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

Xiang Wang [1] **Shuai Yuan** [1] **Chenwei Wu** [1] **Rong Ge** [1]

Abstract

Choosing the right parameters for optimization algorithms is often the key to their success in practice. Solving this problem using a learning-to-learn approach—using meta-gradient descent on a meta-objective based on the trajectory that the optimizer generates—was recently shown to be effective. However, the meta-optimization problem is difficult. In particular, the meta-gradient can often explode/vanish, and the learned optimizer may not have good generalization performance if the meta-objective is not chosen carefully. In this paper we give meta-optimization guarantees for the learning-to-learn approach on a simple problem of tuning the step size for quadratic loss. Our results show that the naïve objective suffers from meta-gradient explosion/vanishing problem. Although there is a way to design the meta-objective so that the meta-gradient remains polynomially bounded, computing the meta-gradient directly using backpropagation leads to numerical issues. We also characterize when it is necessary to compute the meta-objective on a separate validation set to ensure the generalization performance of the learned optimizer. 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 hyperparameters is important for training a deep neural network. Even for simple algorithms like gradient descent and stochastic gradient descent, choosing a good step size can be important to the convergence speed and generalization performance. Empirically, the parameters are often chosen based on past experiences or grid search. Recently, Maclaurin et al. (2015) considered the idea of tuning these parameters by optimization—that is, consider a meta-optimization problem where the goal is to find the best parameters for an optimizer. A series of works (e.g., Andrychowicz et al. (2016); Wichrowska et al. (2017)) extended such ideas and parametrized the set of optimizers by neural networks.

Although this approach has shown empirical success, there are very few theoretical guarantees for learned optimizers. Gupta & Roughgarden (2017) gave sample complexity bounds for tuning the step size, but they did not address how one can find the learned optimizer efficiently. In practice, the meta-optimization problem is often solved by meta-gradient descent—define a meta-objective function based on the trajectory that the optimizer generates, and then compute the meta-gradient using back-propagation (Franceschi et al., 2017). The optimization for meta-parameters is usually a nonconvex problem, therefore it is unclear why simple meta-gradient descent would find an optimal solution.

In this paper we consider using learning-to-learn approach to tune the step size of standard gradient descent/stochastic gradient descent algorithm. 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.

1.1. Our Results

In this paper we focus on two basic questions on learning-to-learn for gradient descent optimizer. First, will the meta-gradient explode/vanish and is there a way to fix the problem? Second, how could we guarantee that the learned optimizer has good generalization properties?

Our first result shows that meta-gradient can explode/vanish even for tuning the 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 version of Theorem 3 and Theorem 4)**.** *For tuning the step size of gradient descent on a quadratic objective, if the meta-objective is the loss of the last iteration,*

[1]Department of Computer Science, Duke University, Durham, North Carolina, USA. Correspondence to: Xiang Wang <xwang@cs.duke.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

then the meta-gradient will 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 meta step size of* $1/\sqrt{k}$ *(where* k *is the number of meta-gradient steps) provably converges to the optimal step size for the inner-optimizer.*

Surprisingly, even though taking the $\log$ of the objective solves the meta-gradient explosion/vanishing problem, one cannot simply implement such an algorithm using back-propagation (which is standard in 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, back-propagation algorithm will compute the meta-gradient as the ratio of two exponentially large/small numbers, which causes numerical issues. Detailed discussion for the first result appears in Section 3.

Our second result shows that defining meta-objective on the same training set (later referred to as the "train-by-train" approach) could lead to overfitting; while defining meta-objective on a separate validation set ("train-by-validation", see Metz et al. (2019)) can solve this issue. We consider 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 version of Theorem 5 and Theorem 6)**.** *For a 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 the 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—an optimizer parametrized by a neural network.

1.2. Related Work

Learned optimizer The idea of learning an optimizer has appeared in early works decades ago (Bengio et al., 1990; 1992; Hochreiter et al., 2001). Recently, with the rise of deep learning, researchers started to consider more complex optimizers on more challenging tasks. One line of research views the optimizer as a policy and apply reinforcement learning techniques to train it (Li & Malik, 2016; 2017; Bello et al., 2017). The other line of papers use gradient descent on the meta-objective to update the optimizer parameters (Maclaurin et al., 2015; Andrychowicz et al., 2016; Lv et al., 2017; Wichrowska et al., 2017; Metz et al., 2019).

