International Journal of Machine Learning and Cybernetics (2022) 13:3785–3805
https://doi.org/10.1007/s13042-022-01625-4

ORIGINAL ARTICLE

Adaptive hierarchical hyper-gradient descent

Renlong Jie[1] · **Junbin Gao**[1] · **Andrey Vasnev**[1] · **Minh-Ngoc Tran**[1]

Received: 26 October 2021 / Accepted: 22 July 2022 / Published online: 13 August 2022

Abstract
Adaptive learning rate strategies can lead to faster convergence and better performance for deep learning models. There are some widely known human-designed adaptive optimizers such as Adam and RMSProp, gradient based adaptive methods such as hyper-descent and practical loss-based stepsize adaptation (L4), and meta learning approaches including learning to learn. However, the existing studies did not take into account the hierarchical structures of deep neural networks in designing the adaptation strategies. Meanwhile, the issue of balancing adaptiveness and convergence is still an open question to be answered. In this study, we investigate novel adaptive learning rate strategies at different levels based on the hyper-gradient descent framework and propose a method that adaptively learns the optimizer parameters by combining adaptive information at different levels. In addition, we show the relationship between regularizing over-parameterized learning rates and building combinations of adaptive learning rates at different levels. Moreover, two heuristics are introduced to guarantee the convergence of the proposed optimizers. The experiments on several network architectures, including feed-forward networks, LeNet-5 and ResNet-18/34, show that the proposed multi-level adaptive approach can significantly outperform many baseline adaptive methods in a variety of circumstances.

Keywords Deep learning · Hypergradient descent · Learning rate adaptation · Hierarchical learning rate system · Adabound

1 Introduction

Deep learning has become the most powerful technique in modern artificial intelligence systems, changing many aspects of our real life and improving the efficiency of a variety of industrial fields [20]. The successful training of deep neural networks requires carefully designed optimization algorithms. Not only because these algorithms can determine the model performance after training, but also they can determine the efficiency and convergence speed of training. With the wide-range application of large models with complex structures in recent years, training models can be expensive and time-consuming in practice, while the selection of optimizers could be of great importance [15, 21, 55].

The basic optimization algorithm for training deep neural networks is gradient descent method (GD), including stochastic gradient descent (SGD), mini-batch gradient descent and batch gradient descent [3]. Model parameters are updated according to the first-order gradients of the objective function with respect to the parameters being optimized, while back-propagation is implemented for calculating the gradients [24, 29, 46]. Traditionally, people consider learning rate as a global hyper-parameter to be tuned. However, with less or no adaptiveness, the training is difficult to converge and sensitive to the selection of learning rate. Training models with basic SGD algorithm usually requires a relatively long training time as well as carefully designed learning rate schedules [10, 18]. Rule-based adaptive optimizers such as Adagrad, RMSProp and Adam achieve faster convergence speed in many scenarios, but their adaptation power is limited by the corresponding pre-designed updating rules [12, 26, 53].

✉ Junbin Gao
junbin.gao@sydney.edu.au

Renlong Jie
renlong.jie@sydney.edu.au

Andrey Vasnev
andrey.vasnev@sydney.edu.au

Minh-Ngoc Tran
minh-ngoc.tran@sydney.edu.au

[1] Discipline of Business Analytics, The University of Sydney Business School, The University of Sydney, Camperdown, NSW 2006, Australia

Thankfully, auto-differentiation provides a technique for updating hyper-parameters with gradient-descent methods [7, 13]. This makes it possible for achieving learning rate adaptation beyond manually designed methods. One example is hyper-gradient descent [6], which introduced the global learning rate adaptation framework for gradient-based optimizers such as SGD and Adam. Their method is shown to be successful in improving the convergence speed for multiple optimizers, also it demonstrates that the way of using hyper-gradient gradient for learning rate adaptation is a promising technique for improving existing optimizers. However, their study did not further investigate the detailed structures of parameters and corresponding learning rates adaptation techniques in complex neural network architectures. This actually limits the potential of auto-differentiation for learning rate adaptation. As is known, deep neural networks are composed of different levels of components such as blocks, layers and parameters, and it is reasonable to assume each component of the model is in favor of a specific learning rate in training. Thus, considering the detailed architectures of networks and exploring hyper-gradient descent for structured learning rate adaptation is a topic of interest and importance.

In this study, we propose a novel family of adaptive optimization algorithms based on the framework of hyper-gradient descent. By considering the regular hierarchical structures of deep neural networks, we introduce hierarchical learning rate structures correspondingly, which enables flexible and controllable learning rate adaptation. Meanwhile, to make the most of the gradient information from the training process, we apply both hyper-gradient descent method in multi-levels and the gradient-based updating of the combination weights of different levels. The main contribution of our study can be summarized as the following four points:

- We introduce hierarchical learning rate structures for neural networks and apply hyper-gradient descent to obtain adaptive learning rates at different levels.
- We introduce a set of regularization techniques for learning rates to address the balance of global and local adaptations and show the relationship with weighted combinations.
- We propose an algorithm implementing trainable weighted combination of adaptive learning rates at multiple levels for model parameter updating.
- Two techniques including weighted approximation and clipping are introduced to guarantee the convergence of the proposed optimization methods in training.
- The experiments demonstrate that the proposed adaptation method can improve the performance of corresponding baseline optimizers in a variety of tasks with statistical significance.

The paper is organised as follows: Sect. 2 is a literature review of the related adaptive optimization algorithms in deep learning. Section 3 introduces the main idea of hyper-gradient descent algorithm. Section 4 is a detailed explanation of the proposed multi-level adaptation methods as well as a discussion on their convergence properties. Section 5 is the main experimental part that compares the proposed algorithm with a set of baselines on several benchmark datasets. In Sect. 6 we provide a further discussion on the hyper-parameter settings, the learning behavior of combination weights, time and space complexity, etc. Section 7 is the conclusion of the whole paper.

2 Literature review

Naïve gradient descent methods apply fixed learning rates without any adaptation mechanism. However, considering the change of available information during the learning process, SGD with fixed learning rates can result in slow convergence speed and requires a relatively large amount of computing resources in hyper-parameter searching. One solution is to introduce a learning rate adaptation rule, where "adaptation" means that the global or local learning rates or effective step sizes can be refined continuously during the training process in response to the change of inputs or other parameters. This idea can be traced back to the work on gain adaptation for connectionist learning methods [51] and related extensions for non-linear cases [48, 59]. In recent years, optimizers with adaptive updating rules were developed in the context of deep learning, while the hyper-parameter learning rates are still fixed in training. The proposed methods include AdaGrad [12], Adadelta [61], RMSProp [53], and Adam [26]. In these methods, pre-designed updating rules provide adaptive step-sizes for parameter updating. The most widely used one, Adam, is shown to be quite effective in speeding up the training, which computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

Convergence is an essential property for optimization algorithms. There are many optimizers aiming to address the convergence issue in Adam. For example, it is noticed that Adam does not converge to the optimal solution for some stochastic convex optimization problems, while AMSGrad is introduced as a substitute with a convergence garantee [43]. Adabound further applies dynamic bound for gradient methods and build a gradual transition between adaptive approach and SGD [36]. RAdam was proposed to rectify the variance of the adaptive learning rate [34]. Adabelief optimizer [64] can achieve fast convergence, good generalization and training stability by adapting the stepsize according to the "belief" in the current gradient direction. Other techniques, such as *Lookahead*, can also achieve variance reduction and stability improvement with negligible extra computational cost [62]. Some analysis of adaptive

optimizers in nonconvex stochastic optimization problems are provided in [60], which discovered that increasing mini-batch sizes could circumvent the nonconvergence issues. In fact, through recent years, more studies with solid theoretical analysis are providing novel techniques and analyzing frameworks for the convergence of adaptive optimizers [1, 9, 33, 54]. Moreover, to address the issue of large memory overheads for adaptive methods, memory-efficient adaptive optimization is developed, which could retains the benefits of standard per-parameter adaptivity [5].

Even though the adaptive optimizers with designed updating rules can converge faster than SGD in a wide range of tasks, the gradient information obtained during the training is not applied, while more hyper-parameters are introduced. Another idea is to use objective function information and update the learning rates as trainable parameters. These methods were introduced as automatic differentiation, where the hyper-parameters can be optimized with backpropagation [7, 38]. As gradient-based hyper-parameter optimization methods, they can be implemented as an online approach [16]. With the idea of auto-differentiation, learning rates can be updated in real-time with the corresponding derivatives of the empirical risk [2], which can be generated to all types of optimizers for deep neural networks [6]. Another step size adaptation approach called "L4", is based on the linearized expansion of the objective functions, which rescales the gradient to make fixed predicted progress on the loss [45]. Meanwhile, layer-wise adaptation methods are also shown to be effective in accelerating the speed of large-batch training, which has been successfully applied in training large models or large datasets [57, 58].

Another set of approaches train an RNN (recurrent neural network) agent to generate the optimal learning rates in the next step given the historical training information, which is known as "learning to learn" [4]. It empirically outperforms hand-designed optimizers in a variety of learning tasks, but another study shows that it may not be effective for long horizon [37]. The generalization ability can be improved by using meta training samples and hierachical LSTMs (Long Short-Term Memory) [56]. Still there are studies focusing on incorporating domain knowledge with LSTM-based optimizers to improve the performance in terms of efficacy and efficiency [17].

Beyond the adaptive learning rate, learning rate schedules can also improve the convergence of optimizers, including time-based decay, step decay, exponential decay [32]. The most fundamental and widely applied one is a piecewise step-decay learning rate schedule, which could vastly improve the convergence of SGD and even adaptive optimizers [34, 36]. It can be further improved by introducing a statistical test to determine when to apply step-decay [28, 63]. Also, there are works on warm-restart [35, 41], which could improve the performance of SGD anytime when training deep neural networks.

The limitations of existing optimization algorithms are mainly in the following two aspects: (a) The existing gradient or model-based learning rate adaptation methods including hyper-gradient descent, L4 and learning to learn only focus on global adaptation. Meanwhile, some studies [57, 58] show that updating rules with layer-wise adaptation is a promising technique for improving the convergence speed. Therefore, it is necessary to try to further extended adaptive optimizers to multi-level cases. (b) In the framework of hyper-gradient descent, no constraints or prior knowledge for learning rates are introduced, which limits its potential in balancing the local and global adaptiveness.

To tackle these limitations, our proposed algorithm is based on hyper-gradient descent but further introduce locally shared adaptive learning rates such as layer-wise, unit-wise and parameter-wise learning rates adaptation. Meanwhile, we introduce a set of regularization techniques for learning rates in order to balance the global and local adaptations.

3 Hyper-gradient descent

This section is dedicated to reviewing the auto-differentiation and hyper-gradient descent with detailed explanation and math formulas. The study of hyper-gradient descent in [6] provides a re-discovery the work of [2], in which the gradient with respect to the learning rate is calculated by using the updating rule of the model parameters in the last iteration. The gradient descent updating rule for model parameter θ can is given by Eq. (1):

$$\theta_t = \theta_{t-1} - \alpha \nabla f(\theta_{t-1}). \tag{1}$$

Note that $\theta_{t-1} = \theta_{t-2} - \alpha \nabla f(\theta_{t-2})$, the gradient of objective function with respect to the learning rate can then by calculated:

$$\frac{\partial f(\theta_{t-1})}{\partial \alpha} = \nabla f(\theta_{t-1}) \cdot \frac{\partial(\theta_{t-2} - \alpha \nabla f(\theta_{t-2}))}{\partial \alpha} = \nabla f(\theta_{t-1}) \cdot (-\nabla f(\theta_{t-2})).$$

A whole learning rate updating rule can be written as:

$$\alpha_t = \alpha_{t-1} - \beta \frac{\partial f(\theta_{t-1})}{\partial \alpha} = \alpha_{t-1} + \beta \nabla f(\theta_{t-1}) \cdot \nabla f(\theta_{t-2}).$$

In a more general prospective, assume that we have an updating rule for model parameters $\theta_t = u(\Theta_{t-1}, \alpha_t)$. We need to update the value of α_t towards the optimum value α_t^* that minimizes the expected value of the objective in the next iteration. The corresponding gradient can be written as:

$$\frac{\partial \mathbb{E}[f(\theta_t)]}{\partial \alpha_t} = \frac{\partial \mathbb{E}[f \circ u(\Theta_{t-1}, \alpha_t)]}{\partial \alpha_t} = \mathbb{E}[\nabla_\theta f(\theta_t)^T \nabla_\alpha u(\Theta_{t-1}, \alpha_t)],$$

where $u(\Theta_{t-1}, \alpha_t)$ denotes the updating rule of a gradient descent method. Then the additive updating rule of learning rate α_t can be written as:

$$\alpha_t = \alpha_{t-1} - \beta \tilde{\nabla}_\theta f(\theta_{t-1})^T \nabla_\alpha u(\Theta_{t-2}, \alpha_{t-1}),$$

where $\tilde{\nabla}_\theta f(\theta_t)$ is the noisy estimator of $\nabla_\theta f(\theta_t)$. On the other hand, the multiplicative rule is given by:

$$\alpha_t = \alpha_{t-1}\left(1 - \beta' \frac{\tilde{\nabla}_\theta f(\theta_{t-1})^T \nabla_\alpha u(\Theta_{t-2}, \alpha_{t-1})}{\left\|\tilde{\nabla}_\theta f(\theta_{t-1})\right\| \left\|\nabla_\alpha u(\Theta_{t-2}, \alpha_{t-1})\right\|}\right).$$

These two types of updating rules can be implemented in any optimizers including SGD and Adam, denoted by corresponding $\theta_t = u(\Theta_{t-1}, \alpha_t)$.

