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. [85], which suffers from short-horizon bias.
- In Section F, we provide a derivation of Eq. 4.
- 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. 3).
- 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. [85] 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 [13]	✗	✗	✓	✗	Any hyperparameter
PBT [34]	✗	✓	✓	✓	Any hyperparameter
BayesOpt [72]	✗	✗	✓	✗	Any hyperparameter
BPTT [50]	✓	✓	✓	✗	Differentiable hyperparams
RTHO [22]	✗	✓	✓	✓	Differentiable hyperparams
IFT [47]	✓	✓	✓	✓	Differentiable *regularization* hyperparams
L4 [59]	✗	✓	✓	✓	LR
MARTHE [17]	✗	✓	✓	✓	LR
FDS [58]	✗	✗	✓	✗	LR, momentum, WD
Meta-SGD [45]	✗	✗	✗	$\sim$	Diagonal preconditioner
Meta-Curvature [66]	✓	✗	✗	$\sim$	Preconditioner parameterized by tensor products
WarpGrad [21]	✓	✗	✗	✓	Preconditioning "warp-layers"
MetaMD [23]	✓	$\sim$	✗	✗	Parameterization of Bregman Divergence
Learned Optimizers [42, 2, 84, 57]	✓	✓	✗	✗	Learned optimizer parameters
HD [6]	✗	✓	✓	✓	LR
FOP [61]	✓	✓	✓	✓	$\mathbf{M}\mathbf{M}^\top$ 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 [76], KF-RTRL [62], OK [12]) 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 [82] 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. [22], and a method for LR adaptation that interpolates between hypergradient descent and RTRL, called MARTHE, was introduced in [17].

Black-Box Hyperparameter Optimization. Finding good optimization hyperparameters is a longstanding problem [10]. Black-box methods for hyperparameter optimization, such as grid search, random search [13], and Bayesian optimization [72, 75, 73], are expensive, as they require performing many complete training runs, and can only find fixed hyperparameter values (e.g., a constant learning rate). Hyperband [44] can reduce the cost by terminating poorly-performing runs early, but is still limited to finding fixed hyperparameters. Population Based Training (PBT) [34] 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 [28, 80, 81, 46] for automatic learning rate adaptation. In a similar vein, Rolinek & Martius [70] 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 [42, 43, 2, 84, 57, 84]. Li & Malik [42] 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 [74]. 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 [45] extends MAML [20] 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) [66] 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 [41] 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 [41, 21].

C Experimental Details

C.1 Computing Environment

All experiments were implemented using the PyTorch [67] and JAX [14] 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. [27]. 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 [54] 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 [86] 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 [86] 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. [49]. 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 [86].

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 [79] 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. [87], we used a diagonal block matrix in the structured preconditioner of the embedding weight matrix to reduce memory overhead. We used the Fairseq toolkit [65] 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.

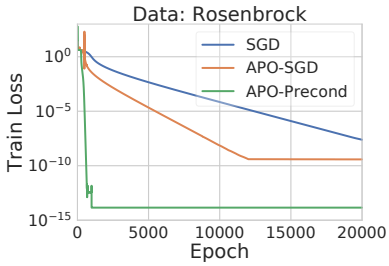


Figure 6: Training loss on the Rosenbrock function. Both the learning rate schedules and the preconditioner adapted by APO outperform the optimal fixed learning rate for SGD.

D.2 SVHN

We also evaluated APO on the Google Street View House Numbers dataset (SVHN) [63], using a WideResNet 16-8. We followed the experimental setup of Zagoruyko & Komodakis [86], 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).

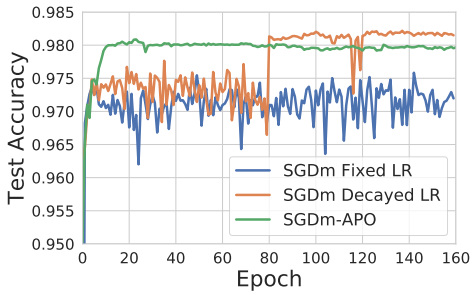


Figure 7: Test Accuracy on SVHN with SGDm.

D.3 CIFAR-10

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

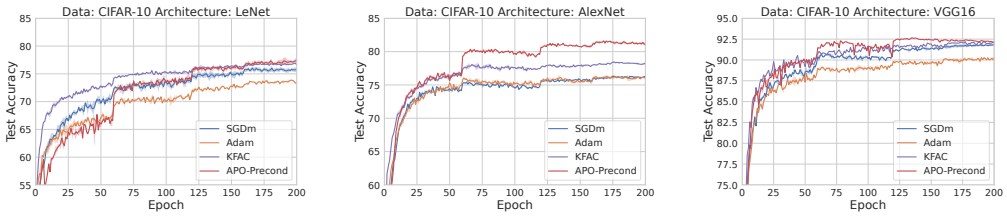


Figure 8: LeNet, AlexNet, and VGG16 on CIFAR-10, using SGDm, Adam, KFAC, and APO-Precond.

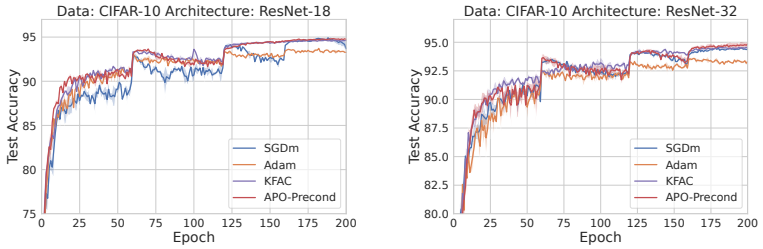


Figure 9: ResNet-18 and ResNet-32 on CIFAR-10, using SGDm, Adam, KFAC, and APO-Precond.

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.

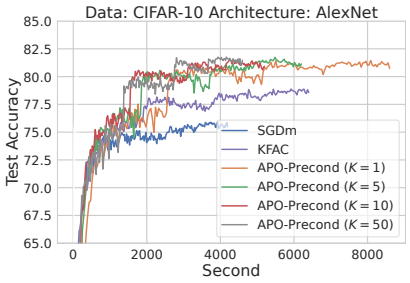


Figure 10: AlexNet on CIFAR-10, using SGDm, KFAC, and APO-Precond with various values of K. We show the test accuracy as a function of wall-clock time.

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.

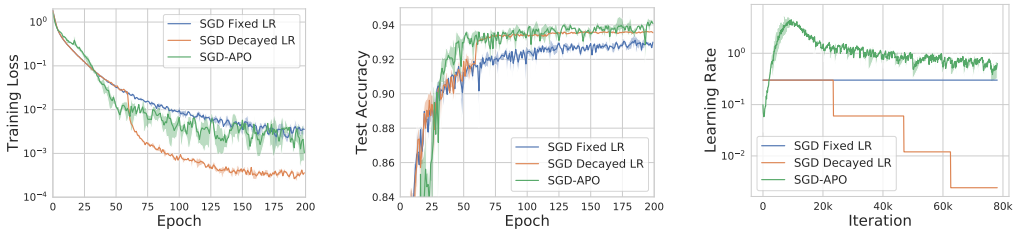


Figure 11: ResNet34 on CIFAR-10, using SGD. The shaded regions show the min/max values over 4 random restarts.