Mostly relevant to our work, Metz et al. (2019) highlighted several challenges in the meta-optimization for learning-to-learn approach. First, they observed the meta-gradient exploding/vanishing issue and proposed to use a gradient estimator for a variational meta-objective. They also observed that train-by-train approach can overfit the training tasks while train-by-validation generalizes well.

Data-driven algorithm design In data-driven algorithm design, we aim to find an algorithm that works well on a particular distribution of tasks. Gupta & Roughgarden (2017) first modeled this algorithm-selection process as a statistical learning problem. In particular, they analyzed the sample complexity of choosing the step size for gradient descent. But they didn't consider the meta-optimization problem. They also restricted the step size into a small range so that gradient descent is guaranteed to converge on every task. We don't have such a restriction and allow the meta-learning to choose a more aggressive step size.

Following the work by Gupta & Roughgarden (2017), data-driven algorithms have been studied in many problems, including partitioning and clustering (Balcan et al., 2016a), tree search (Balcan et al., 2018a), pruning (Alabi et al., 2019) and machanism design (Morgenstern & Roughgarden, 2015; 2016; Balcan et al., 2016b; 2018b).

Step size schedule for GD/SGD Shamir & 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 number of training steps 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.

A closely related paper that appeared later than our work also studied the comparison between train-by-train and train-by-validation (Bai et al., 2020). They considered a very different meta-learning problem, where the goal is to find the best common initialization for adapting to a linear predictor on each task. They proved train-by-train can work better than train-by-validation in the noiseless setting.

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 or simply as I when the dimension is clear. 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, $u(w; \eta) = w - \eta\nabla\hat{f}(w)$ for gradient descent and $u(w; \eta) = w - \eta\nabla_w \ell(w, x, y)$ 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 (train-by-train), 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.

3. Alleviating Gradient Explosion/Vanishing Problems

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 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 with unit norm, and the loss function is $f(w) = \frac{1}{2} w^\top H w$ for some fixed positive definite matrix $H \in \mathbb{R}^{d \times d}$.

Let $\{w_{\tau,\eta}\}_{\tau=0}^t$ be the GD sequence running on $f(w)$ starting from w_0 with step size η. We consider two ways of defining meta-objective: 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 meta-gradient explosion/vanishing problem. We use $\hat{F}'(\eta)$ to denote the derivative of $\hat{F}$ in η.

In the analysis, we use eigen-decomposition to transform H into a diagonal matrix. We introduce related notations here: suppose the eigenvalue decomposition of H is $\sum_{i=1}^d \lambda_i u_i u_i^\top$. We denote $L := \lambda_1(H)$ and $\alpha := \lambda_d(H)$ as the largest and smallest eigenvalues of H. For each $i \in [d]$, let c_i be $\langle w_0, u_i \rangle$ and let $c_{\min}$ be $\min(|c_1|, |c_d|)$. We assume $c_{\min} > 0$ and $L > \alpha$ for simplicity[2].

Theorem 3. *Let the meta-objective be $\hat{F}(\eta) = f(w_{t,\eta})$, we know $\hat{F}(\eta)$ is a strictly convex function in η with an unique minimizer. However, for any step size $0 < \eta < 2/L$,*

$$|\hat{F}'(\eta)| \leq tL^2 \max(|1 - \eta\alpha|^{2t-1}, |1 - \eta L|^{2t-1});$$

for any step size $\eta > 2/L$,

$$|\hat{F}'(\eta)| \geq c_1^2 L^2 t(\eta L - 1)^{2t-1} - L^2 t.$$

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

[2] If w_0 is uniformly sampled from the unit sphere, with high probability $c_{\min}$ is at least $\Omega(1/\sqrt{d})$; if H is $XX^\top$ with $X \in \mathbb{R}^{d \times 2d}$ as a random Gaussian matrix, with constant probability, both α and $L - \alpha$ are at least $\Omega(d)$.