4 Multi-level adaptation methods

Deep neural networks are composed of components in different levels, including blocks, layers, cell, neurons and parameters. It is natural to consider different parts in the hierarchical architectures are in favor of different learning rate in training. Based on hyper-gradient descent discussed in Sect. 3, and further motivated by the success of layer-wise adaptation methods in [57, 58], we further consider the hierarchical structures of neural networks and introduce learning rate adaptation in different levels.

4.1 Layer-wise, unit-wise and parameter-wise adaptation

In the paper of hyper-descent [6], the learning rate is set to be a scalar. However, to make the most of learning rate adaptation for deep neural networks, we introduce updating rules in different levels of the network architectures, including layer-wise or even parameter-wise updating rules, where the learning rate $\boldsymbol{\alpha}_t$ in each time step is considered to be a vector (layer-wise) or even a list of matrices (parameter-wise). For the sake of simplicity, we collect all the learning rates in a vector: $\boldsymbol{\alpha}_t = (\alpha_1, \ldots, \alpha_N)^T$. Correspondingly, the objective $f(\boldsymbol{\theta})$ is a function of $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_N)^T$, collecting all the model parameters. In this case, the derivative of the objective function f with respect to each learning rate can be written as:

$$\begin{aligned}\frac{\partial f(\boldsymbol{\theta}_{t-1})}{\partial \alpha_{i,t-1}} &= \frac{\partial f(\theta_{1,t-1}, \ldots, \theta_{i,t-1}, \ldots, \theta_{n,t-1})}{\partial \alpha_{i,t-1}} \\ &= \sum_{j=1}^{N} \frac{\partial f(\theta_{1,t-1}, \ldots, \theta_{i,t-1}, \ldots, \theta_{n,t-1})}{\partial \theta_{j,t-1}} \frac{\partial \theta_{j,t-1}}{\partial \alpha_{i,t-1}},\end{aligned} \tag{2}$$

where N is the total number of all the model parameters. Eq. (2) can be generalized to group-wise updating, where we associate a learning rate with a special group of parameters, and each parameter group is updated according to its only learning rate. Assume $\boldsymbol{\theta}_t = u(\boldsymbol{\Theta}_{t-1}, \alpha)$ is the updating rule, where $\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_s\}_{s=0}^t$ and α is the learning rate, then the basic gradient descent method for each group i gives $\boldsymbol{\theta}_{i,t} = u(\boldsymbol{\Theta}_{t-1}, \alpha_{i,t-1}) = \boldsymbol{\theta}_{i,t-1} - \alpha_{i,t-1}\nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}_{t-1})$. Hence for gradient descent,

$$\frac{\partial f(\boldsymbol{\theta}_{t-1})}{\partial \alpha_{i,t-1}} = \nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}_{t-1})^T \nabla_{\alpha_{i,t-1}} u(\boldsymbol{\Theta}_{t-1}, \alpha_t) = -\nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}_{t-1})^T \nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}_{t-2}).$$

Here $\alpha_{i,t-1}$ is a scalar with index i at time step $t-1$, corresponding to the learning rate of the ith group, while the shape of $\nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta})$ is the same as the shape of $\boldsymbol{\theta}_i$.

We particularly consider three special cases: (1) In **layer-wise adaptation**, $\boldsymbol{\theta}_i$ is the weight matrix of ith layer, and α_i is the particular learning rate for this layer. (2) In **parameter-wise adaptation**, θ_i corresponds to a certain parameter involved in the model, which can be an element of the weight matrix in a certain layer. (3) We can also introduce **unit-wise adaptation**, where $\boldsymbol{\theta}_i$ is the weight vector connected to a certain neuron, corresponding to a column or a row of the weight matrix depending on whether it is the input or the output weight vector to the neuron concerned. [6] mentioned the case where the learning rate can be considered as a vector, which corresponds to layer-wise adaptation in this paper.

4.2 Regularization on learning rate

For the model involving a large number of learning rates for different groups of parameters, the updating for each learning rate only depends on a small number of examples. Therefore, when the batch size is also not large, the updating will involve a lot of noise due to applying small random samples, while over-parameterization is an issue to be concerned.

To address this issue, our original idea is to apply regularization on learning rates, where we introduce prior knowledge and assume that the low level adaptive learning rates (e.g. parameter-wise learning rates) are distributed around the high-level ones (e.g. global learning rates). Different regularization terms can be implemented to control the flexibility of learning rate adaptation and achieve the effect of variance reduction. First, for layer-wise adaptation, we can add the following regularization term to the cost function:

$$L_{\text{lr\_reg\_layer}} = \lambda_{\text{layer}} \sum_l (\alpha_l - \alpha_g)^2,$$

where l is the indices for each layer, λ_{layer} is the layer-wise regularization coefficient, α_l and α_g are the layer-wise and global-wise adaptive learning rates. A large λ_{layer} can push the learning rate of each layer towards the average learning rate across all the layers. In the extreme case, this will lead

to very similar learning rates for all layers, and the algorithm will be reduced to that in [6].

In addition, we can also consider the case where three levels of learning rate adaptations are involved, including global-wise, layer-wise and parameter-wise adaptation. If we introduce two more regularization terms to control the variation of parameter-wise learning rate with respect to layer-wise learning rate and global learning rates, the regularization loss can be written as:

$$L_{\text{lr\_reg\_para}} = \lambda_{\text{layer}} \sum_{l} (\alpha_l - \alpha_g)^2 + \lambda_{\text{para}} \sum_{l} \sum_{p} (\alpha_{pl} - \alpha_g)^2 + \lambda_{\text{para\_layer}} \sum_{l} \sum_{p} (\alpha_{pl} - \alpha_l)^2 \tag{3}$$

where p represents the index of each parameter within each layer. The second and third terms are the regularization terms pushing each parameter-wise learning rate towards the layer-wise learning rate, and the term of pushing the parameter-wise learning rate towards the global learning rates, while $\lambda_{\text{para\_layer}}$ and $\lambda_{\text{para\_layer}}$ are the corresponding regularization coefficients.

With these regularisation terms, the flexibility and variances of learning rates at different levels can be neatly controlled, while it can reduce to the basement case where a single learning rate for the whole model is used. In addition, there could still be one more regularization for improving the stability across different time steps, which can be used in the original hyper-descent algorithm where the learning rate in each time step is a scalar:

$$L_{\text{lr\_reg\_ts}} = \lambda_{\text{ts}} (\alpha_{g,t} - \alpha_{g,t-1})^2,$$

where λ_{ts} is the regularization coefficient to control the difference of learning rates between current step and the last step. With this term, the model with learning rate adaptation will be close to the model with fixed learning rate as large regularization coefficients are used. Thus, we can write the objective function of the full model as:

$$L_{\text{full}} = L_{\text{model}} + L_{\text{model\_reg}} + L_{\text{lr\_reg}} + L_{\text{lr\_reg\_ts}},$$

where L_{model} and $L_{\text{model\_reg}}$ are the loss and regularization cost of basement model. $L_{\text{lr\_reg}}$ can be any among $L_{\text{lr\_reg\_layer}}$, $L_{\text{lr\_reg\_unit}}$ and $L_{\text{lr\_reg\_para}}$ depending on the specific requirement of the learning task, while the corresponding regularization coefficients can be optimized with random search for several extra dimensions.

4.3 Updating rules for learning rates

Considering these regularisation terms and take layer-wise adaptation for example, the gradient of the cost function with respect to a specific learning rate α_l in layer l can be written as:

$$\begin{aligned}\frac{\partial L_{\text{full}}(\theta, \alpha)}{\partial \alpha_{l,t}} &= \frac{\partial L_{\text{model}}(\theta, \alpha)}{\partial \alpha_{l,t}} + \frac{\partial L_{\text{lr\_reg}}(\theta, \alpha)}{\partial \alpha_{l,t}} \\ &= \tilde{\nabla}_{\theta_l} f(\theta_{t-1}) \nabla_{\alpha_{l,t-1}} u(\Theta_{t-2}, \alpha_{t-1}) + 2\lambda_{\text{layer}}(\alpha_{l,t} - \alpha_{g,t}),\end{aligned}$$

with the corresponding updating rule by naïve gradient descent:

$$\alpha_{l,t} = \alpha_{l,t-1} - \beta \frac{\partial L_{\text{full}}}{\partial \alpha_{l,t-1}}. \tag{4}$$

The updating rule for other types of adaptation can be derived accordingly. Notice that the time step index of layer-wise regularization term is t rather than $t-1$, which ensures that we push the layer-wise learning rates towards the corresponding global learning rates of the current step. If we assume

$$h_{l,t-1} = -\tilde{\nabla}_{\theta_l} f(\theta_{t-1}) \nabla_{\alpha_{l,t-1}} u(\Theta_{t-2}, \alpha_{l,t-1}),$$

then Eq. (4) can be written as:

$$\alpha_{l,t} = \alpha_{l,t-1} - \beta(-h_{l,t-1} + 2\lambda_{\text{layer}}(\alpha_{l,t} - \alpha_{g,t})). \tag{5}$$

In Eq. (5), both sides include the term of $\alpha_{l,t}$, while the natural way to handle this is to solve for the close form of α_t, which gives:

$$\alpha_{l,t} = \frac{1}{1 + 2\beta\lambda_{\text{layer}}} [\alpha_{l,t-1} + \beta(h_{l,t-1} + 2\lambda_{\text{layer}} \alpha_{g,t})]. \tag{6}$$

Equation (6) gives a close form solution but only applicable in the two-levels case. When there are more levels, components of learning rates at different levels can be interdependent. Meanwhile, there is an extra hyper-parameter λ_{layer} to be tuned. To construct a workable updating scheme for Eq. (6), we replace $\alpha_{l,t}$ and $\alpha_{g,t}$ with their relevant approximations. We take the strategy of using their updated version without considering regularization, i.e.,

$$\hat{\alpha}_{l,t} = \alpha_{l,t-1} + \beta h_{l,t-1}, \quad \hat{\alpha}_{g,t} = \alpha_{g,t-1} + \beta h_{g,t-1} \tag{7}$$

where $h_{g,t-1} = -\tilde{\nabla}_{\theta} f(\theta_{t-1}) \nabla_{\alpha_{g,t-1}} u(\Theta_{t-2}, \alpha_{g,t-1})$ is the global h for all parameters. We define $\hat{\alpha}_{l,t}$ and $\hat{\alpha}_{g,t}$ as the "virtual" layer-wise and global-wise learning rates, where "virtual" means they are calculated based on the equation without regularization, and we do not use them directly for model parameter updating. Instead, we only use them as intermediate variables for calculating the real layer-wise learning rate for model training.

$$\begin{aligned}\alpha^*_{l,t} &= \alpha_{l,t-1} + \beta h_{l,t-1} - 2\beta\lambda_{\text{layer}}(\hat{\alpha}_{l,t} - \hat{\alpha}_{g,t}) \\ &= (1 - 2\beta\lambda_{\text{layer}})\hat{\alpha}_{l,t} + 2\beta\lambda_{\text{layer}}\hat{\alpha}_{g,t}.\end{aligned} \tag{8}$$

Notice that in Eq. (8), the first two terms is actually a weighted average of the layer-wise learning rate $\hat{\alpha}_{l,t}$ and global learning rate $\hat{\bar{\alpha}}_{l,t}$ at the current time step. Since we hope to push the layer-wise learning rates towards the global one, the parameters should meet the constraint: $0 < 2\beta\lambda_{layer} < 1$, and thus they can be optimized using hyper-parameter searching within a bounded interval. Moreover, gradient-based optimization on these hyper-parameters can also be applied. Hence both the layer-wise learning rates and the combination proportion of the local and global information can be learned with back propagation. This can be done in online or mini-batch settings. The advantage is that the learning process may be in favor of taking more account of global information in some periods, and taking more local information in some other periods to achieve the best learning performance, which is not taken into consideration by existing learning adaptation approaches.

Now consider the difference between Eqs. (5) and (8):

$$\alpha^*_{l,t} - \alpha_{l,t} = -2\beta\lambda_{\text{layer}}((\hat{\alpha}_{l,t} - \hat{\alpha}_{g,t}) - (\alpha_{l,t} - \alpha_{g,t})). \tag{9}$$

Based on the setting of multi-level adaptation, on the right-hand side of Eq. (9), global learning rate is updated without regularization $\hat{\alpha}_{g,t} = \alpha_{g,t}$. For the layer-wise learning rates, the difference is given by $\hat{\alpha}_{l,t} - \alpha_{l,t} = 2\beta\lambda_{layer}(\alpha_{l,t} - \alpha_{g,t})$, which corresponds to the gradient with respect to the regularization term. Thus, Eq. (9) can be rewritten as:

$$\alpha^*_{l,t} - \alpha_{l,t} = -2\beta\lambda_{\text{layer}}(2\beta\lambda_{\text{layer}}(\alpha_{l,t} - \alpha_{g,t})) = -4\beta^2\lambda^2_{\text{layer}}\left(1 - \frac{\alpha_{g,t}}{\alpha_{l,t}}\right)\alpha_{l,t}$$

which is the error of the virtual approximation introduced in Eq. (7). If $4\beta^2\lambda^2_{\text{layer}} << 1$ or $\frac{\alpha_{g,t}}{\alpha_{l,t}} \to 1$, this approximation becomes more accurate.

Another way for handling Eq. (5) is to use the learning rates for the last step in the regularization term.

$$\alpha_{l,t} \approx \alpha_{l,t-1} - \beta(-h_{l,t-1} + 2\lambda_{\text{layer}}(\alpha_{l,t-1} - \alpha_{g,t-1})).$$

Since we have $\alpha_{l,t} = \hat{\alpha}_{l,t} - 2\beta\lambda_{\text{layer}}(\alpha_{l,t} - \alpha_{g,t})$ and $\hat{\alpha}_{l,t} = \alpha_{l,t-1} + \beta h_{l,t-1}$, using the learning rates in the last step for regularization will introduce a higher variation from term $\beta h_{l,t-1}$, with respect to the true learning rates in the current step. Thus, we consider the proposed virtual approximation works better than last-step approximation.

