HALO: Hardware-Aware Learning to Optimize

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

Abstract. There has been an explosive demand for bringing machine learning (ML) powered intelligence into numerous Internet-of-Things (IoT) devices. However, the effectiveness of such intelligent functionality requires in-situ continuous model adaptation for adapting to new data and environments, while the on-device computing and energy resources are usually extremely constrained. Neither traditional hand-crafted (e.g., SGD, Adagrad, and Adam) nor existing meta optimizers are specifically designed to meet those challenges, as the former requires tedious hyper-parameter tuning while the latter are often costly due to the meta algorithms' own overhead. To this end, we propose *hardware-aware learning to optimize* **(HALO)**, a practical meta optimizer dedicated to resource-efficient on-device adaptation. Our HALO optimizer features the following highlights: (1) *faster adaptation speed* (i.e., taking fewer data or iterations to reach a specified accuracy) by introducing a new regularizer to promote empirical generalization; and (2) *lower per-iteration complexity*, thanks to a stochastic structural sparsity regularizer being enforced. Furthermore, the optimizer itself is designed as a very light-weight RNN and thus incurs negligible overhead. Ablation studies and experiments on five datasets, six optimizees, and two state-of-the-art (SOTA) edge AI devices validate that, while always achieving a better accuracy (↑0.46% - ↑20.28%), HALO can greatly trim down the energy cost (up to ↓60%) in adaptation, quantified using an IoT device or SOTA simulator. Codes and pre-trained models are at `https://github.com/RICE-EIC/HALO`.

Keywords: On-Device Learning, Learning to Optimize, Meta Learning, Efficient Training, Internet-of-Things

1 Introduction

The record-breaking success of machine learning (ML) algorithms has fueled an explosive demand for bringing ML-powered intelligent functionality into numerous Internet-of-Things (IoT) devices [39,37]. For practical deployment, many of them (such as autonomous vehicles, drones, mobiles, and wearables) require on-site in-situ learning for enabling them to continuously learn from new data and adapt

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

to new environments [50]. However, the realization of on-device continuous model adaptation remains a bottleneck challenge because powerful performance of ML algorithms often comes at a prohibitive training cost while IoT devices are often extremely resource constrained. To tackle this challenge, existing efficient training techniques such as low-precision and pruning training can largely fall short as they are not designed and optimized for on-device model adaptation. Specifically, in contrast to standard training, on-device adaptation needs to (1) achieve fast model convergence (i.e., reduced training iterations) given that limited data is available or can be stored on IoT devices and (2) be realized with much boosted training energy/time efficiency for possibly wide adoption.

To close the aforementioned gap, we explore from a promising yet unexplored perspective motivated by the observation that neither traditional hand-crafted (e.g., SGD, Adagrad, and Adam) nor existing meta optimizers are dedicated to meet the on-device adaptation challenges. This is because the former requires tedious and manual hyper-parameter tuning, while the latter can be automated, they are often more costly due to the meta algorithms' own overhead. Specifically, we propose, develop, and experimentally validate a *hardware-aware learning to optimize* (**HALO**) framework, targeting to aggressively trim down the energy cost of on-device ML adaptation. This paper makes the following contributions:

- We **for the first time** introduce learning to optimize to a practical and explosively demanded application of resource-efficient, on-device ML adaptation, and demonstrate that it largely outperforms the most competitive SOTA optimizers. The proposed HALO framework is achieved using a Long Short-Term Memory (LSTM) aided with an innovative Jacobian regularizer that is dedicated for faster adaptation.
- To further ensure that the proposed HALO can be practically deployed for model adaptation on numerous resource-limited IoT devices, we next introduce (stochastic) structural sparsity as an extra regularizer for the learning optimizer, so that it can be efficiently implemented on hardware. Thanks to the aforementioned two regularizers, the HALO generated optimizers are enforced to naturally achieve the critical specification of on-device adaptation, i.e., both faster adaptation speed and reduced per-iteration complexity.
- We have evaluated and demonstrated the HALO optimizers on various models, datasets, and experiment settings (including going-wider, going-deeper, going-sparser, and going-lower bits), by exhaustively comparing it with existing off-the-shelf traditional hand-crafted and meta-optimizers. Extensive experiments and ablation studies show that HALO consistently outperforms others, by largely reducing on-device adaptation energy consumption (i.e., the energy it takes to adapt for achieving the specified accuracy) while always maintaining a better accuracy given the same energy budget.

2 Related Works

Model Adaptation. Model adaptation techniques are commonly exploited to: (1) continuously improve a model's performance in the same domain, as more data

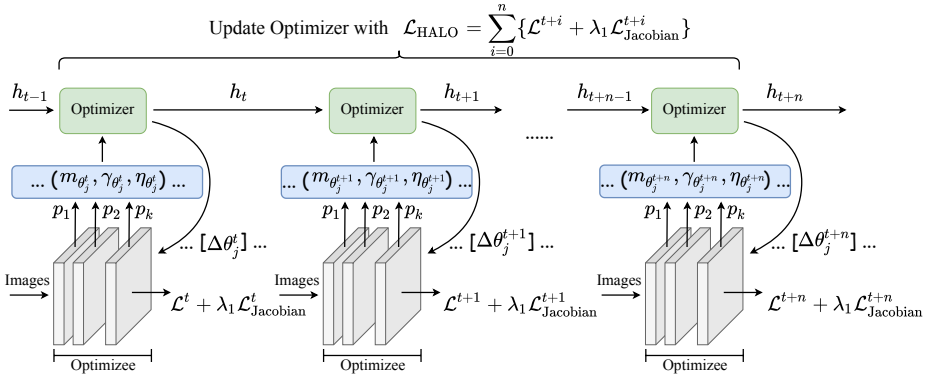

Fig. 1: The overall framework of our proposed hardware-aware learning to optimize (HALO) method. For each time step t, the optimizer will first take the previous hidden vector h_{t-1} and the relative input vector $(m_{\theta_j^t}, \gamma_{\theta_j^t}, \eta_{\theta_j^t})$ which contains the gradient information from the optimizees, and then output a parameter update rule $\Delta\theta_j^t$ for the optimizees. The layers to be updated are selected according to the probability $(p_1, \cdots, p_k)$, i.e., the structural sparsity regularizer. After $(n+1)$ optimization iterations (in our case, $n = 10$), we update the optimizer with the averaged optimizee loss $\mathcal{L}$ and the Jacobian regularizer $\mathcal{L}_{\text{Jacobian}}$.

is collected; or (2) further tune a model already trained on one domain (source domain) to adapt to a new domain (target domain), assuming the source and target domains to have a certain mismatch (either data distribution or task types) [7]. Many adaptation algorithms have been explored for various ML algorithms, from aligning data distributions [10] to utilizing feature or module transferability [16,41,5], for which [59] provides a comprehensive literature review.

Adaptation algorithm is the cornerstone for many intelligent edge platforms to perceive and react to the changing new environments (such as drones and outdoor robots) and for wearable devices to personalize their functionality to individual users [29], and so on.

