processing Systems*, pages 14951–14962.

Gupta, R. and Roughgarden, T. (2017). A pac approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017.

Harvey, N. J., Liaw, C., Plan, Y., and Randhawa, S. (2018). Tight analyses for non-smooth stochastic gradient descent. *arXiv preprint arXiv:1812.05217*.

Jain, P., Nagaraj, D., and Netrapalli, P. (2019). Making the last iterate of sgd information theoretically optimal. *arXiv preprint arXiv:1904.12443*.

Johnson, W. B. and Lindenstrauss, J. (1984). Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1.

Li, K. and Malik, J. (2016). Learning to optimize. *arXiv preprint arXiv:1606.01885*.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

Liu, C. and Belkin, M. (2018). Accelerating sgd with momentum for over-parameterized learning. *arXiv preprint arXiv:1810.13395*.

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

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. (2019). Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565.

Ravi, S. and Larochelle, H. (2016). Optimization as a model for few-shot learning.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2018). Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*.

Shamir, O. and Zhang, T. (2013). Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International conference on machine learning*, pages 71–79.

Vershynin, R. (2010). Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*.

Vershynin, R. (2018). *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. (2017). Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3751–3760. JMLR. org.