Similar to the two-level's case, for the three-level regularization shown in Eq. (3), we have:

$$\begin{aligned}\frac{\partial L_{\text{full}}(\theta,\alpha)}{\partial \alpha_{p,t}} &= \frac{\partial L_{\text{model}}(\theta,\alpha)}{\partial \alpha_{p,t}} + \frac{\partial L_{\text{lr\_reg}}(\alpha)}{\partial \alpha_{p,t}} \\ &= -\tilde{\nabla}_{\theta_l} f(\theta_{t-1}) \nabla_{\theta_l} u(\Theta_{t-2}, \alpha_{t-1}) \\ &\quad + 2\lambda_2(\alpha_{p,t} - \alpha_{g,t}) + 2\lambda_3(\alpha_{p,t} - \alpha_{l,t})\end{aligned}$$

For the sake of simple derivation, we denote $\lambda_2 = \lambda_{\text{layer}}$, and $\lambda_3 = \lambda_{\text{para\_layer}}$ for the regularization parameters in Eq. (3). The updating rule can be written as:

$$\begin{aligned}\alpha_{p,t} &= \alpha_{p,t-1} - \beta(h_p + 2\lambda_2(\alpha_{p,t} - \alpha_{g,t}) + 2\lambda_3(\alpha_{p,t} - \alpha_{l,t})) \\ &\approx \hat{\alpha}_{p,t}(1 - 2\beta\lambda_2 - 2\beta\lambda_3) + 2\hat{\alpha}_{l,t}\beta\lambda_3 + 2\hat{\alpha}_{g,t}\beta\lambda_2\end{aligned}$$

where we assume that $\hat{\alpha}_{p,t}, \hat{\alpha}_{l,t}, \hat{\alpha}_{g,t}$ are independent variables. Define

$$\gamma_1 = 1 - 2\beta\lambda_2 - 2\beta\lambda_3,\ \gamma_2 = 2\beta\lambda_3,\ \gamma_3 = 2\beta\lambda_2,$$

we still have $\alpha = \gamma_1\alpha_p + \gamma_2\alpha_l + \gamma_3\alpha_g$ with $\gamma_1 + \gamma_2 + \gamma_3 = 1$.

Therefore, in the case of three level learning rates adaptation, the regularization effect can still be considered as applying the weighted combination of different levels of learning rates. This conclusion is invariant of the signs in the absolute operators in Eq. (7). In general, we can organize all the learning rates in a tree structure. For example, in three level case above, α_g will be the root node, while $\{\alpha_l\}$ are the children node at level 1 of the tree and $\{\alpha_{lp}\}$ are the children node of α_l as leave nodes at level three of the tree. In a general case, we assume there are L levels in the tree. Denote the set of all the paths from the root node to each of leave nodes as $\mathcal{P}$ and a path is denoted by $p = \{\alpha_1, \alpha_2, \ldots, \alpha_L\}$ where α_1 is the root node and α_L is the leave node on the path. On this path, denote ancestors(i) all the acenstor nodes of α_i along the path, i.e., ancestors(i) $= \{\alpha_1, \ldots, \alpha_{i-1}\}$. We will construct a regularizer to push α_i towards each of its parents. Then the regularization can be written as

$$L_{\text{lr\_reg}} = \sum_{p\in\mathcal{P}} \sum_{\alpha_i \in p} \sum_{\alpha_j \in \text{acenstors}(i)} \lambda_{ij}(\alpha_i - \alpha_j)^2.$$

Under this pair-wise L_2 regularization, the updating rule for any leave node learning rate α_L can be given by the following theorem

Theorem 1 *Under virtual approximation, effect of adding pair-wise L_2 regularization on different levels of adaptive learning rates $L_{\text{reg}} = \sum_i^n \sum_{j<i}^n \lambda_{ij}\|\alpha_i - \alpha_j\|_2^2$ is equal to performing a weighted linear combination of virtual learning rates in different levels $\alpha^* = \sum_i^n \gamma_i\alpha_i$ with $\sum_i^n \gamma_i = 1$, where each component α_i is calculated by assuming there is no regularization.*

Remarks: Theorem 1 actually suggests that the similar updating rule can be obtained for the learning rate at the any level on the path. All these have been demonstrated in Algorithm 1 for the three level case.

Proof Consider the learning regularizer

$$L_{\text{lr\_reg}}(\alpha) = \sum_{p\in\mathcal{P}}\sum_{\alpha_i\in p}\sum_{\alpha_j\in\text{parents}(i)} \lambda_{ij}(\alpha_i - \alpha_j)^2. \tag{10}$$

To apply hyper-gradient descent method to update the learning rate α_L at level L, we need to work the derivative of $L_{\text{lr\_reg}}$ with respect to α_L, the terms in (10) involving α_L are only $(\alpha_i - \alpha_j)^2$ where α_j is an ancestor on the path from the root to the leave node α_L. Hence

$$\begin{aligned}\frac{\partial L_{\text{full}}(\boldsymbol{\theta},\alpha)}{\partial \alpha_{L,t}} &= \frac{\partial L_{\text{model}}(\boldsymbol{\theta},\alpha)}{\partial \alpha_{L,t}} + \frac{\partial L_{\text{lr\_reg}}(\alpha)}{\partial \alpha_{L,t}} \\ &= -\tilde{\nabla}_{\theta_L} f(\boldsymbol{\theta}_{t-1})^T \nabla_{\theta_L} u(\Theta_{t-2}, \alpha_{t-1}) \\ &\quad + \sum_{\alpha_j\in\text{acenstors}(L)} 2\lambda_{Lj}(\alpha_{L,t} - \alpha_{j,t}).\end{aligned}$$

As there are exactly $L-1$ ancestors on the path, we can simply use the index $j = 1, 2, \ldots, L-1$. The corresponding updating function for $\alpha_{n,t}$ is:

$$\begin{aligned}\alpha_{L,t} &= \alpha_{n,t-1} - \beta\left(h_L + \sum_{j=1}^{L-1} 2\lambda_{Lj}(\alpha_{L,t} - \alpha_{j,t})\right) \\ &\approx \hat{\alpha}_{L,t}\left(1 - 2\beta\sum_{j=1}^{L-1}\lambda_{Lj}\alpha_{n,t}) + \sum_{j=1}^{L-1}(2\beta\lambda_{Lj}\hat{\alpha}_{j,t})\right) = \sum_{j=1}^{L}\gamma_j\hat{\alpha}_{j,t}.\end{aligned}$$

where

$$\gamma_L = 1 - 2\beta\sum_{j=1}^{L-1}\lambda_{Lj}, \quad \gamma_j = 2\beta\lambda_{Lj}, \quad \text{for } j = 1, 2, \ldots, L-1.$$

This form satisfies $\alpha_L^* = \sum_{j=1}^{L}\gamma_j\hat{\alpha}_j$ with $\sum_{j=1}^{L}\gamma_j = 1$. This completes the proof. □

Therefore, by applying weighted linear combination of virtual learning rates in different levels as the effective learning rate for parameter updating, the effect of adding regularization on adaptive learning rates in Sect. 4.2 can be approximately achieved. The approximation error can be controlled by fixed parameters. This demonstrates that we can use a more convenient combination form to update the effective learning rates on leaves of the hierarchical model structures. Moreover, the combination form can be extended to the case of many levels.

4.4 Prospective of learning rate combination

Motivated by the analytical derivation and corresponding discussion in Sect. 4.3, we can consider the combination of adaptive learning rates in different levels as a substitute of regularization on the differences of learning rates. As a simple case, the combination of global-wise and layer-wise adaptive learning rates can be written as $\alpha_t = \gamma_1\hat{\alpha}_{l,t} + \gamma_2\hat{\alpha}_{g,t}$, where $\gamma_1 + \gamma_2 = 1$ and $\gamma_1 \geq 0$, $\gamma_2 \geq 0$. In a general form, assume that we have n levels, which could include global-level, layer-level, unit-level and parameter-level, etc, we have:

$$\alpha_t = \sum_{i=1}^{n}\gamma_i\hat{\alpha}_{i,t}. \tag{11}$$

In a more general form, we can implement non-linear models such as neural networks to model the final adaptive learning rates with respect of the learning rates in different levels. Then the function is given by

$$\alpha_t = g(\hat{\alpha}_{1,t}, \hat{\alpha}_{2,t} \cdots \hat{\alpha}_{n,t}; \theta)$$

where θ is the vector of parameters of the non-linear model. In this study, we treat the combination weights $\{\gamma_1, \ldots, \gamma_n\}$ as trainable parameters as demonstrated in Eq. (11). Figure 1 gives an illustration of the linear combination of three-level hierarchical learning rates.

In fact, we only need these different levels of learning rate have a hierarchical relationship, which means the selection of component levels is not fixed. For example, in feed-forward neural networks, we can use parameter level, unit-level, layer level and global level. For recurrent neural networks, the corresponding layer level can either be the "layer of gate" within the cell structure such as LSTM and GRU, or the whole cell in a particular RNN layer. Especially, by "layer of gate" we mean the parameters in each gate of a cell structure share a same learning rate. Meanwhile, for convolutional neural network, we can further introduce "filter level" to replace layer-level if there is no clear layer structure, where the parameters in each filter will share a same learning rate.

As the real learning rates implemented in model parameter updating is a weighted combination, the corresponding Hessian matrices cannot be directly used for learning rate updating. If we take the gradients of the loss with respect to the combined learning rates, and use this to update the learning rate for each parameter, the procedure will be reduced to parameter-wise learning rate updating. To address this issue, we first break down the gradient by the combined learning rate to three levels, use each of them to update the learning rate at each level, and then calculate the combination by the updated learning rates. Especially, $h_{p,t}$, $h_{l,t}$ and $h(g, t)$ are calculated by the gradients of model losses without regularization, as is shown in Eq. (12).

$$
\begin{aligned}
h_{p,t} &= \frac{\partial f(\boldsymbol{\theta},\alpha)}{\partial \alpha_{p,t}} = -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1},\alpha)|_p \cdot \nabla_{\alpha} u(\boldsymbol{\Theta}_{t-2},\alpha)|_p,\\
h_{l,t} &= \frac{\partial f(\boldsymbol{\theta},\alpha)}{\partial \alpha_{l,t}} = -\mathrm{tr}(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1},\alpha)|_l^T \nabla_{\alpha} u(\boldsymbol{\Theta}_{t-2},\alpha)|_l),\\
h_{g,t} &= \frac{\partial f(\boldsymbol{\theta},\alpha)}{\partial \alpha_{t}} = -\sum_{l=1}^{n} \mathrm{tr}(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1},\alpha)|_l^T \nabla_{\alpha} u(\boldsymbol{\Theta}_{t-2},\alpha)_l),
\end{aligned}
\tag{12}
$$

where $h_t = \sum_l h_{l,t} = \sum_p h_{p,t}$ and $h_{l,t} = \sum_{p \in l\text{th layer}} h_p$ and $f(\theta,\alpha)$ corresponds to the model loss $L_{model}(\theta,\alpha)$ in Sect. 4.2.

Algorithm 1 is the full updating rules for the newly proposed optimizer with three levels, which can be denoted as Combined Adaptive Multi-level Hyper-gradient Descent (CAM-HD).

