MetricOpt: Learning to Optimize Black-Box Evaluation Metrics Supplementary Material

Chen Huang Shuangfei Zhai Pengsheng Guo Josh Susskind
Apple Inc.
{chen-huang,szhai,pengsheng_guo,jsusskind}@apple.com

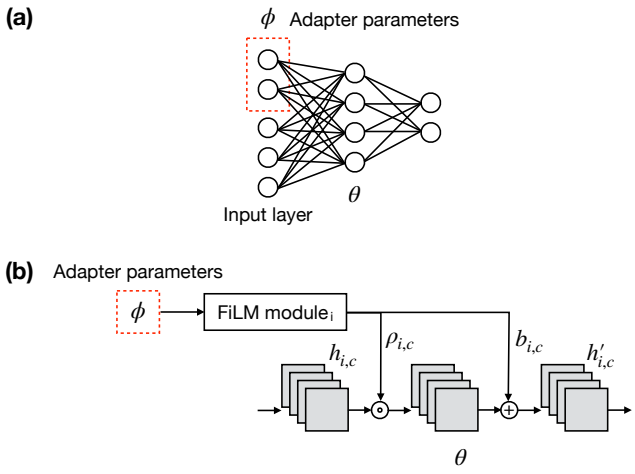

Figure 1. Adapter parameters ϕ that modulate the pre-trained network weights θ. (a) For fully connected networks, ϕ are the dynamic biases concatenated at the input layer. (b) For convolutional networks, ϕ are the input vectors of FiLM layers [7] that linearly transform each feature map $h_{i,c}$ with scale $\rho_{i,c}$ and shift $b_{i,c}$ parameters.

1. Adapter Module Visualization

For the parameter efficiency of model finetuing, we follow [10] to use an adapter module to modulate the pre-trained network weights θ. Specifically, we finetune a small set of parameters ϕ of an adapter module. Fig. 1 illustrates the adapter modules for both fully connected and convolutional networks. We found the two types of adapter parameters achieve a good performance-efficiency tradeoff, and use them to learn our value function throughout the paper.

2. MetricOpt with Jointly Learned Optimizer

In the main paper, we introduce a well-performing MetricOpt that combines our learned value function with hand-crafted optimizers SGD/Adam. Here we show it is also possible to jointly learn the value function *and* the optimizer for better performance. Training signals come from a given surrogate loss and the differentiable value function learned on the fly (as a metric supervision). The main product of such joint training is a neural network parameterized optimizer. During meta-testing, only the learned optimizer is used for a new finetuning task, while the learned value function is dropped. Algorithm 1 shows the pseudocode of a fully learned MetricOpt.

Algorithm 1 Pseudocode of fully learned MetricOpt

Input: Task distribution $p(\mathcal{T})$, hyper-parameters α, η
Input: Number of finetuning steps T per task
Input: Number of metric evaluations K per task
// Meta-training value function & optimizer
Initialize weights of optimizer w_{opt} and value func f_{w_v}
for $\mathcal{T} \sim p(\mathcal{T})$ **do**
  Collect sparse evaluation metrics $\{M(\phi_k)\}_{k=1}^{K}$ over the tuning sequence $\{\phi_t\}_{t=1}^{T}$ of learned optimizer
  Obtain dense interpolation $\{(\hat{M}_t, \hat{\sigma}_t)\}_{t=1}^{T}$
  Update $\tilde{w}_v = w_v - \alpha \nabla_{w_v} L_v(w_v)$ *// Value function*
  Update w_{opt} using adapted $f_{\tilde{w}_v}$ by Eq. (3) *// Optimizer*
  Update $w_v \leftarrow w_v + \eta(\tilde{w}_v - w_v)$ *// Reptile update*
end for
return w_{opt}
// Meta-testing with the learned optimizer
Initialize ϕ_0 for new task $\mathcal{T}$
for $t = 0$ to $T - 1$ **do**
  Update $\phi_{t+1} \leftarrow \phi_t - \alpha_t u_t$ following Eq. (1)
end for
return ϕ_T

