Amortized Proximal Optimization

*Juhan Bae[1,2], *Paul Vicol[1,2], Jeff Z. HaoChen[3], Roger Grosse[1,2]
* denotes equal contribution, [1]University of Toronto, [2]Vector Institute, [3]Stanford University

VECTOR INSTITUTE

UNIVERSITY OF TORONTO

Proximal Point Method

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

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

Here, $D_{\text{F}}(\mathbf{u}, \boldsymbol{\theta}^{(t)}, \tilde{\mathbf{x}}) = \rho(f(\tilde{\mathbf{x}}; \mathbf{u}), f(\tilde{\mathbf{x}}; \boldsymbol{\theta}^{(t)}))$, where ρ is an output-space discrepancy function.

Connections to Second-Order Optimization

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

- Minimizing the proximal objective exactly is uneconomical.
- Various first- and second-order optimization algorithms can be interpreted as minimizing approximations of the proximal objective, using 1st or 2nd order Taylor expansions of the loss or FSD terms.
- When taking a 1st-order approximation to the loss and a 2nd-order approximation to the FSD, the update rule is given in closed form as:

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

where $\mathbf{G}$ is the Hessian of the FSD term.

Amortized Proximal Optimization (APO)

- Consider an update rule u parameterized by a vector ϕ which updates the network weights $\boldsymbol{\theta}$ on a mini-batch $\mathcal{B}^{(t)}$:

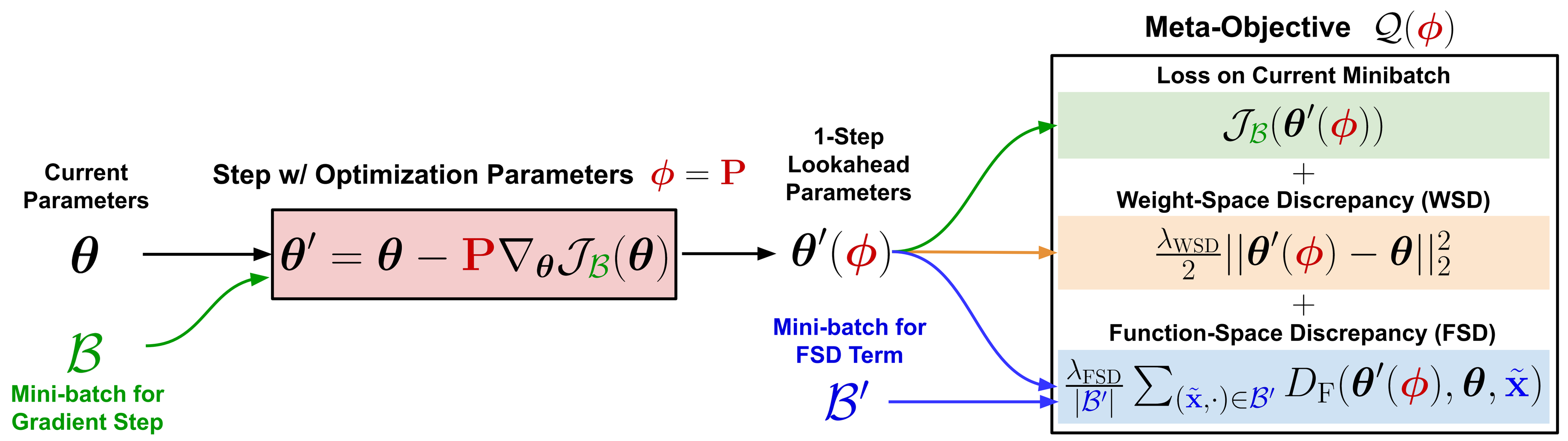
$$\boldsymbol{\theta}^{(t+1)} \leftarrow u(\boldsymbol{\theta}^{(t)}, \phi, \mathcal{B}^{(t)})$$

- We propose to directly minimize a proximal meta-objective with respect to the optimization parameters ϕ:

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

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

APO Algorithm



- In each meta-optimization step, we perform a one-step lookahead to obtain updated parameters $\boldsymbol{\theta}'(\phi)$, where ϕ denotes optimization parameters (e.g. the LR η or preconditioner $\mathbf{P}$). Then ϕ is updated via the meta-gradient $\nabla_{\phi}\mathcal{Q}(\phi)$.