Learning to Optimize. Using machine learning algorithms to design an optimizer is a promising direction towards replacing tedious algorithm crafting and/or hyperparameter tuning. [2] first employs a coordinate-wise LSTM as a learnable optimizer for training neural networks. It takes the gradient of optimizee parameters as inputs and outputs the parameters' update rule. [11] introduces the history of objective values for inputs, and outputs gradients as the actions of reinforcement learning agents. [42] introduces two practical techniques of random scaling and objective convexifying to boost generalization ability. [66] designs a hierarchical RNN architecture, augmenting the inputs with the log gradient magnitudes and the log learning rate; its results remain to be a SOTA among learned optimizers. Lately, [8] combines both point-based and population-based optimization algorithms, and further incorporates the posterior into meta-loss to balance the exploitation-exploration trade-off.

3 The Proposed HALO Framework

In this section, we introduce our HALO framework with two innovations dedicated to the resource-efficient adaptation goal. First, a Jacobian regularizer is designed to boost the *empirical generalization and convergence speed*. Second, we introduce structural sparsity as the desired property to be enforced on the optimizer output, such that the resulting update is more *hardware friendly and energy-efficient*. Both are shown to be experimentally effective in Section 4.

3.1 Faster and Better: A Jacobian-Regularized Learned Optimizer

The backbone of HALO follows the classical setting in [66]. We adopt a similar hierarchical RNN as the learned optimizer. Specifically, the hierarchical RNN architecture contains three levels, named "Parameter RNN", "Tensor RNN", and "Global RNN" from the low to high levels. Specifically, the "Parameter RNN" deals with the inputs and outputs update rules for each parameter of the optimizees; the "Tensor RNN" takes as inputs all hidden states from the "Parameter RNN" which processes parameters belonging to the same tensor and returns a bias term to them; and the "Global RNN" takes as inputs all hidden states from the "Tensor RNN" and returns a bias term. The RNN parameters are shared within each level. In this way, the learned optimizer is able to capture the inter-parameter dependencies.

As shown in Fig. 1, following the prior wisdom of learned optimizers [66,42], the inputs of our optimizer are $(m_{\theta_j^t}, \gamma_{\theta_j^t}, \eta_{\theta_j^t})$, corresponding to the scaled averaged gradients $m_{\theta_j^t}$, the relative log gradient magnitudes $\gamma_{\theta_j^t}$, and the relative log learning rate $\eta_{\theta_j^t}$ of layer j's parameter θ_j in iteration t, respectively, more details of which can be found in the supplement. The output is the parameter update $\Delta\theta_j^t$. Our learnable optimizer performs a coordinate-wise update on the parameter θ so that the learned optimizer can scale to training optimizees with any number of parameters: an important **"one-for-all"** feature desired by mobile applications where a number of different models are typically configured to meet different platforms' resource constraints. Between different coordinates, the weights of the optimizer are shared. In HALO, the optimizer is updated by $\mathcal{L}_{\text{HALO}}$, which is the sum of the average optimizee loss $\mathcal{L}$, plus a new Jacobian regularizer term $\mathcal{L}_{\text{Jacobian}}$ (λ_1 is a hyperparameter, more ablation studies are provided in the supplement): $\mathcal{L}_{\text{HALO}} = \mathcal{L} + \lambda_1 \mathcal{L}_{\text{Jacobian}}$.

We next discuss "*what* and *why*" regarding this new regularizer.

Jacobian Regularizer. We propose a powerful regularizer, called *Jacobian regularizer*, that controls the update magnitudes of the optimizee (i.e., the model to be adapted by HALO). Without loss of generalizability, we define our optimizee with k layers as $f(\theta)$, $\theta = (\theta_1, \theta_2, \cdots, \theta_k)$ ($k = 1$ for shallow models). The Jacobian of the optimizee loss $\mathcal{L}$ can be written as $J = \left[\frac{\partial \mathcal{L}}{\partial \theta_1}, \frac{\partial \mathcal{L}}{\partial \theta_2}, \cdots, \frac{\partial \mathcal{L}}{\partial \theta_k}\right]$, and our new regularizer term can be defined as $\mathcal{L}_{\text{Jacobian}} = ||J||_2^2$. ($||\,\cdot\,||_2^2$ is a Frobenius norm)

The Jacobian regularizer encourages the optimizee's landscape to be **smoother** and **flatter**. Intuitively, such a landscape facilitates an optimizer to explore **faster**

and **more widely** in a neighborhood, which makes it favor our goal of fast adaptation. More formally, recent theories have revealed that optimizing in flat minima leads to more generalizable solutions [21,28]. It is straightforward to see that the larger the components of the Jacobian are, the more unstable the model prediction is with respect to input perturbations. Enforcing $\mathcal{L}_{\text{Jacobian}}$ is therefore a natural way to reduce this instability: it decreases the input-output Jacobian magnitude, potentially reducing the influence of noisy updates during training. That **robustness** is meaningful for practical on-device adaptation whose input samples are often very noisy [6].

Among past works, [51] constrained the Jacobian matrix of the encoder for the regularization of auto-encoders. [22] showed that constraining the Jacobian increases classification margins of neural networks and therefore enhances the model stability. While the above works exploit Jacobian regularizer in classical optimizers, to our best knowledge, we are **the first** to extend this line of ideas into the *learning to optimize* field. Our results demonstrate its effectiveness in improving generalization performance (i.e., adaptation/test accuracy) of the learned optimizer, in addition to the faster empirical convergence speed.

Besides, the analysis in [57] found that a bounded spectral norm of the network's Jacobian matrix is more directly related to the generalization of neural networks. We tested and verified that replacing the Frobenius norm with the spectral norm will yield similar empirical performance and convergence benefits, sometimes the spectral norm being better. However, computing the spectral norm is much more expensive and goes against our goal of resource efficiency: that is why we stay with the Frobenius norm in implementing $\mathcal{L}_{\text{Jacobian}}$.

3.2 More Hardware-Efficient: Stochastic Structural Sparsity

As a learned optimizer, HALO targets faster empirical convergence (e.g., taking fewer iterations to reach a certain accuracy level), which is further boosted by the new Jacobian regularizer. We introduce another regularizer, that enhances the energy efficiency from an orthogonal angle: enforcing structural sparsity on the learned updates (i.e., the outputs of HALO) at each iteration, such that the per-iteration complexity and hence resource costs could be trimmed down.

Structural sparsity is a well-explored regularizer that is typically achieved by weight decay, norm constraints, or various pruning means [65]. In comparison, we choose an extremely cheap "stochastic" way to enforce that. As shown in Fig. 1, for each layer j in the optimizee, we set it to have a probability p_j to be updated by the learned optimizer, at each iteration. Correspondingly, only the layers that are updated at the current iteration will back-propagate to update the learned optimizers.

We note that similar ideas of "randomly not updating all layers every time" were previously exploited for training very deep networks [25] and faster dynamic inference [67]. Lately, it was demonstrated to be helpful for energy-efficient training too [64]. We are **the first** to show this heuristic regularizer to work well for learned optimizers in efficient training.

Compared to enforcing filter- and parameter-wise structural sparsity, the purposed layer-wise structural sparsity regularizer is particularly hardware-friendly, as it requires no massive indexing and gathering processing. As our experiments in Section 4.2.1 show, this alone can save up to 45.42% training energy per iteration on average while sacrificing little accuracy or convergence speed.