Algorithm 1: Updating rule of three-level CAM-HD

input: α_0, β, δ, T
initialization: θ_0, $\gamma_{1,0}$, $\gamma_{2,0}$, $\gamma_{3,0}$, $\alpha_{p,0}$, $\alpha_{l,0}$,
$\alpha_0, \alpha^*_{l,0} = \gamma_{1,0}\alpha_{p,0} + \gamma_{2,0}\alpha_{l,0} + \gamma_{3,0}\alpha_0$
for $t \in 1, 2, ..., T$ **do**
$g_t = \nabla_\theta f(\theta,\alpha)$
$h_{p,t} = \frac{\partial f(\theta,\alpha)}{\partial \alpha_{p,t}} = -\nabla_\theta f(\theta_{t-1},\alpha)|_p \cdot \nabla_\alpha u(\Theta_{t-2},\alpha)|_p$
$h_{l,t} = \frac{\partial f(\theta,\alpha)}{\partial \alpha_{l,t}} = -\mathbf{tr}(\nabla_\theta f(\theta_{t-1},\alpha)|_l^T \nabla_\alpha u(\Theta_{t-2},\alpha)|_l)$
$h_{g,t} = \frac{\partial f(\theta,\alpha)}{\partial \alpha_{t}} = -\sum_{l=1}^{n} \mathbf{tr}(\nabla_\theta f(\theta_{t-1},\alpha)|_l^T \nabla_\alpha u(\Theta_{t-2},\alpha))|_l)$
$\alpha_{p,t} = \alpha_{p,t-1} - \beta_p \frac{\partial f(\theta_{t-1})}{\partial \alpha^*_{p,t-1}} \frac{\partial \alpha^*_{p,t-1}}{\partial \alpha_{p,t-1}} = \alpha_{p,t-1} - \beta_p \gamma_{1,t-1} h_{p,t}$
$\alpha_{l,t} = \alpha_{l,t-1} - \beta_l \sum_p \frac{\partial f(\theta_{t-1})}{\partial \alpha^*_{p,t-1}} \frac{\partial \alpha^*_{p,t-1}}{\partial \alpha_{l,t-1}} = \alpha_{l,t-1} - \beta_l \gamma_{2,t-1} \sum_p h_{p,t} =$
$\alpha_{l,t-1} - \beta_l \gamma_{2,t-1} h_{l,t}$
$\alpha_t = \alpha_{t-1} - \beta_g \sum_l \sum_p \frac{\partial f(\theta)}{\partial \alpha^*_{p,t-1}} \frac{\partial \alpha^*_{p,t-1}}{\partial \alpha_{t-1}} = \alpha_{t-1} - \beta_g \gamma_{3,t-1} h_{g,t}$
$\alpha^*_{p,t} = \gamma_{1,t-1}\alpha_{p,t} + \gamma_{2,t-1}\alpha_{l,t} + \gamma_{3,t-1}\alpha_t$
$\gamma_{1,t} = \gamma_{1,t-1} - \delta \frac{\partial L}{\partial \gamma_{1,t-1}} = \gamma_{1,t-1} - \delta \sum_p \frac{\partial L}{\partial \alpha^*_{p,t-1}} \frac{\partial \alpha^*_{p,t-1}}{\partial \gamma_{1,t-1}} =$
$\gamma_{1,t-1} - \delta \alpha_{p,t-1} \sum_p \frac{\partial L}{\partial \alpha^*_{p,t-1}}$
$\gamma_{2,t} = \gamma_{2,t-1} - \delta \frac{\partial L}{\partial \gamma_{2,t-1}} = \gamma_{2,t-1} - \delta \sum_p \frac{\partial L}{\partial \alpha^*_{p,t-1}} \frac{\partial \alpha^*_{p,t-1}}{\partial \gamma_{2,t-1}} =$
$\gamma_{1,t-1} - \delta \alpha_{l,t-1} \sum_p \frac{\partial L}{\partial \alpha^*_{p,t-1}}$
$\gamma_{3,t} = \gamma_{3,t-1} - \delta \frac{\partial L}{\partial \gamma_{3,t-1}} = \gamma_{3,t-1} - \delta \sum_p \frac{\partial L}{\partial \alpha^*_{p,t-1}} \frac{\partial \alpha^*_{p,t-1}}{\partial \gamma_{3,t-1}} =$
$\gamma_{3,t-1} - \delta \alpha_{t-1} \sum_p \frac{\partial L}{\partial \alpha^*_{p,t-1}}$
$\gamma_1 = \gamma_1/(\gamma_1+\gamma_2+\gamma_3)$, $\gamma_2 = \gamma_1/(\gamma_1+\gamma_2+\gamma_3)$, $\gamma_3 = \gamma_1/(\gamma_1+\gamma_2+\gamma_3)$
$m_t = \phi_t(g_1, ...g_t)$
$V_t = \psi_t(g_1, ...g_t)$
$\theta_t = \theta_{t-1} - \alpha^*_{p,t} m_t / \sqrt{V_t}$
end
return θ_T, $\gamma_{1,T}$, $\gamma_{2,T}$, $\gamma_{3,T}$, $\alpha_{p,T}$, $\alpha_{l,T}$, α_T

where we introduce the general form of gradient descent based optimizers [36, 44].

In Algorithm 1, we use the general form of gradient descent based optimizers [36, 44]. For SGD, $\phi_t(g_1, \ldots g_t) = g_t$ and $\psi_t(g_1, \ldots g_t) = 1$, while for Adam, $\phi_t(g_1, \ldots g_t) = (1-\beta_1)\Sigma_{i=1}^{t} \beta_1^{t-1} g_i$ and $\psi_t(g_1, \ldots g_t) = (1-\beta_2)\mathrm{diag}(\Sigma_{i=1}^{t} \beta_2^{t-1} g_i^2)$. Notice that in each updating time step of Algorithm 1, we re-normalize the combination weights γ_1, γ_2 and γ_3 to make sure that their summation is always 1 even after updating with stochastic gradient-based methods. An alternative way of doing this is to implemented softmax, which require an extra set of intermediate variables c_p, c_l and c_g following: $\gamma_p = \mathrm{softmax}(c_p) = \exp^{c_p}/(\exp^{c_p} + \exp^{c_l} + \exp^{c_g})$, etc. Then the updating of γs will be convert to the updating of c's during training. In addition, the training of γ's can also be extended to multi-level cases, which means we can have different combination weights in different layers. For the updating rates β_p, β_l and β_g of the learning rates at different level, we set:

$$\beta_p = n_p\beta = \beta,\ \beta_l = n_l\beta,\ \beta_g = n\beta$$

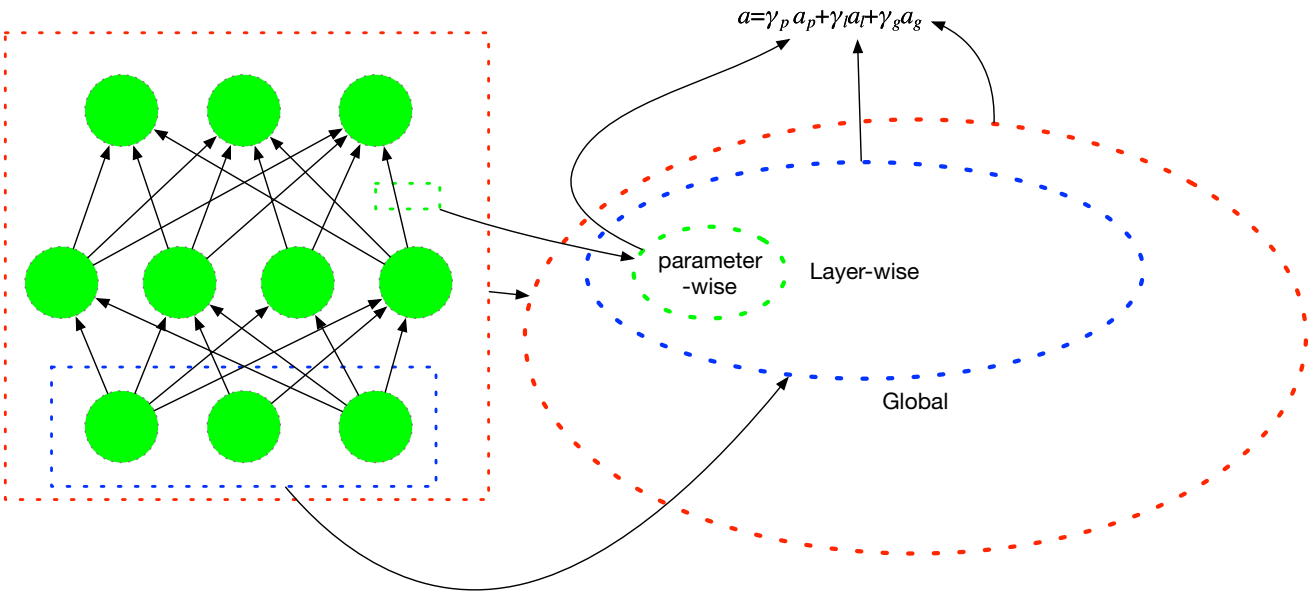

Fig. 1 The diagram of a three-level learning rate combination. Here we consider three levels of adaptive learning rates, which are calculated by global-level, layer-level and parameter-level hyper-gradient descent with different grouping strategies. The final effective learning rate is a weighted combination of the three level adaptive learning rates, while the combination weights are also trainable during back-propagation

where β is a shared parameter. This setting will make the updating steps of learning rates in different levels be in the same scale considering the difference in the number of parameters involved in $h_{p,t}, h_{l,t}, h_{g,t}$. If we take average based on the number of parameters in Eq. (12) at first, this adjustment is not required.

CAM-HD is a higher-level adaptation approach, which can be applied with any gradient-based updating rules and advanced adaptive optimizers. For example, if we apply CAM-HD for Adam optimizer, we have Adam-CAM-HD. Similarly, when we apply CAM-HD for SGDN, we have SGDN-CAM-HD. Further, it can be merged with Adabound by adding an element-wise clipping procedure [36]:

$$\hat{\eta}_t = \text{Clip}\Big(\alpha^*/\sqrt{V_t}, \eta_l(t), \eta_u(t)\Big), \ \eta_t = \hat{\eta}_t/\sqrt{t} \tag{13}$$

where α^* is the final step-size by original CAM-HD, $\eta_l(t)$ and $\eta_u(t)$ are the lower and upper bounds in adabound. η_t can be applied in replacing $\alpha^*_{p,t}/\sqrt{V_t}$ in our algorithm for merging two methods to so called "Adabound-CAM-HD". In the experiment part, we will follow the original paper of Adabound and related discussions to set $\eta_l(t) = 0.1 - \frac{0.1}{(1-\beta_2)t+1}$ and $\eta_u(t) = 0.1 + \frac{0.1}{(1-\beta_2)t}$ for both Adabound and Adabound-CAM-HD [36, 47]. As the effective parameter-wise updating rates and corresponding gradients may change after clipping, the updating rules for other variable should be adjusted accordingly.

Although Algorithm 1 involves many updating steps with intermediate parameters, the time complexity does not increase in large scale, which will be further discussed in Sect. 6.4. In fact, the updating of intermediate variables in the algorithm does not involve the dimension of batch size. Meanwhile, only parameter-wise adaptation requires a proportion of extra computational cost compared with standard back proportion, which can be avoided when layer-wise or cell-wise learning rates are applied as the lowest level adaptation.

4.5 Convergence analysis

The proposed CMA-HD is not an independent optimization method, which can be applied in any kinds of gradient-based updating rules. Its convergence properties highly depends on the base optimizer that is applied. By referring the discussion on convergence in [6], if we introduce $\kappa_{p,t} = \tau(t)\alpha^*_{p,t} + (1-\tau(t))\alpha_\infty$, where the function $\tau(t)$ is selected to satisfy $t\tau(t) \to 0$ as $t \to \infty$, and α_∞ is a selected constant value. Then we demonstrate the convergence analysis for the three level case in the following theorem, where ∇_p is the the gradient of target function w.r.t. a model parameter with index p, ∇_l is the average gradient of target function w.r.t. a parameters in a layer with index l, and ∇_g is the global average gradient of target function w.r.t. all model parameters.

Theorem 2 (Convergence under mild assumptions about *f*) *Suppose that f is convex and L-Lipschitz smooth with* $\|\nabla_p f(\theta)\| < M_p$, $\|\nabla_l f(\theta)\| < M_l$, $\|\nabla_g f(\theta)\| < M_g$ *for some fixed* M_p, M_l, M_g *and all* θ. *Then* $\theta_t \to \theta^*$ *if* $\alpha_\infty < 1/L$ *where L is the Lipschitz constant for all the gradients and* $t \cdot \tau(t) \to 0$ *as* $t \to \infty$, *where the* θ_t *are generated according to* (*non-stochastic*) *gradient descent*.

In the above theorem, ∇_p is the gradient of target function w.r.t. a model parameter with index p, ∇_l is the average gradient of target function w.r.t. parameters in a layer with index l, and ∇_g is the global average gradient of target function w.r.t. all model parameters. The proof of this theorem is given as follows.

Proof We take three-level's case discussed in Sect. 4 for example, which includes global level, layer-level and parameter-level. Suppose that the target function f is convex, L-Lipschitz smooth in all levels, which gives for all θ_1 and θ_2:

$$
\begin{aligned}
&||\nabla_p f(\theta_1) - \nabla_p f(\theta_2)|| \le L_p||\theta_1 - \theta_2|| \\
&||\nabla_l f(\theta_1) - \nabla_l f(\theta_2)|| \le L_l||\theta_1 - \theta_2|| \\
&||\nabla_g f(\theta_1) - \nabla_g f(\theta_2)|| \le L_g||\theta_1 - \theta_2|| \\
&L = \max\{L_p, L_l, L_g\}
\end{aligned}
$$

and its gradient with respect to parameter-wise, layer-wise, global-wise parameter groups satisfy $\|\nabla_p f(\theta)\| < M_p$, $\|\nabla_l f(\theta)\| < M_l$, $\|\nabla_g f(\theta)\| < M_g$ for some fixed M_p, M_l, M_g and all θ. Then the effective combined learning rate for each parameter satisfies:

$$
\begin{aligned}
|\alpha^*_{p,t}| &= |\gamma_{p,t-1}\alpha_{p,t} + \gamma_{l,t-1}\alpha_{l,t} + \gamma_{g,t-1}\alpha_t| \\
&\le (\gamma_{p,t-1} + \gamma_{l,t-1} + \gamma_{g,t-1})\alpha_0 \\
&+ \beta \sum_{i=0}^{t-1} \Big(\gamma_{p,t-1} n_p \max_p \{ \Big| \nabla f(\theta_{p,i+1})^T \nabla f(\theta_{p,i}) \Big| \} \\
&+ \gamma_{l,t-1} n_l \max_l \{ |\nabla f(\theta_{l,i+1})^T \nabla f(\theta_{l,i})| \} + \gamma_{g,t-1} |\nabla f(\theta_{g,i+1})^T \nabla f(\theta_{g,i})| \Big) \\
&\le \alpha_0 + \beta \sum_{i=0}^{t-1} \Big(\gamma_{p,t-1} n_p \max_p \{ \Big\| \nabla f(\theta_{p,i+1}) \Big\| \Big\| \nabla f(\theta_{p,i}) \Big\| \} \\
&+ \gamma_{l,t-1} n_l \max_l \{ \|\nabla f(\theta_{l,i+1})\| \|\nabla f(\theta_{l,i})\| \} + \gamma_{g,t-1} \|\nabla f(\theta_{g,i+1})\| \|\nabla f(\theta_{g,i})\| \Big) \\
&\le \alpha_0 + t\beta(n_p M_p^2 + n_l M_l^2 + M_g^2)
\end{aligned}
$$

where $\theta_{p,i}$ refers to the value of parameter indexed by p at time step i, $\theta_{l,i}$ refers to the set/vector of parameters in layer with index l at time step i, and $\theta_{g,i}$ refers to the whole set of model parameters at time step i. In addition, n_p and n_l are the total number of parameters and number of the layers, and we have applied $0 < \gamma_p, \gamma_l, \gamma_g < 1$. This gives an upper bound for the learning rate in each particular time step, which is $O(t)$ as $t \to \infty$. By introducing $\kappa_{p,t} = \tau(t)\alpha^*_{p,t} + (1 - \tau(t))\alpha_\infty$, where the function $\tau(t)$ is selected to satisfy $t\tau(t) \to 0$ as $t \to \infty$, so we have $\kappa_{p,t} \to \alpha_\infty$ as $t \to \infty$. If $\alpha_\infty < \frac{1}{L}$, for larger enough t, we have $1/(L+1) < \kappa_{p,t} < 1/L$, and the algorithm converges when the corresponding gradient-based optimizer converges for such a learning rate under our assumptions about f. This follows the discussion in [25, 50]. □

This actually provides a convergence of $R(T) = O(T)$ given the assumptions and conditions, while a stronger convergence can be achieved by assuming a more strict form of $\tau(t)$. Notice that when we introduce $\kappa_{p,t}$ instead of $\alpha^*_{p,t}$ in Algorithm 1, the corresponding gradients $\frac{\partial L(\theta)}{\partial \alpha^*_{p,t-1}}$ will also be replaced by $\frac{\partial L(\theta)}{\partial \kappa^*_{p,t-1}} \frac{\partial \kappa^*_{p,t-1}}{\partial \alpha^*_{p,t-1}} = \frac{\partial L(\theta)}{\partial \kappa^*_{p,t-1}} \tau(t)$. Beyond weighted approximation, clipping can also guarantee a convergence. One example is the Adabound-CAM-HD proposed in Sect. 4.4.

