Amortized Proximal Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose a framework for online meta-optimization of parameters that govern optimization, called Amortized Proximal Optimization (APO). We first interpret various existing neural network optimizers as approximate stochastic proximal point methods which trade off the current-batch loss with proximity terms in both function space and weight space. The idea behind APO is to amortize the minimization of the proximal point objective by meta-learning the parameters of an update rule. We show how APO can be used to adapt a learning rate or a structured preconditioning matrix. Under appropriate assumptions, APO can recover existing optimizers such as natural gradient descent and KFAC. It enjoys low computational overhead and avoids expensive and numerically sensitive operations required by some second-order optimizers, such as matrix inverses. We empirically test APO for online adaptation of learning rates and structured preconditioning matrices for regression, image reconstruction, image classification, and natural language translation tasks. Empirically, the learning rate schedules found by APO generally outperform optimal fixed learning rates and are competitive with manually tuned decay schedules. Using APO to adapt a structured preconditioning matrix generally results in optimization performance competitive with second-order methods.

1 Introduction

Many optimization algorithms widely used in machine learning can be seen as approximations to an idealized algorithm called the proximal point method (PPM). When training neural networks, the stochastic PPM iteratively minimizes a loss function $\mathcal{J}_{\mathcal{B}}\colon \mathbb{R}^m \to \mathbb{R}$ on a mini-batch $\mathcal{B}$, plus a proximity term that penalizes the discrepancy from the current iterate:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \operatorname*{arg\,min}_{\mathbf{u}\in\mathbb{R}^m} \mathcal{J}_{\mathcal{B}^{(t)}}(\mathbf{u}) + \lambda D(\mathbf{u}, \boldsymbol{\theta}^{(t)}), \tag{1}$$

where $D(\cdot,\cdot)$ measures the discrepancy between two vectors and $\lambda > 0$ is a hyperparameter that controls the strength of the proximity term. The proximity term discourages the update from excessively changing the parameters, hence preventing aggressive updates. Moreover, the stochastic PPM has good convergence properties [4]. While minimizing Eq. 1 exactly is usually impractical (or at least uneconomical), solving it approximately (by taking first or second-order Taylor series approximations to the loss or the proximity term) has motivated important and widely used optimization algorithms such as natural gradient descent [1] and mirror descent [6]. Stochastic gradient descent (SGD) [68] itself can be seen as an approximate PPM where the loss term is linearized and the discrepancy function is squared Euclidean distance.

Minimization of the exact proximal objective in Eq. 1 is un-economical, as it may require tens to hundreds of optimization steps per parameter update. Instead, we propose to learn a parametric update rule $\boldsymbol{\theta}^{(t+1)} \leftarrow u(\boldsymbol{\theta}^{(t)}, \mathcal{B}^{(t)}, \boldsymbol{\phi})$ described by meta-parameters $\boldsymbol{\phi}$, and use a variant of Eq. 1

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

as a meta-objective to optimize ϕ. By meta-learning the optimization parameters online, we can amortize the cost of minimizing the PPM objective, by performing one meta-step every $K > 1$ steps of the base optimization. We consider adapting optimization hyperparameters (such as the learning rate) for existing optimizers such as SGD and RMSprop [77], as well as learning structured preconditioning matrices. By choosing a structure for the update rule inspired by existing optimizers, we can take advantage of the insights that went into those optimizers while still being robust to cases where their assumptions (such as the use of linear or quadratic approximations) break down. By doing meta-descent on the optimization parameters, we can amortize the cost of minimizing the PPM objective, which would otherwise take many steps per parameter update. Hence, we call our approach *Amortized Proximal Optimization (APO)*.

Eq. 1 leaves a lot of freedom for the proximity term. We argue that many of the most effective neural network optimizers can be seen as trading off two different proximity terms: a *function space discrepancy (FSD)* term which penalizes the average change to the network's predictions, and a *weight space discrepancy (WSD)* term which prevents the weights from moving too far, encouraging smoothness to the update and maintaining the accuracy of second-order approximations. Our meta-objective includes both terms.

Our formulation of APO is general, and can be applied to various settings, from optimizing a single optimization hyperparameter to learning a flexible update rule. We consider two use cases that cover both ends of this spectrum. At one end, we consider the problem of adapting learning rates of existing optimizers, specifically SGD, RMSprop, and Adam [18]. The learning rate is considered one of the most essential hyperparameters to tune [9], and good learning rate schedules are often found by years of trial and error. Empirically, the learning rate schedules found by APO outperformed the best fixed learning rates and were competitive with manual step decay schedules.

Our second use case is more ambitious. We use APO to learn a preconditioning matrix, giving the update rule the flexibility to represent second-order optimization updates such as Newton's method, Gauss-Newton, or natural gradient descent. We show that, under certain conditions, the optimum of our APO meta-objective with respect to a full preconditioning matrix coincides with damped versions of natural gradient descent or Gauss-Newton. While computing and storing a full preconditioning matrix for a large neural network is impractical, various practical approximations have been developed. We use APO to meta-learn a structured preconditioning matrix based on the EKFAC optimizer [23]. APO is more straightforward to implement in current-day deep learning frameworks than EKFAC and is also more computationally efficient per iteration because it avoids the need to compute eigen-decompositions. Empirically, we evaluate APO for learning structured preconditioners on regression, image reconstruction, image classification, and neural machine translation tasks. The preconditioning matrix adapted by APO achieved competitive convergence to other second-order optimizers.

2 Proximal Optimization and 2nd-Order Methods: A Unifying Framework

We first motivate the proximal objective that we use as the meta-objective for APO, and relate it to existing neural network optimization methods. Our framework is largely based on Grosse [24], to which readers are referred for a more detailed discussion.

Preliminaries. Consider a prediction problem with a finite training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)})\}_{i=1}^{N}$. For a data point $(\mathbf{x}, \mathbf{t})$ and parameters $\boldsymbol{\theta} \in \mathbb{R}^m$, let $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta})$ be the prediction of a network parameterized by $\boldsymbol{\theta}$ and $\mathcal{L}(\mathbf{y}, \mathbf{t})$ be the loss. Our goal is to minimize the cost function $\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{t}^{(i)})$. We use $\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$ to denote the mean loss on a mini-batch of examples $\mathcal{B} = \{(\mathbf{x}^{(i)}, \mathbf{t}^{(i)})\}_{i=1}^{B}$. We summarize our notation in Appendix A.

2.1 Proximal Optimization

When we update the parameters on a mini-batch of data, we would like to reduce the loss on that mini-batch, while not changing the predictions on previously visited examples or moving too far in weight space. This motivates the following proximal point update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \operatorname*{arg\,min}_{\mathbf{u} \in \mathbb{R}^m} \mathcal{J}_{\mathcal{B}^{(t)}}(\mathbf{u}) + \lambda_{\text{FSD}} \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[D_{\text{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}})] + \lambda_{\text{WSD}} D_{\text{W}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}), \tag{2}$$

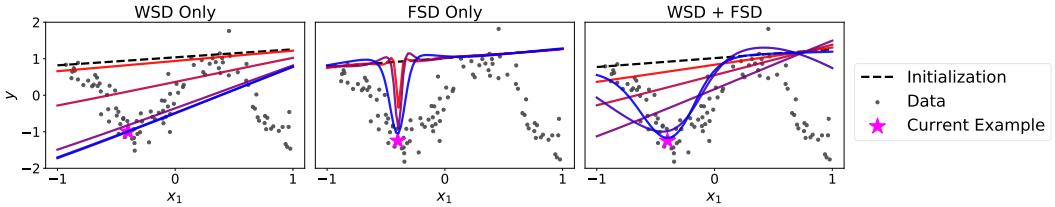


Figure 1: 1D illustration of the exact proximal update on a regression problem with a batch size of 1, inspired by Grosse [24]. The weight of the discrepancy term(s) (λ_{WSD} and λ_{FSD}) is decreased from **red** to **blue**.

where $D_{\text{F}}(\mathbf{u}, \boldsymbol{\theta}, \mathbf{x}) = \rho(f(\mathbf{x}, \mathbf{u}), f(\mathbf{x}, \boldsymbol{\theta}))$ and $D_{\text{W}}(\mathbf{u}, \boldsymbol{\theta}) = {}^1\!/_2\|\mathbf{u} - \boldsymbol{\theta}\|_2^2$ are discrepancy functions (described below), λ_{FSD} and λ_{WSD} are hyperparameters that control the strength of each discrepancy term, $\tilde{\mathbf{x}}$ is a sample from the data-generating distribution $\mathcal{D}$, and $\rho(\cdot, \cdot)$ is the output-space divergence.

The proximal objective in Eq. 2 consists of three terms. The first term is the loss on the current mini-batch. The second term is the *function space discrepancy (FSD)*, whose role is to prevent the update from substantially altering the predictions on other data points. The general idea of the FSD term has been successful in alleviating catastrophic forgetting [10], fine-tuning pre-trained models [34], and training a student model from a teacher network [29].

The final term is the *weight space discrepancy (WSD)*, which encourages the update to move the parameters as little as possible. It can be used to motivate damping in the context of second-order optimization [52]. While weight space distance may appear counterproductive from an optimization standpoint because it depends on the model parameterization, analyses of neural network optimization and generalization often rely on network parameters staying close to their initialization in the Euclidean norm [32, 87, 8]. In fact, Wadia et al. [82] showed that pure second-order optimizers (i.e. ones without WSD regularization) are unable to generalize in the overparameterized setting.

Figure 1 illustrates the effects of the WSD and FSD terms on the exact PPM update for a 1D regression example with a batch size of 1. If the proximal objective includes only the loss and WSD term (i.e. $\lambda_{\text{FSD}} = 0$), the PPM update makes the minimal change to the weights which fits the current example, resulting in a global adjustment to the function which overwrites all the other predictions. If only the loss and FSD terms are used (i.e. $\lambda_{\text{WSD}} = 0$), the update carves a spike around the current data point, failing to improve predictions on nearby examples and running the risk of memorization. When both WSD and FSD are penalized, it makes a local adjustment to the predictions, but one which nonetheless improves performance on nearby examples.

2.2 Connection Between Proximal Optimization and Second-Order Optimization

We further motivate our proximal objective by relating it to existing neural network optimizers. Ordinary SGD can be viewed as an approximate PPM update with a first-order approximation to the loss term and no FSD term. Hessian-Free optimization [50], a classic second-order optimization method for neural networks, approximately minimizes a second-order approximation to the loss on each batch using conjugate gradients. It can be seen as minimizing a quadratic approximation to Eq. 2 with no FSD term. Amari [1] motivated natural gradient descent (NGD) as a steepest descent method with an infinitesimal step size; this justifies a first-order approximation to the loss term and a second-order approximation to the proximity term. Optimizing over a manifold of probability distributions with KL divergence as the proximity term yields the familiar update involving the Fisher information matrix. Natural gradient optimization of neural networks [51, 53] can be interpreted as minimizing Eq. 2 with a linear approximation to the loss term and a quadratic approximation to the FSD term. While NGD traditionally omits the WSD term in order to achieve parameterization invariance, it is typically included in practical neural network optimizers for stability [53].

Method	Loss Approx.	FSD	WSD
Gradient Descent	1st-order	-	✓
Hessian-Free	2nd-order	-	✓
Natural Gradient	1st-order	2nd-order	✗
Proximal Point	Exact	Exact	✓

Table 1: Classical 1st and 2nd optimization algorithms interpreted as minimizing approximations of the proximal objective in Eq. 2, using 1st or 2nd order Taylor expansions of the loss or FSD terms.

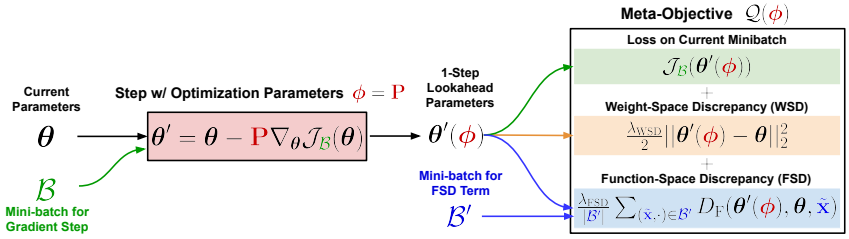


Figure 2: **Amortized Proximal Optimization (APO)**. In each meta-optimization step, we perform a one-step lookahead from the current parameters $\boldsymbol{\theta}$ to obtain updated parameters $\boldsymbol{\theta}'(\boldsymbol{\phi})$, where $\boldsymbol{\phi}$ denotes the optimization parameters (e.g. learning rate η or preconditioner $\mathbf{P}$). The meta-objective $\mathcal{Q}(\boldsymbol{\phi})$ then evaluates the proximal point objective at $\boldsymbol{\theta}'(\boldsymbol{\phi})$. Note that the loss term in $\mathcal{Q}(\boldsymbol{\phi})$ is computed on the same data that was used to compute the gradient for the lookahead step, $\mathcal{B}$, while the FSD term is computed using a different datapoint $(\tilde{\mathbf{x}}, \tilde{\mathbf{t}}) \sim \mathcal{D}_{\text{train}}$. The optimization parameters $\boldsymbol{\phi}$ are updated via the meta-gradient $\nabla_{\boldsymbol{\phi}}\mathcal{Q}(\boldsymbol{\phi})$.

Algorithm 1 Amortized Proximal Optimization (APO) — Meta-Learning Optimization Parameters $\boldsymbol{\phi}$

Require: $\boldsymbol{\theta}$ (initial model parameters), $\boldsymbol{\phi}$ (initial optimization parameters), K (meta-update interval), α (meta-LR)
Require: λ_{WSD} (weight-space discrepancy term weighting), λ_{FSD} (function-space discrepancy term weighting)
while not converged, iteration t **do**
  $\mathcal{B} \sim \mathcal{D}_{\text{train}}$ ▷ Sample mini-batch to compute the gradient and loss term
  if $t \mod K = 0$ **then** ▷ Perform meta-update every K iterations
    $\mathcal{B}' \sim \mathcal{D}_{\text{train}}$ ▷ Sample additional mini-batch to compute the FSD term
    $\boldsymbol{\theta}'(\boldsymbol{\phi}) := u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})$ ▷ Compute the 1-step lookahead parameters
    $\mathcal{Q}(\boldsymbol{\phi}) := \mathcal{J}_{\mathcal{B}}\left(\boldsymbol{\theta}'(\boldsymbol{\phi})\right) + {}^{\lambda_{\text{FSD}}}/_{|\mathcal{B}'|}\sum_{(\tilde{\mathbf{x}},\cdot)\in\mathcal{B}'} D_{\text{F}}(\boldsymbol{\theta}'(\boldsymbol{\phi}), \boldsymbol{\theta}, \tilde{\mathbf{x}}) + {}^{\lambda_{\text{WSD}}}/_{2}\left\|\boldsymbol{\theta}'(\boldsymbol{\phi}) - \boldsymbol{\theta}\right\|_2^2$
    ▷ Compute meta-objective
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha\nabla_{\boldsymbol{\phi}}\mathcal{Q}(\boldsymbol{\phi})$ ▷ Update optimizer parameters (e.g. LR or preconditioner)
  end if
  $\boldsymbol{\theta} \leftarrow u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})$ ▷ Update model parameters
end while

In a more general context, when taking a first-order approximation to the loss and a second-order approximation to the FSD, the update rule is given in closed form as:

$$\boldsymbol{\theta}^{(t+1)} \approx \boldsymbol{\theta}^{(t)} - (\lambda_{\text{FSD}}\mathbf{G} + \lambda_{\text{WSD}}\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}), \tag{3}$$

where $\mathbf{G}$ is the Hessian of the FSD term. The derivation is shown in Appendix F. All of these relationships are summarized in Table 1, and derivations of all of these claims are given in Appendix G.

3 Amortized Proximal Optimization

In this section, we introduce Amortized Proximal Optimization (APO), an approach for online meta-learning of optimization parameters. Then, we describe two use cases that we explore in the paper: (1) adapting learning rates of existing base optimizers such as SGD, RMSProp, and Adam, and (2) meta-learning a structured preconditioning matrix.

3.1 Proximal Meta-Optimization

We assume an update rule u parameterized by a vector $\boldsymbol{\phi}$ which updates the network weights $\boldsymbol{\theta}$ on a batch $\mathcal{B}^{(t)}$:[1] $\boldsymbol{\theta}^{(t+1)} \leftarrow u(\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}, \mathcal{B}^{(t)})$. One use case of APO is to tune the hyperparameters of an existing optimizer, in which case $\boldsymbol{\phi}$ denotes the hyperparameters. For example, when tuning the SGD learning rate, we have $\boldsymbol{\phi} = \eta$ and the update is given by: $u_{\text{SGD}}(\boldsymbol{\theta}, \eta, \mathcal{B}) = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$. More ambitiously, we could use APO to adapt a full preconditioning matrix $\mathbf{P}$. In this case, we define $\boldsymbol{\phi} = \mathbf{P}$ and the update is given by: $u_{\text{Precond}}(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) = \boldsymbol{\theta} - \mathbf{P}\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$.

[1] The update rule may also depend on state maintained by the optimizer, such as the second moments in RMSprop [77]. This state is treated as fixed by APO, so we suppress it to avoid clutter.

In Section 2, we introduced a general proximal objective for training neural networks and observed that many optimization techniques could be seen as an approximation of PPM. Motivated by this connection, we propose to directly minimize the proximal objective with respect to the optimization parameters. As shown in Table 1, under linear and quadratic assumptions on the loss term and FSD term, the proximal objective in Eq. 1 has a closed-form solution; however, when these assumptions do not hold, direct minimization of the proximal objective can still be robust. Adapting a parametric update rule allows us to take advantage of the properties of existing optimizers, and also amortizes the cost of minimizing the proximal objective over the course of training.

We propose to use the following meta-objective, which evaluates the proximal objective at $u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})$:

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\Big[\mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}\left[D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})\right] + \frac{\lambda_{\text{WSD}}}{2}\|u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}\|^2\Big]. \tag{4}$$

In practice, we estimate the expectations in the meta-objective by sampling two different mini-batches, $\mathcal{B}$ and $\mathcal{B}'$, where $\mathcal{B}$ is used to compute the gradient and the loss term, and $\mathcal{B}'$ is used to compute the FSD term. Intuitively, this proximal meta-objective aims to find optimizer parameters $\boldsymbol{\phi}$ that minimize the loss on the current mini-batch, while constraining the size of the step with the FSD and WSD terms so that it does not overfit the current mini-batch and undo progress that has been made by other mini-batches. The optimization parameters $\boldsymbol{\phi}$ are optimized with a stochastic gradient-based algorithm (the *meta-optimizer*). The meta-gradient $\nabla_{\boldsymbol{\phi}}\mathcal{Q}(\boldsymbol{\phi})$ can be computed via automatic differentiation through the one-step unrolled computation graph (Figure 2). We refer to our framework as Amortized Proximal Optimization (APO, Algorithm 1).

