Amortized Proximal Optimization:
Meta-Learning a Parametric Update Rule via a Proximal Meta-Objective

Juhan Bae, Paul Vicol*, Jeff Z. Haochen, Roger Grosse*

Slides by Paul Vicol

Outline

- Background: Proximal Point Method
  - Different types of discrepancy measures (*weight-space* and *function-space*)
  - Connections Between Proximal Optimization and 2nd-Order Optimization
- Amortized Proximal Optimization (APO)
  - Algorithm
  - Computation & Memory Cost
- APO for Structured Preconditioner Adaptation
- APO for Learning Rate Adaptation
- Experimental Results

The Proximal Point Method

- Many algorithms in machine learning can be interpreted as approximations to an idealized algorithm called the *proximal point method (PPM)*

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \arg\min_{\mathbf{u} \in \mathbb{R}^m} \underbrace{\mathcal{J}_{\mathcal{B}^{(t)}}(\mathbf{u})}_{\text{Minimize the loss on the current minibatch}} + \underbrace{\lambda D(\mathbf{u}, \boldsymbol{\theta}^{(t)})}_{\text{While staying close to the current iterate}}$$

The Proximal Point Method

- Many algorithms in machine learning can be interpreted as approximations to an idealized algorithm called the *proximal point method (PPM)*

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\mathbf{u}\in\mathbb{R}^m}{\arg\min}\ \underbrace{\mathcal{J}_{\mathcal{B}^{(t)}}(\mathbf{u})}_{\text{Minimize the loss on the current minibatch}} + \underbrace{\lambda D(\mathbf{u}, \boldsymbol{\theta}^{(t)})}_{\text{While staying close to the current iterate}}$$

- How do we measure proximity to the previous iterate, $D(\mathbf{u}, \boldsymbol{\theta}^{(t)})$?
- Several classic algorithms can be interpreted as *balancing between two types of proximity*:

Function-Space Discrepancy (FSD)

$$D_{\mathrm{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}) = \mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}\left[\rho(f(\tilde{\mathbf{x}}; \mathbf{u}), f(\tilde{\mathbf{x}}; \boldsymbol{\theta}^{(t)}))\right]$$

Weight-Space Discrepancy (WSD)

$$D_{\mathrm{W}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}) = 1/2\|\mathbf{u} - \boldsymbol{\theta}^{(t)}\|_2^2$$

The Proximal Point Method

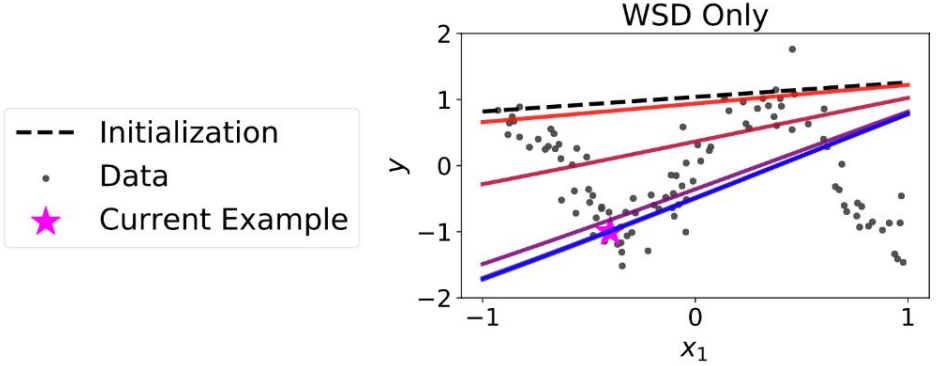
- To gain intuition, let's see what happens when we minimize the proximal objective *exactly*.

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

The Proximal Point Method

- To gain intuition, let's see what happens when we minimize the proximal objective *exactly*.

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

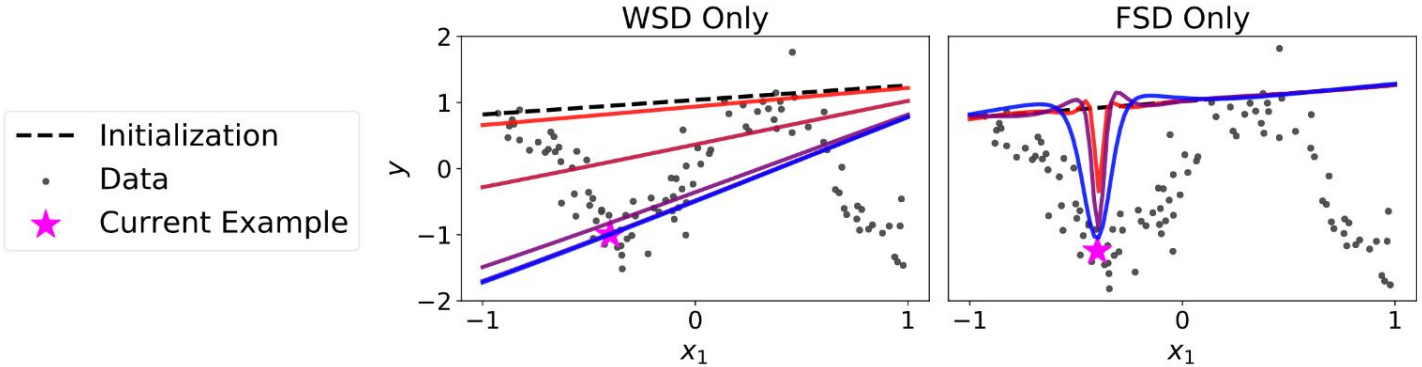


Makes a *small change to the parameters but a global change to the outputs*.
Fits the current example but also changes the predictions on other data.