Note that in Theorem 3, when $0 < \eta < 2/L$, $|\hat{F}'(\eta)|$ is exponentially small because $|1-\eta\alpha|, |1-\eta L| < 1$; when $\eta > 2/L$, $|\hat{F}'(\eta)|$ is exponentially large because $\eta L - 1 > 1$. The strict convexity of $\hat{F}(\eta)$ is proved by showing the second order derivative of $\hat{F}(\eta)$ is positive; the upper and lower bounds of $\hat{F}'(\eta)$ follows from direct calculation.

Intuitively, gradient explosion/vanishing happens because the meta-objective becomes too small or too large. A natural idea to fix the problem is to take the $\log$ of the meta-objective to reduce its range. If we choose $\hat{F}(\eta) = \frac{1}{t}\log f(w_{t,\eta})$, 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 $\eta \geq 0$. 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 $\eta_0 \leq M$. 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 $k \geq k'$, $|\eta_k - \eta^*| \leq \epsilon$.*

For convenience, in the above algorithmic result, we reset η to zero once η goes negative (this corresponds to doing a projected gradient descent on η under constraint $\eta \geq 0$). We give a proof sketch of Theorem 4 in Section 3.1.

Surprisingly, even though we showed that the meta-gradient is well-behaved, 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.*

If we use back-propagation to compute $\hat{F}'(\eta)$, we need to separately compute the numerator and denominator in Eqn. (1), which are exponentially large or small as we showed in Theorem 3. Indeed, in Section 5 we empirically verify that standard auto-differentiation tools can fail in this setting. In contrast, the meta training succeeds if we use the formula derived in Section 3.1 (Eqn. (2)). 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.

3.1. Proof Sketch of Theorem 4

Throughout the proof, we work in the eigenspace of H which reduces the problem to having a diagonal matrix H. The proof goes in three steps:

- Claim 1 shows that the meta-objective $\hat{F}$ has a unique minimizer η^* and the minus meta-gradient always points to the minimizer.
- Claim 2 shows meta-gradient $\hat{F}'(\eta)$ never explodes.
- Claim 3 shows meta-gradient is large when η is far from η^*.

Claim 1. *The meta-objective $\hat{F}$ has only one stationary point that is also its unique minimizer η^*. For any $\eta \in [0, \eta^*)$, $\hat{F}'(\eta) < 0$ and for any $\eta \in (\eta^*, \infty)$, $\hat{F}'(\eta) > 0$.*

The lemma follows from a direct calculation $\hat{F}'(\eta)$:

$$\hat{F}'(\eta) = \frac{-2\sum_{i=1}^d c_i^2\lambda_i^2(1-\eta\lambda_i)^{2t-1}}{\sum_{i=1}^d c_i^2\lambda_i(1-\eta\lambda_i)^{2t}}. \tag{1}$$

Claim 1 is proved by noticing that the denominator in $\hat{F}'(\eta)$ is always positive and the numerator is strictly increasing in η. Next, we show the meta derivative is polynomially upper bounded.

Claim 2. *For any $\eta \in [0, \infty)$, we have $|\hat{F}'(\eta)| \leq \frac{4L^3}{c_{\min}^2\alpha(L-\alpha)}$.*

To prove this claim we observe that the numerator and denominator are both polynomially bounded once we divide them by a common factor, which is $(1-\eta\alpha)^{2t}$ when $\eta \in [0, \frac{2}{\alpha+L}]$. More precisely we have when $\eta \in [0, \frac{2}{\alpha+L}]$

$$\left|\hat{F}'(\eta)\right| = 2\frac{\left|\sum_{i=1}^d \frac{c_i^2\lambda_i^2}{1-\eta\alpha}(\frac{1-\eta\lambda_i}{1-\eta\alpha})^{2t-1}\right|}{c_d^2\alpha + \sum_{i=1}^{d-1} c_i^2\lambda_i(\frac{1-\eta\lambda_i}{1-\eta\alpha})^{2t}} \leq \frac{2\sum_{i=1}^d c_i^2\lambda_i^2}{c_d^2\alpha(1-\eta\alpha)}. \tag{2}$$

This leads to the claimed bounds based on our assumptions. The case when η is large is similar. Finally, we show the meta-gradient is lower bounded if η is away from η^* and is not too large. The proof follows from a similar calculation as above.

