HiLD 2024: 2nd Workshop on High-dimensional Learning Dynamics

Nonconvex Meta-optimization for Deep Learning

Xinyi Chen XINYIC@PRINCETON.EDU
Princeton University and Google DeepMind

Evan Dogariu EDOGARIU@PRINCETON.EDU
Princeton University

Zhou Lu ZHOUL@PRINCETON.EDU
Princeton University

Elad Hazan EHAZAN@PRINCETON.EDU
Princeton University and Google DeepMind

Abstract

Hyperparameter tuning in mathematical optimization is a notoriously difficult problem. Recent tools from online control give rise to meta-optimization, a provable methodology for learning optimal hyperparameters in convex optimization. In this work, we extend this methodology to nonconvex optimization and the training of deep neural networks. We present an algorithm for nonconvex meta-optimization that leverages the reduction from nonconvex optimization to convex optimization, and investigate its applicability for deep learning tasks on academic-scale datasets.

1. Introduction

Hyperparameter tuning for deep learning is notoriously difficult and resource-consuming. It is therefore an extremely well-studied problem, with numerous approaches including Bayesian optimization [20], bandit algorithms [16], meta-gradient methods [3], and spectral methods [13].

A recent paradigm based on online control named meta-optimization [5] gives the first provable guarantees for learning certain hyperparameters for smooth convex optimization. There are several novel aspects to this approach: (1) Instead of learning the hyperparameters in one shot, the goal is to repeatedly solve the given optimization problem, learning from experience and converging to the performance of the best hyperparameters. (2) The parameters of the optimizer are learned via a feedback control algorithm based on novel techniques from online control.

Since meta-optimization is based on minimizing regret, which is intractable in general for nonconvex problems, the existing guarantees of this approach only apply to convex problems. In this work, we extend the online control based framework of meta-optimization to nonconvex optimization. Given the framework's guarantees for smooth convex functions, we leverage a reduction from nonconvex to convex optimization inspired by [1]. We propose an algorithm that can learn to adapt to the problem over many episodes, and eventually reach an approximate stationary point. As the number of episodes increases, it converges at a rate that is determined by the performance of the best optimization algorithm from a class of methods.

We conduct experiments on academic-scale workloads, including image classification and machine translation. These initial experiments demonstrate the applicability of the algorithm. In addition, we ablate over several design choices and empirically verify our assumptions.

1.1. Related work

Parameter-free optimization Parameter-free optimization methods are adaptations of first-order methods that do not need to tune certain hyperparameters, such as the learning rate. Methods in this space include coin-betting [18], adaptation to the diameter of the decision set [7], and many more [14], [6], [17]. The meta-optimization approach is more general in two aspects: (1) it attempts to learn the best algorithm for specific objective functions, rather than a class of functions (for example smooth functions with a specific smoothness parameter) (2) for quadratic problems, it has guarantees over a larger class of methods, including precoditioned methods with a fixed precoditioner. However, it is not parameter-free.

Control for optimization Control and optimization are closely related fields, starting from Lyapunov's work and its application to the design and analysis of optimization algorithms, see [5] for more background. [15] apply control theory to the analysis of optimization algorithms on a single problem instance, giving a general framework for obtaining convergence guarantees for a variety of gradient-based methods. [4] study the characterization of the regret-optimal algorithm given an objective function, using a value function-based approach motivated by optimal control. Our work builds upon [5], which proposes an online control based methodology for regret minimization on convex problems.

2. Preliminaries

Meta-optimization In meta-optimization [5], we are given **a sequence of optimization problems**, called episodes. The goal is to design an algorithm that can, over many episodes, perform as well as the best algorithm in a benchmark algorithm class. We denote the number of episodes as N, and in each episode, we perform T steps of optimization. At the beginning of an episode, the iterate is reinitialized to an arbitrary starting point $x_{i,1}$ to be consistent with the standard optimization setting. Chen and Hazan [5] propose a method based on online control, and guarantees that over N episodes, the method converges to the performance of the best first-order gradient method from a general class of methods.

Extension to nonconvex stochastic optimization We extend the meta-optimization framework to nonconvex optimization in the finite-sum setting. Denote the sequence of N objective functions as $\{f_i\}_{i=1}^N$, in this setting, each function is a finite sum of n nonconvex functions :

$$f_i(x) = \frac{1}{n}\sum_{j=1}^{n} f_{i,j}(x).$$

In each episode, we are given an objective function f_i, and at each time step, we have access to a mini-batch of $f_{i,j}$'s. Notably, this setting formalizes the problem of neural network training, where the objective function is the average loss on training examples.

The goal of stochastic nonconvex optimization is to obtain an ε-stationary point in expectation for each objective function: an x_i such that $\mathbb{E}\left[\|\nabla f_i(x_i)\|\right] \le \varepsilon$ for every $i \in [N]$. Following standard assumptions in the literature, we assume each $f_{i,j}$ is smooth and has bounded function value.

In episode i, at time t (denoted as step (i,t)), an optimization algorithm $\mathcal{A}$ chooses a point $x_{i,t} \in \mathbb{R}^d$. Then it receives a mini-batch of examples $B_{i,t}$ of size b, and suffers the nonconvex cost $f_{i,t}(x_{i,t}) = \frac{1}{b}\sum_{j\in B_{i,t}} f_{i,j}(x_{i,t})$. The protocol of this setting and our assumptions are formally defined in Appendix A. Our goal is to design an algorithm $\mathcal{A}$ whose convergence rate for finding approximate stationary points approaches that of the best algorithm in a benchmark class.

3. Algorithms and guarantees

We first present the meta-optimization algorithm by [5] in Algorithm 1 for completeness. In Algorithm 1, the iterate $x_{i,t}$ is updated using weight decay, gradient descent, and a meta-optimization update. The meta-optimization update is a linear function of past gradients parameterized by a set of matrices M, which are the parameters learned by the meta-optimization process. Algorithm 1 updates M using GD according to the rollout loss, which approximates the objective function value achieved had we used the current M for a number of steps. Algorithm 1's guarantees apply when the objective functions are quadratic; we will use a bandit version of the algorithm that has provable guarantees for general smooth convex functions, Algorithm 5.

Algorithm 1: Convex meta-optimization

```
input : episode number $N$, number of steps per episode $T$, window size $H$, rollout length $L$,
        meta learning rate $\eta_g$, initial points $\{x_{i,1}\}_{i=1}^N$, base learning rate $\eta$, weight decay $\delta$.
Set $M_{1,1} = \{M_{1,1}^{(1)}, \dots, M_{1,1}^{(H)} \in \mathbb{R}^{d \times d}\}$ to 0.
for $i = 1, \dots, N$ do
    Re-initialize iterate to $x_{i,1}$.
    for $t = 1, \dots, T$ do
        Update $x_{i,t} = (1-\delta)x_{i,t-1} - \eta g_{i,t-1} - \sum_{h=1}^H M_{i,t}^{(h)} g_{i,t-h}$.
        Receive $f_{i,t}$, compute $g_{i,t} = \nabla f_{i,t}(x_{i,t})$.
        If $t \geq H + L$, use Algorithm 4 in Appendix B to construct the rollout loss
            $c(M) = \text{Rollout}(M, x_{i,t-L}, \{f_{i,\tau}\}_{\tau=t-L-1}^t\} \{g_{i,\tau}\}_{\tau=t-H-L+1}^t)$
        and update $M_{i,t+1} = M_{i,t} - \eta_g \nabla_{M_{i,t}} c(M_{i,t})$
    end
end
```

Reduction from nonconvex to convex optimization Prior works in the literature propose reducing nonconvex optimization to solving a sequence of strongly convex problems [1, 19, 22]. In this approach, each sub-problem in the sequence, now strongly convex, is the original nonconvex objective with ℓ_2-regularization from a particular point. The reduction is stated in Algorithm 3 in the appendix. Analysis of the reduction shows that if one can minimize the aforementioned sequence of strongly convex problems, then an approximate stationary point can be found among the minimizers. Using this framework, we can apply convex meta-optimization to the sequence of strongly convex functions, and obtain a method whose convergence is characterized by the performance of the best method on that sequence of functions. This guarantee is different from the meta-optimization guarantee one can achieve in convex optimization through regret minimization. In general, regret minimization for nonconvex functions is computationally intractable, and alternative notions of regret were introduced in [12].