The Proximal Point Method

- To gain intuition, let's see what happens when we minimize the proximal objective *exactly*.

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



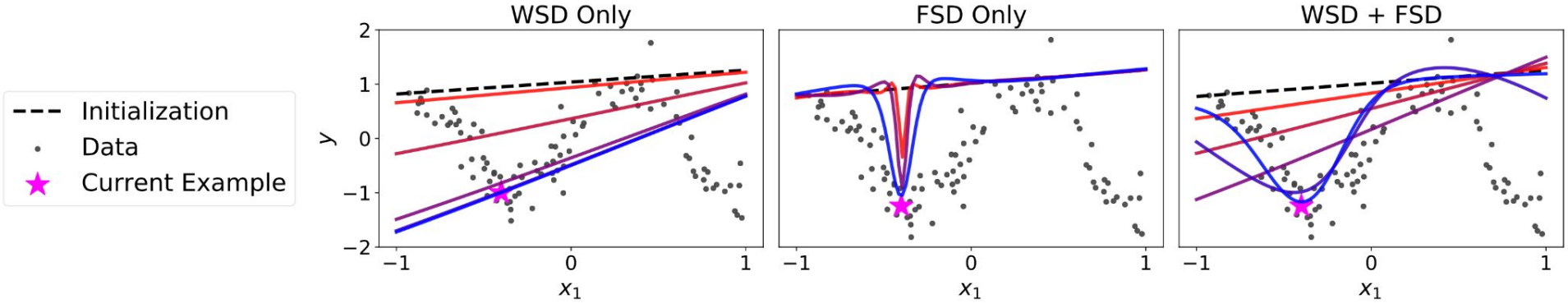
Makes a *small change to the parameters but a global change to the outputs*. Fits the current example but also changes the predictions on other data.

Makes a *sharp change to the outputs* to fit the current example while maintaining the previous outputs on other data.

The Proximal Point Method

- To gain intuition, let's see what happens when we minimize the proximal objective *exactly*.

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



Makes a *small change to the parameters but a global change to the outputs*. Fits the current example but also changes the predictions on other data.

Makes a *sharp change to the outputs* to fit the current example while maintaining the previous outputs on other data.

Including both FSD and WSD terms yields a *smoother update to the function*.

Connections to Second-Order Optimization

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

- Minimizing the proximal objective exactly in each iteration is *uneconomical*
- Various first- and second-order optimization algorithms can be interpreted as *minimizing approximations of the proximal objective*
  - Taking 1st or 2nd order Taylor expansions of the loss or FSD terms allows for *closed-form solutions*

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

Amortized Proximal Optimization (APO)

- Consider an *update rule* u parameterized by a vector of *optimization parameters* $\boldsymbol{\phi}$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow u(\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}, \mathcal{B}^{(t)})$$

- **Examples:**

$$u_{\mathrm{SGD}}(\boldsymbol{\theta}, \eta, \mathcal{B}) = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$$

$$u_{\mathrm{Precond}}(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) = \boldsymbol{\theta} - \mathbf{P} \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$$

Amortized Proximal Optimization (APO)

- Consider an *update rule* u parameterized by a vector of *optimization parameters* $\boldsymbol{\phi}$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow u(\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}, \mathcal{B}^{(t)})$$

- **Examples:**

$$u_{\mathrm{SGD}}(\boldsymbol{\theta}, \eta, \mathcal{B}) = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$$

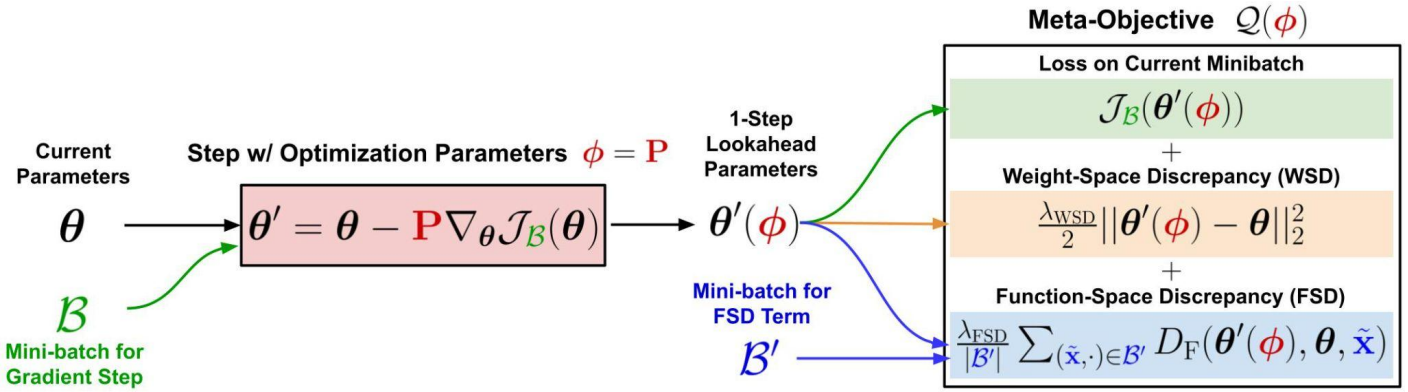
$$u_{\mathrm{Precond}}(\boldsymbol{\theta}, \mathbf{P}, \mathcal{B}) = \boldsymbol{\theta} - \mathbf{P} \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$$