Theorem 3 (Convergence of Adabound-CAM-HD) *Let* $\{\theta_t\}$ *and* $\{V_t\}$ *be the sequences obtained from the modified Algorithm* 1 *for Adabound-CAM-HD discussed in Sect.* 4.4. *The optimizer parameters in Adam satisfy* $\beta_1 = \beta_{11}$, $\beta_{1t} \le \beta_1$ *for all* $t \in [T]$ *and* $\beta_1 < \sqrt{\beta_2}$. *Suppose f is a convex target function on* Θ, $\eta_l(t)$ *and* $\eta_u(t)$ *are the lower and upper bound function applied in the clipping procedure,* $\eta_u(t) \le R_\infty$ *and* $\frac{t}{\eta_l(t)} - \frac{t-1}{\eta_u(t-1)} \le M$ *for all* $t \in [T]$. *Assume that* $||\theta_1 - \theta_2||_\infty \le D_\infty$ *for all* $\theta_1, \theta_2 \in \Theta$ *and* $||\nabla f_t(\theta)|| \le G_2$ *for all* $t \in [T]$ *and* $\theta \in \Theta$. *For* θ_t *generated using Adabound-CAM-HD algorithm, the regret function* $R(T) = \sum_{t=1}^{T} f_t(\theta_t) - \min_{\theta \in \Theta} \sum_{t=1}^{T} f_t(\theta)$ *is upper bounded by* $O(\sqrt{T})$, *which is given by*:

$$R_T \leq \frac{D_\infty^2}{2(1-\beta_1)}\left[2dM\left(\sqrt{T}-1\right)+\sum_{i=1}^{d}\left[\eta_{1,i}^{-1}+\sum_{t=1}^{T}\beta_{1t}\eta_{t,i}^{-1}\right]\right] + (2\sqrt{T}-1)\frac{R_\infty G_2^2}{1-\beta_1}.$$

In general, the main ideas involved in the proof of convergence of Adabound in [36] and [47] is also applicable for Adabound-CAM-HD. The main procedure can be given as follows.

- Let $x^* = \text{argmin}_{x\in\mathcal{F}}\sum_{t-1}^{T} f_t(x)$, which exists since $\mathcal{F}$ is closed and convex. Apply the projection relationship:

$$\begin{aligned} x_{t+1} &= \Pi_{\mathcal{F},\text{diag}(\eta^{-1})}(x_t - \eta_t \odot m_t) \\ &= \min_{x\in\mathcal{F}} ||\eta_t^{-1/2} \odot (x - (x_t - \eta_t \odot m_t))|| \end{aligned}$$

  where η_t is the effective updating rate at step t after clipping and normalizing, while m_t is the momentum at step t.
- Apply Lemma 1 in the original paper [36, 39] with $u_1 = x_{t+1}$ and $u_2 = x^*$ to get the upper bound of $||\eta^{-1/2} \odot (x_{t+1} - x^*)||^2$, and further rearrange the corresponding inequality to get the upper bound of $\langle g_t, x_t - x^*\rangle$, with the auxiliary of Cauchy-Schwarz and Young's inequality.
- Consider the standard approach of bounding the regret at each step using convexity of the functions $f_{t t=1}^{T}$:

$$\begin{aligned} R_T &= \sum_{t=1}^{T} f_t(x_t) - \min_{x\in\mathcal{F}}\sum_{t=1}^{T} f_t(x) \\ &= \sum_{t=1}^{T}(f_t(x_t) - f_t(x^*)) \leq \sum_{t=1}^{T}\langle g_t, x_t - x^*\rangle \end{aligned}$$

- Find the upper bound of R_T given by the summation of upper bounds of $\langle g_t, x_t - x^*\rangle$ with different step t, and further introduce the inequality relationships $\beta_1 = \beta_{11}$, $\beta_{1t} \leq \beta_1$ for all $t \in [T]$ and $\beta_1 < \sqrt{\beta_2}$, bounding conditions of $\eta_u(t) \leq R_\infty$ and $\frac{t}{\eta_l(t)} - \frac{t-1}{\eta_u(t-1)} \leq M$, to get the final upper bound of R_T in terms of $R_\infty, G_\infty M$ and T.

In Adabound-CAM-HD, η_t depends on the format of α^*. If α^* is a matrix for parameter-wise learning rate for a particular layer, η_t should also be a matrix but clipped by a pair of global $\eta_u(t)$ and $\eta_l(t)$ at each time step t. Notice that in [36] with the code provided by github.com/Luolc/AdaBound, the clipping is also parameter-wise because in the clipping function $\hat{\eta}_t = \text{Clip}(\alpha/\sqrt{V_t}, \eta_l(t), \eta_u(t))$, the term $\alpha/\sqrt{V_t}$ will generate a matrix with the same shape as the corresponding parameter matrix in each layer, although the step size α is a scalar. Thus, the clipping on element-wise division $\alpha^*/\sqrt{V_t}$ in Adabound-CAM-HD could achieve the same bounding properties. For example, if the scale is adjusted accordingly, the norm of η_t satisfies $\sqrt{t}||\eta_t||_\infty \leq R_\infty$, which makes the Lemma 3 in [36] holds. Meanwhile, Lemma 1 and Lemma 2 in the original paper can be directly applied as parameter θ_t, θ^* and m_t have element-wise values in both contexts. Although in the proposed algorithm, we introduced the hierarchical learning rate structure, for the model parameter, gradients and momentum, parameter-wise form has already been applied.

This ensures Adabound-CAM-HD achieves a high level of adaptiveness as well as a good convergence property. Notice that in the original version of the convergence theorem of Adabound proposed in [36], the assumptions for upper and lower bound was given by $\eta_l(t+1) \geq \eta_l(t) > 0$, $\eta_u(t+1) < \eta_u(t)$. As $t \to \infty$, $\eta_l(t) \to \alpha^*$, $\eta_u(t) \to \alpha^*$. $L_\infty = \eta_l(1)$ and $R_\infty = \eta_u(1)$. However, in [47], it is pointed out that this original assumption can only guarantee a convergence of $R_T = O(T)$, while it is recommended to suppose $\frac{t}{\eta_l(t)} - \frac{t-1}{\eta_u(t-1)} \leq M$ for all $t \in [T]$ instead to guarantee a better convergence $R_T = O(\sqrt{T})$.

Therefore, both weighted approximation and clipping can be applied to guarantee the convergence of optimizers with CAM. This means that CAM can safely achieve both high-level parameter-specific adaptiveness and a good property of convergence, which gives it the potential of outperforming most of existing optimization algorithms. Meanwhile, it is compatible with any gradient based optimizers and network architectures.

5 Experiments

We use the feed-forward neural network models and different types of convolutions neural networks on multiple benchmark datasets to compare with existing baseline optimizers. For each learning task, the following optimizers will be applied: (a) standard baseline optimizers such as Adam and SGD; (b) hyper-gradient descent in [6]; (c) L4 stepsize adaptation for standard optimizers [45]; (d) Adabound optimizer [36]; (e) RAdam optimizer [34]; and (f) the proposed adaptive combination of different levels of hyper-descent. The implementation of (b) is based on the code provided with the original paper. One NVIDIA Tesla V100 GPU with 16G Memory 61 GB RAM and two Intel Xeon 8 Core CPUs with 32 GB RAM are applied. The program is built in Python 3.5.1 and Pytorch 1.0 [49]. For each experiment, we provide both the average curves and standard error bars for ten runs.

Table 1 Hyperparameter settings for experiments (learning rates: SGD/SGDN (lr1); Adam (lr2); Hyper-grad (SGD/SGDN): lr3; Hyper-grad lr (Adam): lr4; CAM-HD: lr5)

Architecture	Dataset	Batch size	lr1	lr2	lr3	lr4	lr5
MLP 1	MNIST	32	–	0.0003	–	1.00E−07	0.01
MLP 2		64	–	0.001	–	1.00E−07	0.01
MLP 3		128	–	0.001	–	1.00E−07	0.01
LeNet-5	MNIST	256	–	0.001	1.00E−03	1.00E−08	0.03
	CIFAR10	256	–	0.001	1.00E−03	1.00E−08	0.03
	SVHN	128	–	0.001	1.00E−03	1.00E−08	0.03
ResNet-18	CIFAR10	256	0.1	0.001	1.00E−06	1.00E−08	0.001
ResNet-34		256	0.1	0.001	1.00E−06	1.00E−08	0.001

5.1 Hyper-parameter tuning

To compare the effect of CAM-HD with baseline optimizers, we first do hyperparameter tuning for each learning task by referring to related papers [6, 26, 36, 45] as well as implementing an independent grid search [8, 13]. We mainly consider hyper-parameters including batch size, learning rate, and other optimizer parameters for models with different architectures. Other settings in our experiments follow open-source benchmark models. The search space for batch size is the set of $\{2^n\}_{n=3,\ldots,9}$, while the search space for learning rate, hyper-gradient updating rate and combination weight updating rate (CAM-HD-lr) are $\{10^{-1}, 10^{-2}, \ldots, 10^{-4}\}$, $\{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$ and $\{0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$, respectively. The selection criterion is the 5-fold cross-validation loss by early-stopping at the patience of 3 [42]. The optimized hyper-parameters for the tasks in this paper are given in Table 1. For training ResNets with SGDN, we will apply a step-wise learning rate decay schedule as in [34, 36]. Notice that although the hyper-parameters are tuned, it does not mean that the model performance is sensitive to each hyper-parameter.

For training ResNets with SGDN, we will apply a step-wise learning rate decay schedule as in [34, 36]. Notice that although the hyper-parameters are tuned, it does not mean that the model performance is sensitive to each of them.

5.2 Combination ratio and model performances

First, we perform a study on the initialization of the combination weights different level learning rates in the framework of CAM-HD. The simulations are based on image classification tasks on MNIST and CIFAR10 [27, 30]. We use full training sets of MNIST and CIFAR10 for training and full test sets for validation. One feed-forward neural network with three hidden layers of size [100, 100, 100] and two convolutional network models, including LeNet-5 [31] and ResNet-18 [23], are implemented. In each case, two levels of learning rates are considered, which are the global and layer-wise adaptation for FFNN, and global and filter-wise adaptation for CNNs. For LeNet-5 and FFNN, Adam-CAM-HD with fixed and trainable combination weights is implemented, while for ResNet-18, both Adam-CAM-HD and SGDN-CAM-HD with fixed and trainable combination weights are implemented in two independent simulations. We change the initialized combination weights of two levels in each case to see the change of model performance in terms of test classification accuracy at epoch 30 for FFNN, and at epoch 10 for LeNet-5 and ResNet-18. Also we compare CAM-HD methods with baseline Adam and SGDN methods in terms of test accuracy after the same epochs of training. Other hyper-parameters are optimized based on Sect. 5.1. We conduct 10 runs at each combination ratio and draw the average accuracies and corresponding error bars (standard errors). The result is given in Fig. 2,

which leads to the following findings: First, usually the optimal performance is neither at full global level nor full layer/filter level, but a weighted combination of two levels of adaptive learning rates, for both update and no-update cases. Second, CAM-HD methods outperform baseline Adam/SGDN methods for most of the combination ratios initializations. Third, updating of combination weights is effective and helpful in achieving better performance than applying fixed combination weights. This supports our analysis in Sect. 4.3. Also, in real training processes, it is possible that the learning in favor of different combination weights in various stages and this requires the online adaptation of the combination weights.

5.3 Feed forward neural network for image classification

This experiment is conducted with feed-forward neural networks for image classification on MNIST, including 60,000 training examples and 10,000 test examples. We use the full training set for training and the full test set for validation. Three FFNN with three different hidden layer configurations are implemented [14, 52], including [100, 100], [1000, 100], and [1000, 1000]. Adaptive optimizers including Adam, Adabound, Adam-HD with two hyper-gradient updating rates, and proposed Adam-CAM-HD are applied.