3.2 APO for Learning Rate Adaptation

One use case of APO is to adapt the learning rate of an existing base optimizer such as SGD. To do so, we let $u_{\text{SGD}}(\boldsymbol{\theta}, \eta, \mathcal{B})$ be the 1-step lookahead of parameters and minimize the proximal meta-objective with respect to the learning rate η. Although adaptive optimizers such as RMSProp and Adam use coordinate-wise learning rates, they still have a global learning rate which is essential to tune. APO can be applied to such global learning rates to find learning rate schedules.

3.3 APO for Adaptive Preconditioning

More ambitiously, we can use the APO framework to adapt the preconditioning matrix, allowing the update rule to flexibly represent various second-order optimization updates. We let $u_{\text{Precond}}(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B})$ denote the parameters after 1 preconditioned gradient step and adapt the preconditioning matrix $\mathbf{P}$ according to our framework. If the assumptions made when deriving the second-order methods (detailed in Section 2.2) hold, then the optimal preconditioning matrix is equivalent to various second-order updates, depending on the choice of the FSD function.

Theorem 1. *Consider an approximation $\hat{\mathcal{Q}}(\mathbf{P})$ to the meta-objective (Eq. 4) where the loss term is linearized around the current weights $\boldsymbol{\theta}$ and the FSD term is replaced by its second-order approximation around $\boldsymbol{\theta}$. Denote the gradient on a mini-batch as $\mathbf{g} = \nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$, and assume that the second moment matrix $\mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\left[\mathbf{g}\mathbf{g}^\top\right]$ is non-singular. Then, the preconditioning matrix which minimizes $\hat{\mathcal{Q}}$ is given by $\mathbf{P}^\star = (\lambda_{FSD}\mathbf{G} + \lambda_{WSD}\mathbf{I})^{-1}$, where $\mathbf{G}$ denotes the Hessian of the FSD evaluated at $\boldsymbol{\theta}$.*

The proof is provided in Appendix H. As an important special case, when the FSD term is derived from the KL divergence between distributions in output space, $\mathbf{G}$ coincides with the Fisher information matrix $\mathbf{F} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim P_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}}\log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})^\top\right]$, where $P_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\theta})$ denotes the model's predictive distribution over $\mathbf{y}$. Therefore, the optimal preconditioner is the damped natural gradient preconditioner, $\mathbf{P}^\star = (\mathbf{F} + \lambda_{\text{WSD}}\mathbf{I})^{-1}$. Thus, when APO is used to exactly minimize an approximate meta-objective, the update it yields coincides with classical second-order optimization algorithms, depending on the choice of the FSD function.

3.4 Structured Preconditioner Adaptation

In the previous sections, the discussion assumed a full preconditioning matrix for simplicity. However, a full preconditioner is impractical to represent for modern neural networks. Moreover, for practical

stability of the learned preconditioned update, we would like to enforce the preconditioner to be positive semidefinite (PSD) so that the transformed gradient is a descent direction [63].

To satisfy these requirements, we adopt a structured preconditioner analogous to that of the EKFAC optimizer [23]. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{m_i \times m_{i+1}}$ of a layer, we construct the preconditioning matrix as a product of smaller matrices:

$$\mathbf{P}_{\mathrm{S}} = (\mathbf{A} \otimes \mathbf{B})\mathrm{diag}(\mathrm{vec}(\mathbf{S}))^2(\mathbf{A} \otimes \mathbf{B})^\top, \tag{5}$$

where $\mathbf{A} \in \mathbb{R}^{m_{i+1} \times m_{i+1}}$, $\mathbf{B} \in \mathbb{R}^{m_i \times m_i}$, and $\mathbf{S} \in \mathbb{R}^{m_i \times m_{i+1}}$ are small block matrices. Here, $\otimes$ denotes the Kronecker product, $\mathrm{diag}(\cdot)$ denotes the diagonalization operator, and $\mathrm{vec}(\cdot)$ denotes the vectorization operator. This parameterization is memory efficient: it requires $m_i^2 + m_{i+1}^2 + m_i m_{i+1}$ parameters to store, as opposed to $m_i^2 m_{i+1}^2$ parameters for a full preconditioning matrix. It is straightforward to show that the structured preconditioner in Eq. 5 is PSD, as it takes the form $\mathbf{CDC}^\top$, where $\mathbf{D}$ is PSD. The preconditioned gradient can be computed efficiently by using the properties of the Kronecker product: $\mathbf{P}_{\mathrm{S}}\mathrm{vec}(\nabla_{\mathbf{W}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})) = \mathrm{vec}(\mathbf{B}(\mathbf{S}^2 \odot \mathbf{B}^\top \nabla_{\mathbf{W}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})\mathbf{A})\mathbf{A}^\top)$, where $\odot$ denotes elementwise multiplication. This is tractable as it only requires four additional matrix multiplications and elementwise multiplication of small block matrices in each layer when updating the parameters. While EKFAC uses complicated covariance estimation and eigenvalue decomposition to construct the block matrices, in APO, we meta-learn these block matrices directly, where $\boldsymbol{\phi} = [\mathrm{vec}(\mathbf{A})^\top, \mathrm{vec}(\mathbf{B})^\top, \mathrm{vec}(\mathbf{S})^\top]^\top$. As APO does not require inverting (or performing eigendecompositions of) the block matrices, our structured representation incurs less computation per iteration than EKFAC.

While we defined an optimizer with the same functional form as EKFAC, it is not immediately obvious whether the preconditioner which is actually learned by APO will be at all similar. A Corollary of Theorem 1 shows that if certain conditions are satisfied, including the assumptions underlying KFAC [53], then the structured preconditioner minimizing Eq. 4 coincides with KFAC:

Corollary 2. *Suppose that (1) the assumptions for Theorem 1 are satisfied, (2) the FSD term measures the KL divergence, and (3) $\lambda_{WSD} = 0$ and $\lambda_{FSD} = 1$. Moreover, suppose that the parameters $\boldsymbol{\theta}$ satisfy the KFAC assumptions listed in Appendix I. Then, the optimal solution to the approximate meta-objective recovers the KFAC update, which can be represented using Eq. 5.*

The proof is in Appendix I. If the KFAC assumptions are not satisfied, then APO will generally learn a different preconditioner. This may be desirable, especially if differing probabilistic assumptions lead to different update rules, as is the case for KFAC applied to convolutional networks [25, 37].

3.5 Computation and Memory Cost

Computation Cost. Computing the FSD term requires sampling an additional mini-batch from the training set and performing two additional forward passes for $f(\tilde{\mathbf{x}}, \boldsymbol{\theta})$ and $f(\tilde{\mathbf{x}}, u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}))$. Combined with the loss term evaluated on the original mini-batch, one meta-optimization step requires three forward passes to compute the proximal objective. It additionally requires a backward pass through the 1-step unrolled computation graph (Figure 2) to compute the gradient of the proximal meta-objective $\mathcal{Q}(\boldsymbol{\phi})$ with respect to $\boldsymbol{\phi}$. This overhead can be reduced by performing a meta-update only once every K iterations: the overhead will consist of $3/K$ additional forward passes and $1/K$ additional backward passes per iteration, which is small for modest values of K (e.g. $K = 10$).

Memory Cost. APO requires twice the model memory for the 1-step unrolling when computing the proximal meta-objective. In the context of structured preconditioner adaptation, we further need to store block matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{S}$ (in Eq. 5) for each layer, as in KFAC and EKFAC.

4 Related Work

Gradient-Based Learning Rate Adaptation. Maclaurin et al. [49] backpropagate through the full unrolled training procedure to meta-optimize learning rate schedules *offline*. This is expensive, as it requires completing a full training run to make a single hyperparameter update. A related approach is to unroll the optimization for a small number of steps and perform truncated backpropagation [15, 21]. Hypergradient descent [5] adapts the learning rate to minimize the expected loss in the next iteration.

Second-Order Optimization. Although preconditioned methods have better convergence rates than first-order methods [7, 63], storing and computing the inverse of preconditioning matrices is impractical for high-dimensional problems. To mitigate these computational issues, Hessian-free optimization [50, 54] approximates Newton's update by only accessing the curvature matrix through Hessian-vector products. Other works impose a structure on the preconditioner by representing it as a Kronecker product [53, 25, 55, 23, 26, 76], a diagonal matrix [18, 35], or a low-rank matrix [38, 59]. However, these approximate second-order methods may not be easy to implement in deep learning frameworks, and can still be expensive as they often require matrix inversion or eigendecomposition.

Gradient-Based Preconditioner Adaptation. Moskovitz et al. [60] learn the preconditioning matrix with hypergradient descent. Meta-curvature [65] and warped gradient descent [40, 20] adapt the preconditioning matrix that yields effective parameter updates across diverse tasks in the context of few-shot learning. In Appendix B, we provide extended related work, and we provide a conceptual comparison of meta-optimization methods in Table 6.

5 Experiments

Our experiments investigate the following questions: (1) How does the structured preconditioning matrix adapted by APO perform in comparison to existing first- and second-order optimizers? (2) How does the learning rate adapted by APO perform compared to optimal fixed learning rates and manual decay schedules commonly used in the literature?

We used APO to meta-learn the preconditioning matrices for a broad range of tasks, including several regression datasets, autoencoder training, image classification on CIFAR-10 and CIFAR-100 using several network architectures, neural machine translation using transformers, and low-precision (16-bit) training. Several of these tasks are particularly challenging for first-order optimizers. In addition, we used APO to tune the global learning rate for multiple base optimizers – SGD, SGD with momentum (denoted SGDm), RMSprop, and Adam – on CIFAR-10/100 classification with several network architectures. We denote our method that adapts the structured preconditioning matrix as "APO-Precond". The method that tunes the global learning rate of a base optimizer is denoted as "Base-APO" (e.g. SGDm-APO). Experiment details and additional experiments, including ablation studies, are given in Appendix C and J, respectively.

5.1 Meta-Learning Preconditioners

Poorly-Conditioned Regression. We fist considered a regression task traditionally used to illustrate failures of neural network optimization [67]. The targets are given by $\mathbf{t} = \mathbf{A}\mathbf{x}$, where $\mathbf{A}$ is an ill-conditioned matrix with $\kappa(\mathbf{A}) = 10^{10}$. We trained a two-layer linear network $f(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{W}_2\mathbf{W}_1\mathbf{x}$ and minimized the objective $\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|\mathbf{A}\mathbf{x} - \mathbf{W}_2\mathbf{W}_1\mathbf{x}\|^2\right]$. In Figure 4 (left), we show training curves for SGDm, Adam, Shampoo [26], KFAC, and APO-Precond. As the problem is ill-conditioned, 2nd-order optimizers such as Shampoo and KFAC converge faster than 1st-order methods. APO-Precond performs comparably to 2nd-order optimizers with lower loss than KFAC.

UCI Regression. Next, we validated APO-Precond on the Slice, Protein, and Parkinsons datasets from the UCI regression collection [17]. We trained a 2-layer MLP with 100 hidden units per layer and ReLU activations for 500 epochs. The training curves for each optimizer are shown in Figure 3. We found that APO-Precond consistently achieved competitive convergence compared to other second-order optimizers and reached lower training loss than all baselines.

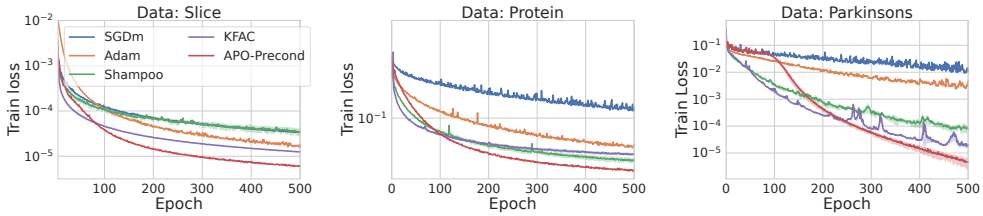


Figure 3: A comparison of SGDm, Adam, KFAC, Shampoo, and APO-Precond on UCI regression tasks. Across all tasks, APO-Precond achieves lower loss with competitive convergence compared to second-order optimizers.

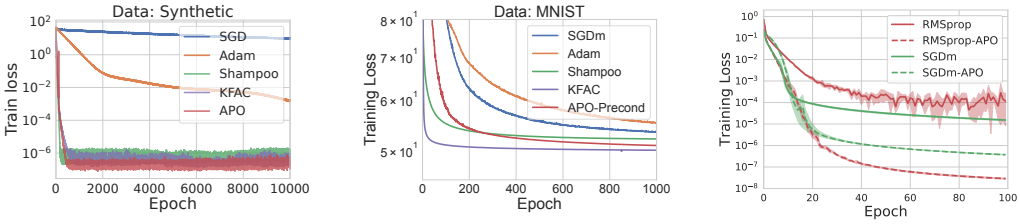


Figure 4: **Left:** Synthetic data for poorly-conditioned regression; **Middle:** Deep autoencoder on MNIST; **Right:** Tuning the global learning rate with APO—we show the training loss for a two-layer MLP trained on MNIST, using SGDm and RMSprop (solid lines), and their APO-tuned variants (dashed lines).

Image Reconstruction. We trained an 8-layer autoencoder on MNIST [39]; this is known to be a challenging optimization task for first-order optimizers [30, 54, 53]. We followed the experimental set-up from Martens & Grosse [53], where the encoder and decoder consist of 4 fully-connected layers with sigmoid activation. The decoder structure is symmetric to that of the encoder, and they do not have tied weights. The logistic activation function and the presence of a bottleneck layer make this a challenging optimization problem compared with most current-day architectures. We compare APO-Precond with SGDm, Adam, Shampoo, and KFAC optimizers and show the training losses for each optimizer in Figure 4 (middle). APO-Precond converges faster than first-order methods and achieves competitive training loss to other second-order methods (although there remains a performance gap compared with KFAC).

Image Classification. To investigate whether adapting the preconditioner with APO improves generalization, we conducted image classification experiments on CIFAR-10 and CIFAR-100. We trained LeNet [39], AlexNet [36], VGG-16 [70] (w/o batch norm [31]), ResNet-18, and ResNet-32 [28] architectures for 200 epochs on batches of 128 images. The test accuracies for SGDm, Adam, KFAC, and APO-Precond are shown in Table 2. We found that APO-Precond achieved competitive generalization performance to SGDm and KFAC. In particular, for architectures without batch normalization (LeNet, AlexNet, and VGG-16), APO-Precond improved the test accuracy substantially.

Task	Model	SGDm	Adam	KFAC	APO-P
C-10	**LeNet**	75.73	73.41	76.63	**77.42**
C-10	**AlexNet**	76.27	76.09	78.33	**81.14**
C-10	**VGG16**	91.82	90.19	92.05	**92.13**
C-10	**ResNet18**	93.69	93.27	94.60	**94.75**
C-10	**ResNet32**	94.40	93.30	94.49	**94.83**
C-100	**AlexNet**	43.95	41.82	46.24	**52.35**
C-100	**VGG16**	65.98	60.61	61.84	**67.95**
C-100	**ResNet18**	76.85	70.87	76.48	**76.88**
C-100	**ResNet32**	**77.47**	56.58	75.70	77.41
SVHN	**ResNet18**	96.19	95.59	96.08	**96.89**
IWSLT14	**Transformer**	31.43	34.60	-	**34.62**

Table 2: Test accuracy on CIFAR-10 ("C-10"), CIFAR-100 ("C-100"), and BLEU score on the WMT'14 Ge→En for APO-Precond ("APO-P").

Neural Machine Translation. We also applied APO-Precond to the IWSLT'14 German-to-English translation task [14]. We used a Transformer [78] composed of 6 encoder and decoder layers, with word embedding and hidden vector dimensionality 512. We compared APO-Precond to SGDm and AdamW [47]. In Table 2, we show the final test BLEU score for SGDm, AdamW, and APO-Precond. While keeping the learning rate fixed, we achieved a BLEU score competitive with AdamW.

Low Precision Training. Low precision training presents a challenge for second-order optimizers such as KFAC which rely on matrix inverses that may be sensitive to quantization noise. We trained LeNet and ResNet-18 with 16-bit floating-point arithmetic to examine if APO-Precond is applicable in training the networks in lower precision. We used the experimental setup from Section 5.1 but stored parameters, activations, and gradients in 16-bit precision. We found that KFAC required a large damping factor to maintain stability, and this prevented it from fully utilizing curvature information. In contrast, as APO-Precond does not require matrix inversion, it remained stable with the same choice of FSD and WSD weights we used in the full precision experiments. The final test accuracies on ResNet-18 for SGDm, KFAC, and APO-Precond are shown in Table 3.

Task	Model	SGDm	KFAC	APO-P
CIFAR-10	**LeNet**	75.65	74.95	**77.25**
CIFAR-10	**ResNet-18**	94.15	92.72	**94.79**
CIFAR-100	**ResNet-18**	73.53	73.12	**75.47**

Table 3: Test acc. of 16-bit nets on CIFAR 10/100.

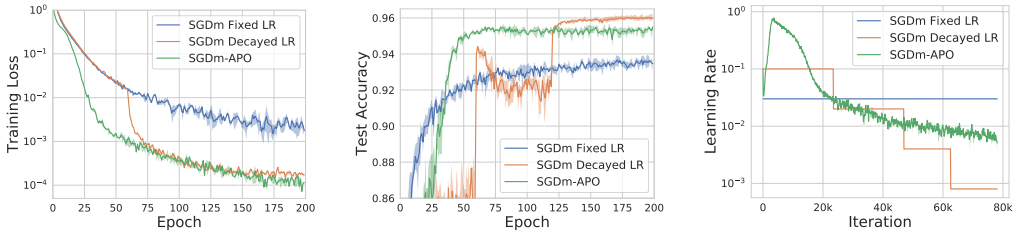


Figure 5: **WideResNet 28-10 on CIFAR-10, using SGD with momentum (SGDm).** We compare the training loss (**left**), test accuracy (**middle**), and learning rate schedules (**right**) of baselines with fixed and decayed learning rates, and APO. The training loss plot uses hyperparameters chosen based on training loss, while the middle and right plots use hyperparameters chosen based on validation accuracy. The shaded regions show the min/max values over 4 random restarts.