- APO tunes the optimization parameters phi to *minimize a proximal meta-objective* $\mathcal{Q}(\boldsymbol{\phi})$:

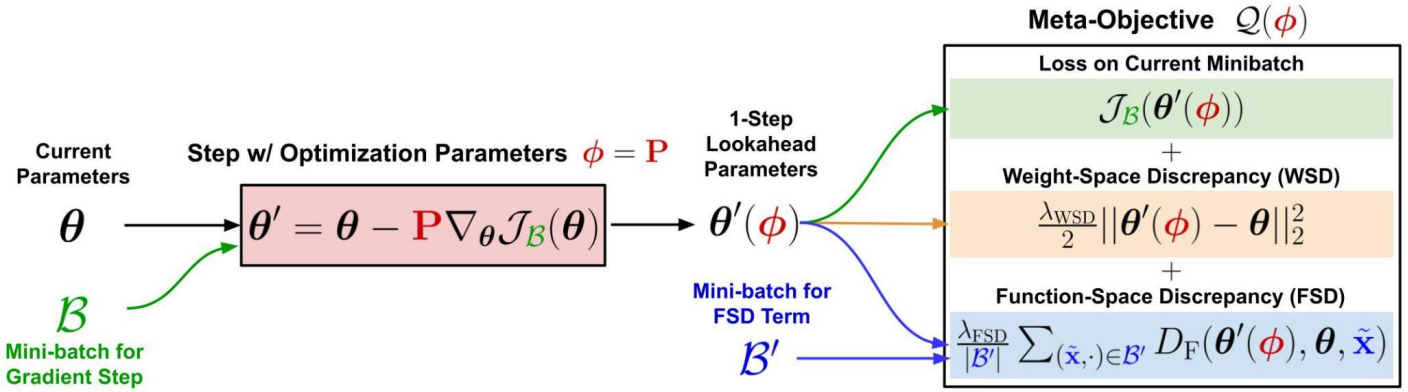
$$\mathcal{Q}(\boldsymbol{\phi}) = \mathbb{E}_{\mathcal{B}\sim\mathcal{D}}\left[\mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B})) + \lambda_{\mathrm{FSD}} \mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}\left[D_{\mathrm{F}}(u(\boldsymbol{\theta}, \boldsymbol{\phi},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

- By adapting a parametric update rule, we can *amortize the cost of minimizing the proximal objective over the course of training*

APO Algorithm



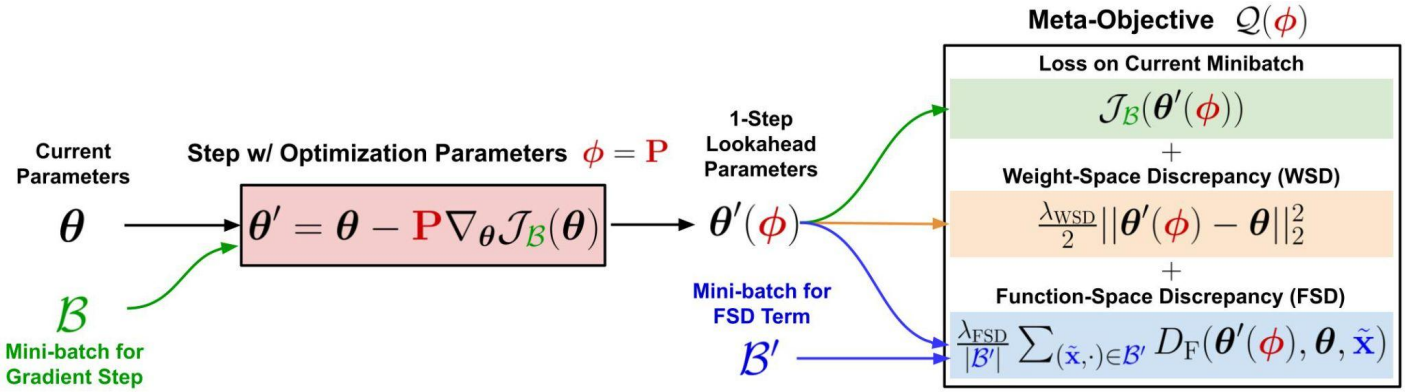
APO Algorithm



Algorithm 1 Amortized Proximal Optimization (APO) — Meta-Learning the Optimization Parameters ϕ

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
    $\boldsymbol{\theta}'(\phi) := u(\boldsymbol{\theta}, \phi, \mathcal{B})$ ▷ Compute the 1-step lookahead parameters
    $\mathcal{Q}(\phi) := \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}'(\phi)) + \lambda_{\text{FSD}}/|\mathcal{B}'| \sum_{(\tilde{\mathbf{x}},\cdot)\in\mathcal{B}'} D_{\text{F}}(\boldsymbol{\theta}'(\phi), \boldsymbol{\theta}, \tilde{\mathbf{x}}) + \lambda_{\text{WSD}}/2 \, \|\boldsymbol{\theta}'(\phi) - \boldsymbol{\theta}\|_2^2$ ▷ Meta-objective
    $\phi \leftarrow \phi - \alpha\nabla_{\phi}\mathcal{Q}(\phi)$ ▷ Update optimizer parameters (e.g. LR or preconditioner)
  end if
  $\boldsymbol{\theta} \leftarrow u(\boldsymbol{\theta}, \phi, \mathcal{B})$ ▷ Update model parameters
end while

Computation and Memory Cost of APO



- **<u>Computation:</u>** Computing the meta-gradient requires 3 forward passes + a backward pass through the 1-step unrolled computation graph.
  - But we only perform a meta-update *once every K iterations*

- **<u>Memory:</u>** APO requires 2x the model memory for the 1-step unroll

APO for Adaptive Preconditioning

- APO can be used to learn the *preconditioning matrix for second-order optimization*
- How should we *parameterize* the preconditioner?