	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.

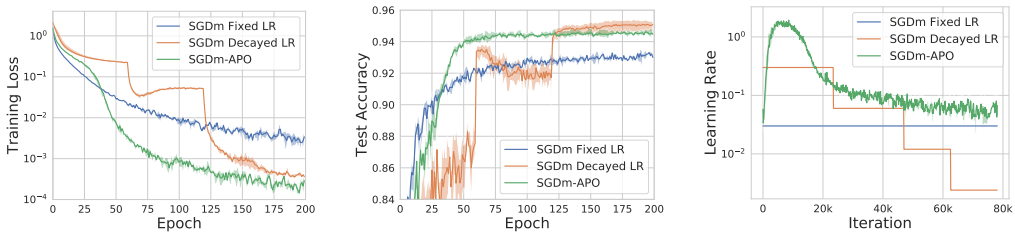


Figure 12: ResNet34 on CIFAR-10, using SGDm. The shaded regions show the min/max values over 4 random restarts.

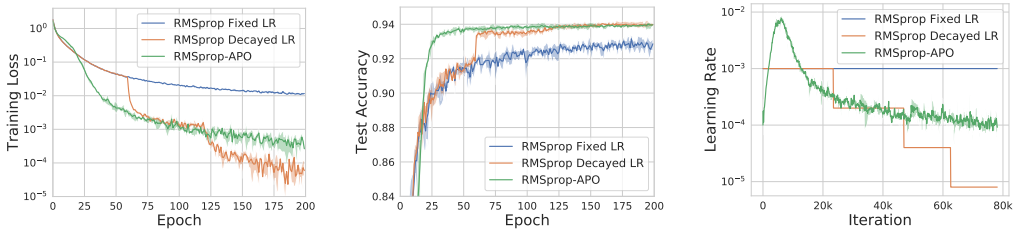


Figure 13: ResNet34 on CIFAR-10, using RMSprop. The shaded regions show the min/max values over 4 random restarts.

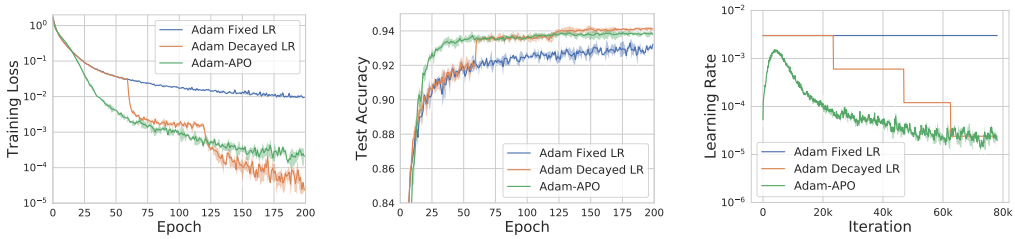


Figure 14: ResNet34 on CIFAR-10, using Adam. The shaded regions show the min/max values over 4 random restarts.

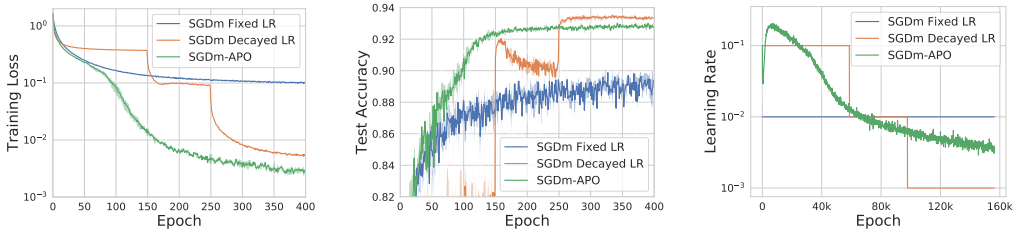


Figure 15: ResNet32 on CIFAR-10, using SGDm. The shaded regions show the min/max values over 4 random restarts.

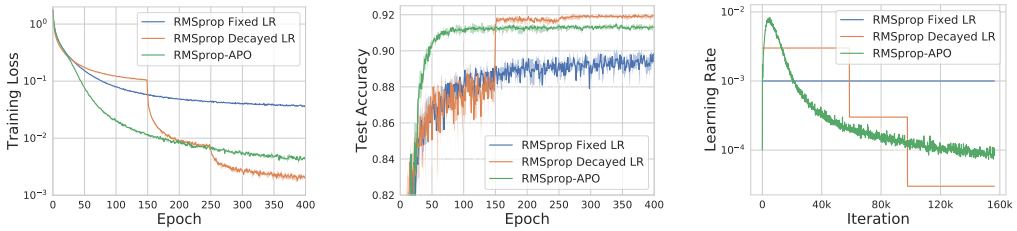


Figure 16: ResNet32 on CIFAR-10, using RMSprop. The shaded regions show the min/max values over 4 random restarts.

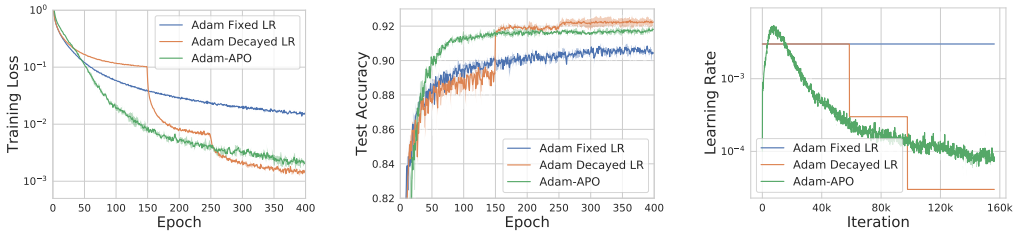


Figure 17: ResNet32 on CIFAR-10, using Adam. The shaded regions show the min/max values over 4 random restarts.

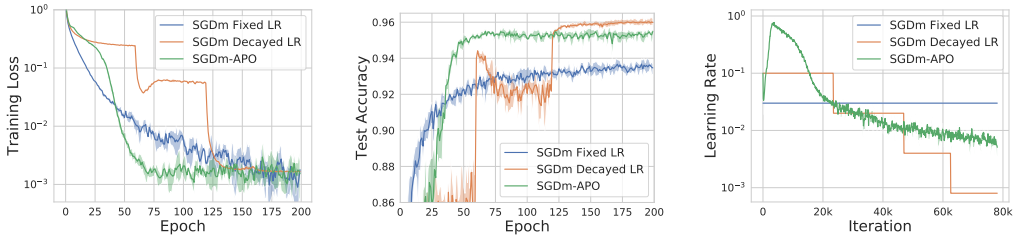


Figure 18: WideResNet 28-10 on CIFAR-10, using SGDm. The shaded regions show the min/max values over 4 random restarts.