4 Experiments and Analysis

In this section, we present ablation studies and evaluation results of the proposed HALO under five datasets, six optimizees (i.e., the wider one in Fig. 2 (b), the wider and deeper one in Fig. 2 (c), ResNet-18 [20] with quantization and high sparsity, two multilayer perceptrons (MLPs), and a CNN+LSTM [52]), and two SOTA edge AI computing devices.

4.1 Experiment Setup

Here we summarize our experiment details including the datasets and baselines, adaptation/test experiment setting, and evaluation metrics, and details of the optimizer design can be found in the supplement.

Datasets and Baselines. To evaluate the potential of the proposed HALO in handling on-device adaptation under different applications and scenarios, we consider a total of **five datasets**, including (1) MNIST [34], (2) CIFAR-10 [32], (3) Thyroid Disease Prediction (TDP) [12], (4) Gas Sensor Array Drift (GSAD) [62], and (5) Smartphones (SP) [3] (more details on the train/test subset splitting could be found in the supplement). These five diverse sets of datasets can emulate on-device ML applications for tasks of object recognition, healthcare monitoring, environmental monitoring, and activity recognition

For benchmarking, we evaluate HALO's generated optimizers against **five baselines** of SOTA optimizers, including three traditional hand-crafted optimizers (i.e., SGD, Adagrad [13], and Adam [31]) and two meta-optimizers (i.e., the DM-L2O [2] and Hierarchical-L2O [66]).

Adaptation/Test Setting. To evaluate each optimizer under a given dataset, we split the dataset into two non-overlapping subsets following the prepossessing in [23,30,18], which are termed as A and B, respectively, with samples from different domains and each consisting of non-overlapping classes. Table 1 summarizes the splitting details for all our considered datasets. Following the strategy in real-world deployments [4,49,19], we first pre-train the model on one subset, and then start from the pre-trained model to retrain it on the other subset to see how accurately and efficiently the corresponding optimizee can adapt to the new domain. The same splitting is applied to the test set for accuracy validation in both the pre-train and adaptation training processes. We observe that the accuracy of the optimizee in the TDP dataset, which is trained on the subset B for the male domain and achieves an accuracy of 73.92%, drops to 55.74% when directly applying to the female domain without adaptation, motivating the need of adaptation.

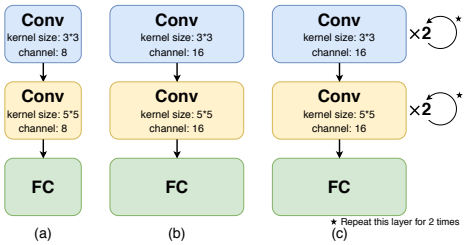

Fig. 2: The convolutional networks adopted for (a) training all the evaluated optimizers, and the optimizee networks including (b) a wider one and (c) a wider and deeper one, as compared to (a).

Table 1: A summary of the splitting details for all the considered datasets.

Dataset	Subset	Domain
MNIST [34]	A	{1, 3, 5, 7, 9}
	B	{0, 2, 4, 6, 8}
CIFAR-10 [32]	A	{plane, bird, deer, frog, ship}
	B	{car, cat, dog, horse, truck}
TDP [12]	A	Female
	B	Male
GSAD [62]	A	{Acetaldehyde, Acetone, Toluene}
	B	{Ethanol, Ethylene, Ammonia}
SP [3]	A	{Walking Upstairs, Sitting, Laying}
	B	{Walking, Walking downstairs, Standing}

Evaluation Metrics. We evaluate all optimizers in terms of the hardware energy consumption in addition to the optimizees' averaged training loss and adaptation/test accuracy over ten random initialization settings. Specifically, for the full-precision optimizees, we obtain the real-measured energy consumption on two SOTA edge AI computing devices, i.e., NVIDIA TX2 [45] (for more complex CNNs in Fig. 3 - Fig. 6 and Table 4) and Raspberry Pi [61] (for simpler MLPs in Tables 2 - 3); for the quantized optimizees, we adopt a SOTA hardware energy simulator, Bit Fusion [55], to obtain the energy consumption (the one in Fig. 7). The real-device energy measurement setup and energy simulation details are provided in the supplement.

4.2 Ablation Studies of the Proposed HALO

Here we perform ablation studies of HALO's effectiveness (Section 4.2.1) and structural sparsity regularizer (Section 4.2.2).

4.2.1 Ablation Studies on the Effectiveness of HALO's Regularizers

For evaluating the effectiveness of HALO's regularizers, we perform a set of experiments using the wider optimizee (see Fig. 2 (b)) and CIFAR-10 dataset. Specifically, the optimizee is evaluated when enforcing the two regularizers of HALO in an incremental manner, with the corresponding optimizer being trained from scratch. Fig. 3 shows the average training loss and adaptation/test accuracy versus the corresponding real-device measured energy over ten random initialization settings, from which we can make the following observation:

First, the vanilla HALO without the two regularizers can not surpass the SOTA traditional hand-crafted optimizer, Adam [31], in terms of both the training loss and adaptation/test accuracy after convergence, while at the same time suffering from a larger variance;

Second, after adding the Jacobian regularizer, the corresponding optimizee achieves a better adaptation/test accuracy **after** convergence, which verifies the

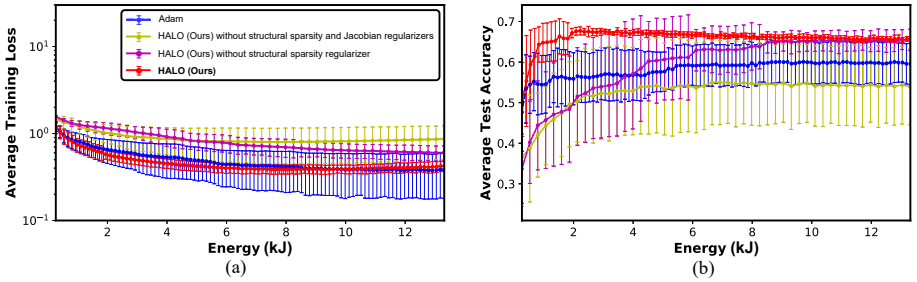

Fig. 3: Ablation studies on **the effectiveness of HALO's regularizers**: (a) The average training loss and (b) adaptation/test accuracy vs. the required energy cost over ten runs, on CIFAR-10-A.

flatten local minima found by HALO with the help of the Jacobian regularizer is beneficial for generalization ability as introduced in Section 3.1, while always performing worse under the same energy budget in the early stage, as compared to the optimizee trained using Adam;

Third, our proposed HALO always leads to a lower training loss (e.g., up to ↓16.72% lower under the same energy) and higher accuracy (e.g., up to ↑11.70% higher under the same energy) while having a smaller variance, as compared to all baselines, which seems to align with recent observations that compressing gradient during training can benefit the efficiency without hurting the performance [64,14].

This set of experiments validate that the two regularizers integrated into HALO lead to not only **faster adaptation** with **reduced energy cost**, but also offer a bonus benefit of **improving convergence stability**.