Two desiderata:

1. We want the preconditioner to be *positive semi-definite*, to maintain descent directions
2. We want the preconditioner to be tractable to store and compute with

APO for Adaptive Preconditioning

- APO can be used to learn the *preconditioning matrix for second-order optimization*
- How should we *parameterize* the preconditioner?

Two desiderata:

1. We want the preconditioner to be *positive semi-definite*, to maintain descent directions
2. We want the preconditioner to be tractable to store and compute with

Parameterization	**Positive Semi-Definite**	**Tractable to Store in Memory**
P	✖	✖

APO for Adaptive Preconditioning

- APO can be used to learn the *preconditioning matrix for second-order optimization*
- How should we *parameterize* the preconditioner?

Two desiderata:

1. We want the preconditioner to be *positive semi-definite*, to maintain descent directions
2. We want the preconditioner to be tractable to store and compute with

Parameterization	**Positive Semi-Definite**	**Tractable to Store in Memory**
$\mathbf{P}$	✖	✖
$\mathbf{M}\mathbf{M}^\top$	✓	✖

APO for Adaptive Preconditioning

- APO can be used to learn the *preconditioning matrix for second-order optimization*
- How should we *parameterize* the preconditioner?

Two desiderata:

1. We want the preconditioner to be *positive semi-definite*, to maintain descent directions
2. We want the preconditioner to be tractable to store and compute with

Parameterization	**Positive Semi-Definite**	**Tractable to Store in Memory**
$\mathbf{P}$	✖	✖
$\mathbf{M}\mathbf{M}^\top$	✓	✖
$\mathbf{P}_\mathrm{S} = (\mathbf{A} \otimes \mathbf{B})\mathrm{diag}(\mathrm{vec}(\mathbf{S}))^2(\mathbf{A} \otimes \mathbf{B})^\top$	✓	✓

APO for Adaptive Preconditioning

- In practice, we use the EKFAC-structured preconditioner:

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

- EKFAC uses *complicated covariance estimation and matrix inversion* to construct the block matrices
- In contrast, APO *directly meta-learns the matrices*

- APO does not require inverting (or performing eigendecompositions of) the block matrices
- → Our structured representation incurs *less computation per iteration than EKFAC.*

APO Can Recover Classic 2nd-Order Methods

- If we make the same assumptions used to derive natural gradient, then the optimal preconditioner P that *minimizes the one-step lookahead meta-objective corresponds to the damped natural gradient update*:

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

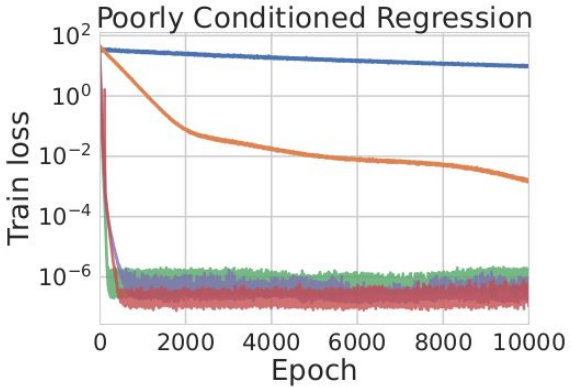
Theorem 1. *Consider an approximation $\hat{\mathcal{Q}}(\mathbf{P})$ to the meta-objective (Eq. 8) where the loss term is linearized around the current weights $\boldsymbol{\theta}$ and the FSD term is replaced by its second-order approximation around $\boldsymbol{\theta}$. Denote the gradient on a 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 $\boldsymbol{\theta}$.*

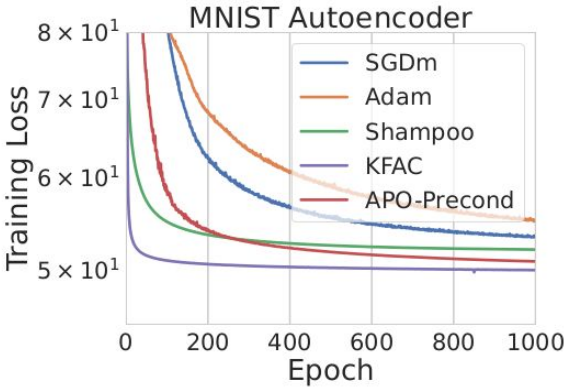
APO Can Recover the KFAC Update

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 H. Then, the optimal solution to the approximate meta-objective recovers the KFAC update, which can be represented using the structured preconditioner in Eq. 9.*

Experiments: Poorly-Conditioned Problems

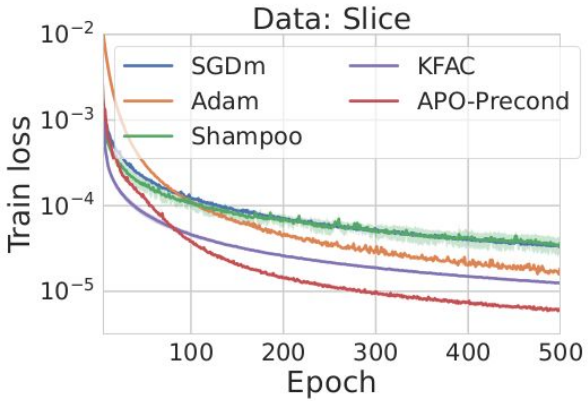
- Using APO to tune the preconditioner works well on *poorly-conditioned tasks where second-order optimizers are typically required*