	C-10, ResNet-32			C-10, ResNet-34			C-10, WRN 28-10			C-100, WRN 28-10		
	Fixed	**Decay**	**APO**	**Fixed**	**Decay**	**APO**	**Fixed**	**Decay**	**APO**	**Fixed**	**Decay**	**APO**
SGD	90.07	93.30	92.71	93.00	93.54	94.27	93.38	94.86	94.85	76.29	77.92	76.87
SGDm	89.40	93.34	92.75	92.99	95.08	94.47	93.46	95.98	95.50	74.81	81.01	79.33
RMSprop	89.84	91.94	91.28	92.87	93.87	93.97	92.91	93.60	94.22	72.06	76.06	74.17
Adam	90.45	92.26	91.81	93.23	94.12	93.80	92.81	94.04	93.83	72.01	75.53	76.33

Table 4: **Tuning the global LR for CIFAR-10 ("C-10") and CIFAR-100 ("C-100"):** We compare the test accuracies achieved by the optimal fixed learning rate, the manual step decay schedule, and the APO-adapted schedule, using ResNet-32 [28], ResNet-34, and WideResNet 28-10 [85]. Results are the mean of four random restarts. APO outperforms optimal fixed learning rates, and is often competitive with manual schedules. APO generally achieves test accuracy comparable to manual schedules in fewer training iterations (App. D).

5.2 Meta-Learning Learning Rates

Image Classification on MNIST. First, we compared SGDm and RMSprop to their APO-tuned variants to train an MLP on MNIST. We used a two-layer MLP with 1000 hidden units per layer and ReLU nonlinearities, and trained on mini-batches of size 100 for 100 epochs. Figure 4 (Right) shows the training loss achieved by each approach; we found that for both base optimizers, APO improved convergence speed and obtained substantially lower loss than the baselines.

Image Classification on CIFAR-10 & CIFAR-100. For learning rate adaptation on CIFAR-10, we experimented with three network architectures: ResNet32 [28], ResNet34, and WideResNet (WRN-28-10) [85]. For ResNet32, we trained for 400 epochs, and the decayed baseline used a step schedule with $10\times$ decay at epochs 150 and 250, following [48]. For ResNet34 and WRN-28-10, we trained for 200 epochs, and the decayed baseline used a step schedule with $5\times$ decay at epochs 60, 120, and 160, following [85]. For CIFAR-100, we used WRN-28-10 with the same schedule as for CIFAR-10. For each of the base optimizers, we compared APO to (1) the optimal fixed learning rate and (2) a manual step learning rate decay schedule. The test accuracies for each base optimizer and their APO-tuned variants are shown in Table 4. In addition, Figure 5 shows the training loss, test accuracy and learning rate adaptation for WRN-28-10 on CIFAR-10, using SGDm as the base optimizer. Using APO to tune the global learning rate yields higher test accuracy than the best fixed learning rate, and is competitive with the manual schedule.

6 Conclusion

We introduced Amortized Proximal Optimization (APO), a framework for online meta-learning of optimization parameters which approximates the proximal point method by learning a parametric update rule. As the meta-parameters are updated only once per K steps of optimization, APO incurs minimal computational overhead. We applied APO to two settings: (1) meta-learning the global learning rate for existing base optimizers (e.g., SGD, RMSprop, and Adam); and (2) meta-learning structured preconditioning matrices, which provides a new approach to 2nd-order optimization. Compared to methods such as KFAC, APO eliminates the need to compute matrix inverses, yielding improved efficiency and numerical stability. On a range of tasks, we showed that APO is competitive with 2nd-order methods, and improves generalization compared to baseline 1st- and 2nd-order optimizers.

References

[1] Amari, S.-I. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] 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.

[3] Arora, S., Li, Z., and Lyu, K. Theoretical analysis of auto rate-tuning by batch normalization. *arXiv preprint arXiv:1812.03981*, 2018.

[4] Asi, H. and Duchi, J. C. Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*, 29(3):2257–2290, 2019.

[5] Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

[6] Beck, A. and Teboulle, M. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

[7] Becker, S. and Le Cun, Y. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 Connecticut Summer School*, 1988.

[8] Belkin, M., Hsu, D., Ma, S., and Mandal, S. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[9] Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pp. 437–478. Springer, 2012.

[10] Benjamin, A. S., Rolnick, D., and Kording, K. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

[11] Benzing, F., Gauy, M. M., Mujika, A., Martinsson, A., and Steger, A. Optimal Kronecker-sum approximation of real time recurrent learning. In *International Conference on Machine Learning*, pp. 604–613, 2019.

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

[13] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: Composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[14] Cettolo, M., Niehues, J., Stüker, S., Bentivogli, L., and Federico, M. Report on the 11th IWSLT evaluation campaign. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, volume 57, 2014.

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

[16] Donini, M., Franceschi, L., Pontil, M., Majumder, O., and Frasconi, P. MARTHE: Scheduling the learning rate via online hypergradients. *arXiv preprint arXiv:1910.08525*, 2019.

[17] Dua, D. and Graff, C. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[18] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[19] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

[20] Flennerhag, S., Rusu, A. A., Pascanu, R., Visin, F., Yin, H., and Hadsell, R. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

[21] Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pp. 1165–1173, 2017.

[22] Gao, B., Gouk, H., Lee, H. B., and Hospedales, T. M. Meta mirror descent: Optimiser learning for fast convergence. *arXiv preprint arXiv:2203.02711*, 2022.

[23] George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast approximate natural gradient descent in a Kronecker-factored eigenbasis. *arXiv preprint arXiv:1806.03884*, 2018.

[24] Grosse, R. University of Toronto CSC2541, Topics in Machine Learning: Neural Net Training Dynamics, Chapter 4: Second-Order Optimization. Lecture Notes, 2021. URL `https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2021/readings/L04_second_order.pdf`.

[25] Grosse, R. and Martens, J. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

[26] Gupta, V., Koren, T., and Singer, Y. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850, 2018.

[27] Hazan, E. and Kakade, S. Revisiting the polyak step size. *arXiv preprint arXiv:1905.00313*, 2019.

[28] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[29] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[30] Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[31] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[32] Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[33] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. Population-based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[34] Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Zhao, T. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

[35] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[36] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[37] Laurent, C., George, T., Bouthillier, X., Ballas, N., and Vincent, P. An evaluation of Fisher approximations beyond Kronecker factorization, 2018. URL `https://openreview.net/forum?id=ryVC6tkwG`.

[38] Le Roux, N., Manzagol, P.-A., and Bengio, Y. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems*, pp. 849–856, 2007.

[39] LeCun, Y. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1988.

[40] Lee, Y. and Choi, S. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, pp. 2927–2936, 2018.

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

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

[43] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

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

[45] Loizou, N., Vaswani, S., Laradji, I. H., and Lacoste-Julien, S. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. In *International Conference on Artificial Intelligence and Statistics*, pp. 1306–1314. PMLR, 2021.

[46] Lorraine, J., Vicol, P., and Duvenaud, D. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, 2020.

[47] Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[48] Lucas, J., Sun, S., Zemel, R., and Grosse, R. Aggregated momentum: Stability through passive damping. In *International Conference on Learning Representations*, 2019.

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

[50] Martens, J. Deep learning via Hessian-free optimization. In *International Conference on Machine Learning*, 2010.

[51] Martens, J. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[52] Martens, J. *Second-order optimization for neural networks. PhD Thesis.* University of Toronto, 2016.

[53] Martens, J. and Grosse, R. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pp. 2408–2417, 2015.

[54] Martens, J. and Sutskever, I. Learning recurrent neural networks with Hessian-free optimization. In *International Conference on Machine Learning*, 2011.

[55] Martens, J., Ba, J., and Johnson, M. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[56] Metz, L., Maheswaranathan, N., Nixon, J., Freeman, C. D., and Sohl-Dickstein, J. Learned optimizers that outperform SGD on wall-clock and validation loss. *arXiv preprint arXiv:1810.10180*, 2018.

[57] Micaelli, P. and Storkey, A. Non-greedy gradient-based hyperparameter optimization over long horizons. *arXiv preprint arXiv:2007.07869*, 2020.

[58] Michal, R. and Georg, M. L4: Practical loss-based stepsize adaptation for deep learning. *arXiv preprint arXiv:1802.05074*, 2018.

[59] Mishkin, A., Kunstner, F., Nielsen, D., Schmidt, M., and Khan, M. E. Slang: Fast structured covariance approximations for Bayesian deep learning with natural gradient. *arXiv preprint arXiv:1811.04504*, 2018.

[60] Moskovitz, T., Wang, R., Lan, J., Kapoor, S., Miconi, T., Yosinski, J., and Rawal, A. First-order preconditioning via hypergradient descent. *arXiv preprint arXiv:1910.08461*, 2019.

[61] Mujika, A., Meier, F., and Steger, A. Approximating real-time recurrent learning with random Kronecker factors. *arXiv preprint arXiv:1805.10842*, 2018.

[62] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[63] Nocedal, J. and Wright, S. *Numerical Optimization*. Springer, 1999.

[64] Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[65] Park, E. and Oliva, J. B. Meta-curvature. *arXiv preprint arXiv:1902.03356*, 2019.

[66] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. 2019.

[67] Rahimi, A. and Recht, B. Reflections on random kitchen sinks, 2017. URL `http://www.argmin.net/2017/12/05/kitchen-sinks/`.

[68] Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[69] Rolinek, M. and Martius, G. L4: Practical loss-based stepsize adaptation for deep learning. In *Advances in Neural Information Processing Systems*, pp. 6433–6443, 2018.

[70] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[71] Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

[72] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, P., and Adams, R. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pp. 2171–2180, 2015.

[73] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063, 2000.

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

[75] Tallec, C. and Ollivier, Y. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017.

[76] Tang, Z., Jiang, F., Gong, M., Li, H., Wu, Y., Yu, F., Wang, Z., and Wang, M. SKFAC: Training neural networks with faster Kronecker-factored approximate curvature. In *Conference on Computer Vision and Pattern Recognition*, pp. 13479–13487, 2021.

[77] Tieleman, T. and Hinton, G. Lecture 6.5—RMSprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[78] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[79] Vaswani, S., Mishkin, A., Laradji, I., Schmidt, M., Gidel, G., and Lacoste-Julien, S. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, pp. 3727–3740, 2019.

[80] Vaswani, S., Laradji, I., Kunstner, F., Meng, S. Y., Schmidt, M., and Lacoste-Julien, S. Adaptive gradient methods converge faster with over-parameterization (but you should do a line-search). *arXiv preprint arXiv:2006.06835*, 2020.

[81] Vicol, P., Metz, L., and Sohl-Dickstein, J. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning*, pp. 10553–10563, 2021.

[82] Wadia, N., Duckworth, D., Schoenholz, S. S., Dyer, E., and Sohl-Dickstein, J. Whitening and second order optimization both make information in the dataset unusable during training, and can reduce or prevent generalization. In *International Conference on Machine Learning*, pp. 10617–10629, 2021.

[83] 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 *International Conference on Machine Learning*, pp. 3751–3760, 2017.

[84] Wu, Y., Ren, M., Liao, R., and Grosse, R. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

[85] Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[86] Zhang, G., Li, L., Nado, Z., Martens, J., Sachdeva, S., Dahl, G., Shallue, C., and Grosse, R. B. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *Advances in neural information processing systems*, 32:8196–8207, 2019.

[87] Zhang, G., Martens, J., and Grosse, R. Fast convergence of natural gradient descent for overparameterized neural networks. *arXiv preprint arXiv:1905.10961*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]

(c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work is fundamental in nature and does not enable immediate misuse.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] We list the KFAC assumptions in Section I.

(b) Did you include complete proofs of all theoretical results? [Yes] We provide the proof of Theorem 1 in Section H. In Section I, we list the KFAC assumptions and prove Corollary 2. Additionally, in Section F, we provide a derivation of Eq. 3, and in Section G, we derive classic first- and second-order optimization algorithms (gradient descent, Newton's method, and natural gradient) starting from the proximal objective for the PPM (Eq. 2).

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include code in the supplementary material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide the training details for all of our experiments in Appendix C.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For several of our experiments, we plot shaded regions which indicate the min/max values obtained after running the experiments 4 random seeds.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We provide these details in Appendix C.1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cited all datasets and model architectures we used in our experiments (Section 5).

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

This appendix is structured as follows:

- In Section A, we provide an overview of the notation we use throughout the paper.
- In Section B, we provide a discussion of additional related work.
- In Section C, we provide experiment details.
- In Section D, we provide additional experimental results.
- In Section E, we provide an ablation investigating the difference between evaluating the loss term on the same minibatch used to compute the base optimizer step vs a separate randomly-sampled minibatch. Using a separate random minibatch yields a similar meta-objective to that studied in Wu et al. [84], which suffers from short-horizon bias.
- In Section F, we provide a derivation of Eq. 3.
- In Section G, we derive classic first- and second-order optimization algorithms (gradient descent, Newton's method, and natural gradient) starting from the proximal objective for the PPM (Eq. 2).
- In Section H, we provide the proof for Theorem 1, which shows that exactly optimizing the proximal meta-objective with respect to the preconditioner $\mathbf{P}$ can recover various first- and second-order optimization methods.
- In Section I, we list the KFAC assumptions and prove Corollary 2, which shows that under the assumptions of Theorem 1 and the KFAC assumptions, exactly optimizing the proximal meta-objective yields the KFAC update.
- In Section J, we show an ablation over λ_{WSD} and λ_{FSD}, and evaluate how well APO performs with different meta-update intervals.

A Table of Notation

Notation	Description
$\mathcal{D}$	Data-generating distribution
$\mathcal{D}_{\text{train}}$	Finite training dataset
$(\mathbf{x}, \mathbf{t}) \sim \mathcal{D}$	An input/target pair
$\mathcal{B}$	Mini-batch of data
N	Number of training data points, $N = \lvert\mathcal{D}_{\text{train}}\rvert$
$\boldsymbol{\theta}$	Network parameters
$\boldsymbol{\phi}$	Optimization parameters (e.g. learning rate or preconditioning matrix)
$\mathbf{P}$	Preconditioning matrix
$\mathbf{G}$	Hessian of the function-space discrepancy
$\mathbf{W}$	Weight matrix of some particular layer of the network
$\mathbf{P}_{\text{S}}$	Structured preconditioner, $\mathbf{P}_{\text{S}} = (\mathbf{A} \otimes \mathbf{B})\text{diag}(\text{vec}(\mathbf{S}))^2(\mathbf{A} \otimes \mathbf{B})^\top$
$\mathbf{A}, \mathbf{B}, \mathbf{S}$	Block matrices for EKFAC parameterization
λ	Weighting of the discrepancy term in the PPM
λ_{FSD}	Weighting of the function-space discrepancy term
λ_{WSD}	Weighting of the weight-space discrepancy term
η	Base optimizer learning rate
α	Meta optimizer learning rate
K	Meta update interval
p	Number of optimization hyperparameters
m	Number of parameters
$f(\mathbf{x}, \boldsymbol{\theta})$	Network function with parameters $\boldsymbol{\theta}$ applied to input $\mathbf{x}$
$\mathcal{L}(\mathbf{y}, \mathbf{t})$	Loss function (e.g. mean-squared error or cross-entropy)
$\mathcal{J}(\boldsymbol{\theta})$	Cost function, $\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{t}^{(i)})$
$\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$	Loss on a mini-batch of data, $\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{\lvert\mathcal{B}\rvert}\sum_{(\mathbf{x},\mathbf{t})\sim\mathcal{B}} \mathcal{L}(f(\mathbf{x}, \boldsymbol{\theta}), \mathbf{t})$
$\mathcal{Q}(\boldsymbol{\phi})$	Proximal meta-objective function
$\rho(\cdot, \cdot)$	Output-space divergence (e.g. KL divergence)
$D(\cdot, \cdot)$	Discrepancy function for the proximal point method
$D_{\text{F}}(\cdot, \cdot, \cdot)$	Function-space discrepancy function
$D_{\text{W}}(\cdot, \cdot)$	Weight-space discrepancy function
$\text{vec}(\cdot)$	Vectorization function
$\text{diag}(\cdot)$	Diagonalization function
$u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})$	Base optimizer update (e.g. stochastic gradient descent)
$\boldsymbol{\theta}'(\boldsymbol{\phi})$	Shorthand for $u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})$ where the data $\mathcal{B}$ is implicit
$\otimes$	Kronecker product
$\odot$	Elementwise product

Table 5: A summary of notation used in this paper.

B Extended Related Work

Short-Horizon Bias. Short-horizon bias (SHB) has been identified as a core challenge in the meta-optimization of optimizer parameters such as learning rates. Wu et al. [84] analyzed the setting where the meta-objective is the expected loss after an optimization step $\mathbb{E}_{\mathcal{B}\sim\mathcal{D}_{\text{train}}}[\mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}))]$. Note that the expectation over the entire dataset is intractable and can be typically approximated using a single mini-batch. This meta-objective yields undesirable behaviour where the learning rate decreases rapidly to reduce fluctuations of the loss caused by stochastic mini-batch evaluation. While our proximal meta-objective is greedy (e.g. it relies on a single lookahead step), APO empirically does not suffer from SHB. We attribute this to the fact that the first term in our proximal meta-objective evaluates the loss on the *same mini-batch used to compute the gradient* rather than a randomly-selected mini-batch. We refer readers to Appendix E for an ablation demonstrating the difference between using the same and a different mini-batch in computing the first term in our meta-objective.

Method	High-Dim Meta-Params	Online Adaptation	No Task Distribution	Short Unrolls	Meta-Params
Random [12]	✗	✗	✓	✗	Any hyperparameter
PBT [33]	✗	✓	✓	✓	Any hyperparameter
BayesOpt [71]	✗	✗	✓	✗	Any hyperparameter
BPTT [49]	✓	✓	✓	✗	Differentiable hyperparams
RTHO [21]	✗	✓	✓	✓	Differentiable hyperparams
IFT [46]	✓	✓	✓	✓	Differentiable *regularization* hyperparams
L4 [58]	✗	✓	✓	✓	LR
MARTHE [16]	✗	✓	✓	✓	LR
FDS [57]	✗	✗	✓	✗	LR, momentum, WD
Meta-SGD [44]	✗	✗	✗	∼	Diagonal preconditioner
Meta-Curvature [65]	✓	✗	✗	∼	Preconditioner parameterized by tensor products
WarpGrad [20]	✓	✗	✗	✓	Preconditioning "warp-layers"
MetaMD [22]	✓	∼	✗	✗	Parameterization of Bregman Divergence
Learned Optimizers [41, 2, 83, 56]	✓	✓	✗	✗	Learned optimizer parameters
HD [5]	✗	✓	✓	✓	LR
FOP [60]	✓	✓	✓	✓	$\mathbf{M}\mathbf{M}^\top$ preconditioner
APO (Ours)	✓	✓	✓	✓	EKFAC preconditioner, LR

Table 6: **A comparison of hyperparameter optimization and meta-learning methods.** For each method, we consider: 1) whether it scales to high-dimensional meta-parameters—most gradient-based methods do, except those that rely on forward-mode auto-diff; 2) whether it dynamically adapts the meta-parameters online or only learns a fixed meta-parameter used for a full training run; 3) whether it requires a task distribution for training, or can operate on a single task of interest; 4) whether it operates on short unrolls of the training procedure (as opposed to requiring complete training runs for each meta-parameter update); and 5) which meta-parameters it can tune.