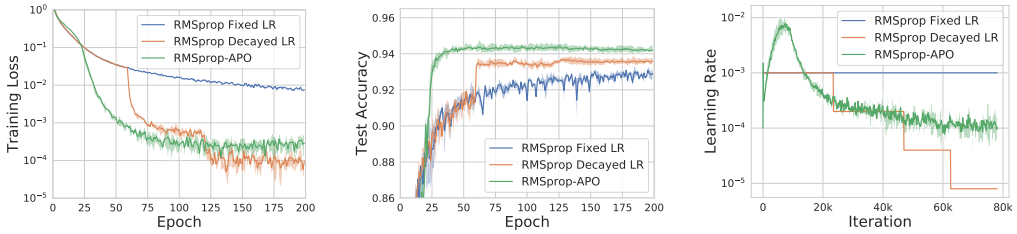


Figure 19: WideResNet 28-10 on CIFAR-10, using RMSprop. The shaded regions show the min/max values over 4 random restarts.

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.

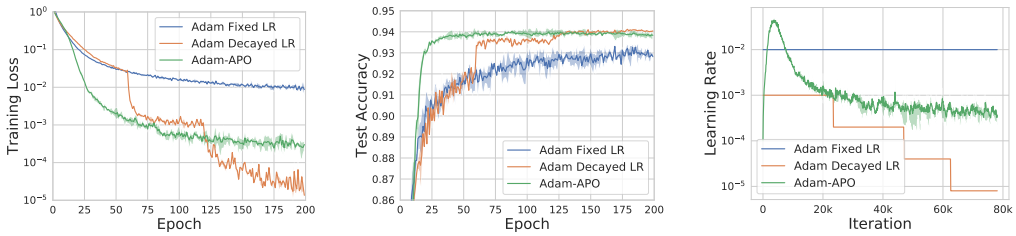


Figure 20: WideResNet 28-10 on CIFAR-10, using Adam. The shaded regions show the min/max values over 4 random restarts.

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.

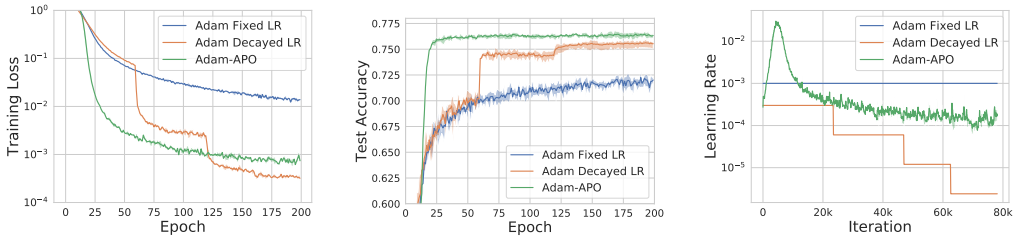


Figure 21: WideResNet on CIFAR-100, using Adam. The shaded regions show the min/max values over 4 random restarts.

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 .

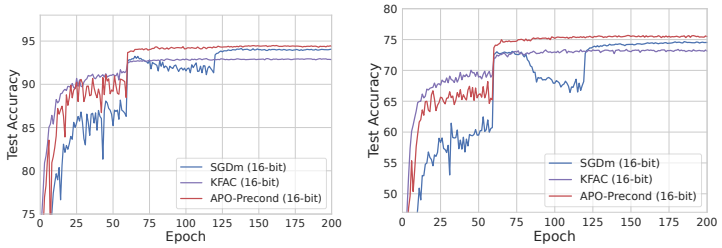


Figure 22: Test accuracy curves for 16-bit neural networks: ResNet-18 on CIFAR-10 (**left**) and CIFAR-100 (**right**).

D.6 Comparison to HD, L4, and FOP

In this section, we compare APO to Hypergradient Descent (HD) [6] and L4 [70] for the learning rate adaptation problem, and First-Order Preconditioning (FOP) [61] for the preconditioning adaptation problem.

D.6.1 Hypergradient Descent

Here, we compare to Hypergradient Descent (HD) [70], 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. [85], that was shown to lead to sub-optimal behavior when the objective is stochastic. In particular, [85] 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 [85]). 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** [6]	**L4** [70]	**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) [6], L4 [70], and the APO-discovered schedule, training ResNet-32 [29] 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.

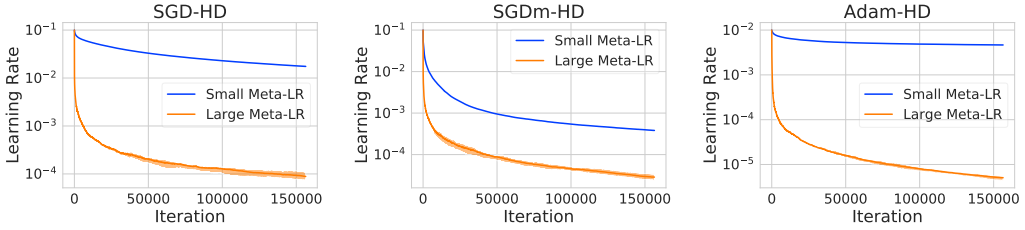


Figure 23: ResNet32, CIFAR-10, Hypergradient Descent (HD) [6] learning rate schedules for each optimizer SGD-HD, SGDm-HD, and Adam-HD. The shaded regions show the min/max values over 4 random restarts.

	CIFAR-10 (ResNet-34)				
	Fixed	**Decay**	**HD** [6]	**L4** [70]	**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) [6], L4 [70], and the APO-discovered schedule, training ResNet-34 [29] 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.

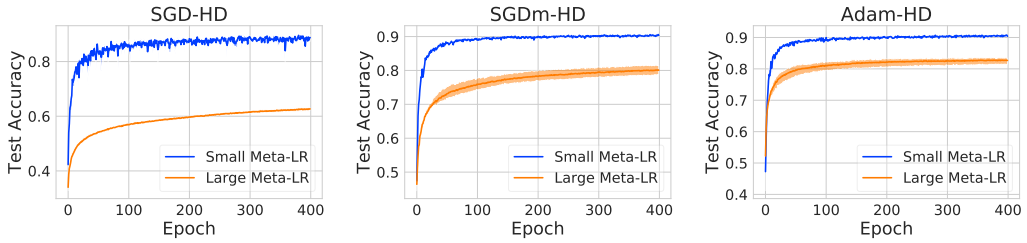


Figure 24: ResNet32, CIFAR-10, Hypergradient Descent (HD) [6] test accuracies for each optimizer SGD-HD, SGDm-HD, and Adam-HD. The shaded regions show the min/max values over 4 random restarts.

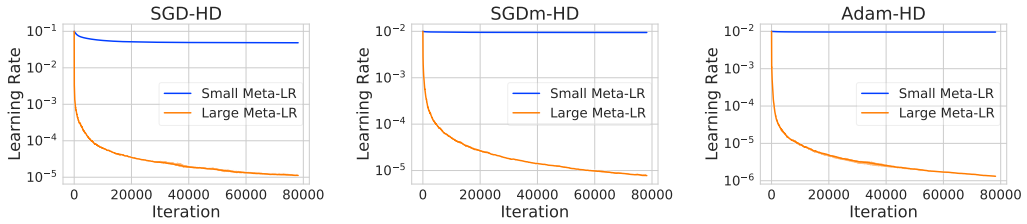


Figure 25: ResNet34, CIFAR-10, Hypergradient Descent (HD) [6] learning rate schedules for each optimizer SGD-HD, SGDm-HD, and Adam-HD. The shaded regions show the min/max values over 4 random restarts.