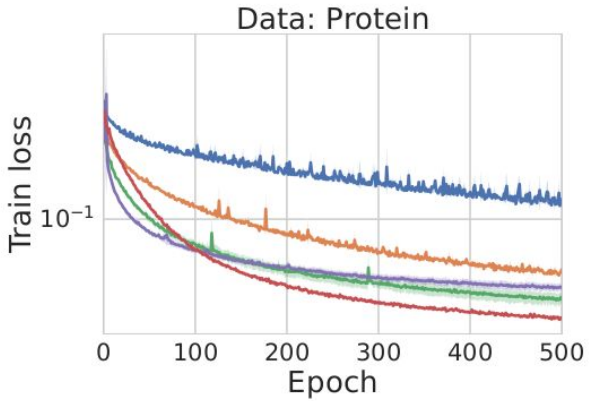


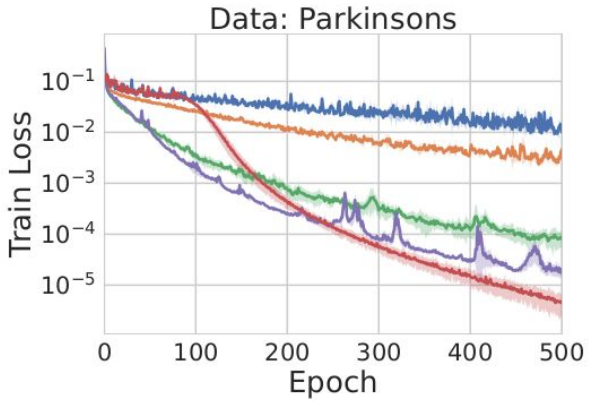


Experiments: UCI Regression Tasks

- APO-Precond *outperforms 1st and 2nd-order baselines (including Shampoo and KFAC)* on UCI regression tasks
  - Typically, APO-Precond *converges more quickly and reaches lower training loss*

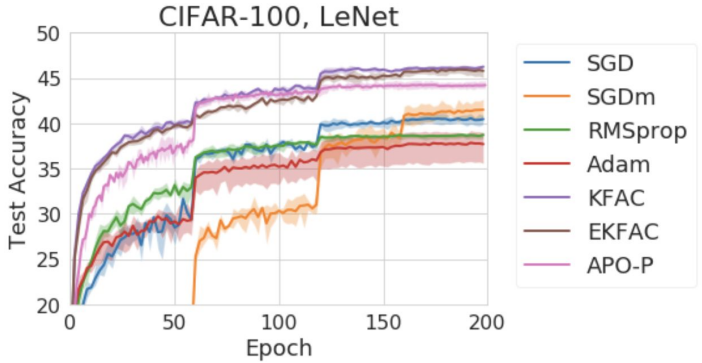


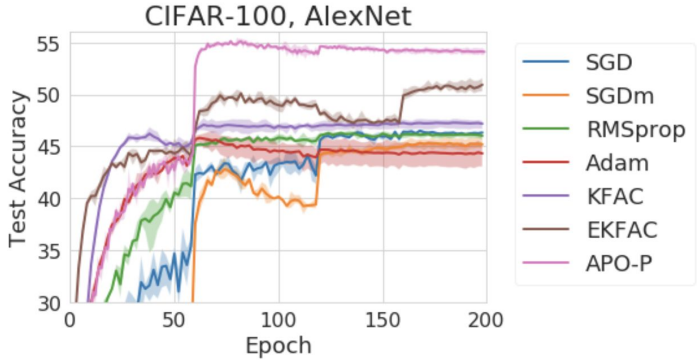


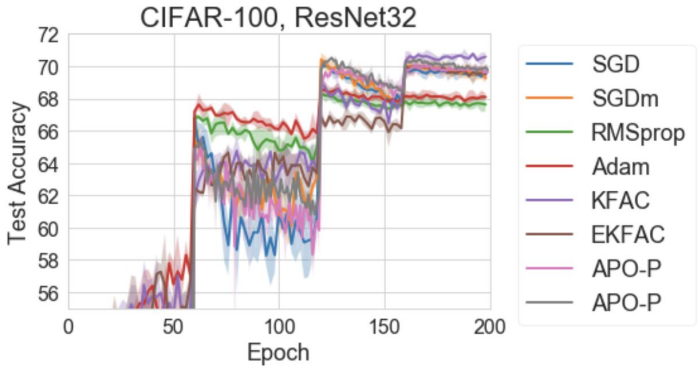


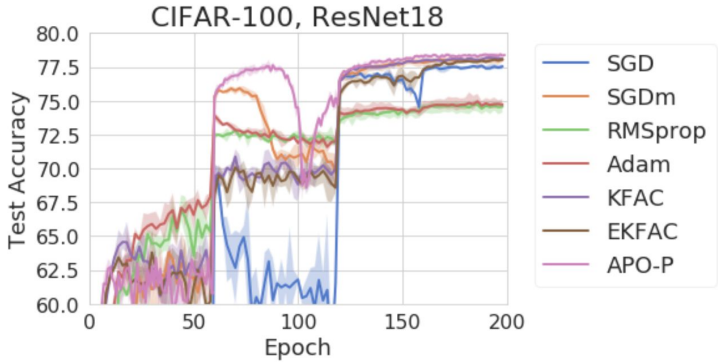
Experiments: Architectures on CIFAR-10/100

- We evaluated the *generalization of a variety of APO-trained models on CIFAR-10/100*
  - Tuned LR, weight decay, etc. separately for each task.