4.2.2 Ablation Studies of HALO's Structural Sparsity Regularizer

Here we present ablation studies on the schedule schemes of HALO's updating probability in the structural sparsity regularizer using the wider optimizee (see Fig. 2 (b)) and CIFAR-10 dataset. Specifically, different updating probability can be adopted for the first, second, and third layers of the optimizee.

For example, "10%-30%-50%" means the corresponding updating probability are 10%, 30%, and 50%, respectively. For HALO, we consider three schedule schemes for the updating probability, i.e., "progressively increased", "uniformly equal", and "progressively decreased"; For Adam, we consider the schedule scheme of updating probability under which HALO performs the best (i.e., "progressively increased", more schemes for Adam and other optimizers could be found in the supplement).

The experiment results in Fig. 4 show that: (1) comparing Adam with Adam + structural sparsity regularizer, we find that the hand-crafted optimizer, Adam, does not benefit from this regularizer, as evidenced by the corresponding decreased adaptation/test accuracy; and (2) comparing the three schedule schemes of updating probability for HALO, we observed that the "progressively increased",

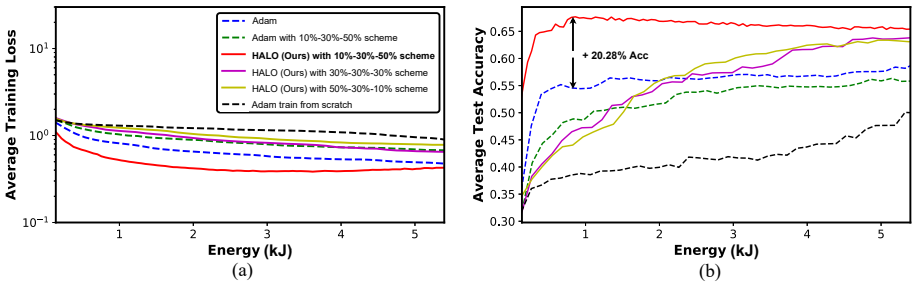

Fig. 4: **Ablation studies of HALO's structural sparsity regularizer** in terms of the updating probability: (a) The average training loss and (b) adaptation accuracy vs. the required energy cost over ten runs, on CIFAR-10-A.

e.g., "10%-30%-50%", significantly outperforms the other two schedule schemes by offering a higher adaptation/test accuracy under the same energy cost.

Note that the advantage of such a "progressively increased" schedule scheme for HALO is consistently observed under different datasets and models, which seems to coincide with recent findings [1,36,69,17,63] that (1) different stages of DNN training call for different treatments and (2) not all layers are equally important for training convergence.

4.3 HALO under Different Datasets/Optimizees

We then evaluate HALO under five different datasets and six different optimizees.

4.3.1 HALO on the CIFAR-10 Dataset

In this subsection, we evaluate HALO's performance and generalization capability when being applied to various optimizees, which are (1) wider (Fig. 2 (b)), (2) wider and deeper (Fig. 2 (c)), and (3) wider, deeper, highly sparse and quantized (pruned and quantized ResNet-18 [20]), as compared to the networks used to train the optimizers.

Experiment settings. For all the aforementioned three optimizees, experiments are performed using the CIFAR-10 dataset. And for all the experiments using the optimizee (3) mentioned above, a compressed ResNet-18 [20] is trained and pruned under a pruning ratio of 70.0%, which leads to a reduction of 43.5% and 61.5% in the computational cost (i.e., FLOPs) and model size over the unpruned one, respectively, while performing quantization-aware training [26] in the optimizee (3) during adaptation. Note that we consider a pruned and quantized optimizee because the current practice often compresses ML models before deploying them into IoT devices [15,46,64].

Experiment results and observations. For each optimizer on the three considered optimizees, we evaluate the optimizees' averaged training loss and

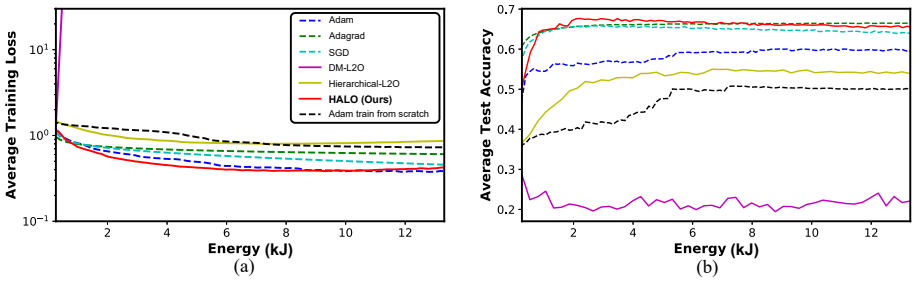

Fig. 5: **HALO for the wider optimizee**: (a) The average training loss and (b) adaptation/test accuracy vs. the energy cost over ten runs, on CIFAR-10-A.

adaptation/test accuracy under ten random initialization settings. The corresponding results are plotted in Figures 5 - 7, from which we can make the following **five observations**:

First, while SOTA learning to optimize works are merely evaluated in terms of the optimizees' training loss [2,42,66], we find that both the training loss and adaptation/test accuracy need to be considered for adaptation tasks, as a lower training loss might not guarantee a higher adaptation/test accuracy. For example, from Fig. 5 we can see that Adam achieves a smaller training loss but a lower adaptation accuracy, as compared to that of Adagrad.

Second, the HALO optimizer outperforms all other meta-optimizers under all the three optimizees, while (1) the meta-optimizer, DM-L2O [2], fails under the evaluation with the wider optimizee as shown in Fig. 5, which is consistent with observations discussed in prior works of learning to optimize [66,42], thus won't be included in the following experiments, and (2) the meta-optimizer, Hierarchical-L2O [66], always leads to a lower adaptation accuracy (e.g., up to ↓16.88% lower in the wider optimizee as shown in Fig. 5 under the same energy budget) and a larger training loss (e.g., up to ↑65.65% larger in the wider

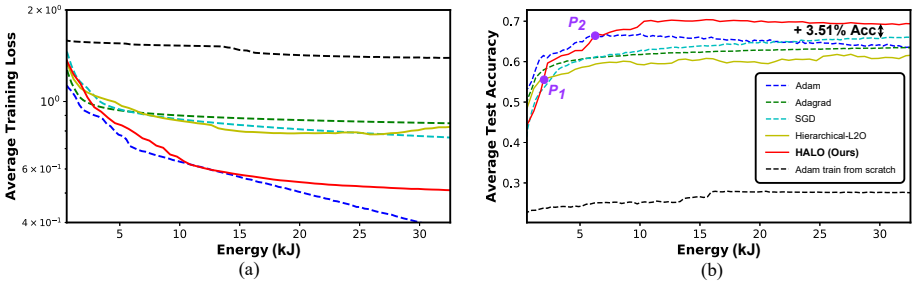

Fig. 6: **HALO for the wider and deeper optimizee**: (a) The average training loss and (b) adaptation/test accuracy vs. the energy cost over ten runs, on CIFAR-10-A.