The current go-to method for optimizer parameterization is to leverage an LSTM network [1, 9]. Unfortunately, LSTM training suffers from either biased or exploding gradients during truncated backpropagation through unrolled optimization [6]. Here we use a simple MLP with weights w_{opt} as in [6]. The MLP-based optimizer $m_{w_{opt}}$ can produce the update direction u_t and learning rate α_t to recur-

rently update our adapter parameters ϕ_t:

$$\begin{aligned} \phi_{t+1} &= \phi_t + \alpha_t u_t, \\ [\alpha_t, u_t] &= m_{w_{opt}}(\nabla_t, \bar{\nabla}_t, \phi_t, \ell_t, \Delta\ell_t, \mathcal{M}_t, \Delta\mathcal{M}_t), \end{aligned} \tag{1}$$

where optimizer inputs are the loss gradient $\nabla_t = \nabla_\phi \ell(\phi_t)$, exponential running average $\bar{\nabla}_t$, current parameters ϕ_t, scale-normalized loss ℓ_t and metric $\mathcal{M}_t$ and their relative changes $\Delta\ell_t$ and $\Delta\mathcal{M}_t$ from the respective moving averages.

As mentioned above, we train w_{opt} with supervisions of both metric L_{metric} and loss L_{loss}:

$$\begin{aligned} L_{opt}(w_{opt}) &= \lambda L_{metric}(w_{opt}) + L_{loss}(w_{opt}), \\ L_{metric} &= \frac{1}{T}\sum_{t=1}^{T} \log\left(1 + \exp\left(\frac{\beta(\hat{\mathcal{M}}_t - \hat{\mathcal{M}}_{t'})}{\hat{\mathcal{M}}_t}\right)\right), \\ \hat{\mathcal{M}}_t &= f_{w_v}(\phi_t),\ t' = \arg\min_{i<t} \hat{\mathcal{M}}_i, \\ L_{loss} &= \frac{1}{T}\sum_{t=1}^{T} \left(\log(\ell(\phi_t) + \epsilon) - \log(\ell(\phi_0) + \epsilon)\right), \end{aligned} \tag{2}$$

where hyper-parameters $\lambda = 50$ and $\beta = T/2$, and the small positive constant ϵ is introduced to avoid numerical instability. Note $\hat{\mathcal{M}}_t$ is the metric prediction from our value function f_{w_v}. The L_{metric} term encourages relative metric improvements for the whole optimization sequence in a logarithmic form. On the other hand, the L_{loss} term evaluates the average log loss (offset by the initial value). This term encourages a low loss at convergence while still providing loss training signals at every step.

One benefit of the multi-task objective L_{opt} is that it continues to penalize metric deterioration when minimizing the loss at convergence. However, the objective surface of L_{opt} can be extremely non-smooth [6], and we are likely to obtain noisy derivatives through the unrolled optimization process. Here, we follow [6] to train our optimizer $m_{w_{opt}}$ using a variational loss as a smoothed objective:

$$\mathbb{E}_{\tilde{w}_{opt} \sim \mathcal{N}(w_{opt}, \varepsilon^2 I)}\ L_{opt}(\tilde{w}_{opt}), \tag{3}$$

where $\varepsilon^2 = 0.01$ is a fixed variance. This smoothing helps stabilize optimizer training. Actual training is based on the two unbiased gradient estimators in [6], with the same learning rate setting.

Computational complexity Our main paper confirms the empirical advantages of jointly learning the value function and optimizer. The resulting **MetricOpt (learned)** outperforms our default **MetricOpt (SGD)** approach when optimizing the image classification metrics MCR and AUCPR on CIFAR-10 dataset. But it is worth noting that such performance gains come at a larger meta-training cost. Specifically, joint training needs more meta-iterations to converge than meta-training the value function alone in our default approach (∼5k vs. <2k, with often more than 3× longer training time). The joint training time is dominated by that of optimizer learning (with costly unrolled derivatives). On the side of value function learning, the corresponding cost is much smaller. During meta-testing, the optimization cost will not affected by value function at all since only the learned optimizer is used.

Table 1. Top-1 and Top-5 classification accuracies (%) on ImageNet. All methods use the same NASNet-A network.