Algorithm 2: Nonconvex stochastic meta-optimization

```
input : episode number $N$, epoch number $K$, inner step number $S$ such that $KS = T$,
        smoothness parameter $\beta$, initial points $\{x_{i,1}\}$, convex meta-optimization algorithm $\mathcal{A}$.
for $i = 1, \dots, N$ do
    Re-initialize iterate to $x_{i,1}$.
    for $k = 1, \dots, K$ do
        Set $x_{i,k,1} = x_{i,k}$, denote $f_{i,k}(x) = f_i(x) + \beta\|x - x_{i,k}\|^2$ as the regularized objective.
        Receive mini-batches of examples and optimize the regularized objective for $S$ time
          steps using $\mathcal{A}$, output $\{x_{i,k,s}\}_{s=1}^S$.
        Update $x_{i,k+1} = \frac{1}{S}\sum_{s=1}^S x_{i,k,s}$.
    end
end
```

We describe our method for nonconvex meta-optimization in Algorithm 2, and present its guarantee in the theorem below. In addition to the smoothness and boundedness assumptions on the objective functions, we also make assumptions to ensure the stability of our algorithm, similar to [5]. The class of benchmark algorithms Π contains methods whose updates are linear functions of past gradients. If the losses are quadratic, it can simulate first-order gradient-based methods such as gradient descent, momentum, and preconditioned methods. Due to limited space, we defer formal definition of our assumptions and other technical details to Appendix B.

Theorem 1 *Let $x^*_{i,k}$ be a minimizer of $f_{i,k}$, $x^\pi_{i,k,s}$ be the iterate under another algorithm π, and let $\nabla_i^2 = \frac{1}{K}\sum_{k=1}^K \|\nabla f_i(x_{i,k})\|^2$ be the average squared gradient norm. Under Assumptions 1, 2, 3 and 4, using the bandit meta-optimization algorithm (Algorithm 5) as $\mathcal{A}$ in Algorithm 2 yields the following guarantee:*

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^N \nabla_i^2\right] \le O\left(\frac{1}{K}\right) + \tilde{O}((NKS)^{-\frac{1}{4}}) + \frac{6\beta}{NKS}\min_{\pi\in\Pi}\mathbb{E}\left[\sum_{i=1}^N\sum_{k=1}^K\sum_{s=1}^S \left(f_{i,k,s}(x^\pi_{i,k,s}) - f_{i,k}(x^*_{i,k})\right)\right].$$

4. Experiments

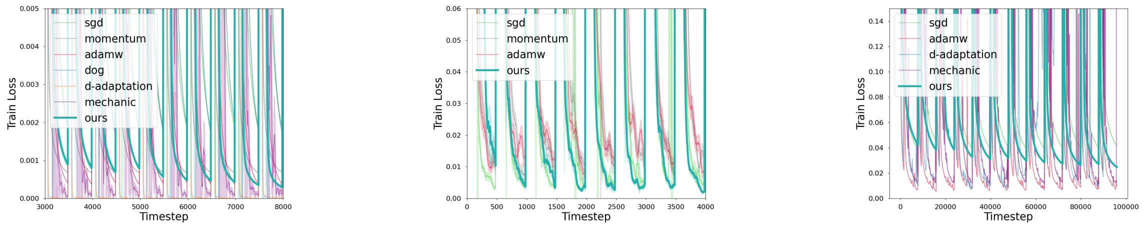
We run experiments on three deep learning workloads of increasing scale: MNIST handwritten digit recognition, CIFAR-10 image classification, and WMT-17 English-to-German machine translation. On each dataset, we apply a standard deep learning architecture (MLP, CNN, and transformer, respectively) and compare our method against the common deep learning optimization techniques. We train in two optimization settings: gradient descent on a fixed batch (i.e. $f_{i,j} \equiv f_i$ for a fixed f_i) and stochastic gradient descent (i.e. a minibatch of $f_{i,j}$'s is sampled i.i.d. each step). For the methods labeled ”ours”, we make practical modifications to Algorithm 2 with the considerations mentioned in Appendix E.1 to arrive at Algorithm 7; this is done in Jax and the code may be found at https://github.com/edogariu/meta_opt/tree/v1.0/.

We compare against the following fully-tuned baselines: vanilla gradient descent, momentum, Adam with weight decay (AdamW), hypergradient descent [3], Distance-over-Gradients [14], D-Adaptation [7], and the Mechanic algorithm [6]. Each of the latter 4 optimizers is based on either vanilla gradient descent or vanilla Adam; to reduce clutter, we only plot these final 4 optimizers when they match or outperform their tuned vanilla counterparts. We refer to AdamW, D-Adaptation, and Mechanic collectively as the ”adaptive methods”, which are not captured theoretically by our benchmark algorithm class. See Appendix E.2 for more information about baselines and experimental hyperparameters such as batch size and number of iterations.

5. Results

We present our main experimental results below, for additional ablations and empirical verification of our assumptions, see Appendix E.3.

Deterministic optimization The training loss in the deterministic setting (i.e. where each training step is over the same subset of data) allows us to inspect the optimization performance of our algorithm without the effect of noise. In Figure 1, the training losses of the various optimization algorithms are plotted across episodes. We see that our method improves over time until the meta-optimization parameters converge – during the first episode it is worse than gradient descent, but after a handful of deterministic episodes it matches or outperforms many other baselines (note that on WMT there is a separation between adaptive methods and non-adaptive methods). Though complexity and the required number of iterations vary between tasks, we find compelling evidence that even on large deep learning workloads our method finds consistent improvement over episodes.

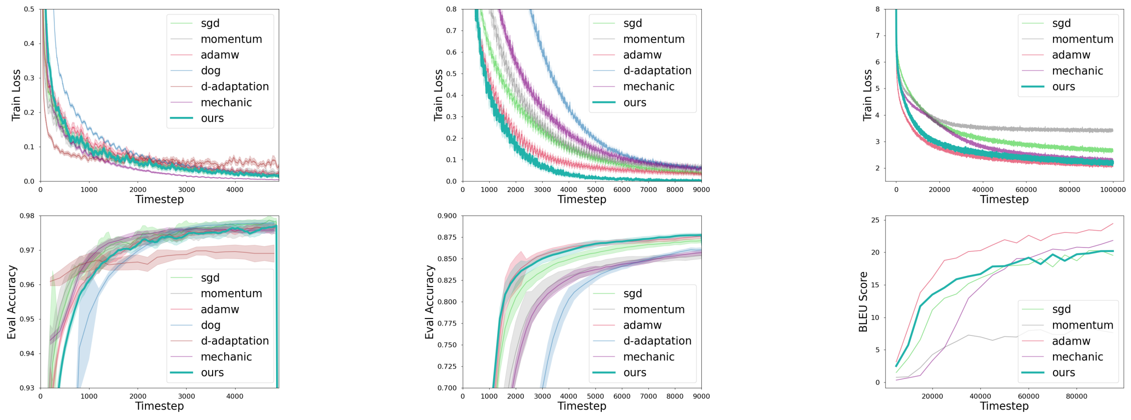


(*a*) MNIST full GD (*b*) CIFAR full GD (*c*) WMT full GD

Figure 1: Full gradient descent training with a fixed batch on the three workloads. MNIST and CIFAR are averaged over 5 trials. Losses smoothed with a mean filter.

Furthermore, since meta-optimization is a convex relaxation of the learning to learn problem, we expect that the meta-optimization process is well-behaved in the deterministic setting. This is indeed the case, as the improvement across episodes is monotonic, and the parameters eventually converge. Moreover, for each fixed workload, the resulting optimal parameters are independent of the initial learning rate η and other hyperparameters. This stability allows our algorithm to converge properly every time we run it; by contrast, none of the self-tuning baselines converged on the CIFAR workload and only the adaptive methods converged on WMT.

Stochastic optimization We also test meta-optimization in the stochastic deep learning setting. Experimentally, we found that the meta-optimization algorithm in the stochastic setting was not as stable on large workloads (see Appendix E.3 for a short explanation). To mitigate this, we take the meta-optimization parameters learned in the deterministic setting and deploy them with frozen parameters ($M_{i,t}$ in Algorithm 1) to the stochastic setting. Figure 2 shows the performance with this approach. Our method is able to outperform the non-adaptive baselines in terms of training loss, demonstrating that the optimal parameters transfer from deterministic to the stochastic setting. For evaluation metrics, we see that on MNIST and CIFAR our method is able to generalize as well as the baselines. On the WMT workload, however, we once again see a qualitative separation between the adaptive methods and non-adaptive methods (those roughly captured by our benchmark algorithm class). We are investigating this generalization gap in our ongoing work (see Appendix F for a discussion of current questions and future work). When we compare our method to the self-tuning baselines, we see that meta-optimization is consistently competitive in training while methods like DoG, D-Adaptation, and Mechanic can unpredictably suffer on certain workloads.



(*a*) MNIST stochastic (*b*) CIFAR stochastic (*c*) WMT stochastic

Figure 2: Stochastic minibatch gradient descent on the three workloads. MNIST and CIFAR are averaged over 5 trials. Losses smoothed with a mean filter.