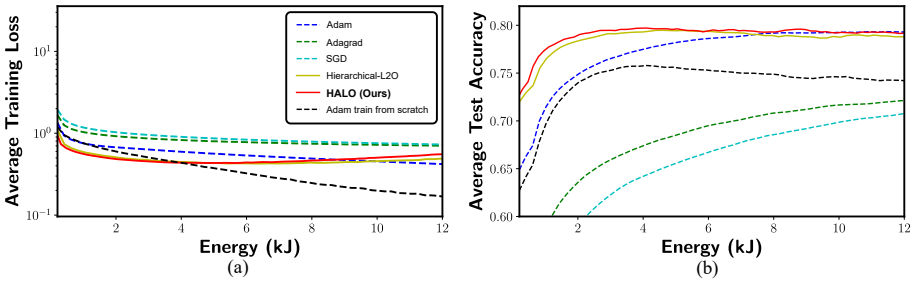

Fig. 7: **HALO for the wider, deeper, highly sparse, and quantized optimizee**: (a) The average training loss and (b) est accuracy vs. the energy cost over ten runs, on CIFAR-10-A.

optimizee as shown in Fig. 5 under the same energy budget), as compared to that of HALO.

Third, as shown in Fig. 6, in the early training stage before the cross-points P_1 or P_2, the HALO optimizer does not outperform other optimizers, while in the later stage after the cross-point P_2, HALO significantly surpasses others by a large performance margin (at least $\uparrow$3.51% higher adaptation/test accuracy when the energy cost is around 30 kJ). For this interesting and expected phenomenon, we conjecture the possible reasons, which are consistent with empirical observations in [21,28]: (1) with the assistance of the Jacobian regularizer, HALO in the early training stage can explore the training landscape to locate a flatter minima, whereas other optimizers who are not flatness-aware can be easily over-fitting and get stuck in some narrow local valleys, leading to results that the adaptation/test accuracy increases quickly at first, and then decays in the later stage; and (2) flatten local minima found by HALO is beneficial for improving its generalization capability [21,28], favoring HALO's large performance advantages.

Forth, wider, deeper, highly sparse and quantized optimizee targets on a difficult (e.g., exploding gradient is a common issue in training quantized network with traditional hand-crafted optimizers as described in [24]) yet practical setting for on-device adaptation [15,46,64]. As shown in Fig. 7, HALO outperforms all traditional hand-crafted optimizers obviously, regardless of the latter being extensively tuned, and show a marginal improvement over the Hierarchical-L2O, indicating the superiority of our purposed HALO (i.e., higher energy efficiency while having one-for-all generalization capability) even in such stringent cases.

Fifth, while the baseline optimizers don't have a fixed performance ranking under various optimizees (e.g., Adam performs better than Adagrad in the wider and deeper optimizee but worse in the wider optimizee, as shown in Fig. 6 and 5), HALO **always** achieves both a comparable or lower training loss (e.g., up to $\downarrow$16.72% lower in the wider optimizee as shown in Fig. 5 under the same energy budget) and a comparable or higher adaptation/test accuracy (e.g., up to $\uparrow$3.51% in the deeper optimizee as shown in Fig. 6 under the same energy budget) among all considered optimizers (including both traditional hand-crafted and mete-

Table 2: The evaluation results of the trained HALO on the TDP dataset.

Methods	Avg. Test Acc. (%)				Avg. Loss			Energy (kJ)		
	# of Iters.				# of Iters.			# of Iters.		
	1k	2k	3k	Final	1k	2k	3k	1k	2k	3k
Adam	79.41	79.85	79.85	79.85	0.49	0.44	0.43	0.04	0.09	0.13
Adagrad	75.81	75.81	75.82	75.83	0.77	0.73	0.71	0.04	0.09	0.13
SGD	77.08	78.07	78.29	78.29	0.73	0.69	0.67	0.04	0.09	0.13
Hierarchical-L2O	78.42	78.92	79.17	79.19	0.56	0.50	0.48	0.22	0.44	0.66
HALO	**79.42**	**79.88**	**80.16**	**80.28**	0.58	0.53	0.50	0.15	0.30	0.45
Improv. over SOTA L2O	**↑1.00%**	**↑0.96%**	**↑0.99%**	**↑1.09%**	-0.02	-0.03	-0.02	**↓31.8%**	**↓31.8%**	**↓31.8%**

optimizers), indicating **HALO's consistently best capability** in balancing the models' accuracy and on-device adaptation cost in various optimizees. Notably, even in some cases HALO outperforms other optimizers marginally (e.g., HALO and Adadgrad in Fig. 5), the latter ones need massive manual hyperparameters tuning while the former one could be used without any hyperparameter tuning while having the one-for-all generalization capability.

4.3.2 HALO on the Thyroid Disease Prediction Dataset (TDP)

In this subsection, we evaluate HALO's performance on the TDP dataset, the tasks of which can emulate on-device healthcare monitoring applications, one of the most popular applications on resource-limited IoT devices [54,43,33].

Experiment settings. As introduced in Section 4.1, the input data of the TDP dataset is a 26-dimensional vector, we thus adopt a three-layer MLP for the optimizee. Specially, we evaluate HALO in terms of the average training loss and adaptation/test accuracy under the same training iterations, where the adaptation energy budget is set to be no more than 0.5 kJ in Raspberry Pi [61] which is 2% of the most commonly-used Li-Po battery's capacity (27 kJ) [47] adopted by massive IoT devices [56].

Experiment results and observations. Table 2 summarizes the experiment results, from which we can see that (1) HALO outperforms all the baseline optimizers (both traditional hand-crafted and automatically learned ones) in terms of the achieved average adaptation/test accuracy under the same number of iterations; and (2) HALO achieves a higher adaptation/test accuracy (i.e., ↑0.96% - ↑1.00%) while requiring ↓31.8% less adaptation energy as compared to the SOTA learning to optimize optimizer, Hierarchical-L2O, thanks to its reduced energy cost per iteration, indicating the advantage and effectiveness of the proposed HALO for on-device adaptation. Note that although the hand-crafted traditional optimizers require a lower energy cost (e.g., ↓0.32 kJ less energy with a ↓0.31% lower accuracy) over the proposed HALO, they are not applicable for widely adopted on-device adaptation into numerous IoT applications due to their required tedious and manual hyper-parameter tuning.

Table 3: The evaluation results of the trained HALO on the GSAD Dataset.

Methods	Avg. Test Acc. (%)				Avg. Loss			Energy (kJ)		
	# of Iters.				# of Iters.			# of Iters.		
	1k	2k	3k	Final	1k	2k	3k	1k	2k	3k
Adam	69.75	69.91	69.91	69.91	0.42	0.38	0.37	0.02	0.04	0.06
Adagrad	59.06	60.31	61.38	62.16	0.93	0.79	0.71	0.02	0.04	0.06
SGD	58.62	59.66	60.56	60.97	0.78	0.69	0.67	0.02	0.04	0.06
Hierarchical-L2O	77.87	79.34	80.12	80.31	0.44	0.34	0.31	0.15	0.29	0.44
HALO	**79.25**	**85.22**	**86.81**	**87.00**	0.46	0.30	0.24	0.06	0.12	0.18
Improv. over SOTA L2O	**↑1.38%**	**↑5.88%**	**↑6.69%**	**↑6.69%**	-0.02	0.04	0.07	**↓60.0%**	**↓58.6%**	**↓59.1%**

4.3.3 HALO on the Gas Sensor Array Drift Dataset (GSAD)

Here we evaluate HALO's performance on the GSAD dataset, the tasks of which aim to classify the individual gas components in the gas mixtures based on the response of various metal oxide collected by the corresponding IoT sensors [27,35].