Forward-Mode Differentiation. In principle, short horizon meta-objectives cannot model long-term behavior. Forward-mode autodiff like Real-Time Recurrent Learning (RTRL) and its approximations (UORO [75], KF-RTRL [61], OK [11]) can perform online optimization without suffering from SHB in principle (but still suffering from hysteresis). However, they do not scale well to many parameters, and thus are be used to tune the preconditioner. Other approaches like PES [81] use a finite-differences approximation to the gradient, which does not scale to the dimensionality of the preconditioner. RTRL has been applied to hyperparameter adaptation by Franceschi et al. [21], and a method for LR adaptation that interpolates between hypergradient descent and RTRL, called MARTHE, was introduced in [16].

Black-Box Hyperparameter Optimization. Finding good optimization hyperparameters is a longstanding problem [9]. Black-box methods for hyperparameter optimization, such as grid search, random search [12], and Bayesian optimization [71, 74, 72], are expensive, as they require performing many complete training runs, and can only find fixed hyperparameter values (e.g., a constant learning rate). Hyperband [43] can reduce the cost by terminating poorly-performing runs early, but is still limited to finding fixed hyperparameters. Population Based Training (PBT) [33] trains a population of networks simultaneously, and throughout training it terminates poorly-performing networks, replaces their weights with a copy of the weights of a better-performing network, perturbs the hyperparameters, and continues training from that point. PBT can find a coarse-grained learning rate schedule, but because it relies on random search, it is far less efficient than gradient-based meta-optimization.

Loss-Based Learning Rate Adaptation. Several works have proposed modern variants of Polyak step size methods [27, 79, 80, 45] for automatic learning rate adaptation. In a similar vein, Rolinek & Martius [69] proposed a method that tracks an estimated minimal loss and tunes a global learning rate to drive the loss on each mini-batch to the minimal loss.

Learned Optimization. Some authors have proposed learning entire optimization algorithms [41, 42, 2, 83, 56, 83]. Li & Malik [41] view this problem from a reinforcement learning perspective, where the state consists of the objective function $\mathcal{J}$ and the sequence of prior iterates $\{\boldsymbol{\theta}^{(t)}\}$ and gradients $\{\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}^{(t)})\}$, and the action is the step $\Delta\boldsymbol{\theta}$. In this setting, the update rule α is a policy that can be found via policy gradient methods [73]. Approaches that learn optimizers must be trained on a *set* of objective functions $\{f_1, \ldots, f_n\}$ drawn from a distribution $\mathcal{F}$. This setup can be restrictive if we only have one instance of an objective function. In addition, the initial phase of training the optimizer on a distribution of functions can be expensive. Learned optimizers require offline training, and typically must use long unrolls to ameliorate short horizon bias, making training expensive. APO requires only the objective function of interest as it does not have trainable parameters.

Few-Shot Meta-Learning Methods. Meta-SGD [44] extends MAML [19] by learning not only a shared initialization which can be fine-tuned in a few steps for many different tasks but also a diagonal preconditioning matrix used for the fine-tuning gradient steps (e.g., the inner-loop of MAML, adapting to a specific task). Meta-SGD requires a task distribution, and only learns a fixed diagonal preconditioner which is not adapted during the task-adaptive optimization procedure. In contrast, APO learns a more expressive, block-diagonal preconditioner from a single training run, which is adapted online during training.

Meta-Curvature (MC) [65] meta-learns a preconditioning matrix to improve generalization after a small number of (preconditioned) gradient steps in MAML-style meta-learning. Meta-Curvature requires a task distribution for training, and relies on the assumption that related tasks have similar curvature; the method learns a single, fixed preconditioning matrix that is applied across tasks (and to new tasks at meta-test time). In contrast, APO does not require a task distribution, and dynamically adapts the meta-learned preconditioning matrix during a single training run, rather than keeping it fixed as done in MC.

Lee and Choi [40] proposed Transformation Networks (T-nets), which learn an activation-space-metric that modifies the update direction and step size taken to adapt to each task in a few-shot setting, and Mask Transformation Networks (MT-nets), which extend T-nets by learning which subset of weights to update for each task. Similarly to Meta-SGD and Meta-Curvature, MT-nets require a task distribution for meta-training, and only learn fixed meta-parameters which are not adapted during training of new tasks.

APO is a generic framework for online meta-learning of optimization parameters. We demonstrate its effectiveness for two specific types of meta-parameters: the preconditioning matrix and the global learning rate. Our main contribution is the use of a principled meta-objective based on the proximal point method, which we prove can recover existing first- and second-order optimization algorithms given appropriate approximations to the loss term and FSD term. We focused on tuning the KFAC-style preconditioner as this is a popular block-diagonal second-order method, and allows us to draw the connection to KFAC in Corollary 1. However, the APO meta-objective can be applied to tune different parameterizations as well; in particular, one could use interleaved warp-layers to parameterize a preconditioner similarly to [40, 20].

C Experimental Details

C.1 Computing Environment

All experiments were implemented using the PyTorch [66] and JAX [13] libraries, and we ran all experiments on NVIDIA P100 GPUs.

C.2 Meta-Optimization Setup

In all experiments, we perform 1 meta-update every 10 updates of the base optimizer (e.g. $K = 10$ in Algorithm 1). Hence, the computational overhead of APO is just a small fraction of the original

training procedure. We perform grid searches over λ_{FSD} and λ_{WSD} for both learning rate and preconditioner adaptation.

Learning Rate Adaptation Setup. For our learning rate adaptation experiments, we used RMSProp as the meta-optimizer with a meta-learning rate of 0.1. For SGD-APO and SGDm-APO, we set the initial learning rate to 0.1, while for RMSProp-APO and Adam-APO, we set the initial learning rate to 3e-4. These specific values are not important for good performance; we show in Appendix J.2 that APO is robust to the choice of initial learning rate.

Preconditioner Adaptation Setup. For preconditioner adaption experiments, we used Adam as the meta-optimizer with a meta-learning rate chosen from $\{1\text{e-}4, 3\text{e-}5\}$. We initialized the block matrices such that the preconditioner is the identity, so that our update is equivalent to SGD at initialization. Furthermore, we used a warm-up phase of 3000 iterations at the beginning of training that updates the parameters with SGDm (or Adam for Transformer) but still meta-learns the preconditioning matrix (e.g. performing meta-gradient steps on $\mathbf{P}$). When updating the parameters, we scaled the preconditioned gradient by a fixed constant of 0.9 for stability.

C.3 Regression on UCI Collection

We used Slice, Protein, and Parkinson data from the UCI collection. In training, we normalized the input features and targets to have a zero mean and unit variance. We used batch size of 128 and trained the network for 500 epochs without weight decay.

We conducted hyperparameter searches over learning rates for all baseline models, making choices based on the final training loss. With the chosen set of hyperparameters, we repeated the experiments 3 times with different random seeds. For SGDm, we set the momentum to 0.9 and swept over the learning rates {1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001}. For Adam, we performed a grid search on learning rate of {1e-2, 3e-3, 1e-3, 3e-4, 3e-4, 1e-4}. For Shampoo, we swept over the learning rates {10, 5, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001} as suggested by Gupta et al. [26]. For KFAC, we did a grid search on learning rates of {0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001} and damping values of {3e-2, 1e-2, 3e-3, 1e-3, 3e-4, 1e-4}. For APO-Precond, we set $\lambda_{\text{FSD}} = 0$ and grid searched over $\lambda_{\text{WSD}} = \{3, 1, 0.1, 0.01\}$.

C.4 Image Reconstruction

We used the same experimental set-up from Martens & Grosse [53] for the deep autoencoder experiment. The loss function was defined to be the binary entropy and we further added a regularization term $\frac{\lambda_{\text{WD}}}{2}\|\boldsymbol{\theta}\|^2$ to the loss function, where $\lambda_{\text{WD}} = 10^{-5}$. The layer widths for the autoencoder were set to be [784, 1000, 500, 250, 30, 250, 500, 1000, 784] and we used the sigmoid activation function. We trained the network for 1000 epochs with the batch size of 512.

We conducted extensive hyperparameter searches for all baseline models, making choices based on the final training loss. With the chosen set of hyperparameters, we repeated the experiments 3 times with different random seeds. For SGDm, we set the momentum to 0.9 and performed a grid search over the learning rates {1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001}. For Adam, we swept over learning rate of {3e-2, 1e-2, 3e-3, 1e-3, 3e-4, 3e-4, 1e-4, 3e-5, 1e-5}. For Shampoo, we tried setting learning rates in range of {10, 5, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001}. For KFAC, we did a grid search on learning rates of {0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001} and damping values of {3e-1, 1e-1, 3e-2, 1e-2, 3e-3, 1e-3, 3e-4, 1e-4}. For APO-Precond, we performed a grid search over $\lambda_{\text{FSD}} = \{0.3, 0.1\}$ and $\lambda_{\text{WSD}} = \{3, 1, 0.3, 0.1\}$. We set the FSD to measure the KL divergence.

C.5 Image Classification

CIFAR-10 & CIFAR-100. We used the standard procedure from Zagoruyko & Komodakis [85] for training convolutional neural networks. The images are zero-padded with 4 pixels on each side and

then a random 32×32 crop is extracted from the image, horizontally flipped with the probability of 0.5. We further normalize the inputs with per-channel mean and standard deviations. We performed the extensive search over the hyperparameters for all networks. We held out 5k examples from the CIFAR-10 and CIFAR-100 to form the validation set following Zagoruyko & Komodakis [85] and selected the hyperparameters with the highest validation accuracy. With the chosen hyperparameters, we re-trained the network with the full training dataset and reported the final test accuracy.

Preconditioning Adaptation. Across all experiments, we used the batch size of 128 and trained the network for 200 epochs. With the chosen set of hyperparameters, we repeated the experiments 3 times with different random seeds and reported the mean accuracy on the test dataset. For SGDm, we set the momentum to 0.9 and performed a grid search over the learning rates {0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001}. For Adam, we swept over learning rate of {3e-2, 1e-2, 3e-3, 1e-3, 3e-4, 3e-4, 1e-4}. For KFAC, we did a grid search on learning rates of {0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001} and damping values of {3e-2, 1e-2, 3e-3, 1e-3, 3e-4, 1e-4}. For APO-Precond, we performed a grid search over $\lambda_{\text{FSD}} = \{0.3, 0.1\}$ and $\lambda_{\text{WSD}} = \{3, 1, 0.3, 0.1\}$ for architectures without batch normalization. As batch normalization makes the parameters scale-invariant [3], we imposed a higher regularization in the function space and lower regularization in the weight space, searching over $\lambda_{\text{FSD}} = \{3, 1\}$ and $\lambda_{\text{WSD}} = \{0.3, 0.1, 0.03, 0.01\}$. We set the FSD term to measure the KL divergence. We also searched over the weight decay in range of {5e-4, 1e-4, 5e-5} for all optimizers.

Learning Rate Adaptation. For ResNet32, we trained for 400 epochs, and used a manual schedule that decays the learning rate by a factor of 10 at epochs 150 and 250, following Lucas et al. [48]. For ResNet34 and WideResNet 28-10, we trained for 200 epochs, with a manual schedule that decays the learning rate by a factor of 5 at epochs 60, 120, and 160, following Zagoruyko & Komodakis [85].

For the baseline optimizers, we performed grid searches over the fixed learning rate or initial learning rate for a fixed step schedule, as well as the weight decay. For all base optimizers and APO-tuned variants, we searched over weight decay values in $\{0.01, 0.003, 0.001, 0.0003, 0.0001, 0.0\}$. For SGD and SGDm, we searched over learning rates in $\{1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$. For RMSprop and Adam, we searched over learning rates in $\{0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001\}$. For each of the APO-tuned variants (e.g. SGDm-APO), we kept $\lambda_{\text{WSD}} = 0$ and searched over $\lambda_{\text{FSD}} \in \{0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$.

C.6 Neural Machine Translation

We trained a Transformer [78] composed of 6 encoder and decoder layers, with a word embedding and hidden vector dimension of 512. The architecture has feed-forward size of 1024, 4 attention heads, dropout value 0.3, and weight decay value 0.0001. For the AdamW baseline, we used a warmup-then-decay learning rate schedule widely used in practice, and for the SGD baseline and APO-Precond, we kept the learning rate fixed after the warmup. For APO-Precond, following the practice from Zhang et al. [86], we used a diagonal block matrix in the structured preconditioner of the embedding weight matrix to reduce memory overhead. We used the Fairseq toolkit [64] to conduct all experiments.

For SGDm, we grid searched over the learning rates in {10, 3, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001} and tried both the fixed learning rate and inverse sqrt learning rate schedules. For AdamW, we set $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-1}$ and searched over the learning rates in range {0.05, 0.001, 0.0005, 0.0001}. AdamW also used the inverse sqrt learning rate schedule with 4000 warmup steps. For APO-Precond, we searched over $\lambda_{\text{FSD}} = \{0.3, 0.1\}$ and $\lambda_{\text{WSD}} = \{3, 1, 0.3, 0.1\}$ and let FSD term to measure the KL divergence. As we found a fixed learning rate schedule to work best for SGDm, we also used a fixed learning rate schedule for APO-Precond. We selected the hyperparameters based on the BLEU score on the validation set and reported on the final test BLEU score on the checkpoint, which achieved the highest validation BLEU score.

D Additional Results

D.1 Rosenbrock Function

We validated APO on the two-dimensional Rosenbrock function defined as:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

with initialization $(x, y) = (1, -1.5)$. In Figure 6, we used APO to tune the learning rate for SGD, as well as the full preconditioning matrix. Both APO-tuned methods outperformed vanilla SGD with the optimal fixed learning rate (chosen through a careful grid search). Because Rosenbrock has ill-conditioned curvature, second-order optimization with APO-Precond dramatically speeds up the convergence and achieves a lower loss.

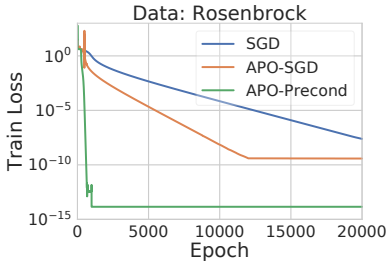


Figure 6: **Training loss on the Rosenbrock function**. Both the learning rate schedules and the preconditioner adapted by APO outperform the optimal fixed learning rate for SGD.

D.2 SVHN

We also evaluated APO on the Google Street View House Numbers dataset (SVHN) [62], using a WideResNet 16-8. We followed the experimental setup of Zagoruyko & Komodakis [85], and used the same decay schedule for the baseline, which decays the learning rate by $10\times$ at epochs 80 and 120. The optimal fixed learning rate achieves test accuracy fluctuating around 97.20%, while the manual schedule achieves 98.17%. APO rapidly converges to test accuracy 97.96%, outperforming the baseline while being slightly worse than the manual schedule (Figure 7).

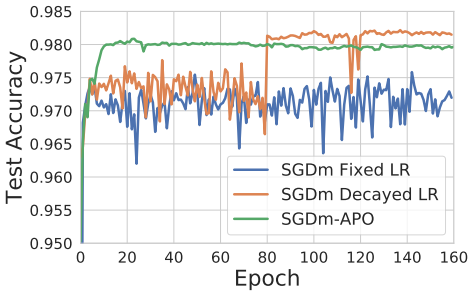


Figure 7: **Test Accuracy on SVHN with SGDm.**

D.3 CIFAR-10

APO Preconditioning Adaptation. We show the plots for for experiments listed in Table 2 in Figure 8 and Figure 9.

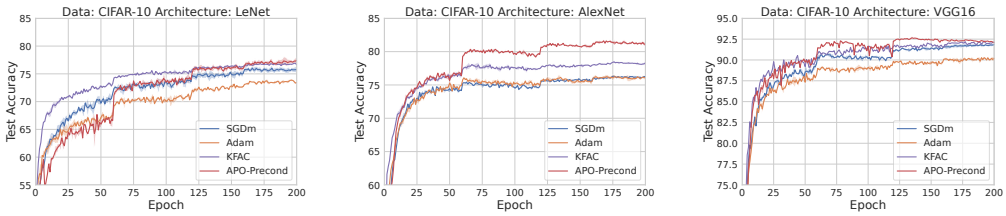


Figure 8: LeNet, AlexNet, and VGG16 on CIFAR-10, using SGDm, Adam, KFAC, and APO-Precond.

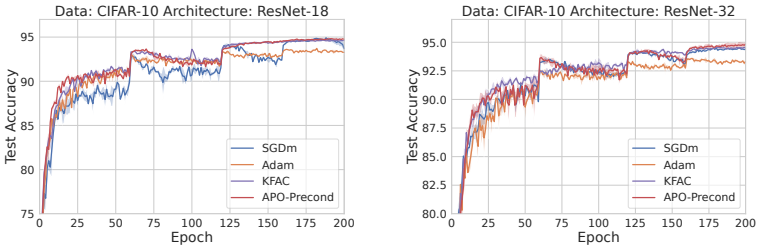


Figure 9: ResNet-18 and ResNet-32 on CIFAR-10, using SGDm, Adam, KFAC, and APO-Precond.

Furthermore, we show the test accuracy as a function of wall-clock time with various values of K in Figure 10. We observed that APO is robust to the values of K and setting $K = 10$, as done in our main experiments, show a significant improvement in the test accuracy over SGDm while only introducing a small computational overhead.

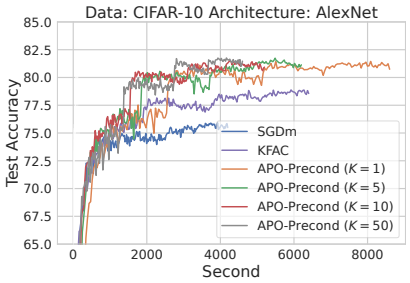


Figure 10: AlexNet on CIFAR-10, using SGDm, KFAC, and APO-Precond with various values of K. We show the test accuracy as a function of wall-clock time.

APO Learning Rate Schedules. We show the plots for each base-optimizer and network architecture as below. We find that APO achieves better performance than the best fixed LR, and is comparable to the manual schedule. The learning rate schedules discovered by APO are nontrivial: most exhibit a large increase in the learning rate at the start of training to make rapid progress, followed by a gradual decrease in the learning rate to fine-tune the solution as optimization approaches a local optimum. The APO learning rate schedules often span two orders of magnitude, similarly to manual decay schedules.