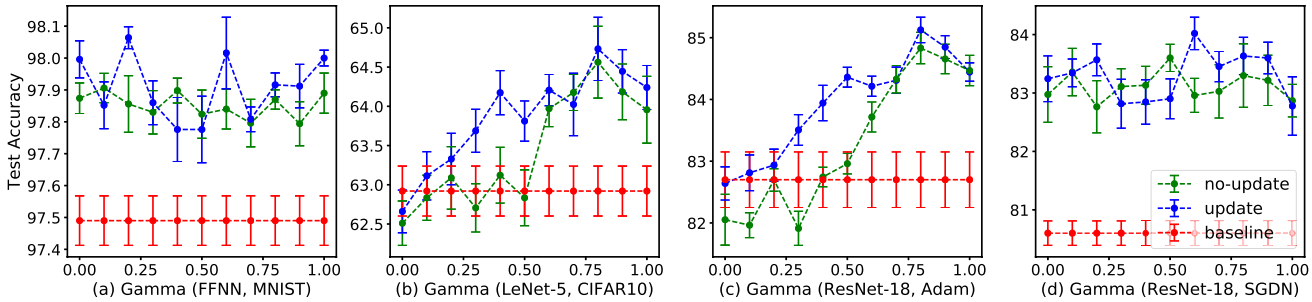

Fig. 2 The diagram of model performances trained by Adam/SGDN-CAM-HD with different combination ratios in the case of two-level learning rates adaptation. The x-axis is the ratio of global-level adaptive learning rates. ResNet-18s are trained for 10 epochs only

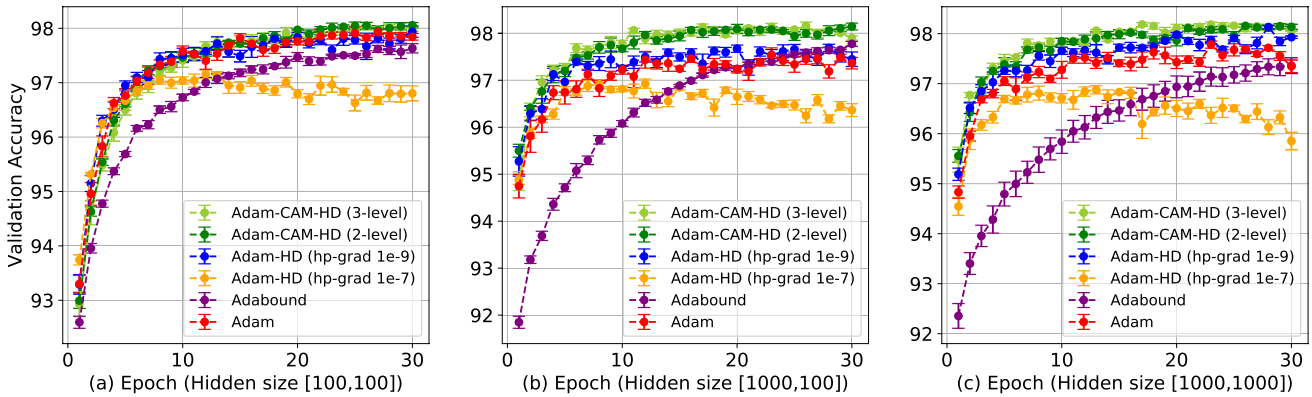

Fig. 3 The comparison of learning curves of FFNN on MNIST with different adaptive optimizers

Table 2 Summary of test performances with FFNNs

	FFNN (100, 100)		FFNN (1000, 100)		FFNN (1000, 1000)	
	Test acc	Test S.E	Test acc	Test S.E	Test acc	Test S.E
Adam-CAM-HD (3-level)	97.91	0.07	97.92	0.15	98.29	0.07
Adam-CAM-HD (2-level)	**98.12**	0.06	**98.09**	0.06	**98.39**	0.04
Adam-HD (hp-grad 1e−9)	97.86	0.07	97.19	0.26	97.83	0.12
Adam	97.93	0.09	97.48	0.14	97.49	0.11

The bold figure means the best performer

For Adam-CAM-HD, we apply three-level parameter-layer-global adaptation with initialization of $\gamma_1 = \gamma_2 = 0.3$ and $\gamma_3 = 0.4$, and two-level layer-global adaptation with $\gamma_1 = \gamma_2 = 0.5$. No decay function of learning rates is applied.

Figure 3 shows the validation accuracy curves for different optimizers during the training process of 30 epochs. We can learn that both the two-level and three-level Adam-CAM-HD outperform the baseline Adam optimizer with optimized hyper-parameters significantly. For Adam-HD, we find that the default hyper-gradient updating rate ($\beta = 10^{-7}$) for Adam applied in [6] is not optimal in our experiments, while an optimized one of 10^{-9} can outperform Adam but still worse than Adam-CAM-HD with $\beta = 10^{-7}$.

The test accuracy of each setting and the corresponding standard error of the sample mean in 10 trials are given in Table 2.

5.4 Lenet-5 for image classification

The second experiment is done with LeNet-5, a classical convolutional neural network without involving many building and training tricks [31]. We compare a set of adaptive Adam optimizers including Adam, Adam-HD, Adam-CAM-HD, Adabound, RAdam and L4 for the image classification learning task of MNIST, CIFAR10 and SVHN [40]. For Adam-CAM-HD, we apply a two-level setting with

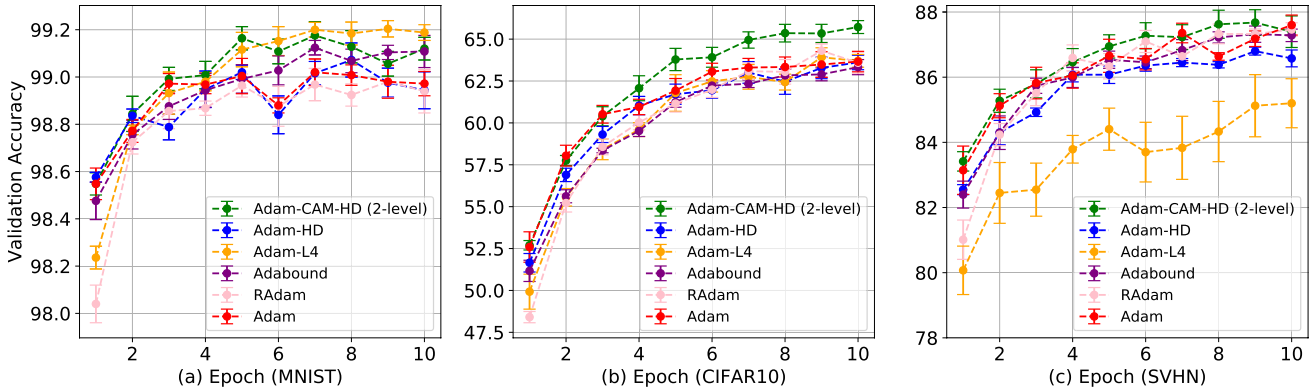

Fig. 4 The comparison of learning curves of training LeNet-5 with different adaptive optimizers

Table 3 Summary of test performances with LeNet-5

	MNIST		CIFAR10		SVHN	
	Test acc	Test S.E	Test acc	Test S.E	Test acc	Test S.E
Adam-CAM-HD	98.93	0.07	**65.55**	0.18	**87.58**	0.37
Adam-HD	98.83	0.05	63.3	0.66	86.94	0.13
Adam-L4	**99.19**	0.05	63.76	0.26	85.44	0.42
Adabound	99.11	0.05	64.06	0.36	87.22	0.14
RAdam	98.94	0.06	63.91	0.34	87.31	0.41
Adam	98.89	0.05	63.88	0.45	86.82	0.16

The bold figure means the best performer

filter-wise and global learning rates adaptation and initialize $\gamma_1 = 0.2, \gamma_2 = 0.8$. We also implement an exponential decay function $\tau(t) = \exp(-rt)$ as was discussed in Sect. 4.5 with rate $r = 0.002$ for all the three datasets, while t is the number of iterations. For L4, we implement the recommended L4 learning rate of 0.15. For Adabound and RAdam, we also apply the recommended hyper-parameters in the original papers. The other hyper-parameter settings are optimized in Sect. 5.1.

As we can see in Fig. 4, Adam-CAM-HD again shows the advantage over other methods in all the three sub-experiments, except MNIST L4 that could perform better in a later stage. The experiment on SVHN indicates that the recommended hyper-parameters for L4 could fail in some cases with unstable accuracy curves. RAdam and Adabound outperform baseline Adam method on MNIST, while Adam-HD does not show a significant advantage over Adam with optimized hyper-gradient updating rate that is shared with Adam-CAM-HD.

The corresponding summary of test performance is given in Table 3, in which the test accuracy of Adam-CAM-HD outperform other optimizers on both CIFAR10 and SVHN. Especially, it gives significantly better results than Adam and Adam-HD for all the three datasets.

5.5 ResNet for image classification

In the third experiment, we apply ResNets for image classification task on CIFAR10 [11, 23] following the code provided by github.com/kuangliu/pytorch-cifar, where a ResNet-18 gives an accuracy of 93.02 with SGD and the cosine annealing learning rate schedule. We compare Adam and Adam-based adaptive optimizers, as well as SGD with Nestorov momentum (SGDN) and corresponding adaptive optimizers for training both ResNet-18 and ResNet-34. For SGDN methods, we apply a learning rate schedule, in which the learning rate is initialized to a default value of 0.1 and reduced to 0.01 or 10% (for SGDN-CAM-HD) after epoch 150. The momentum is set to be 0.9 for all SGDN methods. For Adam-CAM-HD SGDN-CAM-HD, we apply two-level CAM-HD with the same setting as the second experiment. We also implement Adabound-CAM-HD discussed in Sect. 4.4 by sharing the common parameters with Adabound. In addition, we apply an exponential decay function with a decay rate $r = 0.001$ for all the CAM-HD methods. The learning curves for validation accuracy, training loss, and validation loss of ResNet-18 and ResNet-34 are shown in Fig. 5.

We can see that the validation accuracy of Adam-CAM-HD reaches about 90% in 40 epochs and consistently

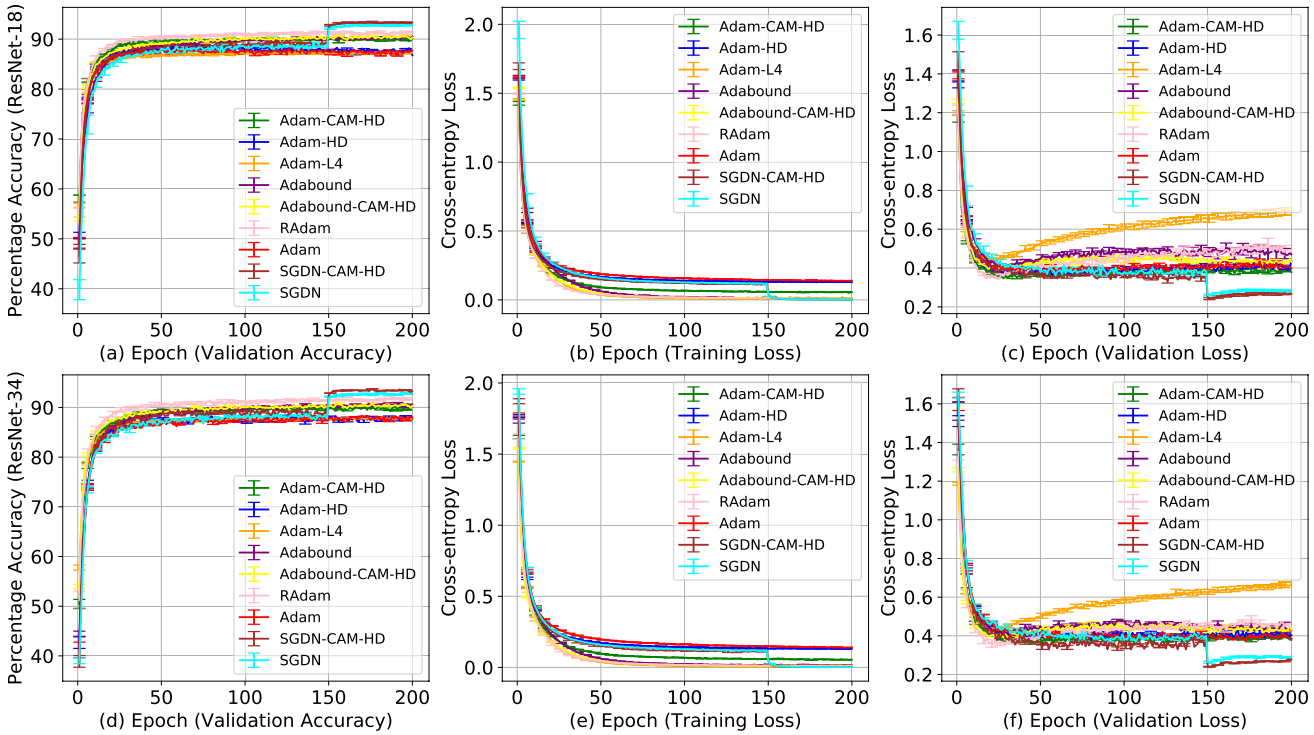

Fig. 5 The learning curves of training ResNet-18/34 on CIFAR10 with adaptive optimizers

outperforms Adam, L4 and Adam-HD optimizers in a later stage. The L4 optimizer with recommended hyper-parameter and an optimized weight-decay rate of 0.0005 (instead of 1e-4 applied in other Adam-based optimizers) can outperform baseline Adam for both ResNet-18 and ResNet-34, while its training loss outperforms all other methods but with potential over-fitting. Adam-HD achieves a similar or better validation accuracy than Adam with an optimized hyper-gradient updating rate of 10^{-9}. RAdam performs slightly better than Adam-CAM-HD in terms of validation accuracy, but the validation cross-entropy of both RAdam and Adabound are outperformed by our method. Also, we find that in training ResNet-18/34, the validation accuracy and validation loss of SGDN-CAM-HD slightly outperform SGDN in most epochs even after the resetting of the learning rate at epoch 150.