---

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
    $\mathcal{Q}(\phi) := \mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta}'(\phi)) + \frac{\lambda_{\text{FSD}}}{|\mathcal{B}'|}\sum_{(\tilde{\mathbf{x}},\cdot)\in\mathcal{B}'} D_{\text{F}}(\boldsymbol{\theta}'(\phi), \boldsymbol{\theta}, \tilde{\mathbf{x}}) + \frac{\lambda_{\text{WSD}}}{2}\|\boldsymbol{\theta}'(\phi) - \boldsymbol{\theta}\|_2^2$
    $\phi \leftarrow \phi - \alpha\nabla_{\phi}\mathcal{Q}(\phi)$ ▷ Update optimizer parameters
  end if
  $\boldsymbol{\theta} \leftarrow u(\boldsymbol{\theta}, \phi, \mathcal{B})$ ▷ Update model parameters
end while

- **Compute Cost:** Computing $\nabla_{\phi}\mathcal{Q}(\phi)$ requires 3 forward passes + a backward pass through the 1-step unrolled computation graph. We perform a meta-update once every K iterations.
- **Memory Cost:** APO requires $2\times$ the model memory for the 1-step unroll.

APO for Learning Rate Adaptation

- One use case of APO is to tune hyperparameters of an existing optimizer: when tuning the LR for SGD, we have $\phi = \eta$ and $u_{\text{SGD}}(\boldsymbol{\theta}, \eta, \mathcal{B}) = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{J}_{\mathcal{B}}(\boldsymbol{\theta})$

APO for Structured Preconditioner Adaptation

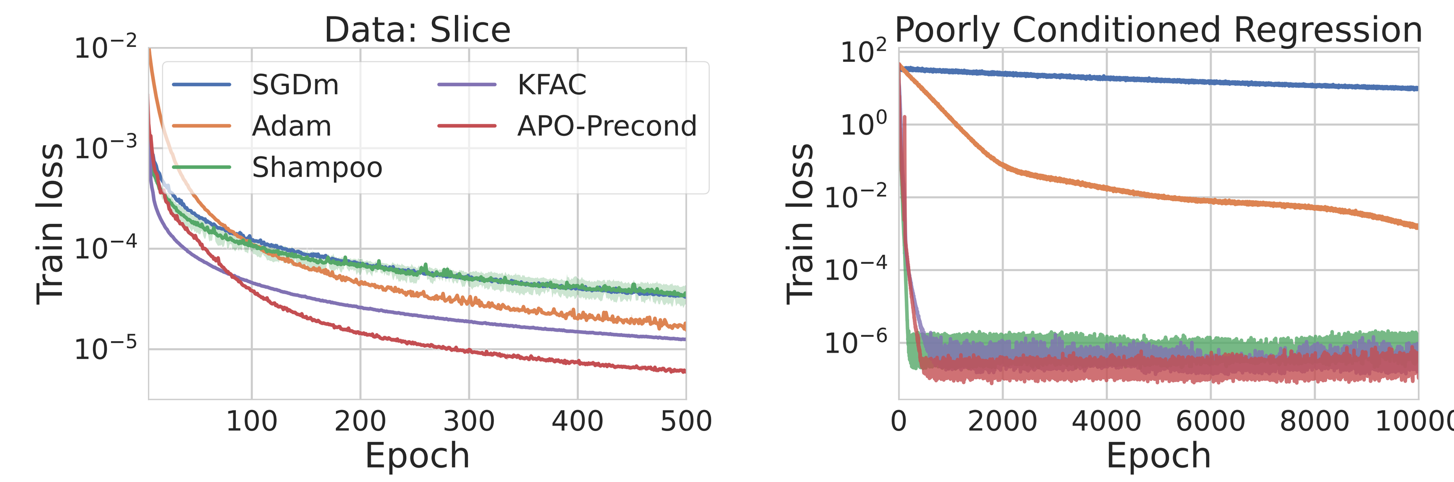
- APO can adapt the preconditioning matrix $\mathbf{P}$, allowing the update rule to flexibly represent various second-order updates.
- Under appropriate assumptions, the optimal $\mathbf{P}$ that minimizes $\mathcal{Q}(\mathbf{P})$ is equivalent to different 2nd-order updates, depending on the choice of FSD.
- To scale to large neural nets, we use the EKFAC structured parameterization, which also ensures that $\mathbf{P}$ is PSD. For the weight matrix $\mathbf{W}$ of a layer, we represent the preconditioning matrix as the product of smaller matrices:

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

- While EKFAC uses complicated covariance estimation and eigenvalue decomposition to construct the block matrices, in APO, we meta-learn these block matrices directly.