	CIFAR-10 (ResNet32)		
	Fixed	**Decayed**	**APO**
SGD	90.07 ± 0.38	93.30 ± 0.18	92.71 ± 0.25
SGDm	89.40 ± 1.38	93.34 ± 0.15	92.75 ± 0.13
RMSprop	89.84 ± 0.43	91.94 ± 0.33	91.28 ± 0.35
Adam	90.45 ± 0.24	92.26 ± 0.34	91.81 ± 0.15

Table 7: **Test accuracy on CIFAR-10 using ResNet32.** APO consistently outperforms the best fixed learning rate for each optimizer, and is on par with carefully-tuned manual schedules. Each reported result is the mean of 4 random restarts, and we report $\pm$ the standard deviation.

	CIFAR-10 (ResNet34)		
	Fixed	**Decayed**	**APO**
SGD	93.00 ± 0.35	93.54 ± 0.01	94.27 ± 0.02
SGDm	92.99 ± 0.14	95.08 ± 0.24	94.47 ± 0.24
RMSprop	92.87 ± 0.32	93.97 ± 0.11	93.97 ± 0.07
Adam	93.23 ± 0.15	94.12 ± 0.10	93.80 ± 0.14

Table 8: **Test accuracy on CIFAR-10 using ResNet34.** APO consistently outperforms the best fixed learning rate for each optimizer, and is on par with carefully-tuned manual schedules. Each reported result is the mean of 4 random restarts, and we report $\pm$ the standard deviation.

	CIFAR-10 (WideResNet-28-10)			CIFAR-100 (WideResNet-28-10)		
	Fixed	**Decayed**	**APO**	**Fixed**	**Decayed**	**APO**
SGD	93.38 ± 0.29	94.86 ± 0.20	94.85 ± 0.21	76.29 ± 0.32	77.92 ± 0.26	76.87 ± 0.31
SGDm	93.46 ± 0.28	95.98 ± 0.17	95.50 ± 0.25	74.81 ± 0.04	81.01 ± 0.08	79.33 ± 0.18
RMSprop	92.91 ± 0.28	93.60 ± 0.21	94.22 ± 0.23	72.06 ± 0.16	76.06 ± 0.11	74.17 ± 0.09
Adam	92.81 ± 0.20	94.04 ± 0.07	93.83 ± 0.13	72.01 ± 0.18	75.53 ± 0.22	76.33 ± 0.17

Table 9: **Test accuracy on CIFAR-10 using WideResNet-28-10.** APO consistently outperforms the best fixed learning rate for each optimizer, and is on par with carefully-tuned manual schedules. Each reported result is the mean of 4 random restarts, and we report $\pm$ the standard deviation.

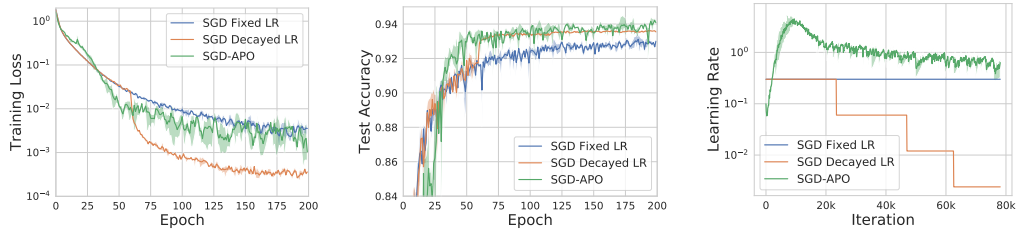


Figure 11: **ResNet34 on CIFAR-10, using SGD.** The shaded regions show the min/max values over 4 random restarts.

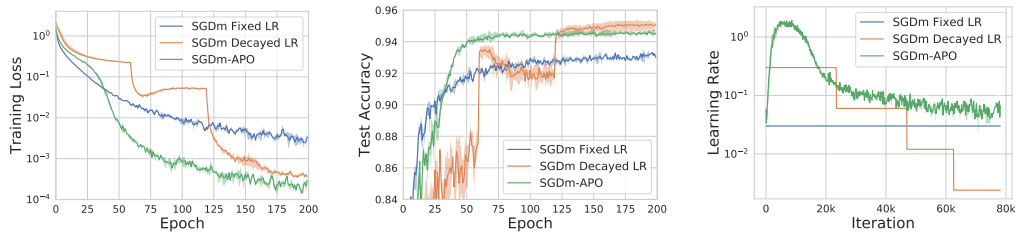


Figure 12: **ResNet34 on CIFAR-10, using SGDm.** The shaded regions show the min/max values over 4 random restarts.

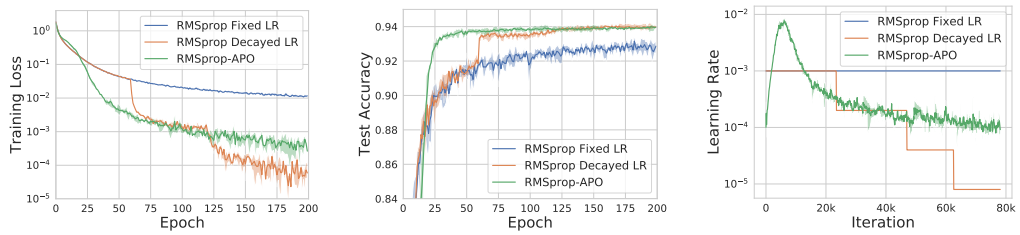


Figure 13: **ResNet34 on CIFAR-10, using RMSprop.** The shaded regions show the min/max values over 4 random restarts.

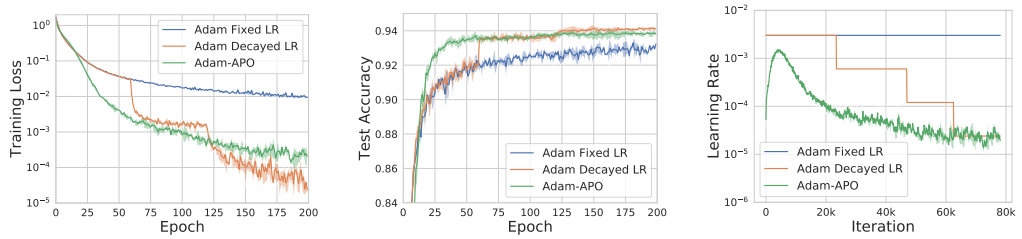


Figure 14: **ResNet34 on CIFAR-10, using Adam.** The shaded regions show the min/max values over 4 random restarts.

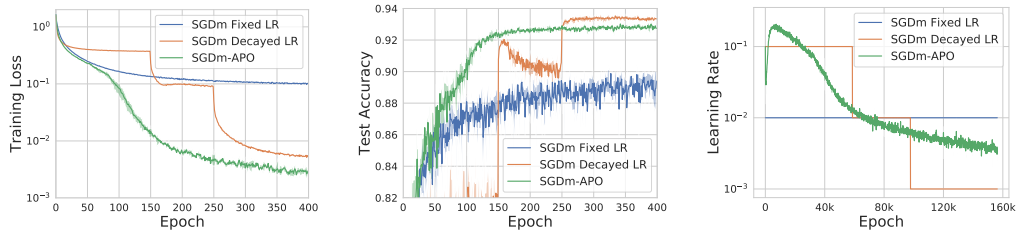


Figure 15: **ResNet32 on CIFAR-10, using SGDm.** The shaded regions show the min/max values over 4 random restarts.

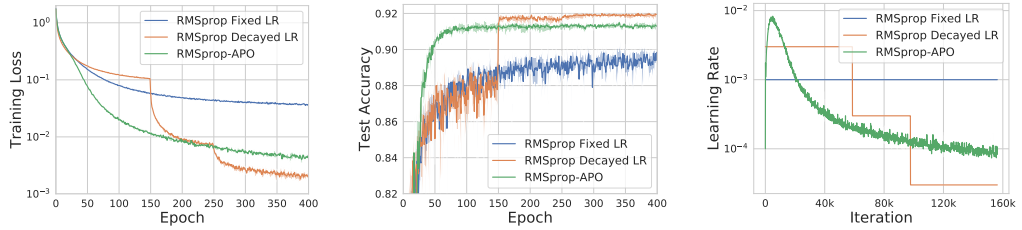


Figure 16: **ResNet32 on CIFAR-10, using RMSprop.** The shaded regions show the min/max values over 4 random restarts.

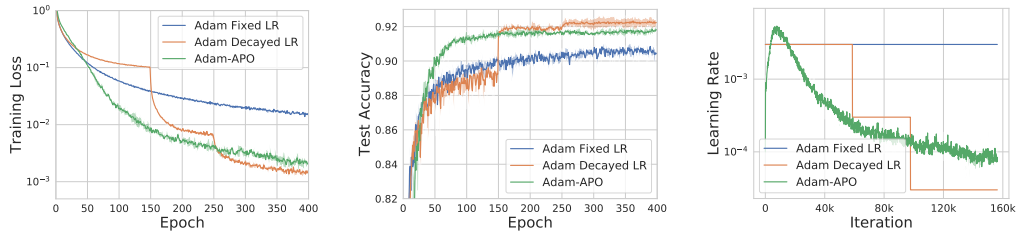


Figure 17: **ResNet32 on CIFAR-10, using Adam.** The shaded regions show the min/max values over 4 random restarts.

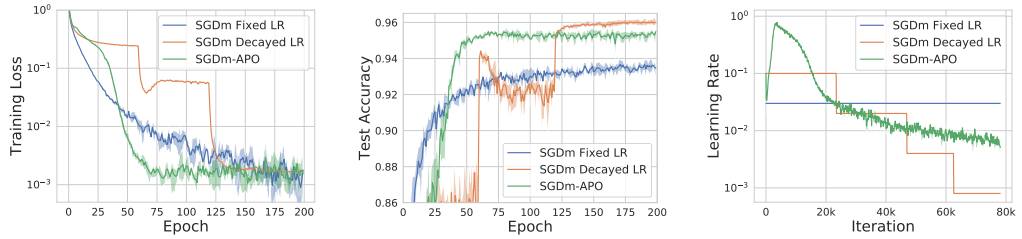


Figure 18: **WideResNet 28-10 on CIFAR-10, using SGDm.** The shaded regions show the min/max values over 4 random restarts.

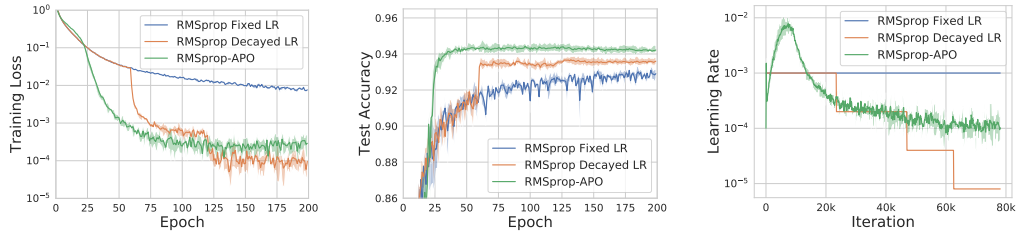


Figure 19: **WideResNet 28-10 on CIFAR-10, using RMSprop.** The shaded regions show the min/max values over 4 random restarts.

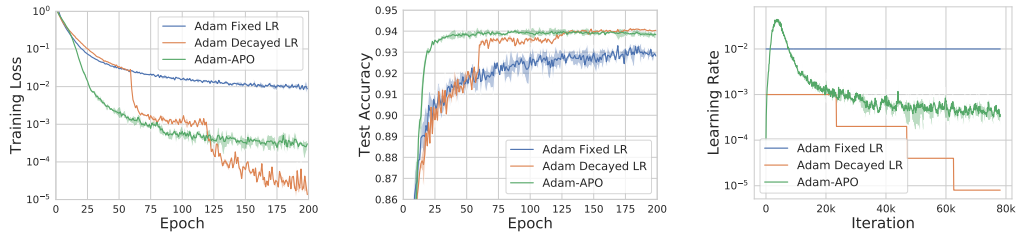


Figure 20: **WideResNet 28-10 on CIFAR-10, using Adam.** The shaded regions show the min/max values over 4 random restarts.

D.4 CIFAR-100

APO Learning Rate Schedules. We show training curve plots for the Adam base-optimizer below. We find that APO achieves better performance than the fixed LR, and is comparable with the manual schedule.

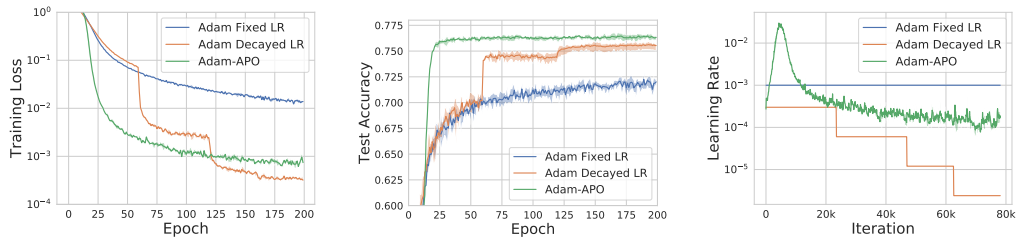


Figure 21: **WideResNet on CIFAR-100, using Adam.** The shaded regions show the min/max values over 4 random restarts.

D.5 16-bit Neural Network Training

We show the training curve plots for training 16-bit ResNet-18 on CIFAR-10 and CIFAR-100 in Figure 22 .

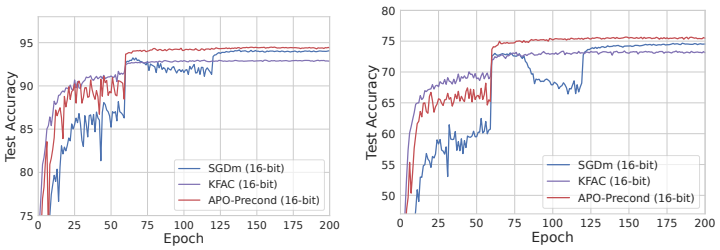


Figure 22: Test accuracy curves for 16-bit neural networks: ResNet-18 on CIFAR-10 (**left**) and CIFAR-100 (**right**).

Task	Model	SGDm	KFAC	APO-P
CIFAR-10	**LeNet**	75.65 $\pm$ 0.24	74.95 $\pm$ 0.55	**77.25** $\pm$ 0.28
CIFAR-10	**ResNet-18**	94.15 $\pm$ 0.23	92.72 $\pm$ 0.13	**94.79** $\pm$ 0.15
CIFAR-100	**ResNet-18**	73.53 $\pm$ 0.29	73.12 $\pm$ 0.22	**75.47** $\pm$ 0.11

Table 10: Test accuracies with $\pm$ standard deviation of 16-bit nets on CIFAR 10/100.

D.6 Comparison to HD, L4, and FOP

In this section, we compare APO to Hypergradient Descent (HD) [5] and L4 [69] for the learning rate adaptation problem, and First-Order Preconditioning (FOP) [60] for the preconditioning adaptation problem.

D.6.1 Hypergradient Descent

Here, we compare to Hypergradient Descent (HD) [69], a method for online learning rate adaptation. The meta-objective used in HD is the expected value of the training loss after updating the learning rate, $\mathbb{E}[\mathcal{L}(u(\theta, \phi)]$. This is the same objective that was analyzed theoretically by Wu et al. [84], that was shown to lead to sub-optimal behavior when the objective is stochastic. In particular, [84] showed that this meta-objective encourages the learning rate to drop rapidly, which reduces the expected loss in the short-term by eliminating fluctuations in the loss caused by random mini-batch sampling, at the cost of worse long-term behavior (as discussed in Appendix B). In our experiments, we found that HD suffers from the short horizon bias issue; the only way to prevent the LR from reducing too quickly is to use a very small meta-learning rate, which becomes more heuristic. Empirically (and as discussed in more detail in Appendix E), the APO meta-objective does not suffer from this issue, allowing us to use larger meta-learning rates without having the LR decrease too quickly. Because the meta-learning rate for HD must be small, HD is not robust to the initial learning rate; the initial LR must be tuned in order to obtain decent performance. In contrast, APO is very robust to the initial learning rate, behaving nearly identically for initial learning rates spanning 6 orders of magnitude, from 1e-1 to 1e-7 (shown in Appendix J.2).

Comparison. We ran hypergradient descent versions of SGD, SGDm, and Adam, denoted SGD-HD, SGDm-HD, and Adam-HD, respectively. There was no existing implementation of RMSprop-HD, so we left out this optimizer. For fair comparison to APO, we used the same exponential learning rate parameterization for both APO and HD. For SGD-HD and SGDm-HD, we performed a grid search over initial learning rates {1e-1, 1e-2, 1e-3}, meta-learning rates {3.0, 1.0, 1e-1, 1e-2, 1e-3}, and weight decay coefficients {1e-2, 1e-3, 5e-4, 1e-4, 0.0}. For Adam-HD, we performed a grid search over initial learning rates {1e-2, 1e-3, 1e-4}, meta-learning rates {3.0, 1.0, 1e-1, 1e-2, 1e-3}, and weight decay coefficients {1e-3, 5e-4, 1e-4, 0.0}.

On the Meta-Learning Rate for HD. In our experiments, we found that HD only works decently when run with a large initial learning rate and a small meta-learning rate, which has the effect of reducing the LR slowly rather than dramatically (as is expected from the analysis in [84]). However, using a larger meta-learning rate, which optimizes the meta-objective more rapidly, exhibits the short horizon bias issue. In Figures 23,25, we plot the learning rates adapted by HD with a small meta-learning rate (0.001) and a large meta-learning rate (1.0), for each of SGD-HD, SGDm-HD, and Adam-HD. We also plot the corresponding test accuracies for each of these experiments in Figures 24,26, to show that the LR adapted using a large meta-learning rate catastrophically affects training and leads to substantially worse test accuracy. In summary, these results provide evidence that the HD meta-objective is not a very good target for meta-optimization.

	CIFAR-10 (ResNet-32)				
	Fixed	**Decay**	**HD** [5]	**L4** [69]	**APO (Ours)**
SGD	90.07 ± 0.38	93.30 ± 0.18	88.75 ± 1.07	83.90 ± 0.19	92.71 ± 0.25
SGDm	89.40 ± 1.38	93.34 ± 0.15	90.34 ± 0.26	89.05 ± 0.35	92.75 ± 0.13
RMSprop	89.84 ± 0.43	91.94 ± 0.33	-	87.35 ± 0.38	91.28 ± 0.35
Adam	90.45 ± 0.24	92.26 ± 0.34	90.38 ± 0.22	89.80 ± 0.09	91.81 ± 0.15

Table 11: We compare the test accuracies achieved by the optimal fixed learning rate, the manual step decay schedule, hypergradient descent (HD) [5], L4 [69], and the APO-discovered schedule, training ResNet-32 [28] on CIFAR-10. We report the mean and standard deviation of four random restarts for each method. APO outperforms the optimal fixed learning rate, HD, and L4, and is competitive with manual schedule.