The test performances (average accuracy and standard error) of different optimizers for ResNet-18 and ResNet-34 after 200 epoch of training are shown in Table 4.[1] We can learn that for both ResNet-18 and ResNet-34, the proposed CAM-HD methods (Adam-CAM-HD, Adabound-CAM-HD and SGDN-CAM-HD) can improve the corresponding baseline methods (Adam, Adabound and SGDN) with statistical significance. Especially, Adabound-CAM-HD outperforms both Adam-CAM-HD and Adabound.

Table 4 Summary of test performances with ResNet-18/34

	ResNet-18		ResNet-34	
	Test acc.	Test S.E.	Test acc.	Test S.E.
Adam	87.03	0.15	87.95	0.22
Adam-HD	87.26	0.35	88.48	0.48
Adabound	90.29	0.15	90.15	0.3
Radam	91.54	0.17	91.76	0.28
Adam-L4	87.81	0.22	88.02	0.15
Adam-CAM-HD	90.31	0.25	90.28	0.09
Adaboud-CAM-HD	90.49	0.31	91.12	0.23
SGDN	93.04	0.21	92.93	0.29
SGDN-CAM-HD	93.35	0.08	93.47	0.23

6 Discussion

The experiments on both small models and large models demonstrate the advantage of the proposed method over baseline optimizers in terms of validation and test accuracy. One explanation of the performance improvement of our method is that it achieves a higher level of adaptation

[1] Here Adam-based methods achieve much lower test accuracies as we only apply learning rate schedules to SGDN and SGDN-CAM-HD.

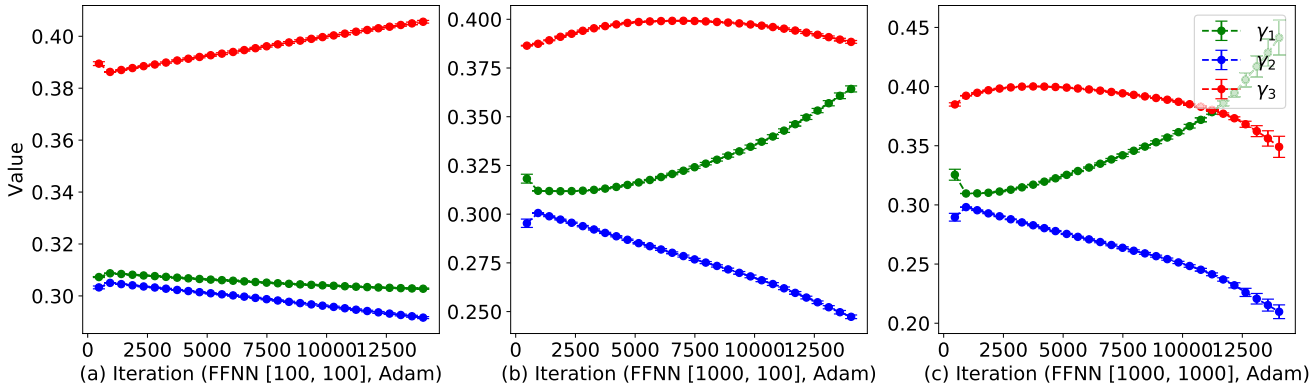

Fig. 6 Learning curves of γs for FFNN on MNIST with Adam

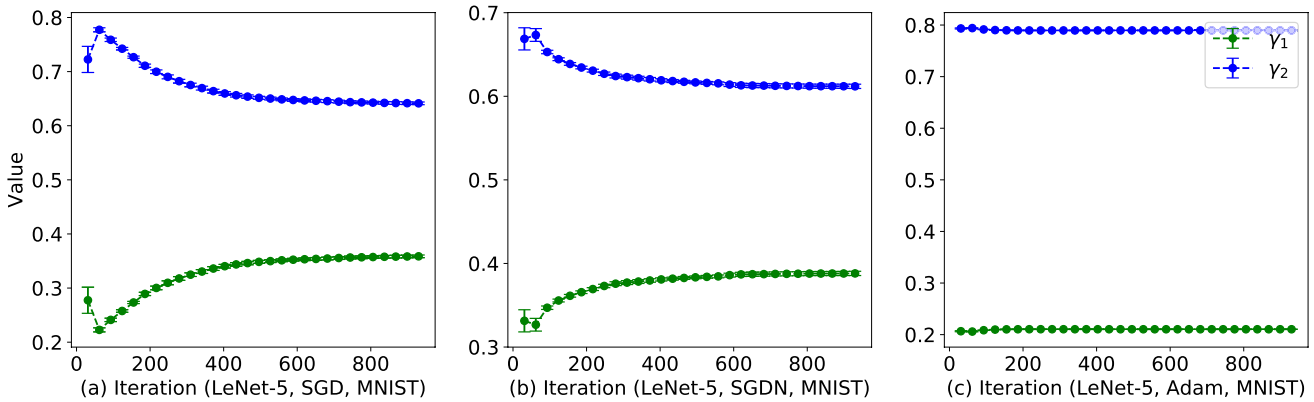

Fig. 7 Learning curves of γs for LeNet-5 on MNIST with SGD, SGDN and Adam ($\tau = 0.002$)

by introducing hierarchical learning rate structures with learn-able combination weights, while the parameterization level of adaptive learning rates is controlled by its intrinsic regularization effects. In addition, both weighted approximation and clipping can be applied to guarantee a convergence. In this section we discuss several aspects of our study, including hyper-parameter settings, learning of combination weights, number of parameters and space and time complexity.

6.1 Performance and hyper-parameter settings

Experiments show that the performance improvement does not require tuning the hyper-parameters independently if the task or model is similar. For example, the hyper-gradient updating rate for LeNet-5, ResNet-18 and ResNet-34 are all set to be 1e-8 in our experiments no matter the dataset being learned. Also, the hyper-parameter CAM-HD-lr is shared among each group of models (FFNNs, LeNet-5, ResNets) for all datasets being learned. For the combination ratio, $\gamma_1 = 0.2, \gamma_2 = 0.8$ works for all our experiments with convolutional networks. However, as the loss surface with respect to the combination weights may not be convex for deep learning models, the learning of combination weights may fall into local optimal. Therefore, it is possible that several trials are needed to find a good initialization of combination weights although the learning of combination weights works locally [13]. In general, the selected hyper-parameters are transferable to a similar task for an improvement from the corresponding baseline, while the optimal hyper-parameter setting may shift a bit.

The proposed CAM-HD method can also apply learning rate schedules in many ways to achieve further improvement. One example is our ResNet experiment on CIFAR10 with SGDN and SGDN-CAM-HD. For more advanced learning rate schedules [19, 28], we can apply strategies like piecewise adaptive scheme by re-initialize all the levels for different steps. Another method is to replace global level learning rate with scheduled learning rate, while adapting the combination weights and other levels continuously.

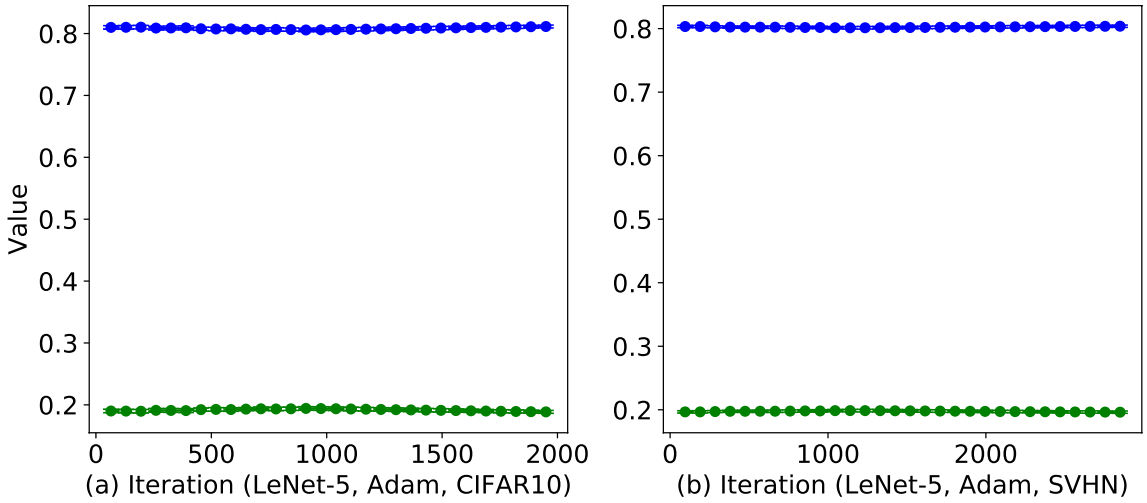

Fig. 8 Learning curves of γs for LeNet-5 with Adam-CAM-HD on CIFAR10 and SVHN ($\tau = 0.002$)

6.2 Learning of combination weights

The following figures including Figs. 6, 7, 8 and 10 give the learning curves of combination weights with respect to the number of training iterations in each experiments, in which each curve is averaged by 5 trials with error bars. Through these figures, we can compare the updating curves with different models, different datasets and different CAM-HD optimizers.

Figure 6 corresponds to the experiment of FFNN on MNIST in Sect. 3.3 of the main paper, which is a three-level case. We can see that for different FFNN architecture, the learning behaviors of γs also show different patterns, although trained on a same dataset. Meanwhile, the standard errors for multiple trials are much smaller relative to the changes of the average combination weight values.

Figure 7 corresponds to the learning curves of γs in the experiments of LeNet-5 for MNIST image classification with SGD, SGDN and Adam, which are trained on 10% of original training dataset. In addition, Fig. 8 corresponds to the learning curves of γs in the experiments of LeNet-5 for CIFAR10 and SVHN image classification with Adam-CAM-HD.

As is shown in Fig. 7, for SGD-CAM-HD, SGDN-CAM-HD and Adam-CAM-HD, the equilibrium values of combination weights are different from each other. Although the initialization $\gamma_1 = 0.2$, $\gamma_2 = 0.8$ and the updating rate $\delta = 0.03$ are set to be the same for the three optimizers, the values of γ_1 and γ_2 only change in a small proportion when training with Adam-CAM-HD, while the change is much more significant towards larger filter/layer-wise adaptation when SGD-CAM-HD or SGDN-CAM-HD is implemented. The numerical results show that for SGDN-CAM-HD, the average value of weight for layer-wise adaptation γ_1 jumps from 0.2 to 0.336 in the first epoch, then drop back to 0.324 before keeping increasing till about 0.388. For Adam-CAM-HD, the average γ_1 moves from 0.20 to 0.211 with about 5% change. In Fig. 8, both the two subplots are about LeNet-5 models trained with Adam-CAM-HD, while the exponential decay rate for weighted approximation is set to be $\tau = 0.002$. For the updating curves in Fig. 8a, which is trained on CIFAR10 with Adam-CAM-HD, the combination weight for filter-wise adaptation moves from 0.20 to 0.188. Meanwhile, for the updating curves in Fig. 8b, which is trained on SVHN, the combination weight for filter-wise adaptation moves from 0.20 to 0.195. Further exploration shows that τ has an impact on the learning curves of combination weights. As is shown by Fig. 9, a smaller $\tau = 0.001$ can result in a more significant change of combination weights during training with Adam-CAM-HD. The similar effect can also be observed from the learning curves of γs for ResNet-18, which is given in Fig. 10 and we only take the first 8000 iterations. Again, we find that in training ResNet-18 on CIFAR10, the combination weights of SGD/SGDN-CAM-HD change much faster than that of Adam-CAM-HD. There are several reasons for this effect: First, in the cases when γs do not move significantly, we apply Adam-CAM-HD, where the main learning rate (1e-3) is only about 1%-6% of the learning rate of SGD or SGDN (1e-1). In Algorithm 1, we can see that the updating rate of γs is in proportion of alpha given other terms unchanged. Thus, for the same tasks, if the same value of updating rate δ is applied, the updating scale of γs for Adam-CAM-HD can be much smaller than that for SGDN-CAM-HD. Second, this does not mean that if we apply a much larger δ for Adam-CAM-HD, the combination weights will still not change significantly or the performance will not be improved. It simply means that using a small δ can also achieve good

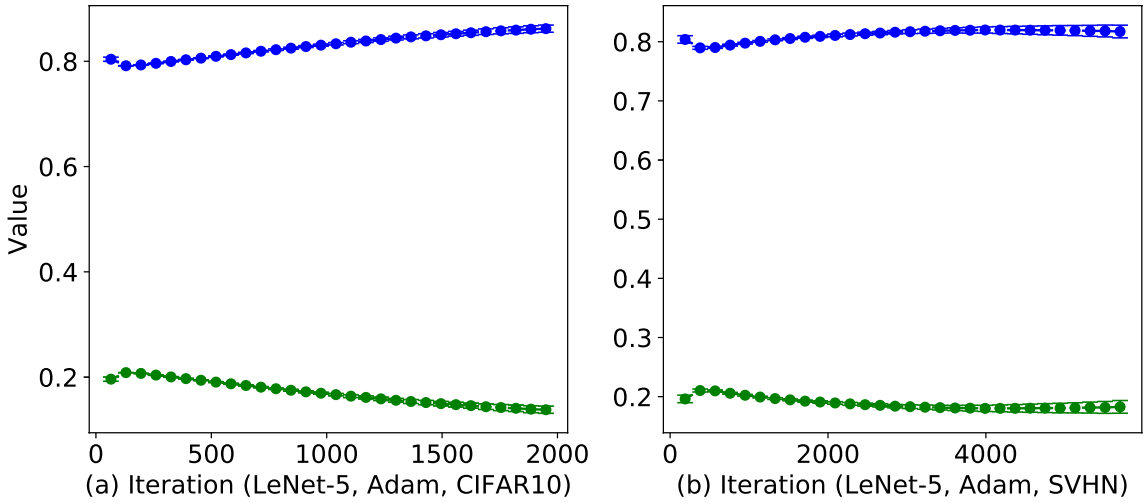

Fig. 9 Learning curves of γs for LeNet-5 with Adam-CAM-HD on CIFAR10 and SVHN ($\tau = 0.001$)