Preconditioner Tuning Experiments

Regression Tasks



Classification Tasks

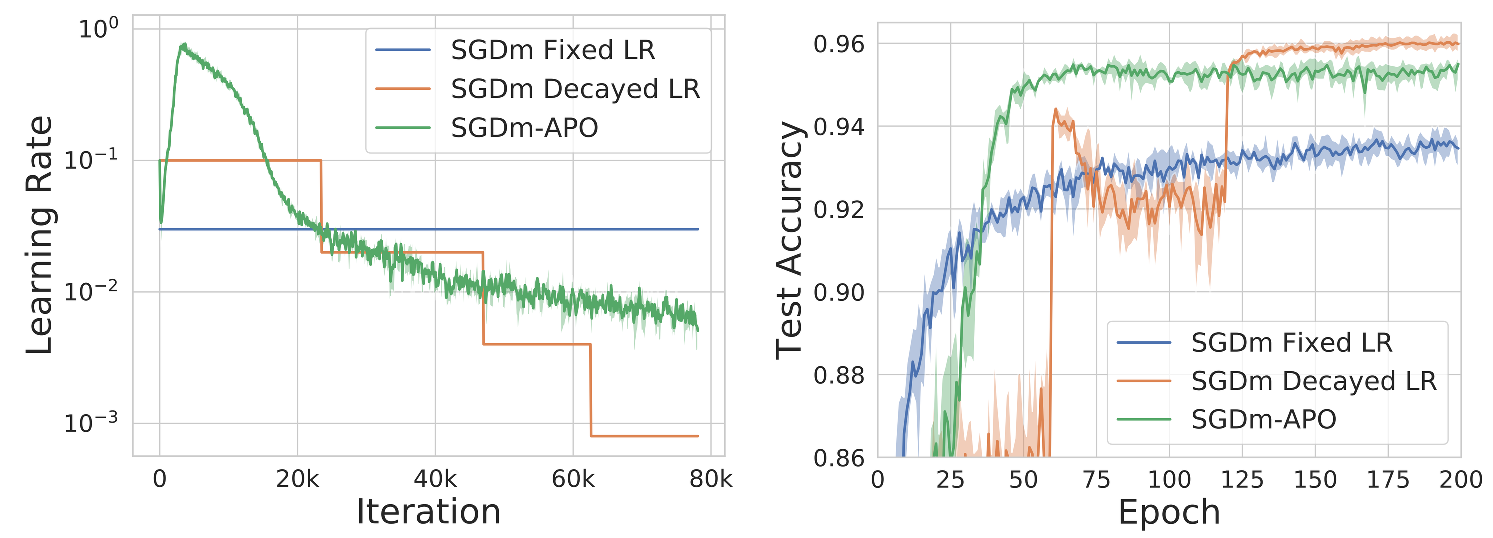
Task	Model	SGDm	Adam	KFAC	APO-P
C-10	**LeNet**	75.73	73.41	76.63	**77.42**
C-10	**AlexNet**	76.27	76.09	78.33	**81.14**
C-10	**VGG16**	91.82	90.19	92.05	**92.13**
C-10	**ResNet-18**	93.69	93.27	94.60	**94.75**
C-100	**AlexNet**	43.95	41.82	46.24	**52.35**
C-100	**VGG16**	65.98	60.61	61.84	**67.95**
C-100	**ResNet-18**	76.85	70.87	76.48	**76.88**
IWSLT14	**Transformer**	31.43	34.60	-	**34.62**

Low Precision (16-bit) Training

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**

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

Learning Rate Adaptation



- Test accuracy and learning rate adaptation for WRN-28-10 on CIFAR-10, using SGDm as the base optimizer.
- APO outperforms the best fixed LR, and is competitive with the step schedule.