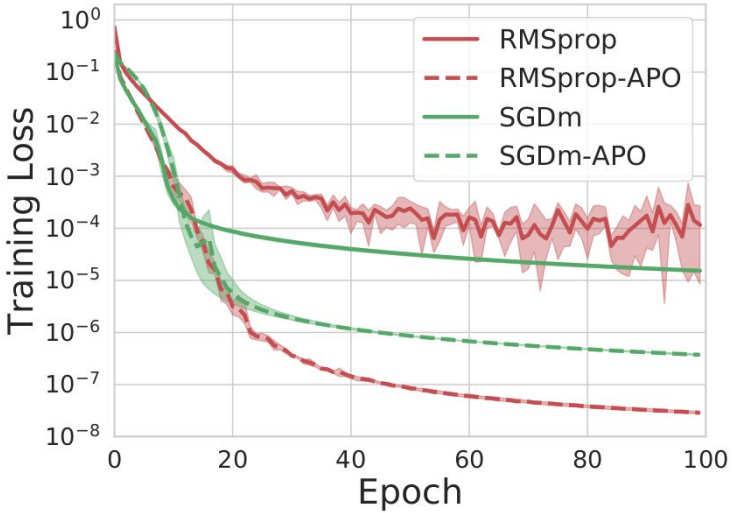
Low-Precision (16-bit) Training

- Low-precision training presents a challenge for second-order optimizers such as KFAC
  - These rely on *matrix inverses that may be sensitive to quantization noise.*
- APO *does not require inversion, and remains stable*

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**

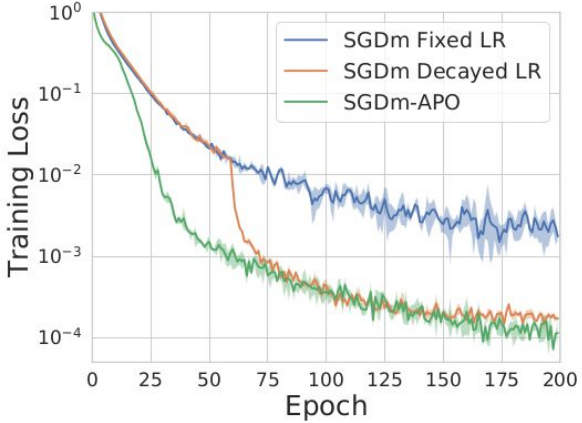
LR Adaptation: MNIST

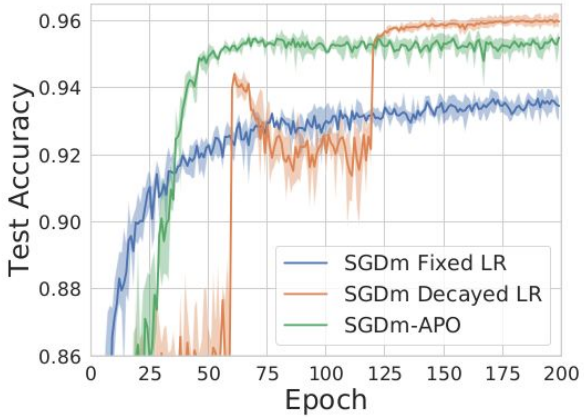
- APO can be used to tune the global learning rate for a variety of *base optimizers*
  - Including *SGD, SGDm, RMSprop, Adam*
- The lambda hyperparameters for APO need to be searched over, but *each lambda setting yields a learning rate schedule*
- Here, we tuned the LR for training an MLP on MNIST, with SGDm and RMSprop

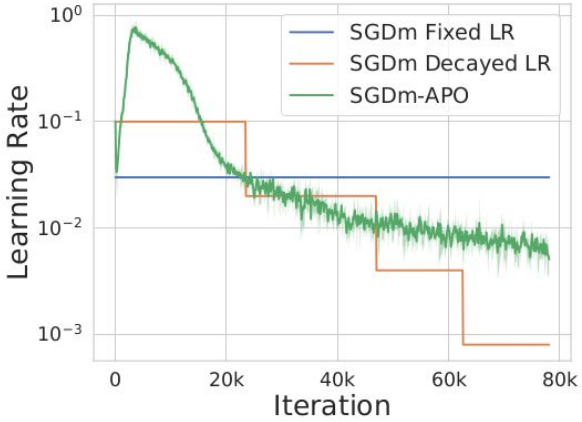


LR Adaptation: MNIST

- The APO-adapted LR schedules usually *outperform optimal fixed LRs, and are competitive with manual step decay schedules*