Claim 3. *Given $\hat{M} \geq 2/\alpha$ and $1/L > \epsilon > 0$, for any $\eta \in [0, \eta^* - \epsilon] \cup [\eta^* + \epsilon, \hat{M}]$, we have $|F'(\eta)| \geq 2\epsilon c_{\min}^2 \min\left(\frac{\alpha^3}{L}, \frac{1}{\hat{M}^2}\right)$.*

With the above three claims, we are ready to sketch the proof of Theorem 4. Due to Claim 1, we know the minus meta-gradient always points to the minimizer η^*. This alone is not sufficient to prove the convergence result because the iterates might significantly overshoot the minimizer if $|\hat{F}'|$ is too large or the iterates might converge very slowly if $|\hat{F}'|$ is too small. Fortunately, these two problematic cases can be excluded by Claim 2 and Claim 3.

4. Generalization for Trained Optimizer

Next we consider the generalization ability of simple trained 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 $w^* \in \mathbb{R}^d$ that is sampled uniformly at random from 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.

We consider two different ways to define the meta-objective.

Train-by-train: 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 weight norm exceeds certain threshold[3]. We can safely do this because we assume the ground truth weight w^* has unit norm, so if the weight norm is too high, it means the inner training has diverged and the step size is too large.

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,

$$\hat{F}_{TbT(n)}(\eta) = \frac{1}{m}\sum_{k=1}^{m}\Delta_{TbT(n)}(\eta, P_k). \tag{3}$$

Train-by-validation: 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 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

$$\hat{F}_{TbV(n_1,n_2)}(\eta) = \frac{1}{m}\sum_{k=1}^{m}\Delta_{TbV(n_1,n_2)}(\eta, P_k), \tag{4}$$

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 in 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* (3) *and Equation* (4) *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 Theorem 5, $w_{t,\eta^*_{\text{train}}}$ and $w_{t,\eta^*_{\text{valid}}}$ are the results obtained on the new task and w^* is the ground truth of the new task. If σ is a large enough constant, we know $\mathbb{E}\left\|w_{t,\eta^*_{\text{train}}} - w^*\right\|^2$ is larger than $\mathbb{E}\left\|w_{t,\eta^*_{\text{valid}}} - w^*\right\|^2$ by some constant. The probability 0.99 is an arbitrary number, which can be replaced by any constant smaller than 1.

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)[4]. However, when the noise is large making the training loss go to 0 will overfit to the noise and hurt the generalization performance. In contrast, train-by-validation will choose a smaller learning rate which allows it to leverage the signal in the training samples without overfitting to noise.

We separately give a proof sketch for the train-by-train setting and train-by-validation setting in Section 4.1 and Section 4.2, respectively. The detailed proof of Theorem 5 is

[3]Specifically, if at the τ-th step $\|w_{\tau,\eta}\| \geq 40\sigma$, we freeze the training on this task and set $w_{\tau',\eta} = 40\sigma u$ for all $\tau \leq \tau' \leq t$, for some arbitrary vector u with unit norm. Setting the weight to a large vector is just one way to declare the loss is high.

[4]In an under-determined problem, there are actually multiple ERM solutions. Here, we focus on the unique ERM solution in the span of training data. This is also the solution that GD converges to when the initialization is 0.

deferred to 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 $d > n$. Therefore in order to train neural networks, it is usually 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 (3). 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, train-by-train can work well. 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.

4.1. Proof Sketch for Train-by-train

In this section, we will give a proof sketch for the first half of Theorem 5 (train-by-train with small number of samples). At the end of this section, we will briefly discuss the proof of Theorem 6 (train-by-train with large number of samples). For convenience, we denote $\hat{F}_{TbT}$ as the empirical meta-objective and F_{TbT} as the population meta-objective. We implicit assume the conditions in Theorem 5 hold in the following lemmas.

Our meta-optimization problem works on a distribution of tasks. Since different tasks can have different smoothness condition, it's possible that under the same step size, the inner training converges on some tasks, but diverges on others. One way to avoid this issue is to restrict the step size into a small range under which the inner training converges on all tasks (Gupta & Roughgarden, 2017). But this is too conservative and may lead to suboptimal step size. Instead, we allow any positive step size and truncate the inner training if the weight norm goes too large. This approach resolves the diverging issues and also allow the meta-learning algorithm to choose a more aggressive step size. As we explain later, this brings some technical challenges into our proof.