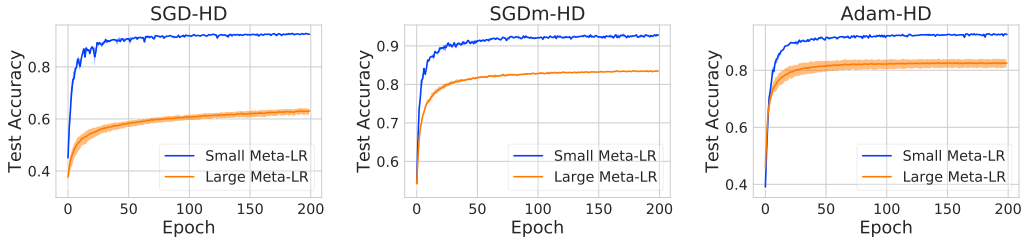


Figure 26: ResNet34, CIFAR-10, Hypergradient Descent (HD) [6] test accuracies for each optimizer SGD-HD, SGDm-HD, and Adam-HD. The shaded regions show the min/max values over 4 random restarts.

D.6.2 Comparison to L4

We also compared APO to the L4 optimizer [70], 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.

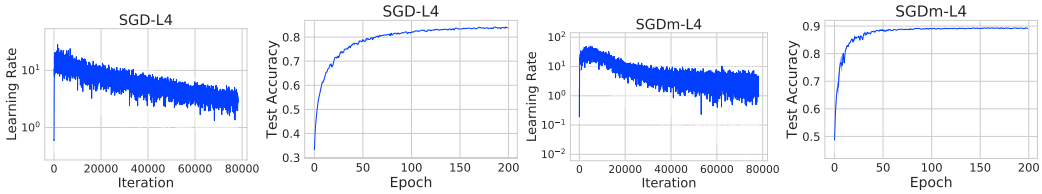


Figure 27: ResNet32 L4-SGDm and RMSprop-L4. The shaded regions show the min/max values over 3 random restarts.

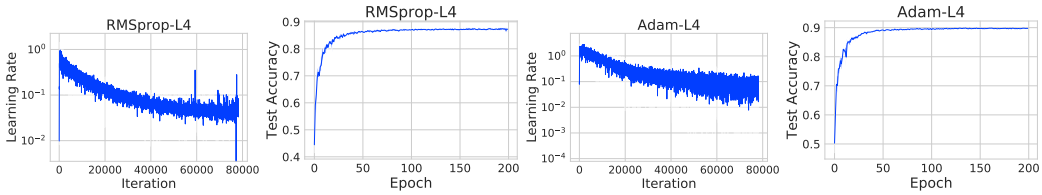


Figure 28: ResNet32 RMSprop-L4 and Adam-L4. The shaded regions show the min/max values over 3 random restarts.

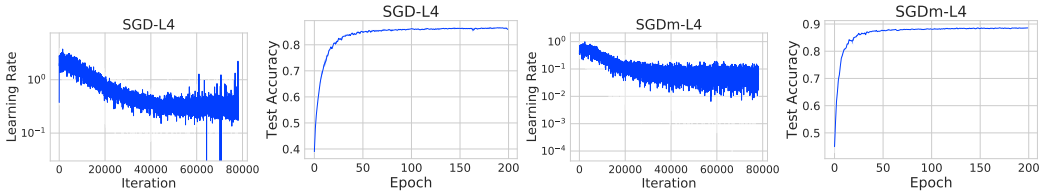


Figure 29: ResNet34 L4-SGDm and RMSprop-L4. The shaded regions show the min/max values over 3 random restarts.

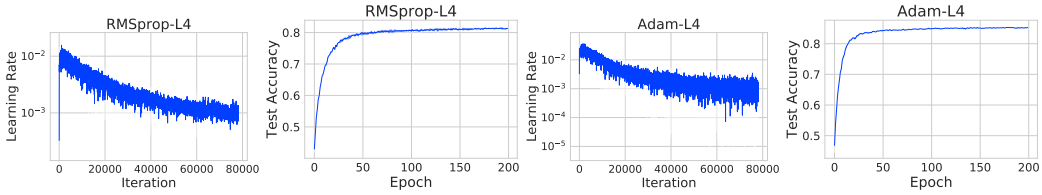


Figure 30: ResNet34 RMSprop-L4 and Adam-L4. The shaded regions show the min/max values over 3 random restarts.

D.6.3 First-Order Preconditioning

Task	Model	SGDm	Adam	KFAC	FOP	APO-Precond
CIFAR-10	**LeNet**	75.73 $\pm$ 0.50	73.41 $\pm$ 0.27	76.63 $\pm$ 0.51	75.25 $\pm$ 0.56	**77.42** $\pm$ 0.55
CIFAR-10	**AlexNet**	76.27 $\pm$ 0.24	76.09 $\pm$ 0.31	78.33 $\pm$ 0.28	76.52 $\pm$ 0.23	**81.14** $\pm$ 0.28
CIFAR-10	**VGG16**	91.82 $\pm$ 0.15	90.19 $\pm$ 0.31	92.05 $\pm$ 0.23	91.65 $\pm$ 0.41	**92.13** $\pm$ 0.17
CIFAR-10	**ResNet-18**	93.69 $\pm$ 0.45	93.27 $\pm$ 0.09	94.60 $\pm$ 0.04	93.76 $\pm$ 0.31	**94.75** $\pm$ 0.09
CIFAR-10	**ResNet-32**	94.40 $\pm$ 0.07	93.30 $\pm$ 0.14	94.49 $\pm$ 0.10	93.90 $\pm$ 0.28	**94.83** $\pm$ 0.19
CIFAR-100	**AlexNet**	43.95 $\pm$ 0.07	41.82 $\pm$ 0.28	46.24 $\pm$ 0.33	44.66 $\pm$ 0.29	**52.35** $\pm$ 0.26
CIFAR-100	**VGG16**	65.98 $\pm$ 0.25	60.61 $\pm$ 0.39	61.84 $\pm$ 0.37	61.64 $\pm$ 0.39	**67.95** $\pm$ 0.30
CIFAR-100	**ResNet-18**	76.85 $\pm$ 0.07	70.87 $\pm$ 0.26	76.48 $\pm$ 0.13	75.93 $\pm$ 0.23	**76.88** $\pm$ 0.06
CIFAR-100	**ResNet-32**	**77.47** $\pm$ 0.25	68.67 $\pm$ 0.65	75.70 $\pm$ 0.13	75.66 $\pm$ 0.27	77.41 $\pm$ 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 6, 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).

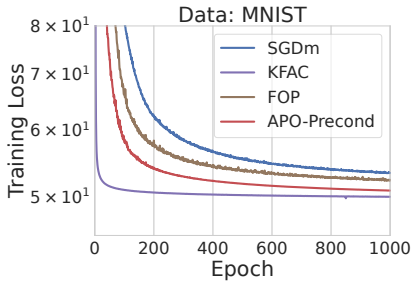


Figure 31: Deep autoencoder on MNIST. We compare the training loss of SGDm, KFAC, FOP, and APO-Precond.