Method	Top-1	Top-5
RMSProp + cross-entropy	73.5	91.5
MetricOpt (SGD)	74.9	92.7
MetricOpt (learned)	**75.0**	**93.0**

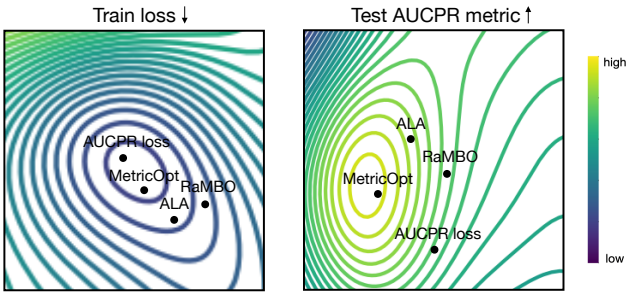

Figure 2. Train loss and test metric surfaces (visualized by the toolbox in [5]) for optimizing the AUCPR metric on CIFAR-10. We compare the solutions obtained from the hand-designed AUCPR loss [3], Adaptive Loss Alignment (ALA) [4], RaMBO (with gradient interpolation) [8] and our MetricOpt.

More results We further benchmark the fully learned MetricOpt on the large-scale ImageNet dataset. Table 1 validates the advantage of MetricOpt (learned) over MetricOpt (SGD) again. The comparison also confirms the efficacy of our value function for gradient-based metric optimization.

3. Understanding Differences among Recent Methods for Metric Optimization

In the field of non-differentiable metric optimization, there are recent strong methods that have been compared against in the main paper. Here we provide more analyses and visualizations to explain the advantages of our MetricOpt method. Recall MetricOpt learns a differentiable value function that generates reliable gradient estimates of metric to augment loss gradients. How does this impact the training dynamics on the optimization landscape? Fig. 2 gives some hints by visualizing the typical optimization surfaces of train loss and test metric, as well as the converged solutions proposed by different methods. The compared methods are of three types: loss function [3] designed to approximate the target metric, adaptively learned loss ALA [4], and RaMBO (with gradient interpolation) [8].

We observe from the figure that:

- loss and metric surfaces are different, verifying the need for some form of metric supervision during optimization.
- All compared methods achieve low training loss values, but their testing metrics have notable differences. This confirms the observation in [2] that low-loss solutions form a connected manifold, and further adds that they tend to be distinct in the metric space due to different ways of metric approximation.
- Specifically, both human-designed and learned losses (*i.e.*, AUCPR and ALA losses) need a relaxed surrogate space to approximate metrics. Obviously, performance will highly depend on the quality of such surrogate relaxations. For RaMBO, gradient interpolation is conducted via black-box differentiation which can be critically affected by the sparsity of supervision signals and interpolation details. Our MetricOpt sidesteps these challenges by a direct function approximation of target metric (with value function). The resulting metric supervision can adjust the optimization trajectory towards better metric, leading to a solution off trajectories purely based on surrogate losses. Fig. 2 shows that MetricOpt is able to converge to the best performance metric while still maintaining a low loss.

4. More Ablation Studies

Fig. 3 ablates the different training aspects of our value function. We observe that:

- Learning with evaluation metrics from a subset of training data with the same size of validation set (default) barely hurts performance. This suggests our advantages mainly come from direct metric optimization, not just from learning validation statistics.
- By default, we collect $K = 5\%T$ evaluation metrics from each finetuning task with T iterations and then interpolate sparse metrics temporally for value function learning. We found a smaller K (*e.g.*, $K = 2.5\%T$) hinders effective value function learning (hurts final performance too). On the other hand, larger Ks slightly improve performance, but lead to increased cost for dense metric evaluation. Note when $K = 100\%T$, we collect evaluation metrics from all iterations, without using any metric interpolation. This proves as unnecessary since the resulting gains are pretty marginal.
- Regarding the value function input, we choose to use the compact adapter parameters ϕ by default (with size 128 in the case here). Other options exist, including using a small portion of the main network θ like the biases of last layer. Results indicate last layer biases are not enough to model metrics well, while using multi-layer biases becomes more parameter-inefficient with even worse results. In our early experiments, we failed to learn value function from the entire last layer for the same reason.

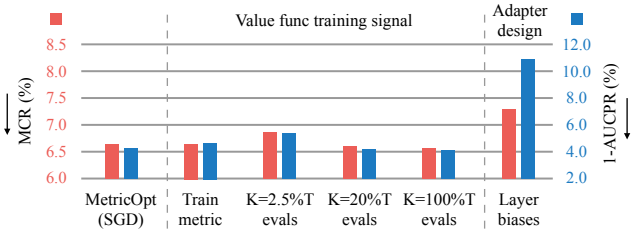

Figure 3. Ablation studies of value function for optimizing Miss-Classification Rate (MCR) and Area Under the Precision Recall Curve (AUCPR) on CIFAR-10 dataset. The AUCPR metric undergoes a $1 - x$ conversion (thus lower is better). In terms of both metrics, we compare different training signals and input parameters for the value function.

