HALO: Hardware-Aware Learning to Optimize Supplement

Chaojian Li[1*], Tianlong Chen[2*], Haoran You[1],
Zhangyang Wang[2], and Yingyan Lin[1]

[1] Rice University, Houston TX 77005, USA
[2] The University of Texas at Austin, Austin TX 78712, USA
{cl114,hy34,yingyan.lin}@rice.edu,{tianlong.chen,atlaswang}@utexas.edu

A More Baselines: SGD with Momentum and Stepwise Learning Rate Decay

We here evaluate the proposed HALO over another handcrafted optimizer, which is SGD paired with momentum and a stepwise learning rate decay following [4,11]. Specifically, this set of baselines adopt a momentum of 0.9 together with the stepwise learning rate decay, in which the initial learning rate is divided by 10 at the 50th and 75th epochs, respectively, for a total of 100 adaptation training epochs. As shown in Fig. 1, we manually tune the baseline's initial learning rate η from 10^{-1} to 10^{-5}. We can see that (1) the automatically generated optimizer by our proposed HALO even outperforms (e.g., $\uparrow$1.67% in terms of average adaptation/test accuracy) the best manually designed baseline (i.e., $\eta = 10^{-4}$); and (2) the baseline's performance is highly dependent on η, e.g., the test accuracy varies from about 20% to 65% when η changes from 10^{-1} to 10^{-5}, limiting its applicability for wide adoption, whereas our proposed HALO does not require such a manual and time-consuming hyperparameter tuning in addition to its advantageous one-for-all generalization capability (i.e., one generated optimizer works for different optimizees with different datasets) as described in Section 4.1 of the main content.

B More Structural Sparsity Schemes for Traditional Hand-crafted Optimizers

Since Section 4.2.2 in the main content shows that applying our structural sparsity regularizer to the most competitive baseline, Adam, does not lead to benefits. To add more support to this conclusion, a grid search to find the optimal structural sparsity schedule of Adam, SGD, and Adagrad are conducted with the same experiment settings in Section 4.2.2 of the main content. As shown in Table 1, traditional handcrafted optimizers cannot show competitive results even with the most carefully designed updating structural sparsity schemes (to our best

*The first two authors Chaojian Li and Tianlong Chen contributed equally. Correspondence should be addressed to Zhangyang Wang and Yingyan Lin.

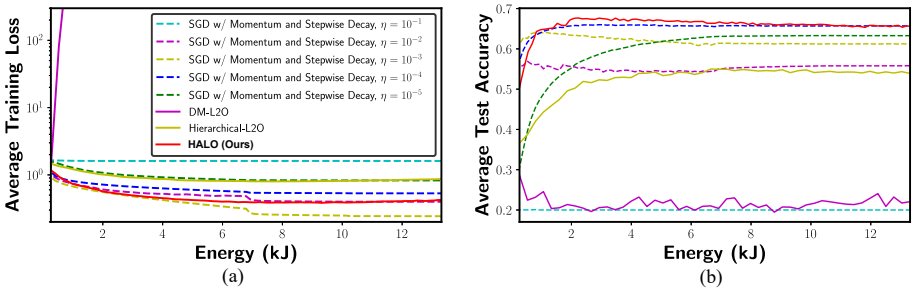

Fig. 1: **HALO for a wider optimizee over the baseline of SGD with momentum and stepwise learning rate decay**: (a) The average training loss and (b) adaptation/test accuracy vs. the energy cost over ten runs, on CIFAR-10-A.