Experiment settings. The input data of GSAD is a 129-dimensional vector, we thus adopt a three-layer MLP for the optimizee, which is similar to the one in Section 4.3.2, except that the first layer's dimension is increased for adapting to the increased input dimension. We also evaluate HALO in terms of the average training loss and adaptation/test accuracy under the same training iterations. The adaptation energy budget of this set of experiments is 0.2 kJ which is smaller than that in Section 4.3.2, considering that GSAD's corresponding tasks of environmental monitoring applications often face circumstances with a very limited energy budget for a long time [38,44].

Experiment results and observations. This set of experiment results are shown in Table 3 from which we can see that: (1) HALO outperforms both the traditional hand-crafted and automatically learned baseline optimizers by achieving a higher average adaptation/test accuracy given the same number of iterations; and (2) HALO performs better than the SOTA learning to optimize optimizer, Hierarchical-L2O, as it achieves a higher average adaptation/test accuracy (i.e., ↑1.38% - ↑6.69%) while requiring ↓58.6% - ↓60.0% lower adaptation energy. Similarly, although the hand-crafted optimizers require a lower energy cost (e.g., ↓0.12 kJ less energy consumption at a cost of a ↓16.90% lower accuracy) over the proposed HALO, their required tedious and manual hyper-parameter tuning limits their applicability to on-device adaptation for numerous IoT applications.

4.3.4 HALO on the Smartphones Dataset (SP)

Here we evaluate HALO's performance on the SP dataset, the tasks of which aim to predict human activities based on data collected from smartphones [58,9].

Experiment settings. Considering the sequence data in the SP dataset, we adopt an optimizee with a CNN+LSTM architecture [52], where a CNN of three convolution layers are used for feature extraction followed by LSTMs to support sequence prediction. For this set of experiment, (1) HALO's structural sparsity

regularizer is only applied to the optimizee's convolution layers for balancing adaptation cost and accuracy; and (2) we relax the energy budget to 5 kJ in the NVIDIA TX2 [45], which is around 8% of the battery capability of commonly used smartphones (e.g., battery capacity of SAMSUNG Galaxy S20 is around 63 kJ [53]), considering the practicability of smartphone-based applications.

Experiment results and observations. The experiment results of HALO and the baseline optimizers in Table 4 show that (1) HALO again performs the best among all the optimizers including both traditional hand-crafted and automatically learned ones in terms of the achieved average adaptation/test accuracy under the same number of iterations; and (2) HALO requires a lower ↓44.3% - ↓45.1% energy while achieving a higher (i.e., ↑0.96% - ↑1.00%) average adaptation/test accuracy, over the SOTA learning to optimize optimizer, Hierarchical-L2O, thanks to its structural sparsity that enforces reduced energy cost per iteration. Note that while being automated and thus more applicable for a wide adoption into numerous IoT applications, the proposed HALO can achieves higher accuracies than all the hand-crafted optimizers which are limited when it comes to on-device adaptation due to their required tedious and manual hyper-parameter tuning needed for changes in data or application.

Table 4: The evaluation results of the trained HALO on the SP Dataset.

Methods	Avg. Test Acc. (%)				Avg. Loss			Energy (kJ)		
	# of Iters.				# of Iters.			# of Iters.		
	1k	2k	3k	Final	1k	2k	3k	1k	2k	3k
Adam	88.57	92.47	95.59	95.66	6.27E-5	1.79E-5	1.21E-5	1.15	2.92	4.62
Adagrad	69.24	72.31	75.64	76.56	1.17E-2	5.99E-3	4.38E-3	1.15	2.92	4.62
SGD	66.47	68.52	68.52	68.52	2.21E-2	1.15E-2	8.01E-3	1.15	2.92	4.62
Hierarchical-L2O	92.31	96.06	97.08	97.20	2.32E-4	5.38E-5	4.81E-5	1.61	4.08	6.44
HALO	**93.22**	**96.52**	**97.76**	**98.16**	2.49E-3	3.55E-4	3.09E-4	1.17	2.96	4.67
Improv. over SOTA L2O	**↑0.91%**	**↑0.46%**	**↑0.68%**	**↑0.96%**	-2.26E-3	-3.01E-4	-2.61E-4	**↓27.3%**	**↓27.5%**	**↓27.5%**

5 Conclusions

We propose a learning to optimize framework, HALO, a practical meta-optimizer dedicated to resource-efficient on-device adaptation. Specifically, Jacobian and structural sparsity regularizers are integrated in HALO to reduce per-iteration complexity and enforce faster adaptation speed, thus contribute to the adaptation efficiency, in additional to the light-weight RNN for reduced optimizer overhead. Furthermore, we demonstrate that HALO outperforms existing off-the-shelf traditional hand-crafted and meta-optimizers based on extensive experiments on six optimizees, five datasets, and two SOTA edge AI computing devices. In the future, we would like to generalize our HALO framework to more compressed (e.g., lower precision) networks.

Acknowledgements

The work is supported by the National Science Foundation (NSF) through the Real-Time Machine Learning (RTML) program (Award number: 1937592, 1937588).

References

1. Achille, A., Rovere, M., Soatto, S.: Critical learning periods in deep networks. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=BkeStsCcKQ
2. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in neural information processing systems. pp. 3981–3989 (2016)
3. 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)
4. Ashqar, B.A., Abu-Naser, S.S.: Identifying images of invasive hydrangea using pre-trained deep convolutional neural networks. International Journal of Academic Engineering Research (IJAER) **3**(3), 28–36 (2019)
5. Bengio, Y.: Deep learning of representations for unsupervised and transfer learning. In: Proceedings of ICML workshop on unsupervised and transfer learning. pp. 17–36 (2012)
6. Bippus, R., Fischer, A., Stahl, V.: Domain adaptation for robust automatic speech recognition in car environments. In: Sixth European Conference on Speech Communication and Technology (1999)
7. Blitzer, J., McDonald, R., Pereira, F.: Domain adaptation with structural correspondence learning. In: Proceedings of the 2006 conference on empirical methods in natural language processing. pp. 120–128 (2006)
8. Cao, Y., Chen, T., Wang, Z., Shen, Y.: Learning to optimize in swarms. In: Advances in Neural Information Processing Systems 32, pp. 15018–15028. Curran Associates, Inc. (2019), http://papers.nips.cc/paper/9641-learning-to-optimize-in-swarms.pdf
9. Chen, H., Mahfuz, S., Zulkernine, F.: Smart phone based human activity recognition. In: 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). pp. 2525–2532. IEEE (2019)
10. Chen, Q., Liu, Y., Wang, Z., Wassell, I., Chetty, K.: Re-weighted adversarial adaptation network for unsupervised domain adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7976–7985 (2018)
11. Chen, Y., Hoffman, M.W., Colmenarejo, S.G., Denil, M., Lillicrap, T.P., Botvinick, M., De Freitas, N.: Learning to learn without gradient descent by gradient descent. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 748–756. JMLR. org (2017)
12. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
13. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(Jul), 2121–2159 (2011)