References

[1] Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, and Yi Zhang. Efficient full-matrix adaptive regularization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 102–110. PMLR, 09–15 Jun 2019.

[2] Oren Anava, Elad Hazan, and Shie Mannor. Online convex optimization against adversaries with memory and application to statistical arbitrage, 2014.

[3] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

[4] Philippe Casgrain and Anastasis Kratsios. Optimizing optimizers: Regret-optimal gradient descent algorithms. In *Proceedings of Thirty Fourth Conference on Learning Theory*, pages 883–926. PMLR, 2021.

[5] Xinyi Chen and Elad Hazan. Online control for meta-optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[6] Ashok Cutkosky, Aaron Defazio, and Harsh Mehta. Mechanic: A learning rate tuner. *Advances in Neural Information Processing Systems*, 36, 2024.

[7] Aaron Defazio and Konstantin Mishchenko. Learning-rate-free learning by d-adaptation. In *International Conference on Machine Learning*, pages 7449–7479. PMLR, 2023.

[8] Abraham D. Flaxman, Adam Tauman Kalai, and H. Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, page 385–394, USA, 2005. Society for Industrial and Applied Mathematics. ISBN 0898715857.

[9] Paula Gradu, John Hallman, and Elad Hazan. Non-stochastic control with bandit feedback. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10764–10774. Curran Associates, Inc., 2020.

[10] Paula Gradu, Elad Hazan, and Edgar Minasyan. Adaptive regret for control of time-varying dynamics. In Nikolai Matni, Manfred Morari, and George J. Pappas, editors, *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, volume 211 of *Proceedings of Machine Learning Research*, pages 560–572. PMLR, 15–16 Jun 2023.

[11] Elad Hazan and Karan Singh. Introduction to online nonstochastic control, 2023.

[12] Elad Hazan, Karan Singh, and Cyril Zhang. Efficient regret minimization in non-convex games. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1433–1441. PMLR, 06–11 Aug 2017.

[13] Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: a spectral approach. In *International Conference on Learning Representations*, 2018.

[14] Maor Ivgi, Oliver Hinder, and Yair Carmon. Dog is sgd's best friend: A parameter-free dynamic step size schedule. In *International Conference on Machine Learning*, pages 14465–14499. PMLR, 2023.

[15] Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

[16] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[17] Zhou Lu, Wenhan Xia, Sanjeev Arora, and Elad Hazan. Adaptive gradient methods with local guarantees. *arXiv preprint arXiv:2203.01400*, 2022.

[18] Francesco Orabona and Dávid Pál. Coin betting and parameter-free online learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[19] Courtney Paquette, Hongzhou Lin, Dmitriy Drusvyatskiy, Julien Mairal, and Zaid Harchaoui. Catalyst for gradient-based nonconvex optimization. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 613–622. PMLR, 09–11 Apr 2018.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[22] Weiran Wang and Nathan Srebro. Stochastic nonconvex optimization with large minibatches. In Aurélien Garivier and Satyen Kale, editors, *Proceedings of the 30th International Conference on Algorithmic Learning Theory*, volume 98 of *Proceedings of Machine Learning Research*, pages 857–882. PMLR, 22–24 Mar 2019.

Appendix A. Additional preliminaries

We make the following assumptions on the nonconvex objective functions, and present the reduction from nonconvex to convex optimization below.

Definition 2 *We say a function is β-smooth if for every x, y, $\|\nabla f(x) - \nabla f(y)\| \le \beta \|x - y\|$.*

Assumption 1 *For all i, j, $0 \le f_{i,j}(x) \le M$ for all x, and $f_{i,j}$ is β-smooth.*

Algorithm 3: Reduction from nonconvex to convex optimization

```
input : epoch number $K$, number of inner steps $S$ such that $T = KS$, convex optimization
        algorithm $\mathcal{A}$, smoothness parameter $\beta$, initial point $x_1$, nonconvex function $f$.
for $k = 1, \dots, K$ do
    Consider the function $f_k(x) = f(x) + \beta\|x - x_k\|^2$.
    Starting from $x_k$, apply $\mathcal{A}$ for $S$ steps on $f_k$ with mini-batch access, obtain $x_{k+1}$.
end
output: $x^* = \arg\min_{\{x_k\}_{k=1}^K} \|\nabla f(x_k)\|$.
```

Appendix B. Algorithm details

Algorithm 4: Rollout

```
input : controller parameters $\{M^{(h)}\}_{h=1}^H$, initial model parameters $x_0$, buffer of cost functions
        $\{f_\tau\}_{\tau=0}^L$, buffer of disturbances $\{g_\tau\}_{\tau=-H}^{L-1}$, rollout length $L$, initial learning rate $\eta$,
        weight decay $\delta$.
for $t = 0, \dots, L-1$ do
    Suffer loss $f_t(x_t)$.
    Update $x_{t+1} = (1-\delta)x_t - \eta\nabla f_t(x_t) - \sum_{h=1}^H M^{(h)} g_{t-h+1}$.
end
output: $f_L(x_L)$
```

The framework of meta-optimization applies online control methods to a particular dynamical system that describes the optimization process. We apply meta-optimization to the stochastic, ℓ_2-regularized strongly convex functions $f_{i,k,s}$, and describe the dynamical system below. The dynamical system is similar to the one for smooth convex optimization put forth in [5].

The dynamical system For each episode i, denote $H_{i,k,s}$ to be the matrix that satisfies

$$\nabla f_{i,k,s}(x_{i,k,s}) = H_{i,k,s}(x_{i,k,s} - x_{i,k,s-1}) + \nabla f_{i,k,s}(x_{i,k,s-1}). \tag{1}$$

If each $f_{i,j}$ is quadratic, then $H_{i,k,s}$ has the following explicit form,

$$H_{i,k,s} = \frac{1}{b}\sum_{j \in B_{i,k,s}} \nabla^2 f_{i,j} + 2\beta.$$

For general smooth functions that are twice differentiable, this matrix exists and each row contains certain second-order information of $f_{i,k,s}$. To see this, observe that we can apply the mean value theorem to each coordinate of $\nabla f_{i,k,s}$ to obtain $H_{i,k,s}$.

The linear dynamical system we consider is

$$\begin{bmatrix} x_{i,k,s+1} \\ x_{i,k,s} \\ \nabla f_{i,k,s}(x_{i,k,s}) \end{bmatrix} = \begin{bmatrix} (1-\delta)I & 0 & -\eta I \\ I & 0 & 0 \\ H_{i,k,s} & -H_{i,k,s} & 0 \end{bmatrix} \times \begin{bmatrix} x_{i,k,s} \\ x_{i,k,s-1} \\ \nabla f_{i,k,s-1}(x_{i,k,s-1}) \end{bmatrix} \tag{2}$$

$$+ \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times u_{i,k,s} + \begin{bmatrix} 0 \\ 0 \\ \nabla f_{i,k,s}(x_{i,k,s-1}) \end{bmatrix}. \tag{3}$$

For notational convenience, we write the dynamical system as

$$z_{i,k,s+1} = A_{i,k,s} z_{i,k,s} + B u_{i,k,s} + w_{i,k,s},$$

where $A_{i,k,s}$ is the system dynamics, B is a constant control-input matrix, and $w_{i,k,s}$ is the non-stochastic disturbance that contains the gradient.

The system above is linear time-varying (LTV), and we introduce the following notion of stability for LTV systems standard in the non-stochastic control literature [5, 10].

Definition 3 (Sequentially stable) *A time-varying linear dynamical system with dynamics $A_1, \ldots, A_T$ is (κ, γ) sequentially stable if for all intervals $I = [r, s] \subseteq [T]$, $\|\prod_{t=s}^{r} A_t\| \le \kappa^2 (1-\gamma)^{|I|}$.*

Assumption 2 *We assume that the dynamical system (2) is (κ, γ) sequentially stable with $\kappa \ge 1$.*

We in addition make the following two assumptions on the system dynamics and the iterates, following the meta-optimization framework. In meta-optimization, the iterates are re-initialized to starting points with bounded norm in each episode; in our algorithm, we essentially have NK episodes, and the iterates are re-initialized with $x_{i,k}$ at the beginning of each episode. We note that since each $f_{i,k,s}$ is strongly convex, the iterates effectively stay within a bounded region, potentially justifying the latter assumption below.