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:

$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma)\mathbf{g}_t^2 \tag{11}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\mathbf{s}_t^{\rho} + \epsilon}\mathbf{g}_t \tag{12}$$

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 \{$1e-3, 1e-4, 1e-5$\}$ and $\lambda_{\text{WSD}} \in \{$1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6$\}$. We compared to baseline RMSprop with its optimal fixed learning rates ($\eta = 0.001$ for CIFAR-10 and $\eta = 0.0001$ for FashionMNIST) and with the default values $\epsilon =$ 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.

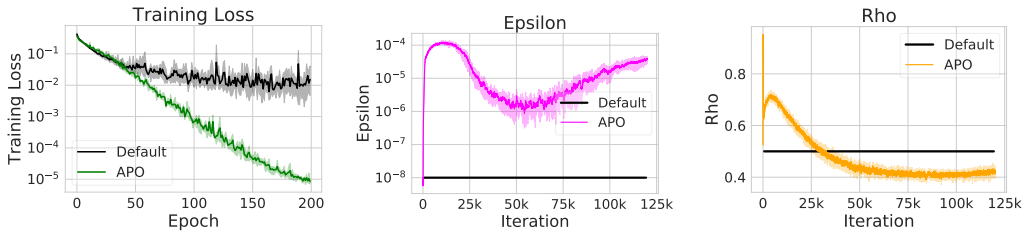


Figure 32: Using APO to tune the $\{\epsilon, \rho\}$ hyperparameters of RMSprop, for an MLP trained on FashionMNIST. Here, we used the best fixed learning rate obtained from a grid search using the best optimizer.

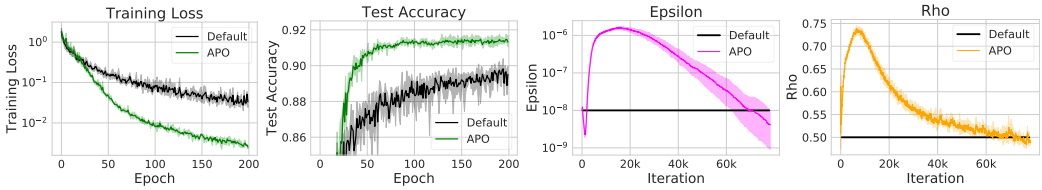


Figure 33: Using APO to tune the $\{\epsilon, \rho\}$ hyperparameters of RMSprop, for ResNet32 trained on CIFAR-10.

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:

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \lambda_{\text{FSD}} \mathbb{E}_{(\tilde{\mathbf{x}}, \cdot) \sim \mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2} ||u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}||_2^2 \quad (13)$$

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:

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}'}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \lambda_{\text{FSD}} \mathbb{E}_{(\tilde{\mathbf{x}}, \cdot) \sim \mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2} ||u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}||_2^2 \quad (14)$$

If we set $\lambda_{\text{FSD}} = \lambda_{\text{WSD}} = 0$, then Eq. 14 becomes the meta-objective analyzed by Wu et al. [85], which was used to illustrate the short-horizon bias issue in stochastic meta-optimization.

Figures 34,35 show the result of using Eq. 13 vs. Eq. 14 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.

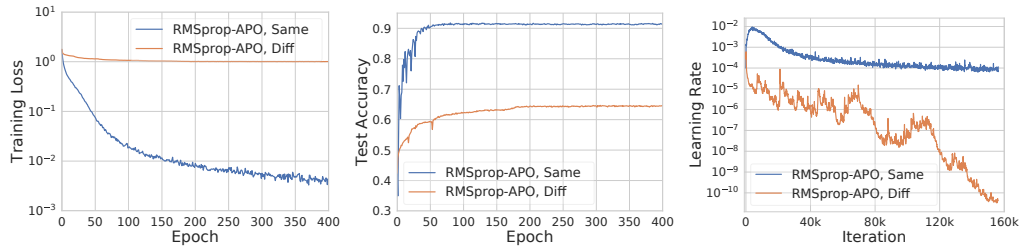


Figure 34: **Left, Center:** Training loss/Test accuracy or RMSprop-APO compared between 1) using the same mini-batch for the first term and 2) using a different mini-batch for the first term. **Right:** Comparison of learning rates for the two conditions.

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. 15.

$$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \frac{\lambda_{\text{FSD}}}{|\mathcal{B}|} \sum_{(\tilde{\mathbf{x}}, \cdot) \in \mathcal{B}} D_{\text{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}}) + \frac{\lambda_{\text{WSD}}}{2} ||u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) - \boldsymbol{\theta}||_2^2 \quad (15)$$

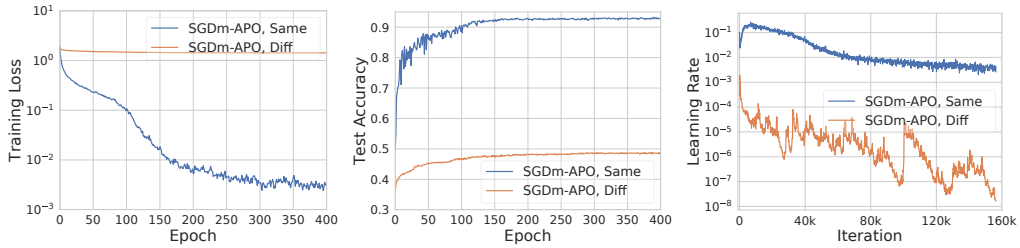


Figure 35: **Left, Center:** Training loss/Test accuracy of SGDm-APO compared between 1) using the same mini-batch for the first term and 2) using a different mini-batch for the first term. **Right:** Comparison of learning rates for the two conditions.

Recall that $D_{\mathrm{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}}) = \rho(f(\tilde{\mathbf{x}}, u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})), f(\tilde{\mathbf{x}}, \boldsymbol{\theta}))$. 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. 15 as a Monte Carlo estimate of the expectation $\mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[D_{\mathrm{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}), \boldsymbol{\theta}, \tilde{\mathbf{x}})]$, 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.

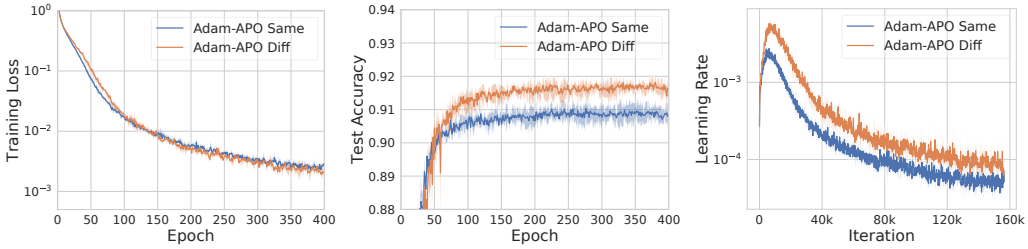


Figure 36: **Left, Center:** Training loss/Test accuracy of Adam-APO compared between 1) using the same mini-batch for the second term as for the first term of the meta-objective and 2) using a different mini-batch for the second term. **Right:** Comparison of learning rates for the two conditions.