But when learning with ϕ, performance is reasonably robust to ϕ's size which does not need careful tuning.

5. More Object Detection Results

The main paper (Table 6) shows notable improvements over the Faster R-CNN detector by our direct metric optimization during finetuning. But what contributes to the gains, and how does finetuning impact the different modules of Faster R-CNN (*i.e.*, region proposal network and detection network)? Fig. 4 sheds some light on these questions. Through finetuning for the AP metric, we observe the region proposal network seems to have an improved recall on the proposed bounding boxes. Intuitively, such improved object proposals should benefit the following detection module. This inspires an interesting future work, which is to investigate different impacts on a multi-staged pipeline when optimizing for different metrics.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

[2] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML*, 2018.

[3] Elad Eban, Mariano Schain, Alan Mackey, Ariel Gordon, Ryan Rifkin, and Gal Elidan. Scalable learning of non-decomposable objectives. In *AISTATS*, 2017.

[4] Chen Huang, Shuangfei Zhai, Walter Talbott, Miguel Ángel Bautista, Shih-Yu Sun, Carlos Guestrin, and Joshua M. Susskind. Addressing the loss-metric mismatch with adaptive loss alignment. In *ICML*, 2019.

[5] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

[6] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and

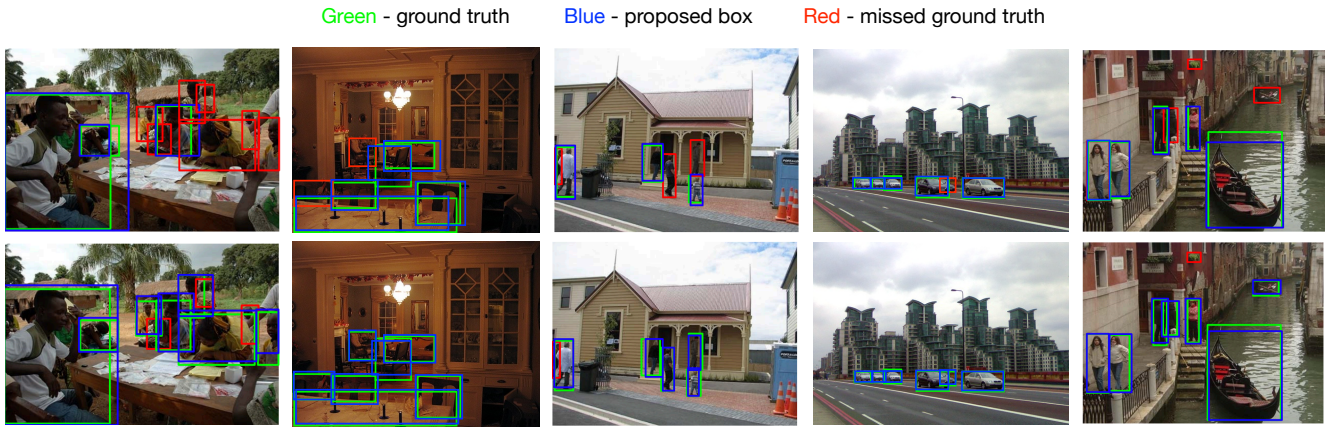

Figure 4. Visual comparison between the region proposal networks of two Faster R-CNN baselines. Top row: object proposals (after IoU thresholding) with standard Cross-Entropy loss (CE). Bottom row: object proposals with MetricOpt + CE. By direct optimization of the target metric AP^{50}, MetricOpt is found to improve the region proposal module of the two-staged Faster R-CNN detector.

correcting pathologies in the training of learned optimizers. In *ICML*, 2019.

[7] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

[8] Michal Rolinek, Vit Musil, Anselm Paulus, Marin Vlastelica, Claudio Michaelis, and Georg Martius. Optimizing rank-based metrics with blackbox differentiation. In *CVPR*, 2020.

[9] Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gmez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *ICML*, 2017.

[10] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *ICML*, 2019.