Assumption 3 *For all $i \in [N]$, $k \in [K]$, $s \in [S]$, $\rho(H_{i,k,s}) \le \beta$.*

Assumption 4 *For all $i \in [N]$, $k \in [K]$, $\|x_{i,k}\| \le R$.*

The benchmark algorithm class The benchmark algorithm class we consider consists of methods whose updates are linear functions of past gradients. Since we view optimization from the perspective of online control, this class of methods correspond to a general class of controllers that often appear in the online control literature. This class of controllers is called Disturbance-feedback controllers (DFCs), and for linear time-invariant systems, they can approximate any stabilizing state-feedback controllers. Consequently, if our objective functions are quadratic with uniformly bounded Hessians, this benchmark class of algorithms can simulate gradient descent, momentum, and preconditioned methods with a fixed preconditioner. Since these algorithms have to be stabilizing on the dynamical system described above, only certain values of learning rate, momentum, and preconditioners are allowed. For example, the learning rate can be at most $O(1/\beta)$, and the preconditioner P needs to satisfy $\rho(PH) < 1/8$, where H is the Hessian. For more detailed specifications of the range of parameters, see the full version of [5].

Appendix C. Proofs for Section 3

Proof [Proof of Theorem 1] Since all $f_{i,k,s}(x)$ are convex and smooth, we can use the bandit meta-optimization algorithm (Algorithm 5) as the black-box optimizer $\mathcal{A}$. For any policy $\pi \in \Pi$, let $x^{\pi}_{i,k,s}$ be its updates, and by Corollary 6 we have the following guarantee:

$$\min_{\pi\in\Pi} \mathbb{E}\left[\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{s=1}^{S}\left(f_{i,k,s}(x_{i,k,s}) - f_{i,k,s}(x^{\pi}_{i,k,s})\right)\right] = \tilde{O}((NKS)^{\frac{3}{4}}). \tag{4}$$

Denote $\pi^* = \text{argmin}_{\pi\in\Pi}\mathbb{E}\left[\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{s=1}^{S} f_{i,k,s}(x^{\pi}_{i,k,s})\right]$ as the optimal algorithm in Π.

By Theorem A.3 in [1], since each $f_{i,k}$ is smooth and strongly convex, for any i, k,

$$f_{i,k}(x_{i,k}) - \min_x f_{i,k}(x) \geq \frac{\|\nabla f_{i,k}(x_{i,k})\|^2}{6\beta}.$$

In addition, we have the following decomposition

$$\begin{aligned}
f_i(x_{i,k}) - f_i(x_{i,k+1}) &\geq f_{i,k}(x_{i,k}) - \min_x f_{i,k}(x) - (f_{i,k}(x_{i,k+1}) - \min_x f_{i,k}(x)) \\
&= f_{i,k}(x_{i,k}) - \min_x f_{i,k}(x) - \left(\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x^{\pi^*}_{i,k,s}) - \min_x f_{i,k}(x)\right) \\
&\quad + \left(\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x^{\pi^*}_{i,k,s}) - f_{i,k}(x_{i,k+1})\right) \\
&\geq f_{i,k}(x_{i,k}) - \min_x f_{i,k}(x) - \left(\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x^{\pi^*}_{i,k,s}) - \min_x f_{i,k}(x)\right) \\
&\quad + \left(\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x^{\pi^*}_{i,k,s}) - \frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s})\right).
\end{aligned}$$

Rearranging the terms, we have

$$f_{i,k}(x_{i,k}) - \min_x f_{i,k} \leq f_i(x_{i,k}) - f_i(x_{i,k+1}) + \frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x^{\pi^*}_{i,k,s}) - \min_x f_{i,k}(x) \tag{5}$$

$$+ \left(\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s}) - \frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x^{\pi^*}_{i,k,s})\right). \tag{6}$$

Using the lower bound on the left hand side and summing up over $k = 1, 2, \ldots, K$,

$$\begin{aligned}
\sum_{k=1}^{K}\frac{\|\nabla f_{i,k}(x_{i,k})\|^2}{6\beta} &\leq M + \frac{1}{S}\sum_{s,k}\left(f_{i,k}(x^{\pi^*}_{i,k,s}) - \min_x f_{i,k}(x)\right) \\
&\quad + \left(\frac{1}{S}\sum_{s,k} f_{i,k}(x_{i,k,s}) - \frac{1}{S}\sum_{s,k} f_{i,k}(x^{\pi^*}_{i,k,s})\right).
\end{aligned}$$

Summing over i and taking an average,

$$\frac{1}{NK}\sum_{i,k}\|\nabla f_i(x_{i,k})\|^2 \le O\left(\frac{1}{K}\right) + \frac{1}{NKS}\left(\sum_{i,s,k} f_{i,k}(x_{i,k,s}^{\pi^*}) - \min_x f_{i,k}(x)\right)$$
$$+ \frac{1}{NKS}\left(\sum_{i,s,k} f_{i,k}(x_{i,k,s}) - \sum_{i,s,k} f_{i,k}(x_{i,k,s}^{\pi^*})\right).$$

The theorem follows by taking an expectation over the randomness of the batches, and using the guarantee (4). ■

Refinement of Theorem 1 We use an approach inspired by [1] to refine our guarantee. Define $\lambda_{i,k}$ as the ratio

$$\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s}^{\pi^*}) - \min_x f_{i,k}(x) \le \lambda_{i,k}\sqrt{\frac{f_{i,k}(x_{i,k}) - \min_x f_{i,k}(x)}{\beta S}}. \tag{7}$$

Then in the inequality (5) above, we have

$$f_{i,k}(x_{i,k}) - \min_x f_{i,k} - \lambda_{i,k}\sqrt{\frac{f_{i,k}(x_{i,k}) - \min_x f_{i,k}(x)}{\beta S}} \le f_i(x_{i,k}) - f_i(x_{i,k+1})$$
$$+ \left(\frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s}) - \frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s}^{\pi^*})\right).$$

Observe that when a variable y satisfies $y^2 - ay \le b$, we can complete the squares and obtain $(y - \frac{a}{2})^2 \le b + \frac{a^2}{4}$. Taking a square root, we have $y \le \sqrt{b} + a$, and squaring both sides, we arrive at $y^2 \le 2b + 2a^2$. Using this result, we have

$$\frac{\|\nabla f_i(x_{i,k})\|^2}{6\beta} \le \frac{\lambda_{i,k}^2}{\beta S} + 2\left(f_i(x_{i,k}) - f_i(x_{i,k+1}) + \frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s}) - \frac{1}{S}\sum_{s=1}^{S} f_{i,k}(x_{i,k,s}^{\pi^*})\right).$$

Summing over i, k, and taking an average,

$$\frac{1}{NK}\sum_{i,k}\|\nabla f_i(x_{i,k})\|^2 \le O\left(\frac{1}{K}\right) + O\left(\frac{\sum_{i,k}\lambda_{i,k}^2}{NKS}\right) + \frac{6\beta}{NSK}\sum_{i,s,k} f_{i,k}(x_{i,k,s}) - f_{i,k}(x_{i,k,s}^{\pi^*}).$$

In particular, we have that

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\min_k\|\nabla f_i(x_{i,k})\|^2\right] \le O\left(\frac{1}{K} + \frac{\sum_{i,k}\lambda_{i,k}^2}{NKS} + \tilde{O}((NKS)^{-\frac{1}{4}})\right).$$

Let λ_0 denote an upper bound of $\lambda_{i,k}$. As N grows large, the right hand side is dominated by the first two terms, and therefore in this regime we can write

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\min_k\|\nabla f_i(x_{i,k})\|^2\right] \le O\left(\frac{1}{K} + \frac{\lambda_0^2}{S}\right).$$

As defined in Equation (7), $\lambda_{i,k}$ scales with the function value optimality gap of the average iterate under π^*, and $\sqrt{\frac{1}{\beta S}}$. By the online to batch reduction, SGD on strongly convex functions converges at a rate of $\tilde{O}(\frac{1}{S})$. The learning rate of SGD that attains this rate depends on the strong convexity parameter and the gradient upper bound of the loss function. If all the f_i's have the same smoothness parameter, and under the assumption of bounded domain, taking π^* to be SGD with the optimal learning rate, λ_0 can be as small as $\tilde{O}\left(\frac{1}{\sqrt{S}}\right)$.

Appendix D. Bandit meta-optimization

We give the details of the bandit meta-optimization algorithm in this section. For any set $\mathcal{M}$ and $\delta_M > 0$, define the Minkowski subset $\mathcal{M}_{\delta_M} = \{x : \frac{1}{1-\delta_M} x \in \mathcal{M}\}$, and let $\mathbb{S}_1^d$ be the d-dimensional unit sphere.