In order to prove $\mathbb{E}\left\|w_{t,\eta^*_{\text{train}}} - w^*\right\|^2$ is large, we only need to show the population meta-objective $F_{TbT}(\eta^*_{\text{train}})$ is small. This is because $F_{TbT}(\eta^*_{\text{train}})$ measures the distance between $w_{t,\eta^*_{\text{train}}}$ and the ERM solution while ERM solution is far from w^*. Since η^*_{train} minimizes the empirical meta-objective, we know $\hat{F}_{TbT}(\eta^*_{\text{train}})$ is small. Thus we only need to show F_{TbT} and $\hat{F}_{TbT}$ are similar. This is easy to prove for small step sizes when the inner training always converges, but is difficult when the inner training can diverge and gets truncated. To address this problem we break the step size into three intervals separated by $1/L$ and $\tilde{\eta}$ (L is a large constant that bounds the smoothness on all tasks). Intuitively, when $\eta \leq 1/L$ almost all inner training converges and larger step size leads to faster convergence and smaller $\hat{F}_{TbT}$; on the other hand, when $\eta > \tilde{\eta}$, we show $\hat{F}_{TbT}(\eta)$ is always large so the minimizer of $\hat{F}_{TbT}$ cannot be in this region. Therefore, the optimal step size must be in $[1/L, \tilde{\eta}]$. We only need to prove in the interval $[1/L, \tilde{\eta}]$ the empirical meta-objective $\hat{F}_{TbT}$ is close to the population meta-objective F_{TbT}. This proof is still nontrivial since the inner training can still diverge on a small fraction of sampled tasks.

We first show that for $\eta \in [0, 1/L]$, the empirical meta-objective $\hat{F}_{TbT}$ strictly decreases as η increases and $\hat{F}_{TbT}$ is exponentially small in t at step size $1/L$.

Lemma 1. *With probability at least $1 - m\exp(-\Omega(d))$, $\hat{F}_{TbT}(\eta)$ is monotonically decreasing in $[0, 1/L]$ and $\hat{F}_{TbT}(1/L) \leq 2L^2\sigma^2\left(1 - \frac{1}{L^2}\right)^t$.*

Next we show that the minimizer cannot be larger than $\tilde{\eta}$ for suitably chosen $\tilde{\eta}$ (see the precise definition in the appendix). Intuitively, this is because when η is too large the inner-optimizer would diverge on a significant fraction of the sampled tasks.

Lemma 2. *With probability at least $1 - \exp(-\Omega(m))$, $\hat{F}_{TbT}(\eta) \geq \frac{\sigma^2}{10L^8}$ for all $\eta > \tilde{\eta}$.*

By Lemma 1 and Lemma 2, we know when t is large enough, the optimal step size η^*_{train} must lie in $[1/L, \tilde{\eta}]$. We can also show $1/L < \tilde{\eta} < 3/L$, so η^*_{train} is a constant. To relate the empirical loss at η^*_{train} to the population loss, we prove the following uniform convergence result when $\eta \in [1/L, \tilde{\eta}]$.

Lemma 3. *With probability at least $1 - m\exp(-\Omega(d)) - O(t+m)\exp(-\Omega(m))$, $|F_{TbT}(\eta) - \hat{F}_{TbT}(\eta)| \leq \frac{\sigma^2}{L^3}$, for all $\eta \in [1/L, \tilde{\eta}]$.*

The proof of this Lemma involves constructing special ϵ-nets for F_{TbT} and $\hat{F}_{TbT}$ and showing that for each fixed η, $|F_{TbT}(\eta) - \hat{F}_{TbT}(\eta)|$ is small with high probability using concentration inequalities.

Combining the above lemmas, we know the population meta-objective F_{TbT} is small at η^*_{train}, which means $w_{t,\eta^*_{\text{train}}}$ is close to the ERM solution. Since the ERM solution

overfits to the noise in the training samples, we know $\mathbb{E}\left\|w_{t,\eta^*_{\text{train}}} - w^*\right\|$ has to be large.