	CIFAR-10 (ResNet-34)				
	Fixed	**Decay**	**HD** [5]	**L4** [69]	**APO (Ours)**
SGD	93.00 ± 0.35	93.54 ± 0.01	92.53 ± 0.14	85.93 ± 0.25	94.27 ± 0.02
SGDm	92.99 ± 0.14	95.08 ± 0.24	92.79 ± 0.35	88.35 ± 0.09	94.47 ± 0.24
RMSprop	92.87 ± 0.32	93.87 ± 0.11	-	80.93 ± 0.36	93.97 ± 0.07
Adam	93.23 ± 0.15	94.12 ± 0.10	92.54 ± 0.05	85.10 ± 0.10	93.80 ± 0.14

Table 12: We compare the test accuracies achieved by the optimal fixed learning rate, the manual step decay schedule, hypergradient descent (HD) [5], L4 [69], and the APO-discovered schedule, training ResNet-34 [28] on CIFAR-10. We report the mean and standard deviation of four random restarts for each method. APO outperforms the optimal fixed learning rate, HD, and L4, and is competitive with manual schedule.

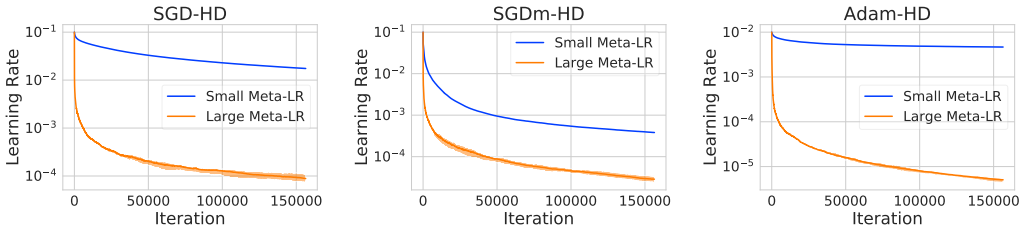


Figure 23: **ResNet32, CIFAR-10, Hypergradient Descent (HD) [5] learning rate schedules for each optimizer SGD-HD, SGDm-HD, and Adam-HD.** The shaded regions show the min/max values over 4 random restarts.

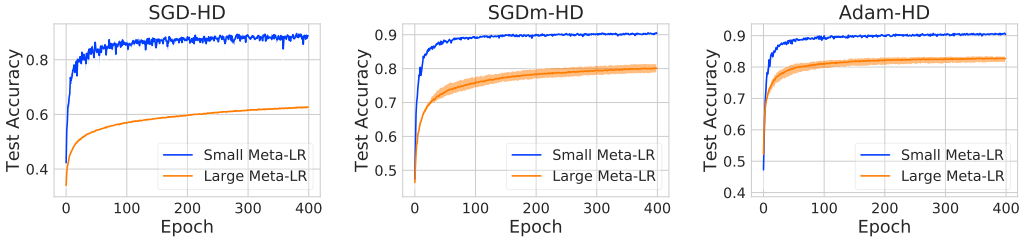


Figure 24: **ResNet32, CIFAR-10, Hypergradient Descent (HD) [5] test accuracies for each optimizer SGD-HD, SGDm-HD, and Adam-HD.** The shaded regions show the min/max values over 4 random restarts.

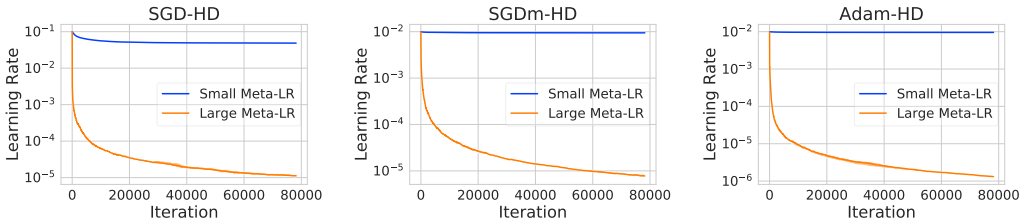


Figure 25: **ResNet34, CIFAR-10, Hypergradient Descent (HD) [5] learning rate schedules for each optimizer SGD-HD, SGDm-HD, and Adam-HD.** The shaded regions show the min/max values over 4 random restarts.

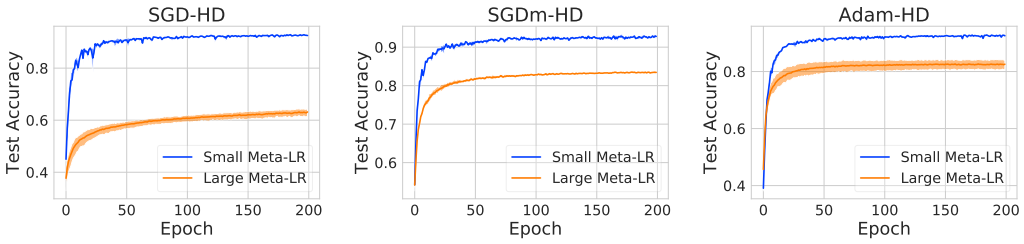


Figure 26: **ResNet34, CIFAR-10, Hypergradient Descent (HD) [5] test accuracies for each optimizer SGD-HD, SGDm-HD, and Adam-HD.** The shaded regions show the min/max values over 4 random restarts.

D.6.2 Comparison to L4

We also compared APO to the L4 optimizer [69], which adapts a global learning rate in an online fashion based on Polyak's update rule. We applied L4 to tune the global learning rate for four base optimizers—vanilla SGD without momentum, SGD with momentum coefficient 0.9 (denoted SGDm), RMSprop, and Adam—used to train ResNet32 and ResNet34 on CIFAR-10. For each experiment, we performed a grid search over weight decay coefficients {1e-3, 5e-4, 1e-4, 0.0} and α values {0.1, 0.15, 0.3}, where α controls the scaling of the step size, and is the primary hyperparameter to tune for L4. For the other L4 hyperparameters, we used the defaults $\gamma = 0.9$ and $\tau = 0.001$.

The results are shown in Table 11 and Table 12. We found that L4 obtained poor generalization, underperforming the other baselines and APO. Figures 27,28 and Figures 29,30 show the test accuracy and learning rate adaptation obtained with each L4-tuned optimizer, on CIFAR-10 classification with ResNet32 and ResNet34.

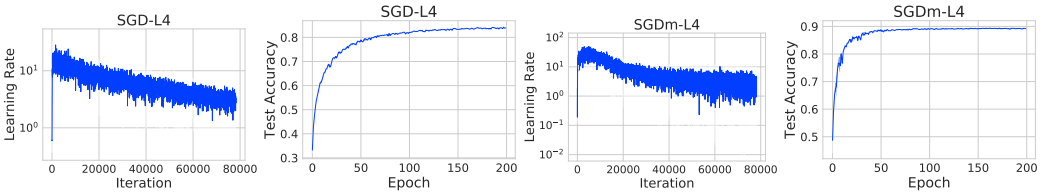


Figure 27: **ResNet32 L4-SGDm and RMSprop-L4.** The shaded regions show the min/max values over 3 random restarts.

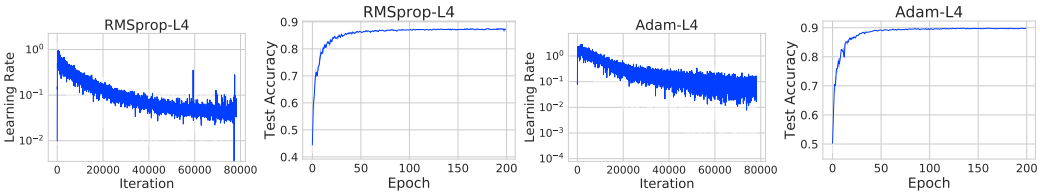


Figure 28: **ResNet32 RMSprop-L4 and Adam-L4.** The shaded regions show the min/max values over 3 random restarts.

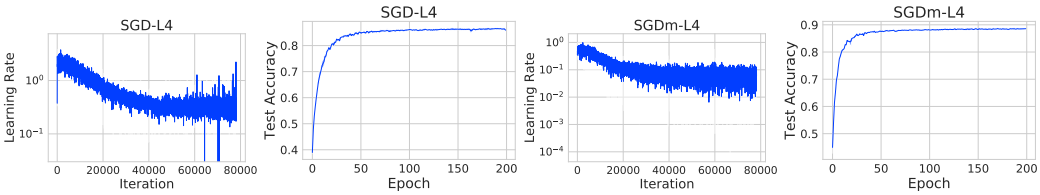


Figure 29: **ResNet34 L4-SGDm and RMSprop-L4.** The shaded regions show the min/max values over 3 random restarts.

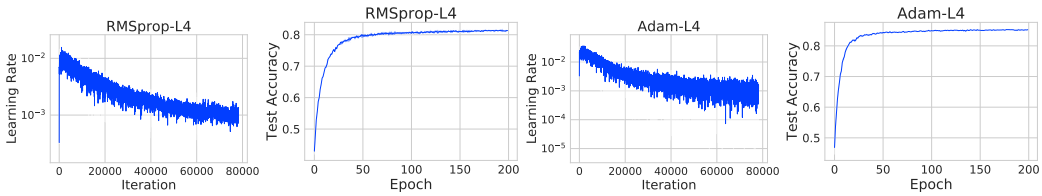


Figure 30: **ResNet34 RMSprop-L4 and Adam-L4.** The shaded regions show the min/max values over 3 random restarts.

D.6.3 First-Order Preconditioning

Task	Model	SGDm	Adam	KFAC	FOP	APO-Precond
CIFAR-10	**LeNet**	75.73 ± 0.50	73.41 ± 0.27	76.63 ± 0.51	75.25 ± 0.56	$\mathbf{77.42} \pm 0.55$
CIFAR-10	**AlexNet**	76.27 ± 0.24	76.09 ± 0.31	78.33 ± 0.28	76.52 ± 0.23	$\mathbf{81.14} \pm 0.28$
CIFAR-10	**VGG16**	91.82 ± 0.15	90.19 ± 0.31	92.05 ± 0.23	91.65 ± 0.41	$\mathbf{92.13} \pm 0.17$
CIFAR-10	**ResNet-18**	93.69 ± 0.45	93.27 ± 0.09	94.60 ± 0.04	93.76 ± 0.31	$\mathbf{94.75} \pm 0.09$
CIFAR-10	**ResNet-32**	94.40 ± 0.07	93.30 ± 0.14	94.49 ± 0.10	93.90 ± 0.28	$\mathbf{94.83} \pm 0.19$
CIFAR-100	**AlexNet**	43.95 ± 0.07	41.82 ± 0.28	46.24 ± 0.33	44.66 ± 0.29	$\mathbf{52.35} \pm 0.26$
CIFAR-100	**VGG16**	65.98 ± 0.25	60.61 ± 0.39	61.84 ± 0.37	61.64 ± 0.39	$\mathbf{67.95} \pm 0.30$
CIFAR-100	**ResNet-18**	76.85 ± 0.07	70.87 ± 0.26	76.48 ± 0.13	75.93 ± 0.23	$\mathbf{76.88} \pm 0.06$
CIFAR-100	**ResNet-32**	$\mathbf{77.47} \pm 0.25$	68.67 ± 0.65	75.70 ± 0.13	75.66 ± 0.27	77.41 ± 0.11

Table 13: Average test accuracy over 4 random restarts on CIFAR-10 and CIFAR-100 dataset for various optimizers. We also report $\pm$ the standard deviation.

Image Reconstruction. Following Section 5, we trained an 8-layer autoencoder on MNIST and compared APO with FOP using the same experimental set-up from Appendix C.4. For FOP, we set the momentum to 0.9 and performed grid search over the learning rates and hypergradient learning rates in range of $\{1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$ and $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, respectively.

We show the training losses in Figure 31. APO-Precond converges faster than FOP and achieves competitive training loss to KFAC (although there remains some performance gap with KFAC).

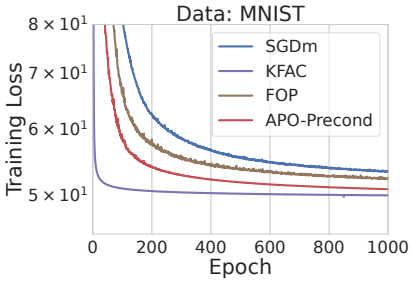


Figure 31: Deep autoencoder on MNIST. We compare the training loss of SGDm, KFAC, FOP, and APO-Precond.

Image Classification. We also compared APO with FOP on image classification task with the same set-up from Appendix C.5. For FOP, we fixed the momentum to 0.9 and again performed grid search over the learning rates and hypergradient learning rates in $\{1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$ and $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, respectively. We also searched over the weight decay in range of {5e-4, 1e-4, 5e-5}.

The test accuracies of APO-Precond and FOP are summarized in Table 13. While APO-Precond achieved competitive generalization performance to SGDm and KFAC on all tasks, FOP consistently underperformed compared to KFAC.

D.7 Tuning Other Optimization Parameters

Here, we show proof-of-concept results for tuning different optimizer parameters, other than the learning rate or preconditioner. In particular, we focus on two parameters of the RMSprop optimizer that are rarely tuned by hand. The RMSprop update rule is:

$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma)\mathbf{g}_t^2 \tag{6}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\mathbf{s}_t^{\rho} + \epsilon}\mathbf{g}_t \tag{7}$$

We consider adapting ϵ and ρ (the power to which $\mathbf{s}_t$ is raised in the denominator, which is fixed to $\rho = \frac{1}{2}$ in standard RMSprop). Both ϵ and ρ can be interpreted as having a damping effect on the update. We investigated the effect of tuning $\{\epsilon, \rho\}$ using APO on two tasks: 1) training a two-hidden-layer MLP with 1000 hidden units per layer on FashionMNIST; and 2) training a ResNet32 on CIFAR-10. We set the learning rate to the best fixed value for the baseline: $\eta = 0.001$ for CIFAR-10 and $\eta = 0.0001$ for FashionMNIST. We used RMSprop with learning rate 0.01 for meta-optimization, and we performed a grid search over $\lambda_{\text{FSD}} \in \{\text{1e-3}, \text{1e-4}, \text{1e-5}\}$ and $\lambda_{\text{WSD}} \in \{\text{1e-1}, \text{1e-2}, \text{1e-3}, \text{1e-4}, \text{1e-5}, \text{1e-6}\}$. We compared to baseline RMSprop with its optimal fixed learning rates ($\eta = 0.001$ for CIFAR-10 and $\eta = 0.0001$ for FashionMNIST) and with the default values $\epsilon = \text{1e-8}$ and $\rho = 0.5$. The results for FashionMNIST and CIFAR-10 are shown in Figures 32 and 33, respectively. We found that on each of these tasks, tuning $\{\epsilon, \rho\}$ resulted in faster training and lower final training loss.

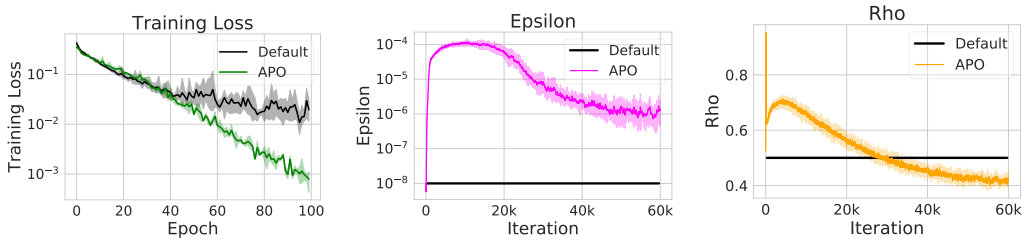


Figure 32: Using APO to tune the $\{\epsilon, \rho\}$ hyperparameters of RMSprop, for an MLP trained on FashionMNIST. Here, we used the best fixed learning rate obtained from a grid search using the best optimizer.

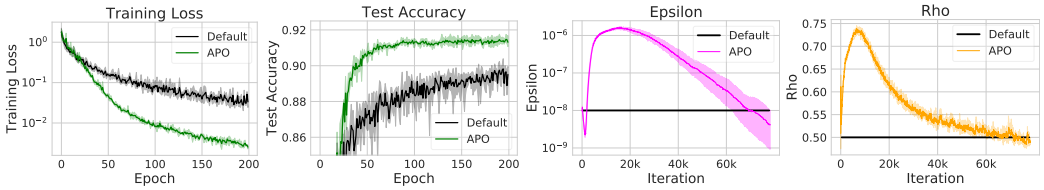


Figure 33: Using APO to tune the $\{\epsilon, \rho\}$ hyperparameters of RMSprop, for ResNet32 trained on CIFAR-10.

E Importance of Using the Same Mini-batch

In this section, we verify empirically the importance of computing the first term of the meta-objective on the *same mini-batch* that is used to compute the gradient of the base optimizer. Recall that our meta-objective is:

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2}||u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}||_2^2 \tag{8}$$

Note that the loss term is computed using the same mini-batch $\mathcal{B}$ used for the update $u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})$. An alternative is to evaluate the loss term on a randomly-sampled mini-batch $\mathcal{B}' \sim \mathcal{D}$, yielding:

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}'}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2}||u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}||_2^2 \tag{9}$$

If we set $\lambda_{\text{FSD}} = \lambda_{\text{WSD}} = 0$, then Eq. 9 becomes the meta-objective analyzed by Wu et al. [84], which was used to illustrate the short-horizon bias issue in stochastic meta-optimization.

Figures 34,35 show the result of using Eq. 8 vs. Eq. 9 for meta-optimizing the learning rate of a ResNet32 model on CIFAR-10. We observe that when using a random mini-batch $\mathcal{B}'$ to compute the loss term, the learning rate decays rapidly, preventing long-term progress; in contrast, using the same mini-batch $\mathcal{B}$ for the loss term yields a reasonable LR schedule which does not suffer from the short-horizon bias issue empirically.

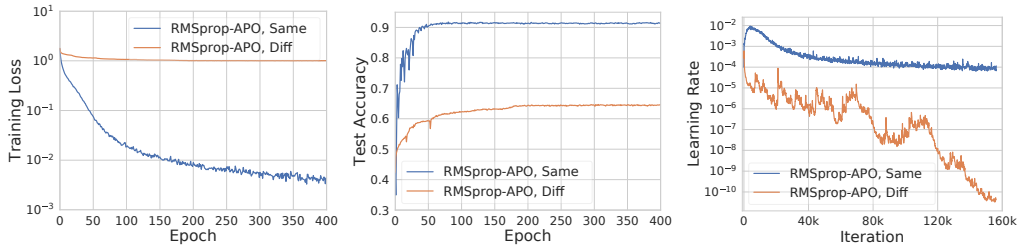


Figure 34: **Left, Center:** Training loss/Test accuracy or RMSprop-APO compared between 1) using the same mini-batch for the first term and 2) using a different mini-batch for the first term. **Right:** Comparison of learning rates for the two conditions.