Algorithm 5: Bandit meta-optimization

```
input : episode number $K$, system parameters $\eta, \delta, \kappa, \gamma$, learning rates $\{\eta^M_{i,k,s}\}$, history length
        $L$, $\delta_M$, starting points $\{x_{i,1}\}_{i=1}^N$.
Set: $\mathcal{M} = \{M = \{M^1, \dots, M^L\} : \|M^l\| \le \kappa^3(1-\gamma)^l\}$.
Initialize any $M_{1,1} = \dots = M_{L,1} \in \mathcal{M}_{\delta_M}$, $z_{i,1,1} = [x_{i,1}^\top \; x_{i,1}^\top \; 0]^\top$.
Sample $\epsilon_{1,1}, \dots, \epsilon_{L,1} \in_{\mathbb{R}} \mathbb{S}_1^{L\times 3d\times 3d}$, set $\widetilde{M}_{l,1} = M_{l,1} + \delta_M \epsilon_{l,1}$ for $l = 1, \dots, L$.
for $i = 1, \dots, N$ do
    If $i > 1$, set $z_{i,1,1} = z_{i-1,K,S+1}$, $M_{i,1,1} = M_{i-1,K,S+1}$.
    for $k = 1, \dots, K$ do
        If $k > 1$, set $z_{i,k,1} = z_{i,k-1,S+1}$, $M_{i,k,1} = M_{i,k-1,S+1}$.
        for $s = 1, \dots, S$ do
            Choose $u_{i,k,s} = \sum_{l=1}^L \widetilde{M}^l_{i,k,s} w_{i,k,s-1}$.
            Receive $f_{i,k,s}$, compute $w_{i,k,s} = \nabla f_{i,k,s}(x_{i,k,s-1})$. If $s = S$, compute
             $x_{i,k+1} = \frac{1}{S}\sum_{s=1}^S x_{i,k,s}$, and
                $$w_{i,k,S} = \begin{bmatrix} x_{i,k+1} - ((1-\delta)x_{i,k,S} - \eta\nabla f_{i,k,S-1}(x_{i,k,S-1}) + \bar{u}_{i,k,S}) \\ x_{i,k+1} - x_{i,k,S} \\ \nabla f_{i,k,S}(x_{i,k,S-1}) - \nabla f_{i,k,S}(x_{i,k,S}) \end{bmatrix}, \quad (8)$$
             where $\bar{u}_{i,k,S}$ is the first $d$ coordinates of the control signal $u_{i,k,s}$.
            Suffer control cost $f_{i,k,s}(x_{i,k,s})$.
            Store the gradient estimator $g_{i,k,s} = \frac{9d^2L}{\delta_M} f_{i,k,s}(x_{i,k,s}) \sum_{l=1}^{L} \epsilon_{i,k,s-l}$ if $s \ge L$, else 0.
            Perform gradient update on the controller parameters:
                $$M_{i,k,s+1} = \Pi_{\mathcal{M}_{\delta_M}}(M_{i,k,s} - \eta^M_{i,k,s} \cdot g_{i,k,s-L}).$$
            Sample $\epsilon_{i,k,s+1} \in_{\mathrm{R}} \mathbb{S}_1^{L\times 3d\times 3d}$, set $\widetilde{M}_{i,k,s+1} = M_{i,k,s+1} + \delta_M \epsilon_{i,k,s+1}$.
        end
        If $k = K$, compute $w_{i,K,S}$ similar to (8), so the next state evolves to
         $z_{i,K,S+1} = [x_{i,1,1}^\top \; x_{i,1,1}^\top \; 0^\top]$.
    end
end
```

Theorem 4 (Theorem 5.1 in [9], Theorem 3.3 in [5]) *Under Assumptions 1, 2, 3, 4, Algorithm 5 with $\eta \le 1$, $L = \Theta(\log NKS)$, and setting $\eta^M_{i,k,s} = \Theta((N(i-1)+K(k-1)+s)^{-3/4}L^{-3/2}G^{-2/3})$, and perturbation constant $\delta_M = \Theta((NKS)^{-1/4}L^{-1/2})$ gives the guarantee*

$$\mathbb{E}\left[\sum_{i,k,s} f_{i,k,s}(x_{i,k,s})\right] - \min_{\mathcal{A}\in\Pi}\sum_{i,k,s} f_{i,k,s}(x^{\mathcal{A}}_{i,k,s}) \le \tilde{O}((NKS)^{3/4}),$$

where $\tilde{O}$, Θ contain polynomial factors in $\gamma^{-1}, \beta, \kappa, R, b, d, M$, and $\tilde{O}$ in addition contains logarithmic factors in K, S, N. The benchmark algorithm class Π is the class of DFCs discussed in Appendix B.

The theorem above guarantees the performance of Algorithm 5 under any adversarially chosen functions $f_{i,k,s}$. However, for our setting of nonconvex stochastic optimization, is it more useful to derive a guarantee in expectation for randomly chosen functions. We show that such extension is possible, and we start from the bandit convex optimization with memory (BCOwM) problem [9]. The guarantee for bandit online control and hence bandit meta-optimization can be derived as corollaries of the BCOwM guarantee.

Consider the basic online learning with memory problem under bandit feedback, where the loss functions f_t are β-smooth, G-Lipschitz, and M-bounded. The domain of decisions $\mathcal{K}$ has diameter D, and the f_t's are random functions determined by previous decisions of the player. Same as before, let $\mathcal{K}_\delta$ be the Minkowski set of $\mathcal{K}$, and let $\mathbb{S}_1^d$ be the d-dimensional unit sphere. The algorithm below, proposed by [9], is an application of zeroth-order method [8] to the Online Convex Optimization with Memory (OCOwM) setting [2].

Algorithm 6: BCO with Memory

```
input : Decision set $\mathcal{K}$, time horizon $T$, history length $L$, learning rates $\{\eta_t\}$ and noise
        magnitude $\delta$.
Initialize $x_1 = \cdots = x_L \in \mathcal{K}_\delta$ arbitrarily, and sample noise $u_1, \ldots, u_L \in \mathbb{S}_1^d$.
Set $y_i = x_i + \delta u_i$ for $i = 1, \ldots, L$, $g_i = 0$ for $i = 1, \ldots, L-1$.
Predict $y_i$ for $i = 1, \ldots, L-1$.
for $t = L, \ldots, T$ do
    Play $y_t$, and suffer loss $f_t(y_{t-L+1:t})$.
    Store gradient estimate $g_t = \frac{d}{\delta} f_t(y_{t-L+1:t}) \sum_{i=0}^{L-1} u_{t-i}$.
    Set $x_{t+1} = \prod_{\mathcal{K}_\delta} [x_t - \eta_t \cdot g_{t-L+1}]$.
    Sample $u_{t+1} \in \mathbb{S}_1^d$, set $y_{t+1} = x_{t+1} + \delta u_{t+1}$.
end
```

Theorem 5

Suppose the loss functions f_t are random functions determined by previous iterates $y_1, \ldots, y_{t-1}$. Let O denote polynomial dependence on D, d, M, L, G, β. Taking $\eta_t = O(t^{-3/4}), \delta = O(T^{-1/4})$, Algorithm 6 produces y_t's that satisfy

$$\mathbb{E}\left[\sum_{t=L}^{T} f_t(y_{t-L+1:t})\right] - \min_{x \in \mathcal{K}} \mathbb{E}\left[\sum_{t=L}^{T} f_t(x, \ldots, x)\right] \le O(T^{3/4}).$$

Proof We largely follow the proof of Theorem 3.1 in [9]. Let x^* be any comparator in $\mathcal{K}$, and x_δ^* be the projection of x^* in the Minkowski set. Let $\tilde{f}(x) = f(x, \ldots, x)$ be the shorthand notation.

$$\mathbb{E}\left[\sum_{t=L}^{T} f_t(y_{t-L+1:t}) - \sum_{t=L}^{T} \tilde{f}_t(x^*)\right] = \mathbb{E}\left[\sum_{t=L}^{T} (f_t(y_{t-L+1:t}) - \tilde{f}_t(x^*))\right] - \mathbb{E}\left[\sum_{t=L}^{T} \tilde{f}_{t-L+1}(x_t) - \tilde{f}_{t-L+1}(x_\delta^*)\right] \tag{9}$$

$$+ \mathbb{E}\left[\sum_{t=L}^{T} \tilde{f}_{t-L+1}(x_t) - \tilde{f}_{t-L+1}(x_\delta^*)\right] \tag{10}$$