14. Elibol, M., Lei, L., Jordan, M.I.: Variance reduction with sparse gradients (2020)
15. Fang, B., Zeng, X., Zhang, M.: Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. pp. 115–127 (2018)
16. Glorot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: A deep learning approach. In: Proceedings of the 28th International Conference on Machine Learning (2011)
17. Greff, K., Srivastava, R.K., Schmidhuber, J.: Highway and residual networks learn unrolled iterative estimation. arXiv preprint arXiv:1612.07771 (2016)
18. Grothmann, T., Patt, A.: Adaptive capacity and human cognition: the process of individual adaptation to climate change. Global environmental change **15**(3), 199–213 (2005)
19. Habibzadeh, M., Jannesari, M., Rezaei, Z., Baharvand, H., Totonchi, M.: Automatic white blood cell classification using pre-trained deep learning models: Resnet and inception. In: Tenth International Conference on Machine Vision (ICMV 2017). vol. 10696, p. 1069612. International Society for Optics and Photonics (2018)
20. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European conference on computer vision. pp. 630–645. Springer (2016)
21. Hochreiter, S., Schmidhuber, J.: Flat minima. Neural Computation **9**(1), 1–42 (1997)
22. Hoffman, J., Roberts, D.A., Yaida, S.: Robust learning with jacobian regularization. arXiv preprint arXiv:1908.02729 (2019)
23. Hoffman, J., Rodner, E., Donahue, J., Darrell, T., Saenko, K.: Efficient learning of domain-invariant image representations. arXiv preprint arXiv:1301.3224 (2013)
24. Hou, L., Zhu, J., Kwok, J., Gao, F., Qin, T., Liu, T.y.: Normalization helps training of quantized lstm. In: Advances in Neural Information Processing Systems. pp. 7344–7354 (2019)
25. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: European conference on computer vision. pp. 646–661. Springer (2016)
26. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (Jun 2018). https://doi.org/10.1109/cvpr.2018.00286, http://dx.doi.org/10.1109/CVPR.2018.00286
27. Keshamoni, K., Hemanth, S.: Smart gas level monitoring, booking & gas leakage detector over iot. In: 2017 IEEE 7th International Advance Computing Conference (IACC). pp. 330–332. IEEE (2017)
28. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836 (2016)
29. Kikui, K., Itoh, Y., Yamada, M., Sugiura, Y., Sugimoto, M.: Intra-/inter-user adaptation framework for wearable gesture sensing device. In: Proceedings of the 2018 ACM International Symposium on Wearable Computers. pp. 21–24 (2018)
30. Kikui, K., Itoh, Y., Yamada, M., Sugiura, Y., Sugimoto, M.: Intra-/inter-user adaptation framework for wearable gesture sensing device. In: Proceedings of the 2018 ACM International Symposium on Wearable Computers. p. 21–24. ISWC '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3267242.3267256, https://doi.org/10.1145/3267242.3267256

31. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014)
32. Krizhevsky, A., et al.: Learning multiple layers of features from tiny images (2009)
33. Lane, N.D., Bhattacharya, S., Mathur, A., Georgiev, P., Forlivesi, C., Kawsar, F.: Squeezing deep learning into mobile and embedded devices. IEEE Pervasive Computing **16**(3), 82–88 (2017)
34. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1999)
35. Lee, J., Kim, J., Im, J., Lim, S., Kwon, J., Lee, S.M., Moon, S.: Mems-based no 2 gas sensor using zno nano-rods for low-power iot application. Journal of the Korean Physical Society **70**(10), 924–928 (2017)
36. Li, Y., Wei, C., Ma, T.: Towards explaining the regularization effect of initial large learning rate in training neural networks. arXiv preprint arXiv:1907.04595 (2019)
37. Lin, Y., Sakr, C., Kim, Y., Shanbhag, N.: Predictivenet: An energy-efficient convolutional neural network via zero prediction. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–4 (2017)
38. Liu, C.H., Fan, J., Branch, J.W., Leung, K.K.: Toward qoi and energy-efficiency in internet-of-things sensory environments. IEEE Transactions on Emerging Topics in Computing **2**(4), 473–487 (Dec 2014). https://doi.org/10.1109/TETC.2014.2364915
39. Liu, S., Lin, Y., Zhou, Z., Nan, K., Liu, H., Du, J.: On-demand deep model compression for mobile devices: A usage-driven model selection framework. In: Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services. pp. 389–400. ACM (2018)
40. 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`
41. Long, M., Zhu, H., Wang, J., Jordan, M.I.: Deep transfer learning with joint adaptation networks. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. p. 2208–2217. ICML'17, JMLR.org (2017)
42. Lv, K., Jiang, S., Li, J.: Learning gradient descent: Better generalization and longer horizons. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2247–2255. JMLR. org (2017)
43. Miotto, R., Wang, F., Wang, S., Jiang, X., Dudley, J.T.: Deep learning for healthcare: review, opportunities and challenges. Briefings in bioinformatics **19**(6), 1236–1246 (2018)
44. Moreno, M., Úbeda, B., Skarmeta, A.F., Zamora, M.A.: How can we tackle energy efficiency in iot basedsmart buildings? Sensors **14**(6), 9582–9614 (2014)
45. NVIDIA Inc.: NVIDIA Jetson TX2, `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/`, accessed 2019-09-01
46. Park, K., Yi, Y.: Bpnet: Branch-pruned conditional neural network for systematic time-accuracy tradeoff in dnn inference: work-in-progress. In: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion. pp. 1–2 (2019)
47. PARTICLE Inc.: PARTICLE LP103450, `https://store.particle.io/products/li-po-battery`, accessed 2020-02-29
48. 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
49. Peters, M., Ruder, S., Smith, N.A.: To tune or not to tune? adapting pretrained representations to diverse tasks. arXiv preprint arXiv:1903.05987 (2019)