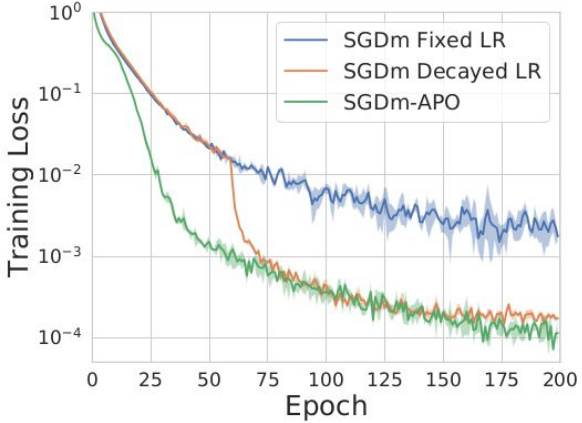


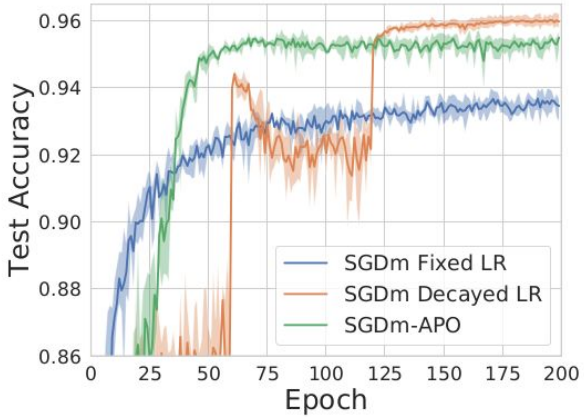


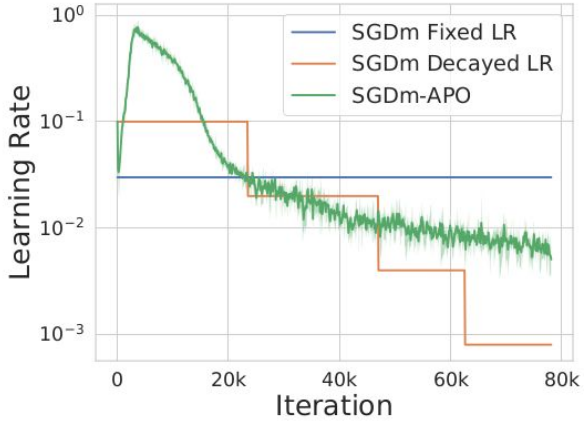


LR Adaptation: MNIST

- The APO-adapted LR schedules usually *outperform optimal fixed LRs, and are competitive with manual step decay schedules*



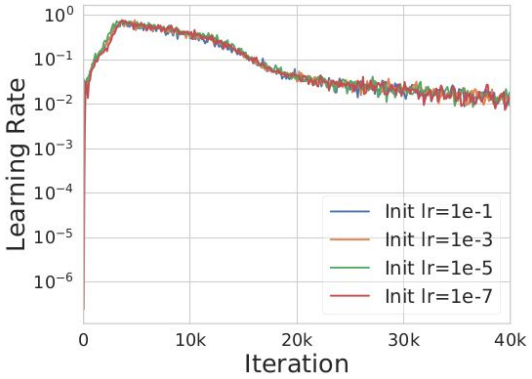


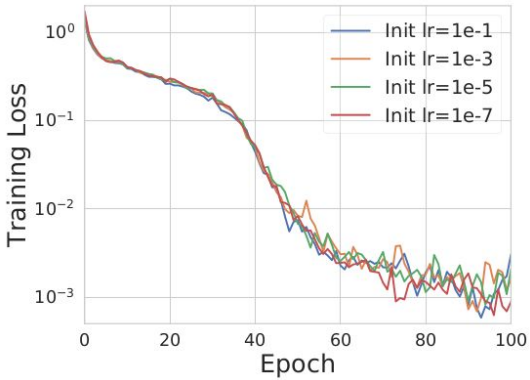


	CIFAR-10 ResNet-32			CIFAR-10 ResNet-34			CIFAR-10 WRN 28-10			CIFAR-100 WRN 28-10		
	Fixed	**Decay**	**APO**	**Fixed**	**Decay**	**APO**	**Fixed**	**Decay**	**APO**	**Fixed**	**Decay**	**APO**
SGD	90.07	93.30	92.71	93.00	93.54	94.27	93.38	94.86	94.85	76.29	77.92	76.87
SGDm	89.40	93.34	92.75	92.99	95.08	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

LR Adaptation: Robustness to the Initial LR

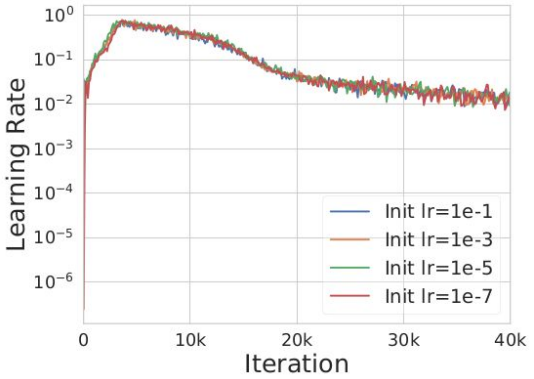
Robustness to the Initial Learning Rate — Across 6 Orders of Magnitude