We bound (9) and (10) separately. We start with (9), which can be bounded for any sequence of random variables $u_1, \dots, u_T$. Fix $u_1, \dots, u_T$, we have

$$\begin{aligned} f_t(y_{t-L+1:t}) - \tilde{f}_t(x_{t+L-1}) &= f_t(x_{t-L+1:t} + \delta u_{t-L+1:t}) - \tilde{f}_t(x_{t+L-1}) \\ &\le f_t(x_{t-L+1:t}) - \tilde{f}_t(x_{t+L-1}) + \delta G\sqrt{L} \\ &\le G\|x_{t-L+1:t} - (x_{t+L-1}, \dots, x_{t+L-1})\| + \delta G\sqrt{L} \\ &\le \frac{2dMGL^2\eta_{t-L+1}}{\delta} + \delta G\sqrt{L}, \end{aligned}$$

where the first and second inequalities hold by the Lipschitz property of f_t, and the last inequality is due to Lemma 7. Furthermore, the Lipschitz property of f_t gives

$$|\tilde{f}_t(x_\delta^*) - \tilde{f}_t(x^*)| \le G\|(x_\delta^*, \dots, x_\delta^*) - (x^*, \dots, x^*)\| \le \delta GD\sqrt{L}.$$

Putting the two inequalities together, and accounting for the shift in the index of $\tilde{f}_{t-L+1}(x_\delta^*)$,

$$(2) \le 2\delta GD\sqrt{L}T + \frac{2dMGL^2}{\delta}\sum_{t=1}^{T}\eta_t + 2LM.$$

The term (10) can be decomposed as follows,

$$\begin{aligned} \mathbb{E}\left[\sum_{t=L}^{T} \tilde{f}_{t-L+1}(x_t) - \tilde{f}_{t-L+1}(x_\delta^*)\right] &\le \mathbb{E}\left[\sum_{t=L}^{T} \nabla\tilde{f}_{t-L+1}(x_t)^\top (x_t - x_\delta^*)\right] \\ = \mathbb{E}\left[\sum_{t=L}^{T}(g_{t-L+1} + (\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - g_{t-L+1})\right] \\ + \mathbb{E}\left[(\nabla\tilde{f}_{t-L+1}(x_t) - \mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}]))^\top (x_t - x_\delta^*)\right]. \end{aligned}$$

Since x_{t+1} is a projected gradient descent step from x_t with the gradient estimator g_{t-L+1}, we have

$$2g_{t-L+1}^\top (x_t - x_\delta^*) \le \frac{1}{\eta_t}(\|x_t - x_\delta^*\|^2 - \|x_{t+1} - x_\delta^*\|^2) + \eta_t\|g_{t-L+1}\|^2, \qquad \text{(Eq. 3.2 in [9])}$$

and

$$\begin{aligned} \sum_{t=L}^{T} g_{t-L+1}^\top (x_t - x_\delta^*) &\le \frac{1}{2}\sum_{t=L}^{T}\left(\|x_t - x_\delta^*\|^2\left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) + \eta_t\|g_{t-L+1}\|^2\right) + \frac{\|x_L - x_\delta^*\|^2}{\eta_{L-1}} \\ &\le \frac{D^2}{2}\left(\frac{1}{\eta_T} + \frac{1}{\eta_{L-1}}\right) + \frac{d^2M^2L}{2\delta^2}\sum_{t=L}^{T}\eta_t, \end{aligned}$$

where the bound on $\|g_t\|$ is in the proof of Lemma 7.

By Lemma 8, we also have

$$\begin{aligned} &\mathbb{E}\left[(\nabla\tilde{f}_{t-L+1}(x_t) - \mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}]))^\top (x_t - x_\delta^*)\right] \\ &\le \mathbb{E}\left[\|\nabla\tilde{f}_{t-L+1}(x_t) - \mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}]\|\|x_t - x_\delta^*\|\right] \\ &\le D\mathbb{E}\left[\|\nabla\tilde{f}_{t-L+1}(x_t) - \mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}]\|\right] \\ &\le 2\eta_{t-L+1}\frac{dML^{5/2}\beta D}{\delta} + \frac{d\delta L^2 D}{2}. \end{aligned}$$

Lastly, by Lemma 9,

$$\mathbb{E}\left[\sum_{t=L}^{T}(\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - g_{t-L+1})^\top(x_t - x_\delta^*)\right] \le \frac{2d^2M^2L^2}{\delta^2}\sum_{t=L}^{T}\eta_{t-L+1}.$$

Summing up the the three inequalities, (3) can be bounded by

$$(3) \le \frac{D^2}{\eta_T} + \left(\frac{3d^2M^2L^2}{\delta^2} + \frac{2dML^{5/2}\beta D}{\delta}\right)\sum_{t=1}^{T}\eta_t + \frac{d\delta L^2 DT}{2}.$$

Putting everything together, the expected regret can be bounded by

$$\mathbb{E}\left[\sum_{t=L}^{T} f_t(y_{t-L+1:t}) - \sum_{t=L}^{T}\tilde{f}_t(x^*)\right] \le 2LM + \frac{D^2}{\eta_T} + \left(\frac{3d^2M^2L^2}{\delta^2} + \frac{4dGML^{5/2}\beta D}{\delta}\right)\sum_{t=1}^{T}\eta_t + \frac{5d\delta GL^2DT}{2}.$$

Let O denote polynomial dependence on D, d, M, L, G, β. Taking $\eta_t = O(t^{-3/4}), \delta = O(T^{-1/4})$, we have $\sum_{t=1}^{T}\eta_t \le O(T^{1/4})$, and

$$\mathbb{E}\left[\sum_{t=L}^{T} f_t(y_{t-L+1:t}) - \sum_{t=L}^{T}\tilde{f}_t(x^*)\right] \le O(T^{3/4}).$$

■

Corollary 6 *Under the same assumptions as Theorem 5, and setting $\eta_{i,k,s}^M, \delta_M, L$ correctly, Algorithm 5 produces a sequence of controls $M_{i,k,s}$ that satisfy*

$$\mathbb{E}\left[\sum_{i,k,s} f_{i,k,s}(x_{i,k,s})\right] \le \min_{\pi\in\Pi}\mathbb{E}\left[\sum_{i,k,s} f_{i,k,s}(x_{i,k,s}^\pi)\right] + \tilde{O}((NKS)^{3/4}).$$

Lemma 7 (Variant of Lemma A.6 in [9]) *Fixing $u_1, \ldots, u_T$, Algorithm 6 produces a sequence of x_t such that*

$$\|x_{t-H+1:t} - (x_{t+H-1}, \ldots, x_{t+H-1})\| \le 2\eta_{t-H+1}\frac{dCH^2}{\delta}.$$

Proof Fix $u_1, \ldots, u_T$.

$$\begin{aligned}
\|x_{t-L+1:t} - (x_{t+L-1}, \ldots, x_{t+L-1})\|^2 &= \sum_{i=0}^{L-1} \|x_{t-i} - x_{t+L-1}\|^2 \\
&\le \sum_{i=0}^{L-1} \left(\sum_{j=1}^{i+L-1} \|x_{t+L-j} - x_{t+L-j-1}\| \right)^2 \\
&\le \sum_{i=1}^{L-1} \left(\sum_{j=1}^{i+L-1} \eta_{t+L-1-j} \|g_{t-j}\| \right)^2 \\
&\le \eta_{t-L+1}^2 \sum_{i=1}^{L-1} \left(\sum_{j=1}^{i+L-1} \|g_{t-j}\| \right)^2 \\
&\le 4\eta_{t-L+1}^2 L^3 \frac{d^2 M^2 L}{\delta^2},
\end{aligned}$$

where the second-to-last inequality holds since the stepsize is non-increasing, and the last inequality is true because for any t,

$$\|g_t\| = \frac{d}{\delta} \|f_t(y_{t-L-1:t}) \sum_{i=0}^{L-1} u_{t-i}\| \le \frac{dM}{\delta} \sqrt{L}.$$

The lemma follows by taking a square root on both sides. ■

Lemma 8 (Variant of Lemma A.12 in [9]) *Conditioned on $u_1, \ldots, u_{t-2L+1}$, for any sequence of $u_{t-2L+2:t-L+1}$ that determines x_t,*

$$\|\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - \nabla \tilde{f}_{t-L+1}(x_t)\| \le 2\eta_{t-L+1} \frac{dML^{5/2}\beta}{\delta} + \frac{d\delta L^2}{2}.$$

Proof By definition, after fixing $u_{1:t-2L+1}$, the following quantities and functions are deterministic: $g_1, \ldots, g_{t-2L+1}$, $x_1, \ldots, x_{t-L+1}$, and $f_1, \ldots, f_{t-L+2}$. For a function f that takes in L inputs, let $\nabla_i f(x_0, \ldots, x_{L-1}) = \frac{\partial f(x_0, \ldots, x_{L-1})}{x_i}$ denote the gradient of f with respect to x_i.