Table 1: Grid search on structural sparsity schemes for traditional hand-crafted optimizers

Method	Avg. Test Acc. (%)					
	Energy = 2.0 kJ			Energy = 4.0 kJ		
	SGD	Adagrad	Adam	SGD	Adagrad	Adam
"100%-100%-100%" scheme	65.61	65.71	56.51	65.44	66.15	56.91
"10%-10%-10%" scheme	62.21	63.43	37.20	64.51	64.71	41.47
"30%-30%-30%" scheme	65.45	65.32	58.07	65.63	66.12	58.67
"50%-50%-50%" scheme	65.11	65.24	53.39	65.68	65.98	58.39
"50%-30%-10%" scheme	60.98	62.40	56.91	64.03	63.94	59.53
"10%-30%-50%" scheme	65.70	66.17	55.78	65.55	66.42	57.70
HALO (Ours)	**67.50**			**67.57**		
HALO Acc. Improv.	**↑1.33 - ↑30.30**			**↑1.15 - ↑26.10**		

possible effort). HALO outperforms all other optimizers with an obvious higher accuracy (i.e., ↑1.15% - ↑30.30%) under the same energy cost.

C HALO Also Benefits the Convergence Speed

As described in the main content, the proposed HALO is the first learning-to-optimize framework that is dedicated to on-device adaptation applications, i.e., explicitly design to improve the two key metrics of on-device adaptation including the energy efficiency and the required adaptation time. We here evaluate HALO in terms of the convergence speed (i.e., the required adaptation time on an Edge GPU [5]) in addition to the evaluation experiments in Section 4 in terms of the required energy consumption. Specifically, we re-evaluate all the optimizers in terms of the average training loss and adaptation/test accuracy

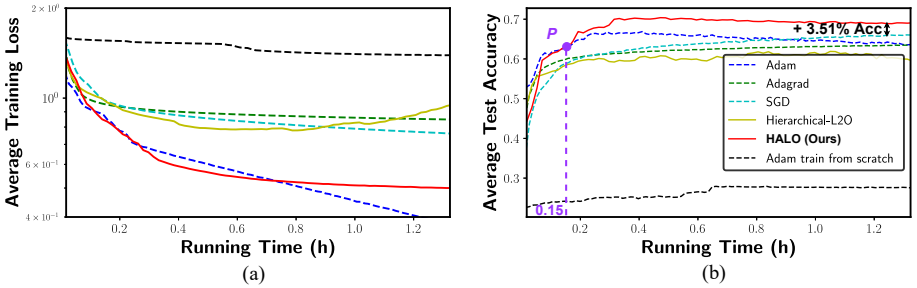

Fig. 2: **HALO for a wider and deeper optimizee over all the baseline optimizers**: (a) The average training loss and (b) adaptation/test accuracy vs. the **running time** over ten runs, on CIFAR-10-A.

verus the *real-device running time*, on the wider and deeper optimizees (see Section 4.3.1 of the main content) with the CIFAR-10-A dataset over ten random initialization settings. As shown in Fig. 2, HALO outperforms (e.g., ↑3.51% average adaptation/test accuracy over the best baseline) all other optimizers after 9 minutes (see the point marked as P in Fig. 2) running time in terms of the average adaptation/test accuracy under the same running time budget.

D Ablation Studies of HALO's Jacobian Regularizer

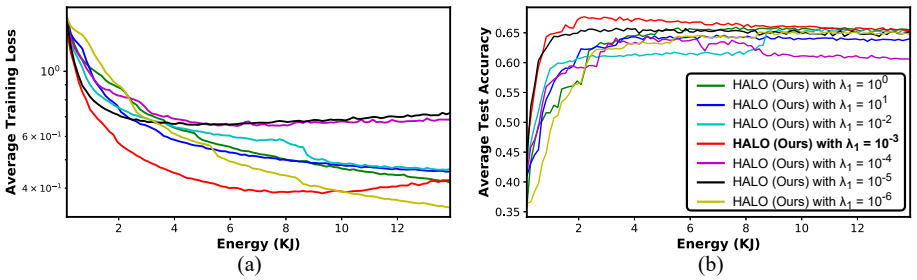

Fig. 3: **Ablation studies of HALO's Jacobian regularizer** in terms of the hyperparameter λ_1: the average training loss (a) and adaptation accuracy (b) vs. the required energy cost over ten runs, on CIFAR-10-A.