F Approximate Closed-Form Solution for the PPM

In Section 3.1, we introduced a general update rule for the stochastic PPM, which is defined as follows:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\mathbf{u} \in \mathbb{R}^m}{\arg\min}\, \mathcal{J}_{\mathcal{B}}(\mathbf{u}) + \lambda_{\mathrm{FSD}} \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[D_{\mathrm{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \mathbf{x})] + \lambda_{\mathrm{WSD}} D_{\mathrm{W}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}). \tag{16}$$

We first take the infinitesimal limit by letting $\lambda_{\mathrm{FSD}} \to \infty$. 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 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:

$$\mathbb{E}_{\tilde{\mathbf{x}} \sim \mathcal{D}}[D_{\mathrm{F}}(f(\tilde{\mathbf{x}}, \mathbf{u}), f(\tilde{\mathbf{x}}, \boldsymbol{\theta}^{(t)}))] \approx \frac{1}{2}(\mathbf{u} - \boldsymbol{\theta}^{(t)})^\top \mathbf{G} (\mathbf{u} - \boldsymbol{\theta}^{(t)}), \tag{17}$$

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. 16 by linearizing the loss and taking a quadratic approximation to the FSD term:

$$\boldsymbol{\theta}^{(t+1)} \approx \underset{\mathbf{u} \in \mathbb{R}^m}{\arg\min} \left[\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^\top (\mathbf{u} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\mathrm{FSD}}}{2} (\mathbf{u} - \boldsymbol{\theta}^{(t)})^\top \mathbf{G} (\mathbf{u} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\mathrm{WSD}}}{2} \|\mathbf{u} - \boldsymbol{\theta}^{(t)}\|^2 \right]. \tag{18}$$

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

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \lambda_{\text{FSD}} \mathbf{G}\mathbf{u} - \lambda_{\text{FSD}} \mathbf{G}\boldsymbol{\theta}^{(t)} + \lambda_{\text{WSD}} \mathbf{u} - \lambda_{\text{WSD}} \boldsymbol{\theta}^{(t)} = \mathbf{0}. \tag{19}$$

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

$$\mathbf{u}^\star \approx \boldsymbol{\theta}^{(t)} - (\lambda_{\text{FSD}} \mathbf{G} + \lambda_{\text{WSD}} \mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}). \tag{20}$$

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 3.2. Given a mini-batch $\mathcal{B}$, recall that we consider the following proximal objective for optimizing the model parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^m} \left\{ \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) + \lambda_{\text{FSD}} \mathbb{E}_{\tilde{\mathbf{x}}}[D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \right\}. \tag{21}$$

Method	**Loss Term Approx.** $\mathcal{J}_{\mathcal{B}}(\mathbf{u})$	**FSD Term Approx.** $D_{\text{F}}(\mathbf{u}, \boldsymbol{\theta}, \mathbf{x})$	**FSD Term Choice of** D_{F}	**WSD Term** $\frac{1}{2}||\mathbf{u} - \boldsymbol{\theta}||^2$
Gradient Descent	1^{st}-order	-	-	✓
Newton's Method	2^{nd}-order	-	-	✗
Damped Newton's Method	2^{nd}-order	-	-	✓
Natural Gradient	1^{st}-order	2^{nd}-order	KL	✗
Damped Natural Gradient	1^{st}-order	2^{nd}-order	KL	✓
Generalized Gauss-Newton	1^{st}-order	2^{nd}-order	Bregman	✗
Damped Generalized Gauss-Newton	1^{st}-order	2^{nd}-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 $P^{\text{GD}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta})$:

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

where:

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

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

$$\nabla_{\boldsymbol{\theta}} \left(\nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \right) = 0 \tag{24}$$

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

$$\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)} - \frac{1}{\lambda_{\text{WSD}}} \nabla \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{26}$$

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 $P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\boldsymbol{\theta}} P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}), \tag{27}$$

where:

$$P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^\top\mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2. \tag{28}$$

In Eq. 28, $\mathbf{H} = \nabla^2_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$. Now, we set the gradient $\nabla_{\boldsymbol{\theta}} P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = 0$ and solve for $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} P^{\text{Newton}}_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = 0 \tag{29}$$

$$\nabla\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \mathbf{H}\boldsymbol{\theta} - \mathbf{H}\boldsymbol{\theta}^{(t)} + \lambda_{\text{WSD}}\boldsymbol{\theta} - \lambda_{\text{WSD}}\boldsymbol{\theta}^{(t)} = 0 \tag{30}$$

$$(\mathbf{H} + \lambda_{\text{WSD}}\mathbf{I})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) = -\nabla\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{31}$$

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

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:

$$P_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \lambda_{\text{FSD}}\mathbb{E}_{\tilde{\mathbf{x}}}[D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}}))] + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \tag{33}$$

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

$$\begin{aligned} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \approx \underbrace{D_{\text{F}}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta}^{(t)})}_{=0} + \underbrace{\nabla_{\boldsymbol{\theta}} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|^\top_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})}_{=0} \\ + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^\top\nabla^2_{\boldsymbol{\theta}} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}). \end{aligned} \tag{34}$$

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

$$\nabla^2_{\boldsymbol{\theta}} D_{\text{F}}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} = \underbrace{\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}}\right)^\top}_{\mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}}} \underbrace{\left(\frac{\partial^2 \rho}{\partial \mathbf{z}^2}\right)}_{\mathbf{H}_\rho} \underbrace{\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}}\right)}_{\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}}} = \mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}}\nabla^2_{\mathbf{z}}\rho(\mathbf{z}, \mathbf{z}^{(t)})\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}} = \mathbf{J}^\top_{\mathbf{z}\boldsymbol{\theta}}\mathbf{H}_\rho\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}} \tag{35}$$

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

$$P_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\theta}) = \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) + \nabla\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{FSD}}}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^\top\mathbf{G}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}||_2^2 \tag{36}$$

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

$$\boldsymbol{\theta} = \boldsymbol{\theta} - (\lambda_{\text{FSD}}\mathbf{G} + \lambda_{\text{WSD}}\mathbf{I})^{-1}\nabla\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \tag{37}$$

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:

$$u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}) = u(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) = \boldsymbol{\theta} - \mathbf{P}\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \mathbf{P}\mathbf{g} \tag{38}$$

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. 8) where the loss term is linearized around the current weights $\boldsymbol{\theta}$ and the FSD term is replaced by its second-order approximation around $\boldsymbol{\theta}$:*

$$\hat{\mathcal{Q}}(\mathbf{P}) = \mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\Big[\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})^{\top}(u(\boldsymbol{\theta},\mathbf{P},\mathcal{B})-\boldsymbol{\theta}) + \lambda_{FSD}(u(\boldsymbol{\theta},\mathbf{P},\mathcal{B})-\boldsymbol{\theta})^{\top}\mathbf{G}(u(\boldsymbol{\theta},\mathbf{P},\mathcal{B})-\boldsymbol{\theta}) + \frac{\lambda_{WSD}}{2}\|u(\boldsymbol{\theta},\mathbf{P},\mathcal{B})-\boldsymbol{\theta}\|^2\Big]. \quad (39)$$