By triangle inequality,

$$\begin{aligned}
\|\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - \nabla \tilde{f}_{t-L+1}(x_t)\| &\le \|\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - \sum_{i=0}^{L-1} \nabla_i f_{t-L+1}(x_{t-2L+2:t-L+1})\| \\
&\quad + \|\sum_{i=0}^{L-1} \nabla_i f_{t-L+1}(x_{t-2L+2:t-L+1}) - \nabla \tilde{f}_{t-L+1}(x_t)\| \\
&\le \frac{d\delta L^2}{2} + \|\sum_{i=0}^{L-1} \nabla_i f_{t-L+1}(x_{t-2L+2:t-L+1}) - \nabla \tilde{f}_{t-L+1}(x_t)\|,
\end{aligned}$$

where the second inequality is due to Corollary A.10 in [9]. The norm of the sum can be bounded by smoothness: for any sequence of $u_{t-2L+2:t-L+1}$,

$$\begin{aligned}\|\sum_{i=0}^{L-1}\nabla_i f_{t-L+1}(x_{t-2L+2:t-L+1}) - \nabla\tilde{f}_{t-L+1}(x_t)\|^2 &\leq L\sum_{i=0}^{L-1}\|\nabla_i f_{t-L+1}(x_{t-2L+2:t-L+1}) - \nabla_i f_{t-L+1}(x_{t:t})\|^2\\ &= L\|\nabla f_{t-L+1}(x_{t-2L+2:t-L+1}) - \nabla f_{t-L+1}(x_{t:t})\|^2\\ &\leq L\beta^2\|x_{t-2L+2:t-L+1} - (x_t,\ldots,x_t)\|^2\\ &\leq 4\eta_{t-L+1}^2\frac{d^2M^2L^5\beta^2}{\delta^2},\end{aligned}$$

by Lemma 7. Hence

$$\|\sum_{i=0}^{L-1}\nabla_i f_{t-L+1}(x_{t-2L+2:t-L+1}) - \nabla\tilde{f}_{t-L+1}(x_t)\| \leq 2\eta_{t-L+1}\frac{dML^{5/2}\beta}{\delta},$$

and

$$\|\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - \nabla\tilde{f}_{t-L+1}(x_t)\| \leq 2\eta_{t-L+1}\frac{dML^{5/2}\beta}{\delta} + \frac{d\delta L^2}{2}$$

■

Lemma 9 *Conditioned on* $u_1,\ldots,u_{t-2L+1}$,

$$\mathbb{E}_{u_{t-2L+2:t-L+1}}\left[\left(\mathbb{E}_{u_{t-2L+2:t-L+1}}[g_{t-L+1}] - g_{t-L+1}\right)^\top(x_t - x_\delta^*)\right] \leq \eta_{t-L+1}\frac{2d^2M^2L^2}{\delta^2}.$$

Proof For convenience, let $\mathbb{E}$ denote the expectation over $u_{t-2L+2:t-L+1}$. Note that x_t is a function of g_{t-L+1}, which depends on $u_{t-2L+2:t-L+1}$. We have

$$\begin{aligned}\mathbb{E}\left[\left(\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\right)^\top(x_t - x_\delta^*)\right] &= \mathbb{E}\left[\left(\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\right)^\top(x_{t-L+1} - x_\delta^*)\right]\\ &+ \mathbb{E}\left[\left(\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\right)^\top(x_t - x_{t-L+1})\right].\end{aligned}$$

Note that x_{t-L+1} is fixed conditioned on $u_1,\ldots,u_{t-2L+1}$, so

$$\mathbb{E}\left[\left(\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\right)^\top(x_{t-L+1} - x_\delta^*)\right] = 0.$$

The second term satisfies, for any sequence of $u_{t-2L+2:t-L+1}$,

$$\begin{aligned}\left(\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\right)^\top(x_t - x_{t-L+1}) &\leq \|\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\|\,\|x_t - x_{t-L+1}\|\\ &\leq 2\max_{u_{t-2L+1:t-L+1}}\|g_{t-L+1}\|\|x_t - x_{t-L+1}\|\\ &\leq \frac{2dM\sqrt{L}}{\delta}\|x_t - x_{t-L+1}\|.\end{aligned}$$

Similarly to the proof of Lemma 7, we have

$$\|x_t - x_{t-L+1}\| \le \sum_{i=0}^{L-2} \|x_{t-i} - x_{t-i-1}\| \le \sum_{i=0}^{L-2} \eta_{t-i-1}\|g_{t-i-L}\| \le \eta_{t-L+1}\frac{dML^{3/2}}{\delta}.$$

Therefore,

$$\left(\mathbb{E}[g_{t-L+1}] - g_{t-L+1}\right)^\top (x_t - x_{t-L+1}) \le \eta_{t-L+1}\frac{2d^2M^2L^2}{\delta^2},$$

and the lemma follows by summing the two terms. ■

Appendix E. Experiments

E.1. Meta-optimization implementation

The implementation used for the deep learning experiments is based on the convex stochastic meta-optimization algorithm detailed in Algorithm 1. This has two key differences with what is used in our proofs: (1) we do not use the regularized loss functions of the form $f(x) + \beta\|x - x_k\|^2$ and (2) we compute gradients of Algorithm 4 instead of using bandit feedback. Note that to differentiate the surrogate loss that is computed through counterfactual rollouts, the implemented algorithm must backpropagate through several training steps in order to perform each meta-update. In the control language, this amounts to applying the gradient perturbation controller (GPC) to the dynamical system defined in Appendix B; for more information on nonstochastic control and the counterfactual nature of the GPC algorithm, please see Chapter 7 of [11]. We learn scalar meta-parameters $M_{i,t}$ instead of full matrices for computational efficiency (though we observed no difference when using diagonal matrices), and we use the Adam optimizer with learning rate 10^{-4} and $\beta_1 = 0.9$, $\beta_2 = 0.999$ to update the $M_{i,t}$ parameters.

An efficient open-source implementation of the algorithm as an `optax` optimizer is available at https://github.com/edogariu/meta_opt/tree/v1.0/. For clarity and reproducibility, we also provide as Algorithm 7 a specification (in more standard deep learning terminology) of the practical meta-optimization algorithm that was used in our experiments. The algorithm is phrased to handle mini-batches and stochastic gradients for user convenience, but in our experiments we only use the full Algorithm 7 in the deterministic setting with a fixed batch and full gradients. As discussed in the main paper, in the stochastic setting we freeze the $M_{i,t}$ parameters. In the algorithm below, we set $H = 32$ (except for WMT, where $H = 16$ due to memory constraints), $L = 2$, $\mathcal{O}$ as Adam with learning rate 10^{-4} and $(\beta_1, \beta_2) = 0.9, 0.999$, and the initializers selected at random; however, the behavior is quite robust to all these parameters.

Algorithm 7: Meta-optimization, deep learning implementation

```
input : number of episodes $N$, number of steps per episode $T$, window size $H$, rollout length
        $L$, first-order optimizer $\mathcal{O}$, initial points $\{x_{i,1}\}_{i=1}^N$, initial learning rate $\eta$, weight decay
        $\delta$.
Initialize buffers of the past $L$ model parameters, the past $L+1$ loss functions, and the past
  $H+L$ stochastic gradients.
Initialize scalar parameters $\{M_{1,1}^{(h)}\}_{h=1}^H \subset \mathbb{R}$ with $M_{1,1}^{(h)} = 0$ for all $h$.
for $i = 1, \dots, N$ do
    for $t = 1, \dots, T$ do
        Play $x_{i,t} = x_{i,t-1} - \eta \widetilde{\nabla} f_{i,t-1} - \sum_{h=1}^H M_{i,t}^{(h)} \widetilde{\nabla} f_{i,t-h}$ if $t > 1$; else play $x_{i,1}$ (if
          $t - h < 1$, set $\widetilde{\nabla} f_{i,t-h} = 0$).
        Receive a mini-batch of examples $B_{i,t}$ of size $b$.
        Obtain the loss function $f_{i,t} = \frac{1}{b}\sum_{j \in B_{i,t}} f_{i,j}$ and append it to the buffer of loss
          functions.
        Suffer loss $f_{i,t}(x_{i,t})$ and compute the stochastic gradient $\widetilde{\nabla} f_{i,t} = \frac{1}{b}\sum_{j \in B_{i,t}} \nabla f_{i,j}(x_{i,t})$.
        If $t \geq H + L$, calculate the counterfactual GPC rollout gradients

            $g_{i,t} = \nabla_{M_{i,t}} \text{Rollout}\left(M_{i,t}, x_{i,t-L}, \{f_{i,\tau}\}_{\tau=t-L-1}^t, \{\widetilde{\nabla} f_{i,\tau}\}_{\tau=t-H-L+1}^t, L, \eta, \delta\right)$

          by running Algorithm 4 and autodifferentiating. Update the meta-optimizer
          parameters via $M_{i,t+1} = \mathcal{O}(M_{i,t}, g_{i,t})$.
        Append $\widetilde{\nabla} f_{i,t}$ to the gradient buffer and $x_{i,t}$ to the parameter buffer.
    end
end
```

E.2. Experimental setup

Architectures We used the following commonplace deep learning architectures for the three workloads, and note that the deterministic setting uses the same batch of data throughout training:

- **MNIST**: a 3-layer multilayer perceptron (MLP) with ReLU and 784, 100, 100, and 10 neurons in the input layer, two hidden layers, and output layer, respectively, totaling 90K parameters. We used a batch size of 512 in both the deterministic and stochastic settings with no preprocessing. For the deterministic setting we trained for $N = 16$ episodes of $T = 500$ iterations each, and for the stochastic setting we train for 5,000 iterations.