We next presents an ablation study of HALO's hyperparameter λ_1 in the Jacobian regularizer, as introduced in Section **??**, based on the wider optimizee (Fig. **??** (b)) and CIFAR-10 dataset and using an updating probability schedule of "10%-30%-50%". As shown in Fig. 3, we decrease λ_1 of the Jacobian regularizer

from 1 to 10^{-6}, without loss of generality. The experiment results show that (1) our proposed Jacobian regularizer is insensitive to hyper parameters since 85.71% of them result in a small convergent accuracy range (i.e., 64.01% - 65.48%); (2) $\lambda_1 = 10^{-3}$ leads to the best performance, i.e., higher adaptation/test accuracy with the same energy (e.g., up to ↑2.81% higher), among all variants.

E Train/test subset splitting

For datasets other than TDP, we follow their default experiment settings [3,2,9,1]; for the TDP dataset which doesn't provide uniform data pre-processing, we divide the 3221 thyroid cases into four classes (negative, hypothyroid, sick, and hyperthyroid) with each sample including a 26-dimensional diagnostic vector for disease prediction, where we use 75% of the data to train and the remaining to test for each domain and re-sampling to adjust the class distributions since negative cases dominate in the original dataset.

F Details of Optimizer Design

The proposed HALO adopts a hierarchical RNN in a SOTA learning to optimize work [10] as the optimizer network and follows its network parameters. One notable advantage of HALO is that its generated optimizers have a **one-for-all generalization capability**, i.e., optimizers can be designed under the same settings (including the network structure, dataset, and training hyper-parameters) for various targeted tasks, datasets, and optimizees. Specifically, the HALO optimizer in this paper is trained using: (1) the aforementioned adaptation setting (see Section **??**) on the MNIST dataset; (2) a simple 2-layer CNN introduced in [10] and shown in Fig. **??** (a); (3) 50 training epochs each of which consists of 100 iterations with a batch size of 64; (4) an optimal hyperparameter $\lambda_1 = 1 \times 10^{-3}$ which is resulted from a grid search as shown in the Section D; and (5) an updating probability of 10%, 30% and 50% for the first third, the middle third, and the last third network layers, respectively.

G The Energy Measurement Setup

As described in this supplementary and main content, we evaluate the required energy cost and running time of the proposed HALO and the baseline optimizers based on real-device measurement results except for experiments with quantized optimizees that have been described in Section 4.1 of the main content. Fig. 4 shows the energy/running-time measurement setup using an Egde GPU (i.e., NVIDIA JETSON TX2) [5] (to the right of the figure) or a Raspberry Pi for general IoT applications [8] (to the left of the figure). Specifically, the Edge GPU is connected to a laptop, and the real-time energy consumption and running time are obtained using the sysfs [6] of the embedded INA3221 [7] power rails monitor. For the experiments with the Raspberry Pi, the setup is the same as

Fig. 4: The energy/running-time measurement setup with a laptop, a state-of-the-art Edge GPU [5] and a popular IoT device [8]

that of the Edge GPU except that the energy/time consumption is obtained using an external power monitor meter because of its lack of corresponding on-board power rail monitor.

H The Input of HALO

As mentioned in Section 3.1 of the main content, similar to the learned optimizers in [10], $(m_{\theta^t}, \gamma_{\theta^t}, \eta_{\theta^t})$ are chosen as the optimizer inputs for the parameter θ at the t-th iteration. Here we provide the details about these parameters, i.e., the scaled average gradient m_{θ^t}, the relative log gradient magnitude γ_{θ^t}, and the relative log learning rate η_{θ^t}.