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):

$$\mathbf{P}^{\star} = \arg\min_{\mathbf{P}}\Big\{\mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\Big[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) + \underbrace{\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})^{\top}}_{\mathbf{g}}((\boldsymbol{\theta}-\mathbf{P}\mathbf{g})-\boldsymbol{\theta}) \quad (40)$$

$$+ \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}\left[D_{\text{F}}(u(\boldsymbol{\theta},\mathbf{P},\mathcal{B}),\boldsymbol{\theta},\tilde{\mathbf{x}})\right] + \frac{\lambda_{\text{WSD}}}{2}||\boldsymbol{\theta}-\mathbf{P}\mathbf{g}-\boldsymbol{\theta}||^2\Big]\Big\} \quad (41)$$

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

$$d(\boldsymbol{\theta}') \approx \underbrace{d(\boldsymbol{\theta})}_{=0} + \underbrace{\nabla d(\boldsymbol{\theta})^{\top}(\boldsymbol{\theta}'-\boldsymbol{\theta})}_{=0} + \frac{1}{2}(\boldsymbol{\theta}'-\boldsymbol{\theta})^{\top}\nabla^2_{\boldsymbol{\theta}}d(\boldsymbol{\theta})(\boldsymbol{\theta}'-\boldsymbol{\theta}) \quad (42)$$

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:

$$\nabla^2_{\boldsymbol{\theta}}d(\boldsymbol{\theta}) = \underbrace{\left(\frac{\partial\mathbf{z}}{\partial\boldsymbol{\theta}}\right)^{\top}}_{\mathbf{J}^{\top}_{\mathbf{z}\boldsymbol{\theta}}}\underbrace{\left(\frac{\partial^2\rho}{\partial\mathbf{z}^2}\right)}_{\mathbf{H}_{\rho}}\underbrace{\left(\frac{\partial\mathbf{z}}{\partial\boldsymbol{\theta}}\right)}_{\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}}} = \mathbf{J}^{\top}_{\mathbf{z}\boldsymbol{\theta}}\mathbf{H}_{\rho}\mathbf{J}_{\mathbf{z}\boldsymbol{\theta}} \quad (43)$$

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:

$$\mathbf{P}^{\star} = \arg\min_{\mathbf{P}}\left\{\mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^{\top}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{FSD}}}{2}\mathbf{g}^{\top}\mathbf{P}^{\top}\mathbf{G}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2}\mathbf{g}^{\top}\mathbf{P}^{\top}\mathbf{P}\mathbf{g}\right]\right\} \quad (44)$$

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

$$\nabla_{\mathbf{P}}\left(\mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\left[\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}) - \mathbf{g}^{\top}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{FSD}}}{2}\mathbf{g}^{\top}\mathbf{P}^{\top}\mathbf{G}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2}\mathbf{g}^{\top}\mathbf{P}^{\top}\mathbf{P}\mathbf{g}\right]\right) = 0 \quad (45)$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^{\top} + \frac{\lambda_{\text{FSD}}}{2}(\mathbf{G}\mathbf{P}\mathbf{g}\mathbf{g}^{\top} + \mathbf{G}^{\top}\mathbf{P}\mathbf{g}\mathbf{g}^{\top}) + \lambda_{\text{WSD}}\mathbf{P}\mathbf{g}\mathbf{g}^{\top}\right] = 0 \quad (46)$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^{\top} + \lambda_{\text{FSD}}\mathbf{G}\mathbf{P}\mathbf{g}\mathbf{g}^{\top} + \lambda_{\text{WSD}}\mathbf{P}\mathbf{g}\mathbf{g}^{\top}\right] = 0 \quad (47)$$

$$(\lambda_{\text{FSD}}\mathbf{F}\mathbf{P} + \lambda_{\text{WSD}}\mathbf{P} - \mathbf{I})\mathbb{E}\left[\mathbf{g}\mathbf{g}^{\top}\right] = 0 \quad (48)$$

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

$$\lambda_{\text{FSD}}\mathbf{G}\mathbf{P} + \lambda_{\text{WSD}}\mathbf{P} - \mathbf{I} = 0 \quad (49)$$

$$(\lambda_{\text{FSD}}\mathbf{G} + \lambda_{\text{WSD}}\mathbf{I})\mathbf{P} = \mathbf{I} \quad (50)$$

$$\mathbf{P} = (\lambda_{\text{FSD}}\mathbf{G} + \lambda_{\text{WSD}}\mathbf{I})^{-1} \quad (51)$$

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:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(\boldsymbol{\theta}) + \nabla \mathcal{J}_\mathcal{B}(\boldsymbol{\theta})^\top ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \mathbf{P}\mathbf{g} - \boldsymbol{\theta}||^2 \right] \right\} \tag{52}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} ||\mathbf{P}\mathbf{g}||^2 \right] \right\} \tag{53}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(f(\boldsymbol{\theta}, \mathbf{x}), \mathbf{t}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right\} \tag{54}$$

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

$$\nabla_\mathbf{P} \left(\mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right) = 0 \tag{55}$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^\top + \lambda_{\text{WSD}} \mathbf{P}\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{56}$$

$$\left(\mathbf{P} - \frac{1}{\lambda_{\text{WSD}}} \mathbf{I} \right) \mathbb{E}\left[\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{57}$$

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

$$\mathbf{P} = \frac{1}{\lambda_{\text{WSD}}} \mathbf{I} \tag{58}$$

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:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \Big\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \Big[\mathcal{J}_\mathcal{B}(\boldsymbol{\theta}) + \underbrace{\nabla_{\boldsymbol{\theta}} \mathcal{J}_\mathcal{B}(\boldsymbol{\theta})^\top}_{\mathbf{g}} ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) \tag{59}$$

$$+ \frac{1}{2} ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta})^\top \mathbf{H} ((\boldsymbol{\theta} - \mathbf{P}\mathbf{g}) - \boldsymbol{\theta}) + \frac{\lambda_{\text{WSD}}}{2} ||\boldsymbol{\theta} - \mathbf{P}\mathbf{g} - \boldsymbol{\theta}||^2 \Big] \Big\} \tag{60}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{1}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{H}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} ||\mathbf{P}\mathbf{g}||^2 \right] \right\} \tag{61}$$

$$= \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{1}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{H}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right\} \tag{62}$$

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

$$\nabla_\mathbf{P} \left(\mathbb{E}_{\mathcal{B}\sim\mathcal{D}} \left[\mathcal{J}_\mathcal{B}(\boldsymbol{\theta}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{1}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{H}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right) = 0 \tag{63}$$

$$\mathbb{E}\left[-\mathbf{g}\mathbf{g}^\top + \frac{1}{2} \mathbf{H}\mathbf{P}\mathbf{g}\mathbf{g}^\top + \frac{1}{2} \mathbf{H}^\top \mathbf{P}\mathbf{g}\mathbf{g}^\top + \lambda_{\text{WSD}} \mathbf{P}\mathbf{g}\mathbf{g}^\top \right] = 0 \tag{64}$$