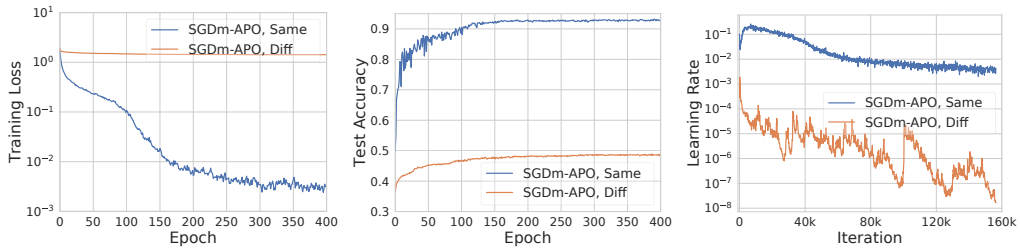


Figure 35: **Left, Center:** Training loss/Test accuracy of SGDm-APO compared between 1) using the same mini-batch for the first term and 2) using a different mini-batch for the first term. **Right:** Comparison of learning rates for the two conditions.

Same Mini-Batch for the FSD Term. Another alternative to the meta-objective is to compute the FSD term using the same mini-batch used to compute the loss term, as shown in Eq. 10.

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \frac{\lambda_{\text{FSD}}}{|\mathcal{B}|} \sum_{(\tilde{\mathbf{x}}, \cdot) \in \mathcal{B}} D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}}) + \frac{\lambda_{\text{WSD}}}{2} ||u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}||_2^2 \quad (10)$$

Recall that $D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}}) = \rho(f(\tilde{\mathbf{x}}, u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})), f(\tilde{\mathbf{x}}, \boldsymbol{\theta}))$. The function evaluation $f(\tilde{\mathbf{x}}, u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}))$ must be performed to compute the loss term; thus, using the same mini-batch $\mathcal{B}$ for the FSD term can allow us to reduce computation by re-using this function output in both the loss and FSD computations. However, with the interpretation of the FSD term in Eq. 10 as a Monte Carlo estimate of the expectation $\mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})]$, using the same mini-batch for the loss and dissimilarity would yield a biased estimate. The effect of using a different vs the same mini-batch to compute the FSD term is shown in Figure 36.

F Approximate Closed-Form Solution for the PPM

In Section 2.1, we introduced a general update rule for the stochastic PPM, which is defined as follows:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \arg\min_{\mathbf{u} \in \mathbb{R}^m} \mathcal{J}_{\mathcal{B}}(\mathbf{u}) + \lambda_{\text{FSD}} \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[D_{\text{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \mathbf{x})] + \lambda_{\text{WSD}} D_{\text{W}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}). \quad (11)$$

We first take the infinitesimal limit by letting $\lambda_{\text{FSD}} \to \infty$. Then, the optimal update $\mathbf{u}^\star$ will stay close to the current parameters $\boldsymbol{\theta}^{(t)}$ and we can approximate the loss function with the first-order Taylor

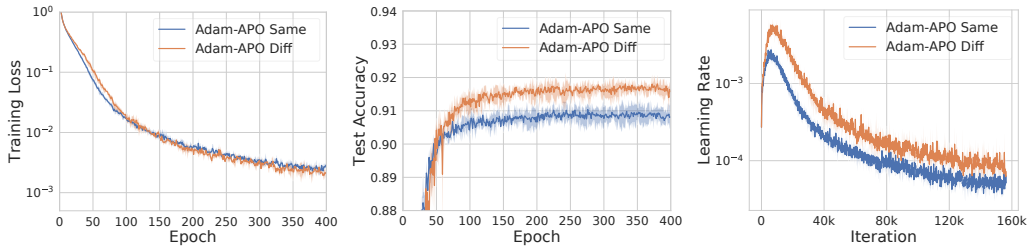


Figure 36: **Left, Center:** Training loss/Test accuracy of Adam-APO compared between 1) using the same mini-batch for the second term as for the first term of the meta-objective and 2) using a different mini-batch for the second term. **Right:** Comparison of learning rates for the two conditions.

approximation. Moreover, since the FSD term $\mathbb{E}_{\tilde{\mathbf{x}}\sim\mathcal{D}}[D_{\mathrm{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \mathbf{x})]$ is minimized when $\mathbf{u} = \boldsymbol{\theta}^{(t)}$, we have $\nabla_{\mathbf{u}}\mathbb{E}_{\tilde{\mathbf{x}}\sim\mathcal{D}}[D_{\mathrm{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \mathbf{x})]|_{\mathbf{u}=\boldsymbol{\theta}^{(t)}} = \mathbf{0}$. Hence, the second-order Taylor expansion of the FSD term would be:

$$\mathbb{E}_{\tilde{\mathbf{x}}\sim\mathcal{D}}[D_{\mathrm{F}}(f(\tilde{\mathbf{x}}, \mathbf{u}), f(\tilde{\mathbf{x}}, \boldsymbol{\theta}^{(t)}))] \approx \frac{1}{2}(\mathbf{u} - \boldsymbol{\theta}^{(t)})^{\top}\mathbf{G}(\mathbf{u} - \boldsymbol{\theta}^{(t)}), \tag{12}$$

where $\mathbf{G} = \nabla^2_{\mathbf{u}}\mathbb{E}_{\tilde{\mathbf{x}}\sim\mathcal{D}}[D_{\mathrm{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \mathbf{x})]|_{\mathbf{u}=\boldsymbol{\theta}^{(t)}}$ is the local Hessian of the FSD term. We further let the weight space discrepancy function to be the squared Euclidean distance $D_{\mathrm{W}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}) = {}^1\!/_2\|\mathbf{u}-\boldsymbol{\theta}^{(t)}\|_2^2$. Combining these insights, we approximate Eqn. 11 by linearizing the loss and taking a quadratic approximation to the FSD term:

$$\boldsymbol{\theta}^{(t+1)} \approx \underset{\mathbf{u}\in\mathbb{R}^m}{\arg\min}\left[\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^{\top}(\mathbf{u} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\mathrm{FSD}}}{2}(\mathbf{u} - \boldsymbol{\theta}^{(t)})^{\top}\mathbf{G}(\mathbf{u} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\mathrm{WSD}}}{2}\|\mathbf{u} - \boldsymbol{\theta}^{(t)}\|^2\right]. \tag{13}$$

Taking the gradient with respect to $\mathbf{u}$ and setting it equal to $\mathbf{0}$, we have:

$$\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \lambda_{\mathrm{FSD}}\mathbf{G}\mathbf{u} - \lambda_{\mathrm{FSD}}\mathbf{G}\boldsymbol{\theta}^{(t)} + \lambda_{\mathrm{WSD}}\mathbf{u} - \lambda_{\mathrm{WSD}}\boldsymbol{\theta}^{(t)} = \mathbf{0}. \tag{14}$$

Rearranging the terms, we arrive at the approximate closed-form solution:

$$\mathbf{u}^{\star} \approx \boldsymbol{\theta}^{(t)} - (\lambda_{\mathrm{FSD}}\mathbf{G} + \lambda_{\mathrm{WSD}}\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}). \tag{15}$$

G Optimization Algorithms Derived from the Proximal Objective

In this section, we show how several classical optimization algorithms can be derived from the proximal objective introduced in Section 2.2. Given a mini-batch $\mathcal{B}$, recall that we consider the following proximal objective for optimizing the model parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{(t+1)} = \underset{\boldsymbol{\theta}\in\mathbb{R}^m}{\arg\min}\left\{\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) + \lambda_{\mathrm{FSD}}\mathbb{E}_{\tilde{\mathbf{x}}}[D_{\mathrm{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}})] + \frac{\lambda_{\mathrm{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2\right\}. \tag{16}$$

Method	**Loss Term Approx.** $\mathcal{J}_{\mathcal{B}}(\mathbf{u})$	**FSD Term Approx.** $D_{\mathrm{F}}(\mathbf{u}, \boldsymbol{\theta}, \mathbf{x})$	**FSD Term Choice of** D_{F}	**WSD Term** $\frac{1}{2}\|\|\mathbf{u} - \boldsymbol{\theta}\|\|^2$
Gradient Descent	1st-order	-	-	✓
Newton's Method	2nd-order	-	-	✗
Damped Newton's Method	2nd-order	-	-	✓
Natural Gradient	1st-order	2nd-order	KL	✗
Damped Natural Gradient	1st-order	2nd-order	KL	✓
Generalized Gauss-Newton	1st-order	2nd-order	Bregman	✗
Damped Generalized Gauss-Newton	1st-order	2nd-order	Bregman	✓
Direct Proximal Optimization	Exact	Exact	Any	✓

G.1 Gradient Descent

To derive vanilla gradient descent, we take the first-order Taylor series approximation to the loss term $\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$ about $\boldsymbol{\theta}^{(t)}$, and set $\lambda_{\text{FSD}} = 0$. We need to have $\lambda_{\text{WSD}} > 0$ to ensure that we do not take an infinitely large step. Thus, each step of gradient descent minimizes an approximation to the proximal objective which we denote by $P^{\text{GD}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\boldsymbol{\theta}} P^{\text{GD}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}), \tag{17}$$

where:

$$P^{\text{GD}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^{\top}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2. \tag{18}$$

Now, we set the gradient $\nabla_{\boldsymbol{\theta}} P^{\text{GD}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = 0$ and solve for $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \left(\nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^{\top}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \right) = 0 \tag{19}$$

$$\nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \lambda_{\text{WSD}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) = 0 \tag{20}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)} - \frac{1}{\lambda_{\text{WSD}}} \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{21}$$

G.2 Newton's Method

To derive Newton's method, we take the second-order Taylor series approximation to the loss term $\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$ about $\boldsymbol{\theta}^{(t)}$, and set $\lambda_{\text{FSD}} = 0$. Below we derive the general form for the *damped* Newton update, which incorporates the WSD term; setting $\lambda_{\text{WSD}} = 0$ leads to the undamped Newton update as a special case. Each step of Newton's method minimizes an approximation to the proximal objective which we denote by $P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\boldsymbol{\theta}} P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}), \tag{22}$$

where:

$$P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^{\top}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^{\top}\mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2. \tag{23}$$

In Eq. 23, $\mathbf{H} = \nabla^2_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$. Now, we set the gradient $\nabla_{\boldsymbol{\theta}} P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = 0$ and solve for $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = 0 \tag{24}$$

$$\nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \mathbf{H}\boldsymbol{\theta} - \mathbf{H}\boldsymbol{\theta}^{(t)} + \lambda_{\text{WSD}}\boldsymbol{\theta} - \lambda_{\text{WSD}}\boldsymbol{\theta}^{(t)} = 0 \tag{25}$$

$$(\mathbf{H} + \lambda_{\text{WSD}}\mathbf{I})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) = -\nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{26}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)} - (\mathbf{H} + \lambda_{\text{WSD}}\mathbf{I})^{-1} \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{27}$$

G.3 Generalized Gauss-Newton Method

Next, we consider taking the first-order Taylor series approximation to the loss term, and an arbitrary function-space discrepancy D_{F} approximated by its second-order Taylor series expansion. We will use the notation $\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta})$ to denote the outputs of a neural network with parameters $\boldsymbol{\theta}$ on inputs $\mathbf{x}$. We have the following proximal objective:

$$P_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^{\top}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \lambda_{\text{FSD}}\mathbb{E}_{\tilde{\mathbf{x}}}[D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}}))] + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \tag{28}$$

Taking the second-order approximation to D_{F} (with respect to $\boldsymbol{\theta}$), we have:

$$\begin{aligned} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \approx \underbrace{D_{\text{F}}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta}^{(t)})}_{=0} + \underbrace{\nabla_{\boldsymbol{\theta}} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|^{\top}_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})}_{=0} \\ + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^{\top} \nabla^2_{\boldsymbol{\theta}} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}). \end{aligned} \tag{29}$$

Using the chain rule, we can derive $\nabla^2_{\boldsymbol{\theta}} D_{\mathrm{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$ as follows:

$$\nabla^2_{\boldsymbol{\theta}} D_{\mathrm{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} = \underbrace{\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}}\right)^\top}_{\mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}}} \underbrace{\left(\frac{\partial^2 \rho}{\partial \mathbf{z}^2}\right)}_{\mathbf{H}_\rho} \underbrace{\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}}\right)}_{\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}}} = \mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}} \nabla^2_{\mathbf{z}} \rho(\mathbf{z}, \mathbf{z}^{(t)}) \mathbf{J}_{\mathbf{z}\boldsymbol{\theta}} = \mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}} \mathbf{H}_\rho \mathbf{J}_{\mathbf{z}\boldsymbol{\theta}} \tag{30}$$

Letting $\mathbf{G}$ denote the expectation of the Hessian of the FSD function, $\mathbf{G} = \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[\nabla^2_{\boldsymbol{\theta}} D_{\mathrm{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}})]$, we have:

$$P_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{FSD}}}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^\top \mathbf{G} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \tag{31}$$

If we set the gradient $\nabla_{\boldsymbol{\theta}} P_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = 0$ and solve for $\boldsymbol{\theta}$, we obtain:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - (\lambda_{\text{FSD}} \mathbf{G} + \lambda_{\text{WSD}} \mathbf{I})^{-1} \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{32}$$

H Optimizing the 1-Step Meta-Objective Recovers Classic Methods

In this section, we show that when we approximate the loss term and FSD term of our meta-objective $\mathcal{Q}(\boldsymbol{\phi})$ and solve for the analytic $\arg\min_{\boldsymbol{\phi}} \mathcal{Q}(\boldsymbol{\phi})$, we recover the preconditioners corresponding to classic first- and second-order algorithms, including gradient descent, Gauss-Newton, Generalized Gauss-Newton, and natural gradient. The preconditioners used by each of these methods are shown in Table 1. These results parallel those for directly optimizing the parameters $\boldsymbol{\theta}$ using the proximal objective, but optimizing for the meta-parameters $\boldsymbol{\phi} = \mathbf{P}$.

Throughout this exposition, we use the notation:

$$u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) = u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) = \boldsymbol{\theta} - \mathbf{P} \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \mathbf{P}\mathbf{g} \tag{33}$$

where $\mathbf{g} = \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$. Note that $\mathbf{g}$ implicitly depends on the data $\mathcal{B}$; we use this shorthand to simplify the exposition.

Theorem. *Consider an approximation $\hat{\mathcal{Q}}(\mathbf{P})$ to the meta-objective (Eq. 4) where the loss term is linearized around the current weights $\boldsymbol{\theta}$ and the FSD term is replaced by its second-order approximation around $\boldsymbol{\theta}$:*

$$\begin{aligned}\hat{\mathcal{Q}}(\mathbf{P}) =& \mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \Big[\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})^\top (u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) - \boldsymbol{\theta}) \\ &+ \lambda_{FSD} (u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) - \boldsymbol{\theta})^\top \mathbf{G} (u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) - \boldsymbol{\theta}) + \frac{\lambda_{WSD}}{2} \|u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) - \boldsymbol{\theta}\|^2 \Big].\end{aligned} \tag{34}$$

Denote the gradient on a mini-batch as $\mathbf{g} = \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$, and assume that the second moment matrix $\mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathbf{g}\mathbf{g}^\top\right]$ is non-singular. Then, the preconditioning matrix which minimizes $\hat{\mathcal{Q}}$ is given by $\mathbf{P}^\star = (\lambda_{FSD} \mathbf{G} + \lambda_{WSD} \mathbf{I})^{-1}$, where $\mathbf{G}$ denotes the Hessian of the FSD evaluated at $\boldsymbol{\theta}$.

Proof. We have the following meta-objective (where the loss term is expanded using its first-order Taylor series approximation):

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \Big\{ \mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \Big[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) + \underbrace{\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})}_{\mathbf{g}}^\top ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) \tag{35}$$

$$+ \lambda_{\text{FSD}} \mathbb{E}_{(\tilde{\mathbf{x}}, \cdot) \sim \mathcal{D}} \left[D_{\mathrm{F}}(u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}}) \right] + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \mathbf{P}\mathbf{g} - \boldsymbol{\theta}||^2 \Big] \Big\} \tag{36}$$

Let us first derive the second-order Taylor series approximation to the discrepancy term $D_{\mathrm{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})$. To simplify notation, let $d(\boldsymbol{\theta}') \equiv D_{\mathrm{F}}(\boldsymbol{\theta}', \boldsymbol{\theta}, \tilde{\mathbf{x}})$ where $\boldsymbol{\theta}$ and the data $\tilde{\mathbf{x}}$ are implicit. Then, the second-order expansion of d about $\boldsymbol{\theta}$ is:

$$d(\boldsymbol{\theta}') \approx \underbrace{d(\boldsymbol{\theta})}_{=0} + \underbrace{\nabla d(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta})}_{=0} + \frac{1}{2} (\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \nabla^2_{\boldsymbol{\theta}} d(\boldsymbol{\theta}) (\boldsymbol{\theta}' - \boldsymbol{\theta}) \tag{37}$$

We have $d(\boldsymbol{\theta}) = D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}, \tilde{\mathbf{x}}) = \rho(f(\tilde{\mathbf{x}}, \boldsymbol{\theta}), f(\tilde{\mathbf{x}}, \boldsymbol{\theta})) = 0$, and $\nabla d(\boldsymbol{\theta}) = 0$ because $\boldsymbol{\theta}$ is a minimum of d, so the first two terms are 0. Denote the network output on example $\tilde{\mathbf{x}}$ by $\mathbf{z} = f(\tilde{\mathbf{x}}, \boldsymbol{\theta})$. To compute $\nabla^2_{\boldsymbol{\theta}} d(\boldsymbol{\theta})$, we have:

$$\nabla^2_{\boldsymbol{\theta}} d(\boldsymbol{\theta}) = \underbrace{\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}}\right)^\top}_{\mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}}} \underbrace{\left(\frac{\partial^2 \rho}{\partial \mathbf{z}^2}\right)}_{\mathbf{H}_\rho} \underbrace{\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}}\right)}_{\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}}} = \mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}} \mathbf{H}_\rho \mathbf{J}_{\mathbf{z}\boldsymbol{\theta}} \tag{38}$$

To simplify notation, we let $\mathbf{G}$ denote the expectation of the Hessian of the FSD function, $\mathbf{G} \equiv \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[\nabla^2_{\boldsymbol{\theta}} d(\boldsymbol{\theta})]$. Plugging this into the meta-objective, we have:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P} \mathbf{g} + \frac{\lambda_{\text{FSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{G} \mathbf{P} \mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P} \mathbf{g} \right] \right\} \tag{39}$$

Next, we take the gradient with respect to $\mathbf{P}$ and set it to 0 to solve for $\mathbf{P}$:

$$\nabla_{\mathbf{P}} \left(\mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P} \mathbf{g} + \frac{\lambda_{\text{FSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{G} \mathbf{P} \mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P} \mathbf{g} \right] \right) = 0 \tag{40}$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^\top + \frac{\lambda_{\text{FSD}}}{2} (\mathbf{G}\mathbf{P}\mathbf{g}\mathbf{g}^\top + \mathbf{G}^\top \mathbf{P}\mathbf{g}\mathbf{g}^\top) + \lambda_{\text{WSD}} \mathbf{P}\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{41}$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^\top + \lambda_{\text{FSD}} \mathbf{G}\mathbf{P}\mathbf{g}\mathbf{g}^\top + \lambda_{\text{WSD}} \mathbf{P}\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{42}$$

$$(\lambda_{\text{FSD}} \mathbf{F}\mathbf{P} + \lambda_{\text{WSD}} \mathbf{P} - \mathbf{I}) \mathbb{E}\left[\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{43}$$

Assuming that the second moment matrix $\mathbb{E}_{\mathcal{B} \sim \mathcal{D}}[\mathbf{g}\mathbf{g}^\top]$ is non-singular, we have:

$$\lambda_{\text{FSD}} \mathbf{G}\mathbf{P} + \lambda_{\text{WSD}} \mathbf{P} - \mathbf{I} = 0 \tag{44}$$

$$(\lambda_{\text{FSD}} \mathbf{G} + \lambda_{\text{WSD}} \mathbf{I}) \mathbf{P} = \mathbf{I} \tag{45}$$

$$\mathbf{P} = (\lambda_{\text{FSD}} \mathbf{G} + \lambda_{\text{WSD}} \mathbf{I})^{-1} \tag{46}$$

Thus, $\mathbf{P}^\star = (\lambda_{\text{FSD}} \mathbf{G} + \lambda_{\text{WSD}} \mathbf{I})^{-1}$. □

Below, we provide detailed derivations for several special cases (e.g. specific assumptions on the loss term and FSD term) which yield gradient descent, damped Newton's method, and natural gradient descent.