50. Petrolo, R., Lin, Y., Knightly, E.: Astro: Autonomous, sensing, and tetherless networked drones. In: Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications. p. 1–6. DroNet'18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3213526.3213527, https://doi.org/10.1145/3213526.3213527
51. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contractive auto-encoders: Explicit invariance during feature extraction (2011)
52. Sainath, T.N., Vinyals, O., Senior, A., Sak, H.: Convolutional, long short-term memory, fully connected deep neural networks. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 4580–4584 (April 2015). https://doi.org/10.1109/ICASSP.2015.7178838
53. SAMSUNG Inc.: SAMSUNG Galaxy S20, https://www.samsung.com/global/galaxy/galaxy-s20/specs/, accessed 2020-02-29
54. Sannino, G., Pietro, G.D.: A deep learning approach for ecg-based heartbeat classification for arrhythmia detection. Future Generation Computer Systems **86**, 446 – 455 (2018). https://doi.org/https://doi.org/10.1016/j.future.2018.03.057, http://www.sciencedirect.com/science/article/pii/S0167739X17324548
55. Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Chandra, V., Esmaeilzadeh, H.: Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA) (Jun 2018). https://doi.org/10.1109/isca.2018.00069, http://dx.doi.org/10.1109/ISCA.2018.00069
56. Singh, K.J., Kapoor, D.S.: Create your own internet of things: A survey of iot platforms. IEEE Consumer Electronics Magazine **6**(2), 57–68 (April 2017). https://doi.org/10.1109/MCE.2016.2640718
57. Sokolić, J., Giryes, R., Sapiro, G., Rodrigues, M.R.: Robust large margin deep neural networks. IEEE Transactions on Signal Processing **65**(16), 4265–4280 (2017)
58. Subasi, A., Radhwan, M., Kurdi, R., Khateeb, K.: Iot based mobile healthcare system for human activity recognition. In: 2018 15th Learning and Technology Conference (L&T). pp. 29–34. IEEE (2018)
59. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A survey on deep transfer learning. In: International conference on artificial neural networks. pp. 270–279. Springer (2018)
60. Texas Instruments Inc.: INA3221 Triple-Channel, High-Side Measurement, Shunt and Bus Voltage Monitor, http://www.ti.com/product/INA3221, accessed 2019-11-21
61. Upton, E., Halfacree, G.: Raspberry Pi user guide. John Wiley & Sons (2014)
62. 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)
63. Wang, X., Yu, F., Dou, Z.Y., Darrell, T., Gonzalez, J.E.: Skipnet: Learning dynamic routing in convolutional networks. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 409–424 (2018)
64. Wang, Y., Jiang, Z., Chen, X., Xu, P., Zhao, Y., Lin, Y., Wang, Z.: E2-train: Training state-of-the-art cnns with over 80% energy savings. In: Advances in Neural Information Processing Systems. pp. 5139–5151 (2019)
65. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems. pp. 2074–2082 (2016)

66. 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)
67. Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L.S., Grauman, K., Feris, R.: Blockdrop: Dynamic inference paths in residual networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8817–8826 (2018)
68. 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)
69. Zhang, C., Bengio, S., Singer, Y.: Are all layers created equal? CoRR **abs/1902.01996** (2019)

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 [40,68]. 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. 8, 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.

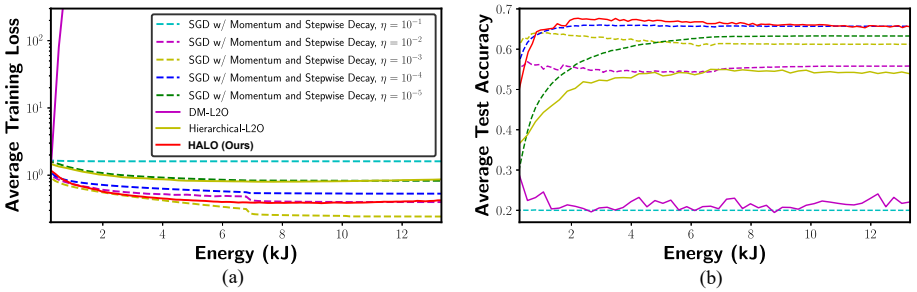

Fig. 8: **HALO for a wider optimizee over the baseline of SGD with momentum and stepwise learning rate decay**: (a) The average training loss and (b) adaptation/test accuracy vs. the energy cost over ten runs, on CIFAR-10-A.

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

Table 5: 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**		

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 5, traditional handcrafted optimizers cannot show competitive results even with the most carefully designed updating structural sparsity schemes (to our best 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 [45]) 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 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. 9, 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. 9) running time in terms of the average adaptation/test accuracy under the same running time budget.

D Ablation Studies of HALO's Jacobian Regularizer

We next presents an ablation study of HALO's hyperparameter λ_1 in the Jacobian regularizer, as introduced in Section 3.1, based on the wider optimizee (Fig. 2

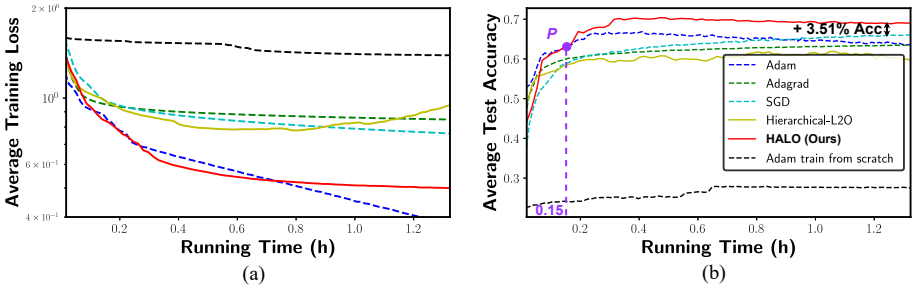

Fig. 9: **HALO for a wider and deeper optimizee over all the baseline optimizers**: (a) The average training loss and (b) adaptation/test accuracy vs. the **running time** over ten runs, on CIFAR-10-A.

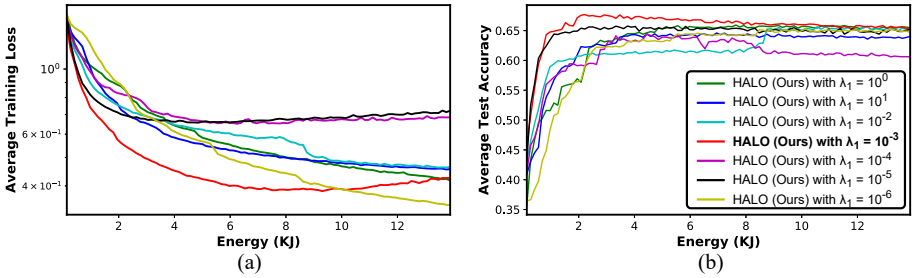

Fig. 10: **Ablation studies of HALO's Jacobian regularizer** in terms of the hyperparameter λ_1: the average training loss (a) and adaptation accuracy (b) vs. the required energy cost over ten runs, on CIFAR-10-A.

(b)) and CIFAR-10 dataset and using an updating probability schedule of "10%-30%-50%". As shown in Fig. 10, 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 [34,32,62,3]; 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 [66] 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 4.1) on the MNIST dataset; (2) a simple 2-layer CNN introduced in [66] and shown in Fig. 2 (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

Fig. 11: The energy/running-time measurement setup with a laptop, a state-of-the-art Edge GPU [45] and a popular IoT device [61]

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. 11 shows the energy/running-time measurement setup using an Egde GPU (i.e., NVIDIA JETSON TX2) [45] (to the right of the figure) or a Raspberry Pi for general IoT applications [61] (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 [48] of the embedded INA3221 [60] power rails

monitor. For the experiments with the Raspberry Pi, the setup is the same as 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 [66], $(m_{\theta^t}, \gamma_{\theta^t}, \eta_{\theta^t})$ 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 [66], $\bar{g}_{\theta^{t+1},s}$ 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.

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

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

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