- m_{θ^t} can be derived from Equations 1, 2, and 3, where s denotes the timescale index following the same definition in [10], $\bar{g}_{\theta^{t+1},s}$ in Equation 1 denotes an exponential moving averages of the gradient (i.e., g_{θ^t}) on several timescales, σ represents the sigmoid function, β_{g,θ^t} and β_{λ,θ^t} represent the momentum logits outputted by our optimizer, $\lambda_{\theta^{t+1},s}$ denotes a running average of the square average gradient, and $m_{\theta^t,s}$ is the scaled average gradients of m_{θ^t} on timescale s.

$$\bar{g}_{\theta^{t+1},s} = \bar{g}_{\theta^t,s} \cdot \sigma(\beta_{g,\theta^t})^{2^{-s}} + g_{\theta^t} \cdot (1 - \sigma(\beta_{g,\theta^t})^{2^{-s}}) \tag{1}$$

$$\lambda_{\theta^{t+1},s} = \lambda_{\theta^t,s} \cdot \sigma(\beta_{\lambda,\theta^t})^{2^{-s}} + (\bar{g}_{\theta^t,s})^2 \cdot (1 - \sigma(\beta_{\lambda,\theta^t})^{2^{-s}}) \tag{2}$$

$$m_{\theta^t,s} = \frac{\bar{g}_{\theta^t,s}}{\sqrt{\lambda_{\theta^t,s}}} \tag{3}$$

- γ_{θ^t} denotes the relative log gradient magnitude defined below:

$$\gamma_{\theta^t} = \log(\lambda_{\theta^t,s}) - \mathbb{E}_s[\log(\lambda_{\theta^t,s})] \tag{4}$$

- η_{θ^t} can be be derived from Equations 5, 6, and 7, where $\hat{\eta}_{\theta^t}$ is the log step length which is specified relative to an exponential running average $\bar{\eta}_{\theta^t}$ with

meta-learned momentum γ for stability reasons, $\Delta\hat{\eta}_{\theta^t}$ is the update outputted by our optimizer, and η_{θ^t} is the relative log learning rate of each parameter θ.

$$\hat{\eta}_{\theta^{t+1}} = \Delta\hat{\eta}_{\theta^t} + \bar{\eta}_{\theta^{t+1}} \tag{5}$$

$$\bar{\eta}_{\theta^{t+1}} = \gamma \cdot \bar{\eta}_{\theta^t} + (1-\gamma) \cdot \hat{\eta}_{\theta^{t+1}} \tag{6}$$

$$\eta_{\theta^t} = \hat{\eta}_{\theta^t} - \mathbb{E}_\theta[\hat{\eta}_{\theta^t}] \tag{7}$$

References

1. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: A public domain dataset for human activity recognition using smartphones. In: Esann (2013)
2. Krizhevsky, A., et al.: Learning multiple layers of features from tiny images (2009)
3. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1999)
4. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=rJlnB3C5Ym`
5. NVIDIA Inc.: NVIDIA Jetson TX2, `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/`, accessed 2019-09-01
6. Patrick Mochel and Mike Murphy.: sysfs - The filesystem for exporting kernel objects., `https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt`, accessed 2019-11-21
7. Texas Instruments Inc.: INA3221 Triple-Channel, High-Side Measurement, Shunt and Bus Voltage Monitor, `http://www.ti.com/product/INA3221`, accessed 2019-11-21
8. Upton, E., Halfacree, G.: Raspberry Pi user guide. John Wiley & Sons (2014)
9. Vergara, A., Vembu, S., Ayhan, T., Ryan, M.A., Homer, M.L., Huerta, R.: Chemical gas sensor drift compensation using classifier ensembles. Sensors and Actuators B: Chemical **166**, 320–329 (2012)
10. Wichrowska, O., Maheswaranathan, N., Hoffman, M.W., Colmenarejo, S.G., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3751–3760. JMLR. org (2017)
11. You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R.G., Wang, Z., Lin, Y.: Drawing early-bird tickets: Towards more efficient training of deep networks (2019)