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

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

$$\mathbf{H}\mathbf{P} + \lambda_{\text{WSD}} \mathbf{P} - \mathbf{I} = 0 \tag{66}$$

$$(\mathbf{H} + \lambda_{\text{WSD}} \mathbf{I}) \mathbf{P} = \mathbf{I} \tag{67}$$

$$\mathbf{P} = (\mathbf{H} + \lambda_{\text{WSD}} \mathbf{I})^{-1} \tag{68}$$

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:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \left\{ \mathbb{E}_{(\mathbf{x},\mathbf{t})\sim\mathcal{D}} \left[\mathcal{L}(f(\boldsymbol{\theta},\mathbf{x}),\mathbf{t}) - \mathbf{g}^\top \mathbf{P}\mathbf{g} + \frac{\lambda_{\text{FSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{F}\mathbf{P}\mathbf{g} + \frac{\lambda_{\text{WSD}}}{2} \mathbf{g}^\top \mathbf{P}^\top \mathbf{P}\mathbf{g} \right] \right\} \tag{69}$$

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

$$(\lambda_{\text{FSD}} \mathbf{F}\mathbf{P} + \lambda_{\text{WSD}} \mathbf{P} - \mathbf{I}) \mathbb{E}\left[\mathbf{g}\mathbf{g}^\top\right] = 0 \tag{70}$$

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:

$$\mathbf{P} = (\lambda_{\text{FSD}} \mathbf{F} + \lambda_{\text{WSD}} \mathbf{I})^{-1} \tag{71}$$

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 ℓ^{th} block (corresponding to the ℓ^{th} 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 [25]. 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:

$$\mathbf{s}_\ell = \mathbf{W}_\ell \mathbf{a}_{\ell-1} \tag{72}$$

$$\mathbf{a}_\ell = \sigma(\mathbf{s}_\ell) \tag{73}$$

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

$$\mathcal{D}\mathbf{a}_\ell = \mathbf{W}^\top \mathcal{D}\mathbf{s}_{\ell+1} \tag{74}$$

$$\mathcal{D}\mathbf{s}_\ell = \mathcal{D}\mathbf{a}_\ell \odot \sigma'(\mathbf{s}_\ell) \tag{75}$$

$$\mathcal{D}\mathbf{W}_\ell = \mathcal{D}\mathbf{s}_\ell \mathbf{a}_{\ell-1}^\top \tag{76}$$

KFAC [54] 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 ℓ^{th} block of the Fisher is:

  $$\hat{\mathbf{G}}_\ell = \mathbf{A}_\ell^{\text{KFAC}} \otimes \mathbf{B}_\ell^{\text{KFAC}} \tag{77}$$

  where:

  $$\mathbf{A}_\ell^{\text{KFAC}} = \mathbb{E}[\mathbf{a}_{\ell-1}\mathbf{a}_{\ell-1}^\top] \tag{78}$$

  $$\mathbf{B}_\ell^{\text{KFAC}} = \mathbb{E}[\mathcal{D}\mathbf{s}_\ell \mathcal{D}\mathbf{s}_\ell^\top] \tag{79}$$

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. 9.*

Proof. If the KFAC assumptions are satisfied, then $\mathbf{F}^{-1}$ is block-diagonal with the block corresponding to the ℓ^{th} 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. 9). 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).

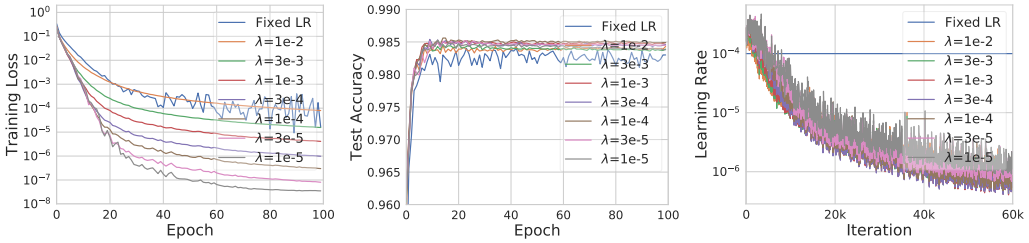


Figure 37: Ablation over λ_{FSD} for training a two-layer MLP with 1000 hidden units per layer on MNIST, using RMSprop-APO.

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.

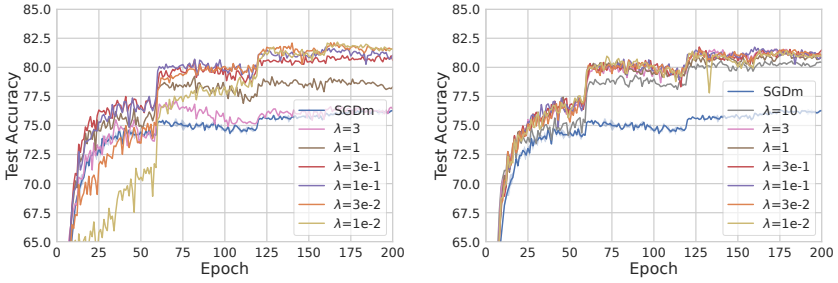
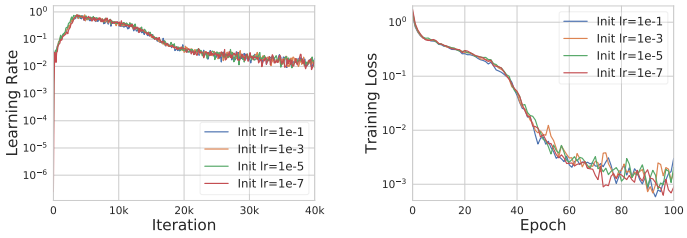


Figure 38: Ablation over (left) λ_{WSD} and (right) λ_{FSD} for training AlexNet on CIFAR-10, using APO-Precond.

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.

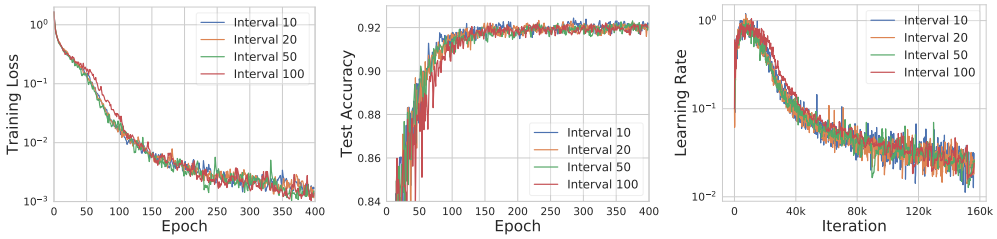


Figure 40: Robustness to the meta-update interval. Here, we trained ResNet32 on CIFAR-10 using SGDm-APO, with meta-updates performed once every $\{10, 20, 50, 100\}$ steps. We observe that APO performs similarly using intervals from 10 to 100, corresponding to computation times approximately $1.3\times$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

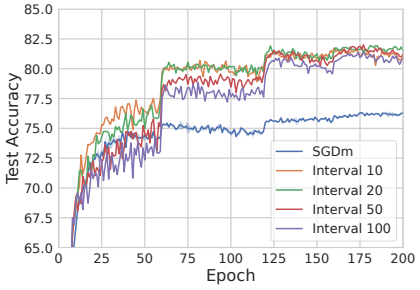


Figure 41: Robustness to the meta-update interval on AlexNet. CIFAR-10, APO-Precond.