Train-by-train with large number of samples: The proof of Theorem 6 follows the same strategy as above. We prove that under the optimal step size η^*_{train}, $w_{t,\eta^*_{\text{train}}}$ converges to the ERM solution. But with more samples, the ERM solution w_{ERM} becomes closer to the ground truth w^*. More precisely, we can prove $\mathbb{E}\left\|w_{\text{ERM}} - w^*\right\|^2$ is roughly $\frac{d\sigma^2}{n}$, which leads to the bound in Theorem 6.

4.2. Proof Sketch for Train-by-Validation

In this section, we give a proof sketch for the second half of Theorem 5. We denote $\hat{F}_{TbV}$ as the empirical meta-objective and F_{TbV} as the population meta-objective.

The overall proof strategy is similar as before: we will show the empirical meta-objective is high when the step size is beyond certain threshold, and only prove generalization result for step sizes below this threshold. Under the train-by-validation meta-objective, the optimal step size η^*_{valid} is in order $\Theta(1/t)$. So we will choose a smaller threshold step size to be $1/L$.

When $\eta < 1/L$, we show that the learned signal is linear in ηt while the fitted noise is quadratic in ηt. So there exists certain step size in the order $\Theta(1/t)$ such that our model can leverage the signal in the training set without overfitting the noise. More precisely, we prove the following lemma.

Lemma 4. *There exist $\eta_1, \eta_2, \eta_3 = \Theta(1/t)$ with $\eta_1 < \eta_2 < \eta_3$ such that*

$$F_{TbV}(\eta_2) \le \frac{1}{2}\left\|w^*\right\|^2 - \frac{9}{10}C + \frac{\sigma^2}{2}$$

$$F_{TbV}(\eta) \ge \frac{1}{2}\left\|w^*\right\|^2 - \frac{6}{10}C + \frac{\sigma^2}{2}, \forall \eta \in [0, \eta_1] \cup [\eta_3, 1/L]$$

where C is a positive constant.

We then show whenever η is large, either the gradient descent diverges and the sequence gets truncated or it converges and overfits the noise. In both cases, the meta-objective must be high.

Lemma 5. *With probability at least $1 - \exp(-\Omega(m))$, $\hat{F}_{TbV}(\eta) \ge C'\sigma^2 + \frac{1}{2}\sigma^2$, for all $\eta \ge 1/L$, where C' is a positive constant independent with σ.*

To relate the behavior of F_{TbV} to the behavior of $\hat{F}_{TbV}$, we prove the following uniform convergence result for step sizes in $[0, 1/L]$. The proof is similar as in Lemma 3.

Lemma 6. *With probability at least $1 - O(1/\epsilon)\exp(-\Omega(\epsilon^2 m))$, $|\hat{F}_{TbV}(\eta) - F_{TbV}(\eta)| \le \epsilon$, for all $\eta \in [0, 1/L]$.*

By choosing a small enough ϵ in Lemma 6, we ensure that the behavior of $\hat{F}_{TbV}$ is similar as that of F_{TbV} in Lemma 4.

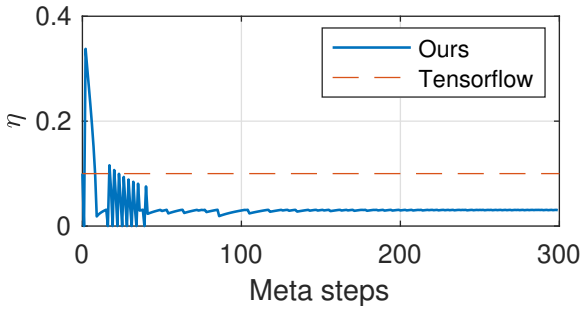


Figure 1. Meta training trajectory for η ($t = 80$, $\eta_0 = 0.1$).

Combing with Lemma 5, we know $\eta^*_{\text{valid}} = \Theta(1/t)$ and $F_{TbV}(\eta^*_{\text{valid}}) \le \frac{1}{2}\left\|w^*\right\|^2 + \frac{1}{2}\sigma^2 - \Omega(1)$. This concludes our proof since $F_{TbV}(\eta) = \frac{1}{2}\mathbb{E}\left\|w_{t,\eta} - w^*\right\|^2 + \frac{1}{2}\sigma^2$.