- **CIFAR**: a VGG-16 architecture with an output layer of 10, totaling 15M parameters. We used a batch size of 512 in both the deterministic and stochastic settings with no preprocessing. For the deterministic setting we trained for $N = 8$ episodes of $T = 500$ iterations each, and for the stochastic setting we train for 9,000 iterations.

- **WMT**: a base Transformer architecture (as specified in [21] and implemented in Flax's WMT tutorial) totaling 65M parameters. We evaluated on the WMT-14 English-to-German dataset, and we used a batch size of 16 in both the deterministic and stochastic settings. For the deterministic setting we trained for $N = 12$ episodes of $T = 8,000$ iterations each, and for the stochastic setting we train for 100,000 iterations.

Baselines For each of the above workloads, we tried the following deep learning optimizers:

- **SGD**: Gradient descent with weight decay. To tune this baseline, we used a grid search over the learning rate η and the weight decay parameter δ taking values $\eta \in [0.001, 0.01, 0.1, 0.2, 0.4, 1.0]$ and $\delta \in [0, 10^{-5}, 10^{-4}, 10^{-3}]$, respectively.

- **MOMENTUM**: Gradient descent with momentum and weight decay. To tune this baseline, we used a grid search over the learning rate η, the momentum parameter μ, and the weight decay parameter δ taking values $\eta \in [0.001, 0.01, 0.1, 0.2, 0.4, 1.0]$, $\mu \in [0.9, 0.95, 0.99]$, and $\delta \in [0, 10^{-5}, 10^{-4}, 10^{-3}]$, respectively.

- **ADAMW**: Adam optimizer, with weight decay. To tune this baseline, we used a grid search and swept the learning rate η, momentum parameters β_1, β_2, and weight decay parameter δ with the values $\eta \in [10^{-4}, 4 \cdot 10^{-4}, 10^{-3}]$, $\beta_1 \in [0.9, 0.99]$, $\beta_2 \in [0.9, 0.99, 0.999]$, and $\delta \in [0, 10^{-5}, 10^{-4}, 10^{-3}]$, respectively.

- **HGD**: Hypergradient descent acting on the standard gradient descent algorithm (Algorithm 4 in [3]). To tune this baseline, we set the initial learning rate to be the tuned SGD learning rate and swept the meta-learning rate β with the values $\beta \in [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$. Hypergradient descent never performed better than tuned vanilla SGD, so we do not plot it in Figures 1 or 2.

- **DOG**: The Distance-over-Gradients (DoG) algorithm [14]. We run this baseline with the given hyperparameters since it is self-tuning, and we use the `optax.contrib` implementation.

- **D-ADAPTATION**: D-Adaptation algorithm acting on the Adam optimizer (Algorithm 5 in [7]). We run this baseline with the given hyperparameters since it is self-tuning, and we use the `optax.contrib` implementation.

- **MECHANIC**: the Mechanic [6] algorithm acting on the AdamW optimizer described earlier. We tune this baseline with the same grid search used to tune the AdamW optimizer, and we use the `optax.contrib` implementation.

E.3. Ablations & other experiments

Sequential stability One assumption we make that is nonstandard in the deep learning optimization literature is Assumption 2 – the sequential stability of the LTV dynamical system. We numerically verify this notion of stability in Figure 3 for the dynamical system induced by training a small neural network on MNIST (since the size of these matrices scales quadratically with number of parameters, computing this for larger networks is infeasible).

Stochastic meta-optimization In Figure 4, we show what may happen if the meta-optimization algorithm is run in the stochastic setting; as can be seen, the performance degrades between episodes. This occurs on the more complex datasets, and so we believe it to be a characteristic of how back-propagation through rollouts responds to noise between batches.

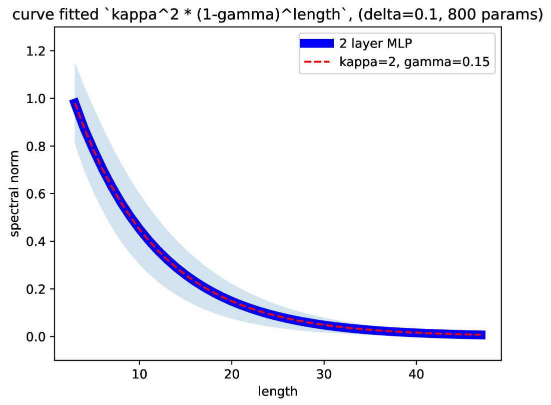


Figure 3: Decay of spectral norm of $\|\prod_{t=s}^{r} A_t\|$ as a function of $|r - s|$ for a small neural network at the beginning of training. Averaged over 10 trials. Assumption 2 is satisfied in this instance with a value of $\kappa \approx 2.0$.

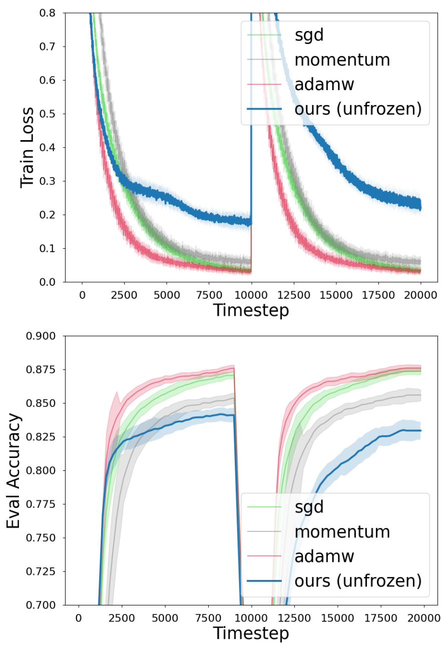


Figure 4: Behavior of the meta-optimization algorithm when training unfrozen in the stochastic regime. The optimizer's performance degrades over time.

Appendix F. Future work

In this work, we presented an initial exploration into the behavior of our meta-optimization algorithm in deep learning environments. As such, our investigations and design decisions leave much room for improvement and discovery, and we hope that the promising results inspire research to make such methods more practical. We list below several directions of investigation that we think will be fruitful.

Generalization As seen in Figure 2, on the WMT workload there is a noticeable generalization gap between adaptive methods (algorithms like Adam and its derivatives) and non-adaptive methods (vanilla gradient descent, momentum, and fixed preconditioners). This mirrors what is seen in many related works, where versions of hyperparameter-tuning algorithms that are built on top of Adam variants perform better than those built on vanilla gradient descent. While training with adaptive methods does not induce a linear dynamical system, we consider it a problem of practical importance to incorporate this adaptivity into the meta-optimization algorithm.

Scaling We have demonstrated that the meta-optimization approach is competitive on workloads of different scales. However, for the largest workloads, it would be valuable to understand the transferability of learned optimizers across model scales. Progress in this direction could allow for one to learn a meta-optimizer on a smaller architecture and transfer it to a larger one, potentially allowing for meta-optimization of large frontier models.

Optimizer pre-training The paradigm we proposed for meta-optimization on general deep learning workloads was to learn an optimizer in the deterministic setting on a fixed batch and deploy it in the stochastic setting. However, we are still investigating the effects of the data selection itself on the downstream performance: how do batch size, dataset complexity, and using the same frozen batch affect the optimal $M_{i,t}$'s, and does this impact their transferability to the stochastic setting? Depending on the answers to these questions, there may be more principled or practical ways to learn a robust optimizer that can be deployed in the stochastic minibatch setting.

Efficient implementation There is much room for improvement in terms of the implementation and parallelization of the meta-optimization algorithm. At the moment, the optimizer state needs to retain the past H gradients, which for large models can be a significant memory burden; however, there is ample structure in the gradient buffer and how it is used, and so we expect that an efficient sharding of optimizer state is possible. Furthermore, we anticipate that there are cleverer ways to use Jax's machinery in order to help with the computational cost of backpropagating through rollouts.