H.1 Gradient Descent

For gradient descent, we consider the first-order Taylor series approximation of the loss term about $\boldsymbol{\theta}$ and set $\lambda_{\text{FSD}} = 0$. We have:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) + \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})^\top ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \mathbf{P}\mathbf{g} - \boldsymbol{\theta}||^2 \right] \right\} \tag{47}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} ||\mathbf{P}\mathbf{g}||^2 \right] \right\} \tag{48}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right\} \tag{49}$$

Next, we take the gradient with respect to $\mathbf{P}$ and set it to 0 to solve for $\mathbf{P}^\star$:

$$\nabla_{\mathbf{P}} \left(\mathbb{E}_{\mathcal{B} \sim \mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right) = 0 \tag{50}$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^\top + \lambda_{\text{WSD}} \mathbf{P}\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{51}$$

$$\left(\mathbf{P} - \frac{1}{\lambda_{\text{WSD}}} \mathbf{I} \right) \mathbb{E}\left[\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{52}$$

Assuming that the second moment matrix $\mathbb{E}_{\mathcal{B} \sim \mathcal{D}}[\mathbf{g}\mathbf{g}^\top]$ is non-singular, we have:

$$\mathbf{P} = \frac{1}{\lambda_{\text{WSD}}} \mathbf{I} \tag{53}$$

Thus, the optimal preconditioner is $\mathbf{P} = \frac{1}{\lambda_{\text{WSD}}} \mathbf{I}$, which corresponds to using a global learning rate of $\frac{1}{\lambda_{\text{WSD}}}$, as we would expect from steepest descent with the WSD term.

H.2 (Damped) Newton's Method

For Newton's method, we consider the second-order Taylor series expansion of the loss term about $\boldsymbol{\theta}$, and set $\lambda_{\text{FSD}} = 0$. Here, we consider the generic case where we have $\lambda_{\text{WSD}} \geq 0$: if $\lambda_{\text{WSD}} > 0$ we obtain a damped Newton update, while if $\lambda_{\text{WSD}} = 0$ we obtain the undamped update. We have:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \Big\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \Big[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) + \underbrace{\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})^\top}_{\mathbf{g}} ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) \tag{54}$$

$$+ \frac{1}{2}((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta})^\top \mathbf{H}((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \mathbf{P}\mathbf{g} - \boldsymbol{\theta}||^2 \Big] \Big\} \tag{55}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{1}{2}\mathbf{g}^\top \mathbf{P}^\top \mathbf{H}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} ||\mathbf{P}\mathbf{g}||^2 \right] \right\} \tag{56}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{1}{2}\mathbf{g}^\top \mathbf{P}^\top \mathbf{H}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right\} \tag{57}$$

Next, we take the gradient with respect to $\mathbf{P}$ and set it to 0 to solve for $\mathbf{P}^\star$:

$$\nabla_{\mathbf{P}} \left(\mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{1}{2}\mathbf{g}^\top \mathbf{P}^\top \mathbf{H}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right) = 0 \tag{58}$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^\top + \frac{1}{2}\mathbf{H}\mathbf{P}\mathbf{g}\mathbf{g}^\top + \frac{1}{2}\mathbf{H}^\top \mathbf{P}\mathbf{g}\mathbf{g}^\top + \lambda_{\text{WSD}} \mathbf{P}\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{59}$$

$$(\mathbf{H}\mathbf{P} + \lambda_{\text{WSD}}\mathbf{P} - \mathbf{I})\, \mathbb{E}\left[\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{60}$$

Assuming that the second moment matrix $\mathbb{E}_{\mathcal{B}\sim\mathcal{D}}[\mathbf{g}\mathbf{g}^\top]$ is non-singular, we have:

$$\mathbf{H}\mathbf{P} + \lambda_{\text{WSD}}\mathbf{P} - \mathbf{I} = 0 \tag{61}$$

$$(\mathbf{H} + \lambda_{\text{WSD}}\mathbf{I})\mathbf{P} = \mathbf{I} \tag{62}$$

$$\mathbf{P} = (\mathbf{H} + \lambda_{\text{WSD}}\mathbf{I})^{-1} \tag{63}$$

Thus, if we take the second-order Taylor series approximation to the loss term and solve for the optimal $\mathbf{P}^\star$ through the 1-step lookahead $\boldsymbol{\theta}'(\mathbf{P})$, we obtain the (damped) inverse Hessian, $\mathbf{P}^\star = (\mathbf{H} + \lambda_{\text{WSD}}\mathbf{I})^{-1}$.

H.3 Natural Gradient

For natural gradient, we consider a first-order approximation to the loss term and a second-order approximation to the FSD term, where the output-space discrepancy measure ρ is taken to be the KL divergence.

The second derivative of the KL divergence yields the Fisher information matrix $\mathbf{F}$; here, we write $\mathbf{F} = \mathbf{J}_{\mathbf{z}\boldsymbol{\theta}}^\top \mathbf{H}_\rho \mathbf{J}_{\mathbf{z}\boldsymbol{\theta}}$. Plugging this into the meta-objective, we have:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{(\mathbf{x},\mathbf{t})\sim\mathcal{D}} \left[\mathcal{L}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{FSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{F}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right\} \tag{64}$$

Next, we take the gradient with respect to $\mathbf{P}$ and set it to 0 to solve for $\mathbf{P}^\star$:

$$(\lambda_{\text{FSD}}\mathbf{F}\mathbf{P} + \lambda_{\text{WSD}}\mathbf{P} - \mathbf{I})\mathbb{E}\left[\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{65}$$

Assuming that the second moment matrix $\mathbb{E}_{(\mathbf{x},\mathbf{t})\sim\mathcal{D}}[\mathbf{g}\mathbf{g}^\top]$ is non-singular, we have:

$$\mathbf{P} = (\lambda_{\text{FSD}}\mathbf{F} + \lambda_{\text{WSD}}\mathbf{I})^{-1} \tag{66}$$

I Optimizing the 1-Step Meta-Objective Yields the KFAC Update

In the previous section, we derived the optimal solutions to various approximate proximal objectives, optimizing over an unconstrained preconditioner $\mathbf{P}$. Here, we consider an analogous setup for a structured preconditioner, where $\mathbf{P}$ is block-diagonal with the ℓth block (corresponding to the ℓth layer in the network) given by $\mathbf{A}_\ell \otimes \mathbf{B}_\ell$.

I.1 KFAC Assumptions

This exposition of the KFAC assumptions is based on [24]. First we briefly introduce some notation that will be used in the assumptions. Suppose a layer of a neural network has weights $\mathbf{W}_\ell$ and that its input is the activation vector from the previous layer, $\mathbf{a}_{\ell-1}$. Then the output of the layer is a linear transformation of the input, followed by a nonlinear activation function σ:

$$\mathbf{s}_\ell = \mathbf{W}_\ell \mathbf{a}_{\ell-1} \tag{67}$$

$$\mathbf{a}_\ell = \sigma(\mathbf{s}_\ell) \tag{68}$$

where $\mathbf{s}_\ell$ are the pre-activations. In the backward pass, we have the following activation and weight derivatives:

$$\mathcal{D}\mathbf{a}_\ell = \mathbf{W}^\top \mathcal{D}\mathbf{s}_{\ell+1} \tag{69}$$

$$\mathcal{D}\mathbf{s}_\ell = \mathcal{D}\mathbf{a}_\ell \odot \sigma'(\mathbf{s}_\ell) \tag{70}$$

$$\mathcal{D}\mathbf{W}_\ell = \mathcal{D}\mathbf{s}_\ell \mathbf{a}_{\ell-1}^\top \tag{71}$$

KFAC [53] makes the following assumptions on the neural network being optimized:

- The layers are independent, such that that the pseudo-derivatives dw_i and dw_j are uncorrelated when w_i and w_j belong to different layers. If this assumption is satisfied, then the Fisher information matrix $\mathbf{F}$ will be block-diagonal.
- The activations $\{\mathbf{a}_\ell\}$ are independent of the pre-activation pseudo-gradients $\{\mathcal{D}\mathbf{s}_\ell\}$. If $\mathbf{a}_{\ell-1}$ is independent of $\mathcal{D}\mathbf{s}_\ell$, then the ℓth block of the Fisher is:

$$\hat{\mathbf{G}}_\ell = \mathbf{A}_\ell^{\text{KFAC}} \otimes \mathbf{B}_\ell^{\text{KFAC}} \tag{72}$$

  where:

$$\mathbf{A}_\ell^{\text{KFAC}} = \mathbb{E}[\mathbf{a}_{\ell-1}\mathbf{a}_{\ell-1}^\top] \tag{73}$$

$$\mathbf{B}_\ell^{\text{KFAC}} = \mathbb{E}[\mathcal{D}\mathbf{s}_\ell \mathcal{D}\mathbf{s}_\ell^\top] \tag{74}$$

I.2 Proof of Corollary 2

Corollary. *Suppose that (1) the assumptions for Theorem 1 are satisfied, (2) the FSD term measures the KL divergence, and (3) $\lambda_{WSD} = 0$ and $\lambda_{FSD} = 1$. Moreover, suppose that the parameters $\boldsymbol{\theta}$ satisfy the KFAC assumptions listed in Appendix I. Then, the optimal solution to the approximate meta-objective recovers the KFAC update, which can be represented using the structured preconditioner in Eq. 5.*

Proof. If the KFAC assumptions are satisfied, then $\mathbf{F}^{-1}$ is block-diagonal with the block corresponding to the ℓth layer expressed as $\mathbf{G}_\ell = \mathbf{A}_\ell^{-1} \otimes \mathbf{B}_\ell^{-1}$ where $\mathbf{A}_\ell = \mathbb{E}[\bar{\mathbf{a}}_{\ell-1}\bar{\mathbf{a}}_{\ell-1}^\top]$ and $\mathbf{B}_\ell = \mathbb{E}[\mathcal{D}\mathbf{s}_\ell \mathcal{D}\mathbf{s}_\ell^\top]$. By Theorem 1, the optimal solution to the approximate meta-objective is $\mathbf{P}^\star = \mathbf{F}^{-1}$. Hence $\mathbf{P}^\star$ is this block-diagonal matrix, which can be expressed using our structured parameterization (Eq. 5). Thus, APO recovers the KFAC update. □

J Ablations

J.1 Ablation Over λ_{WSD} and λ_{FSD}

Figure 37 provides an ablation over λ_{FSD} for training an MLP on MNIST using APO to tune the global learning rate for RMSprop (e.g. RMSprop-APO).

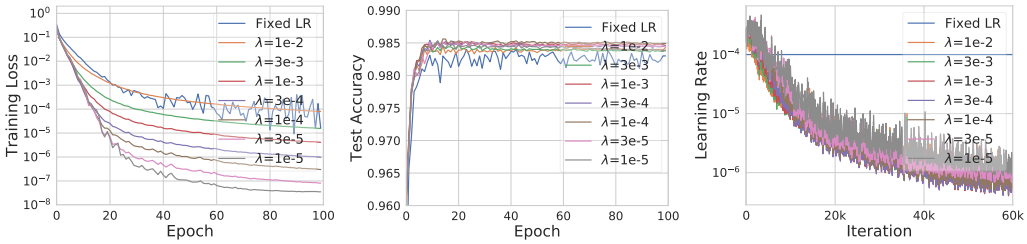


Figure 37: **Ablation over λ_{FSD} for training a two-layer MLP with 1000 hidden units per layer on MNIST, using RMSprop-APO.**

Figure 38 provides an ablation over λ_{WSD} and λ_{FSD} for training AlexNet on CIFAR-10 using APO to adapt the preconditioning matrix. We kept the other proximity weight fixed when performing the experiments. We found that APO-Precond is robust in various ranges of proximity weights λ.

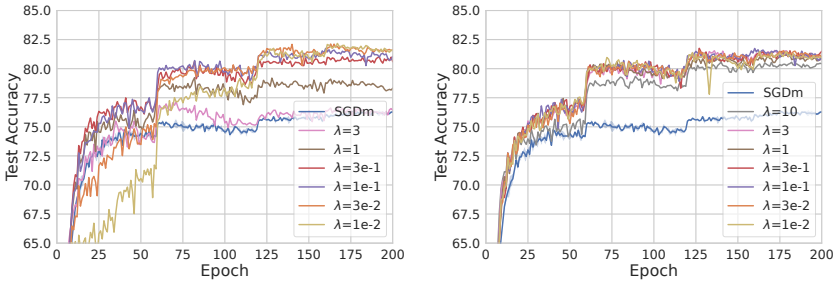
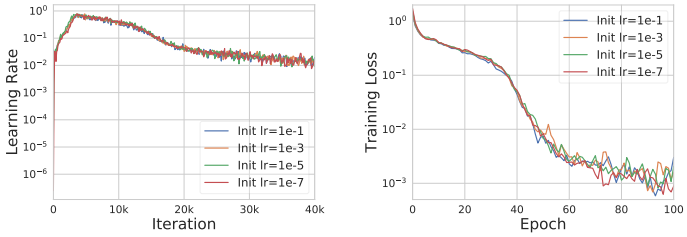


Figure 38: **Ablation over (left) λ_{WSD} and (right) λ_{FSD} for training AlexNet on CIFAR-10, using APO-Precond.**

J.2 Robustness to Initial Learning Rate and Meta-Update Interval

We show in Figure 39(a,b) that when using APO to tune the global learning rate, APO is robust to the initial learning rate of the base optimizer, achieving nearly identical training loss and test accuracy using initial learning rates that span 6 orders of magnitude, from 1e-7 to 1e-1.



(a) (b)

Figure 39: **Robustness to the initial learning rate.** Figures (a) and (b) show that APO achieves almost identical learning rate schedules and training losses for initial learning rates spanning 6 orders of magnitude.

Next, we performed an ablation to evaluate how well APO performs with different meta-update intervals (i.e. when making more or less frequent updates to the learning rate and preconditioning matrix during training). We used APO to tune the learning rate, and experimented with performing meta-updates once every 10, 20, 50, and 100 base optimization iterations. We trained a ResNet32 model on CIFAR-10, and used SGDm as the base optimizer. The results are shown in Figure 40. We found that APO is robust to the meta-update interval, performing almost identically with respect to training loss, test accuracy, and the adaptive learning rate schedule, for each meta-update interval.

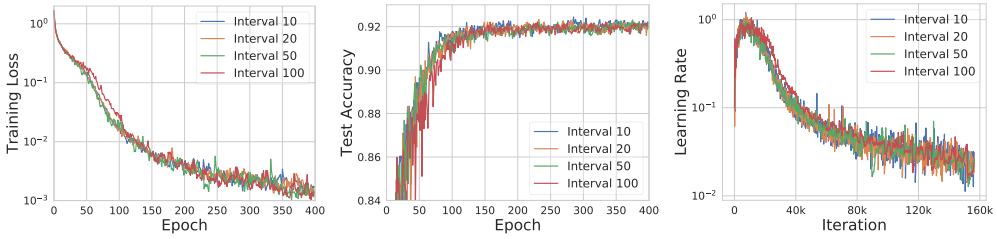


Figure 40: **Robustness to the meta-update interval.** Here, we trained ResNet32 on CIFAR-10 using SGDm-APO, with meta-updates performed once every $\{10, 20, 50, 100\}$ steps. We observe that APO performs similarly using intervals from 10 to 100, corresponding to computation times approximately $1.3\times$ to $1.03\times$ that of the base optimizer.

To further examine how meta-update intervals affect the preconditioning adaptation, we trained AlexNet model on CIFAR-10. We used APO to learn the preconditioning matrix and experimented with performing meta-updates once every 10, 20, 50, and 100 base optimization iterations. The results are shown in Figure 41. As in the learning rate adaptation problem, we found that APO performs similarly using intervals from 10 to 100, achieving higher test accuracy than the baseline method.

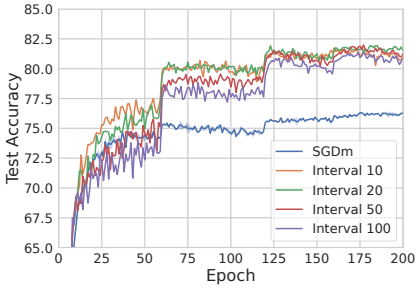


Figure 41: **Robustness to the meta-update interval on AlexNet. CIFAR-10, APO-Precond.**