5. Experiments

In this section, we give experiment results on both synthetic data and realistic data to verify our theory.[5]

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 the formula in Eqn. (2). In this way, we avoid the computation of exponentially small/large intermediate terms. 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 larger initial η_0. 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. In Figure 1, under the TensorFlow implementation, the step size is stuck at the initial value throughout the meta training because the meta-gradient explodes and gives NaN value. More details can be found in Appendix F.

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. 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).

[5] Our code is available at https://github.com/Kolin96/learning-to-learn.

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 2. 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.

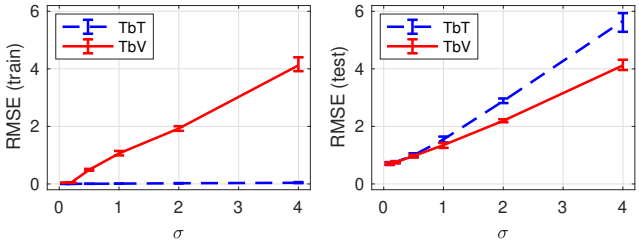


Figure 2. Training and testing RMSE for different σ values (500 samples)

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. From Figure 3, we can see that the gap between these two settings is decreasing as we use more data, as expected by Theorem 6.

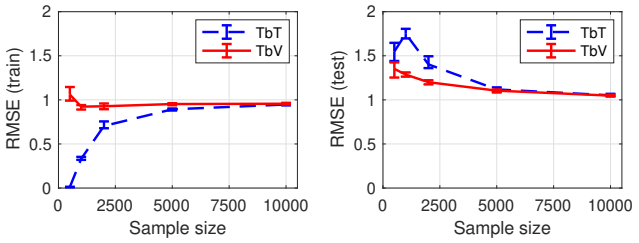
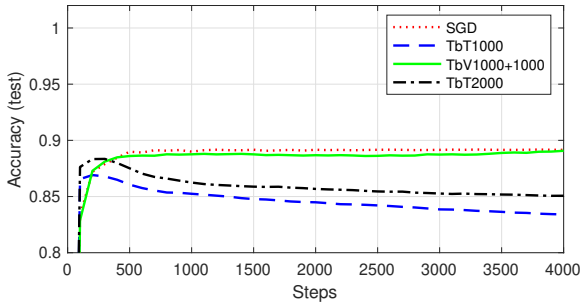
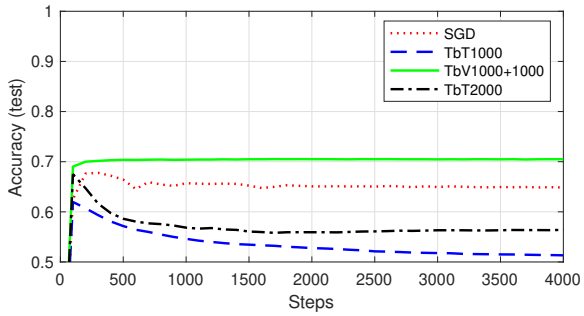


Figure 3. Training and testing RMSE for different samples sizes ($\sigma = 1$)

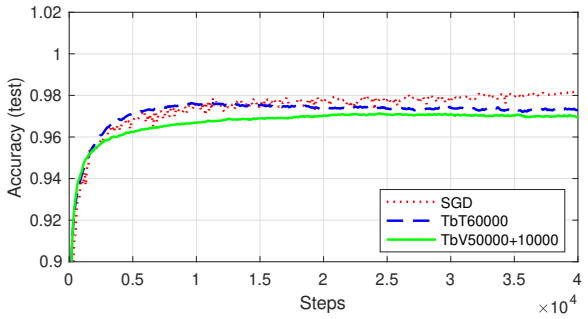
Train-by-train vs. train-by-validation, MLP optimizer on MNIST Here we consider the more interesting case of a multi-layer perceptron (MLP) optimizer on MNIST data set. We use the same MLP optimizer as in Metz et al. (2019), and details of this optimizer is discussed in Appendix F. As the inner problem, 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. In all the following experiments, we use SGD as a baseline with step size tuned by grid search against validation loss. For each optimizer, we run 5 independent tests and collect training accuracy and test accuracy for evaluation. The plots show the mean of the 5 tests[6].