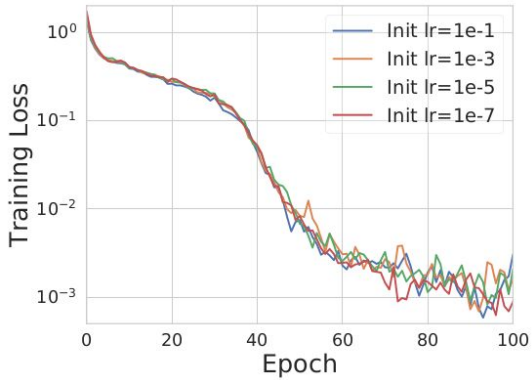




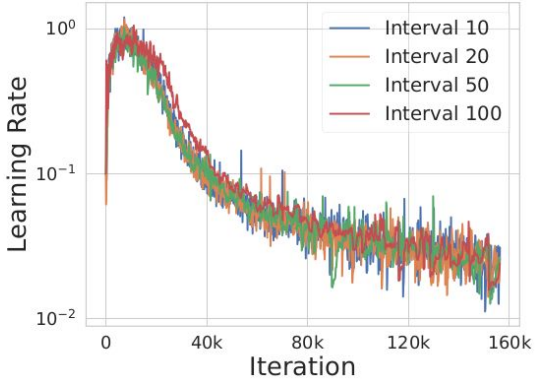
LR Adaptation: Robustness to the Initial LR

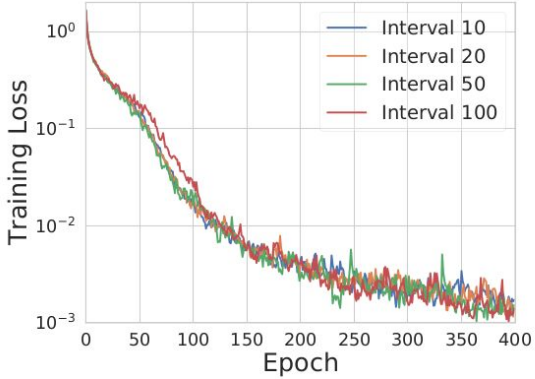
Robustness to the Initial Learning Rate — Across 6 Orders of Magnitude





Robustness to the Meta-Update Frequency





On Short-Horizon Bias

- *APO is a greedy method*—it only considers the effect of the optimization parameters through a one-step lookahead → **Does APO suffer from short-horizon bias?**
- Original short-horizon bias meta-objective: $\mathbb{E}_{\mathcal{B}\sim\mathcal{D}_{\text{train}}}[\mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{B}))]$

On Short-Horizon Bias

- *APO is a greedy method*—it only considers the effect of the optimization parameters through a one-step lookahead → **Does APO suffer from short-horizon bias?**
- Original short-horizon bias meta-objective: $\mathbb{E}_{\mathcal{B}\sim\mathcal{D}_{\text{train}}}[\mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}))]$

1 $$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B})) + \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}),\boldsymbol{\theta},\tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2}||u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}) - \boldsymbol{\theta}||_2^2$$

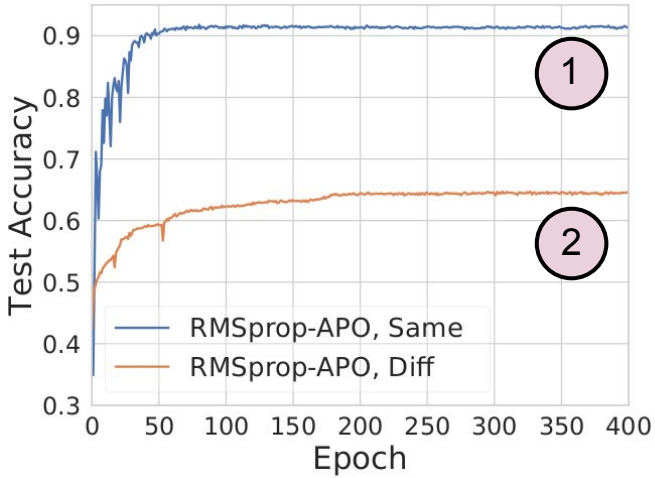
2 $$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}'}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B})) + \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}),\boldsymbol{\theta},\tilde{\mathbf{x}})]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

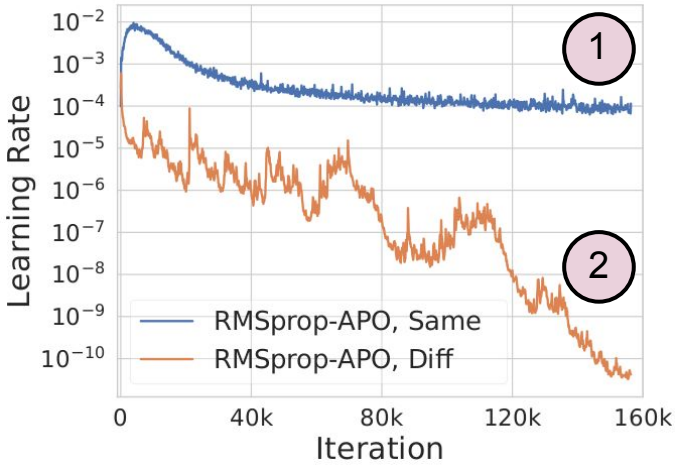
On Short-Horizon Bias

- *APO is a greedy method*—it only considers the effect of the optimization parameters through a one-step lookahead → **Does APO suffer from short-horizon bias?**
- Original short-horizon bias meta-objective: $\mathbb{E}_{\mathcal{B}\sim\mathcal{D}_{\text{train}}}[\mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}))]$

1 $$\mathcal{Q}(\boldsymbol{\phi}) = \mathcal{J}_{\mathcal{B}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B})) + \lambda_{\text{FSD}}\mathbb{E}_{(\tilde{\mathbf{x}},\cdot)\sim\mathcal{D}}[D_{\text{F}}(u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}),\boldsymbol{\theta},\tilde{\mathbf{x}})] + \frac{\lambda_{\text{WSD}}}{2}||u(\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{B}) - \boldsymbol{\theta}||_2^2$$

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





Thank you!