(a) 1000 samples, no noise



(b) 1000 samples, 20% noise



(c) All samples, no noise

Figure 4. The test accuracy of different optimizers in various settings. Comparison between (a) and (b) shows that the advantage of train-by-validation over train-by-train increases when the samples have more noise; comparison between (a) and (c) shows that when the number of samples increases, train-by-train gets comparable performance as train-by-validation.

In Figure 4, we show the test accuracy for different optimizers for different sample size and noise level. In this figure, "TbTx" represents train-by-train approach with x training samples; "TbV$x+y$" represents train-by-validation approach with x training samples and y validation samples. In Figure 4(a) the optimizer is applied to 1000 randomly sub-sampled data (split between training and validation for

[6]We didn't show the measure of the spread because the results of these 5 tests are so close to each other, such that the range or standard deviation marks will not be readable in the plots.

train-by-validation); in Figure 4(b) we use the same amount of data, except we add 20% label noise; in Figure 4(c) we use the whole MNIST dataset without label noise. Comparing Figure 4(a) and (b), we see that when the noise is large train-by-validation significantly outperforms train-by-train. Figure 5 gives the training accuracy in the same setting as Figure 4(b), which clearly shows that train-by-validation can avoid overfitting to noisy labels. Comparing Figure 4(a) and (c), we see that when the number of samples is large enough there is no significant difference between train-by-train and train-by-validation.

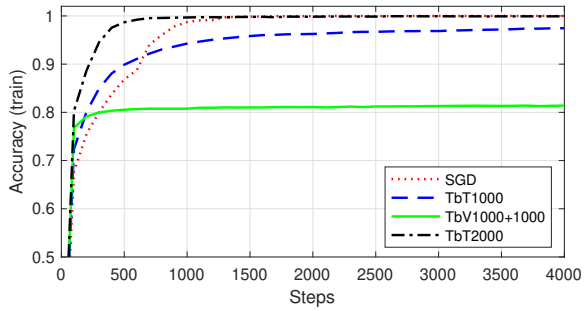


Figure 5. Training accuracy for 1000 samples and 20% noise (same setting as in Figure 4(b))

6. Conclusions

In this paper, we have proved optimization and generalization guarantees for tuning the step size for quadratic loss. From the optimization perspective, we considered a simple task whose objective is a quadratic function. We proved that the meta-gradient can explode/vanish if the meta-objective is simply the loss of the last iteration; we then showed that the log-transformed meta-objective has polynomially bounded meta-gradient and can be successfully optimized. To study the generalization issues, we considered the least squares problem—when the number of samples is small and the noise is large, train-by-validation approach generalizes better than train-by-train; while when the number of samples is large, train-by-train can also work well.

Although our theoretical results are proved for quadratic loss, this simple setting already yields interesting phenomenons and requires non-trivial techniques to analyze. We have also verified our theoretical results on an optimizer parameterized by neural networks and on MNIST dataset. There are still many open problems, including extending similar analysis to more complicated optimizers, or generalizing the idea to prevent numerical issues to neural network optimizers. We hope our work can lead to more theoretical understanding of the learning-to-learn approach.

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 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. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

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

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

Bai, Y., Chen, M., Zhou, P., Zhao, T., Lee, J. D., Kakade, S., Wang, H., and Xiong, C. How important is the train-validation split in meta-learning? *arXiv preprint arXiv:2010.05843*, 2020.

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

Balcan, M.-F., Sandholm, T., and Vitercik, E. Sample complexity of automated mechanism design. *arXiv preprint arXiv:1606.04145*, 2016b.

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

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

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

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2, 1992.

Bengio, Y., Bengio, S., and Cloutier, J. *Learning a synaptic learning rule*. Citeseer, 1990.

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

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

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

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

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

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

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

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

Li, K. and Malik, J. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

Lv, K., Jiang, S., and Li, J. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pp. 2247–2255. PMLR, 2017.

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

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

Morgenstern, J. and Roughgarden, T. Learning simple auctions. In *Conference on Learning Theory*, pp. 1298–1318. PMLR, 2016.

Morgenstern, J. H. and Roughgarden, T. On the pseudo-dimension of nearly optimal auctions advances in neural information processing systems. 136–144. *Google Scholar Google Scholar Digital Library Digital Library*, 2015.

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

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

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

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