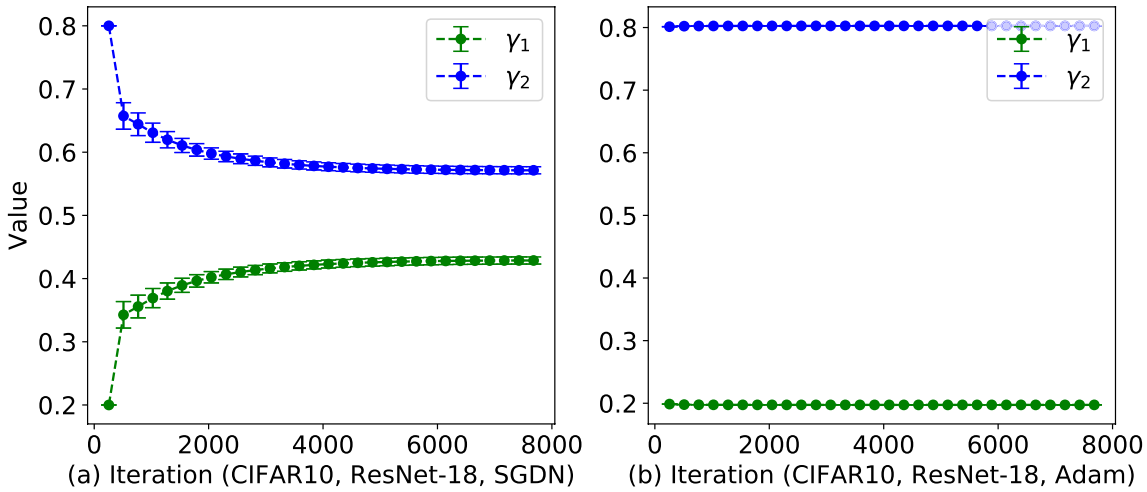

Fig. 10 Learning curves of γs for ResNet-18 with SGDN-CAM-HD and Adam-CAM-HD ($\tau = 0.001$)

performance due to the goodness of initialisation points. Third, it is possible that Adam requires lower level of combination ratio adaptation for the same network architecture compared with SGD/SGDN due to the fact that Adam itself involves stronger adaptiveness.

6.3 Number of parameters and space complexity

The proposed adaptive optimizer is for efficiently updating the model parameters, while the final model parameters will not be increase by introducing CMA-HD optimizer. However, during the training process, several extra intermediate variables are introduced. For example, in the discussed three-level's case for feed-forward neural network with n_{layer} layers, we need to restore $h_{p,t}$, $h_{l,t}$ and $h_{g,t}$, which have the sizes of $S(h_{p,t}) = \sum_{l=1}^{n_{\text{layer}}-1}(n_l+1)n_{l+1}$, $S(h_{l,t}) = n_{\text{layer}}$ and $S(h_{g,t}) = 1$, respectively, where n_i is the number of units in ith layer. Also, learning rates $\alpha_{p,t}$, $\alpha_{l,t}$, $\alpha_{g,t}$ and take the sizes of $S(a_{p,t}) = \sum_{l=1}^{n_{\text{layer}}-1}(n_l+1)n_{l+1}$, $S(a_{l,t}) = n_{\text{layer}}$, $S(a_{g,t}) = 1$, $S(a_{g,t}) = 1$, and $S(a^*_{p,t}) = \sum_{l=1}^{n_{\text{layer}}-1}(n_l+1)n_{l+1}$, respectively. Also we need a small set of scalar parameters to restore γ_1, γ_2 and γ_3 and other coefficients.

Consider the fact that the training the baseline models, we need to restore model parameters, corresponding gradients, as well as the intermediate gradients during the implementation of chain rule, CAM-HD will take twice of the space for storing intermediate variables in the worst case. For two-level learning rate adaptation considering global and layer-wise learning rates, the extra space complexity

by CAM-HD will be one to two orders' smaller than that of baseline model during training.

6.4 Time complexity

In CMA-HD, we need to calculate gradient of loss with respect to the learning rates in each level, which are $h_{p,t}$, $h_{l,t}$ and $h_{g,t}$ in three-level's case. However, the gradient of each parameter is already known during normal model training, the extra computational cost comes from taking summations and updating the lowest-level learning rates. In general, this cost is in linear relation with the number of differentiable parameters in the original models. Here we discuss the case of feed-forward networks and convolutional networks.

Recall that for feed-forward neural network the whole computational complexity is:

$$T(n) = O\left(m \cdot n_{\text{iter}} \cdot \sum_{l=2}^{n_{\text{layer}}} n_l \cdot n_{l-1} \cdot n_{l-2} \right) \tag{14}$$

where m is the number of training examples, n_{iter} is the iterations of training, n_l is the number of units in the l-th layer. On the other hand, when using three-level CAM-HD with, where the lowest level is parameter-wise, we need n_{layer} element products to calculate $h_{p,t}$ for all layers, one n_{layer} matrix element summations to calculate $h_{l,t}$ for all layers, as well as a list summation to calculate $h_{g,t}$. In addition, two element-wise summations will also be implemented for calculating $\alpha_{p,t}$ and α_p^*. Therefore, the extra computational cost of using CAM-HD is $\Delta T(n) = O(m_b \cdot n_{\text{iter}} \sum_{l=2}^{n_{\text{layer}}} (n_l \cdot n_{l-1} + n_l))$, where m_b is the number of mini-batches for training. Notice that m/m_b is the batch size, which is usually larger than 100. This extra cost is more than one-order smaller than the computation complexity of training a model without learning rate adaptation. For the cases when the lowest level is layer-wise, only one element-wise matrix product is needed in each layer to calculate $h_{l,t}$. For convolutional neural networks, we have learned that the total time complexity of all convolutional layers is [22]:

$$O\left(m \cdot n_{\text{iter}} \cdot \sum_{l=1}^{n_{conv\_layer}} (n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2) \right) \tag{15}$$

where l is the index of a convolutional layer, and n_{conv_layer} is the depth (number of convolutional layers). n_l is the number of filters in the l-th layer, while n_{l-1} is known as the number of input channels of the l-th layer. s_l is the spatial size of the filter. m_l is the spatial size of the output feature map. If we consider convolutional filters as layers, the extra computational cost for CAM-HD in this case is $\Delta T(n) = O(m_b \cdot n_{\text{iter}} \sum_{l=1}^{n_{conv\_layer}} ((n_{l-1} \cdot s_l^2 + 1) \cdot n_l))$, which is still more than one order smaller than the cost of model without learning rate adaptation.

Therefore, for large networks, applying CMA-HD will not significantly increase the computational cost from the theoretical prospective.

7 Conclusion

In this study, we propose a gradient-based learning rate adaptation strategy by introducing hierarchical learning rate structures in deep neural networks. By considering the relationship between regularization and the combination of adaptive learning rates in multiple levels, we further propose a joint algorithm for adaptively learning each level's combination weight (CAM). It increases the adaptiveness of the hyper-gradient descent method in any single level, while over-parameterization involved in optimizers can be controlled by adaptive regularization effect. In addition, both weighted approximation and clipping can be applied to guarantee the convergence. The proposed CAM algorithm is compatible with any gradient based optimizers, learning rate schedules and network architectures. Experiments on FFNN, LeNet-5, and ResNet-18/34 show that the proposed methods can outperform the standard ADAM/SGDN and other baseline methods with statistical significance.

Acknowledgements The authors acknowledge the Sydney Informatics Hub and the University of Sydney's high performance computing cluster Artemis for providing the high performance computing resources that have contributed to the research results reported within this paper.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

References

1. Alacaoglu A, Malitsky Y, Mertikopoulos P, Cevher V (2020) A new regret analysis for adam-type algorithms. In: International conference on machine learning, PMLR, pp 202–210
2. Almeida LB, Langlois T, Amaral JD, Plakhov A (1998) Parameter adaptation in stochastic optimization. On-line learning in neural networks, Publications of the Newton Institute, pp 111–134
3. Amari S (1993) Backpropagation and stochastic gradient descent method. Neurocomputing 5(4–5):185–196

4. Andrychowicz M, Denil M, Gomez S, Hoffman MW, Pfau D, Schaul T, Shillingford B, De Freitas N (2016) Learning to learn by gradient descent by gradient descent. In: NeurIPS, pp 3981–3989
5. Anil R, Gupta V, Koren T, Singer Y (2019) Memory efficient adaptive optimization. Adv Neural Inf Process Syst 32
6. Baydin AG, Cornish R, Rubio DM, Schmidt M, Wood F (2017) Online learning rate adaptation with hypergradient descent. ICLR
7. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM (2018) Automatic differentiation in machine learning: a survey. J Mach Learn Res 18:1–43
8. Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. In: NeurIPS, neural information processing systems foundation, vol 24
9. Chen Z, Xu Y, Chen E, Yang T (2018) Sadagrad: strongly adaptive stochastic gradient methods. In: International conference on machine learning, PMLR, pp 913–921
10. Darken C, Moody J (1990) Note on learning rate schedules for stochastic optimization. Adv Neural Inf Process Syst 3
11. DeVries T, Taylor GW (2017) Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552
12. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. JMLR 12:2121–2159
13. Feurer M, Hutter F (2019) Hyperparameter optimization. Automated machine learning. Springer, Cham, pp 3–33
14. Fine TL (2006) Feedforward neural network methodology. Springer, Berlin
15. Floridi L, Chiriatti M (2020) Gpt-3: its nature, scope, limits, and consequences. Mind Mach 30(4):681–694
16. Franceschi L, Donini M, Frasconi P, Pontil M (2017) Forward and reverse gradient-based hyperparameter optimization. In: ICML, JMLR. org, pp 1165–1173
17. Fu J, Ng R, Chen D, Ilievski I, Pal C, Chua TS (2017) Neural optimizers with hypergradients for tuning parameter-wise learning rates. In: JMLR: workshop and conference proceedings, vol 1, pp 1–8
18. Ge R, Kakade SM, Kidambi R, Netrapalli P (2018) Rethinking learning rate schedules for stochastic optimization. In: Submission to ICLR, pp 1842–1850
19. Ge R, Kakade SM, Kidambi R, Netrapalli P (2019) The step decay schedule: a near optimal, geometrically decaying learning rate procedure for least squares. In: Advances in neural information processing systems, pp 14977–14988
20. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press, Cambridge
21. Gusak J, Cherniuk D, Shilova A, Katrutsa A, Bershatsky D, Zhao X, Eyraud-Dubois L, Shlyazhko O, Dimitrov D, Oseledets I, Beaumont O (2022) Survey on large scale neural network training. arXiv: 2202.10435
22. He K, Sun J (2015) Convolutional neural networks at constrained time cost. In: CVPR, pp 5353–5360
23. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: CVPR, pp 770–778
24. Hecht-Nielsen R (1992) Theory of the backpropagation neural network. Neural networks for perception. Elsevier, Oxford, pp 65–93
25. Karimi H, Nutini J, Schmidt M (2016) Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 795–811
26. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. ICLR
27. Krizhevsky A, Hinton G (2012) Learning multiple layers of features from tiny images. University of Toronto
28. Lang H, Xiao L, Zhang P (2019) Using statistics to automate stochastic optimization. In: Advances in neural information processing systems, pp 9540–9550
29. LeCun Y, Touresky D, Hinton G, Sejnowski T (1988) A theoretical framework for back-propagation. In: Proceedings of the 1988 connectionist models summer school, vol 1, pp 21–28
30. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
31. LeCun Y et al (2015) Lenet-5, convolutional neural networks. http://yannlecuncom/exdb/lenet20:5
32. Li Z, Arora S (2019) An exponential learning rate schedule for deep learning. In: International conference on learning representations
33. Li X, Orabona F (2019) On the convergence of stochastic gradient descent with adaptive stepsizes. In: The 22nd International conference on artificial intelligence and statistics, PMLR, pp 983–992
34. Liu L, Jiang H, He P, Chen W, Liu X, Gao J, Han J (2019) On the variance of the adaptive learning rate and beyond. In: ICLR
35. Loshchilov I, Hutter F (2017) Sgdr: stochastic gradient descent with warm restarts. In: ICLR
36. Luo L, Xiong Y, Liu Y, Sun X (2018) Adaptive gradient methods with dynamic bound of learning rate. In: ICLR
37. Lv K, Jiang S, Li J (2017) Learning gradient descent: better generalization and longer horizons. In: ICML, JMLR. org, pp 2247–2255
38. Maclaurin D, Duvenaud D, Adams R (2015) Gradient-based hyperparameter optimization through reversible learning. In: ICML, pp 2113–2122
39. McMahan HB, Streeter M (2010) Adaptive bound optimization for online convex optimization. arXiv:1002.4908
40. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY (2011) Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning 2011
41. O'donoghue B, Candes E (2015) Adaptive restart for accelerated gradient schemes. Found Comput Math 15(3):715–732
42. Prechelt L (1998) Early stopping-but when? Neural networks: tricks of the trade. Springer, Berlin, pp 55–69
43. Reddi SJ, Kale S, Kumar S (2018) On the convergence of adam and beyond. In: ICLR
44. Reddi SJ, Kale S, Kumar S (2019) On the convergence of adam and beyond. In: International conference on learning representations
45. Rolinek M, Martius G (2018) L4: Practical loss-based stepsize adaptation for deep learning. In: NeurIPS, pp 6433–6443
46. Ruder S (2016) An overview of gradient descent optimization algorithms. arXiv:1609.04747
47. Savarese P (2019) On the convergence of adabound and its connection to sgd. arXiv:1908.04457
48. Schraudolph NN (1999) Local gain adaptation in stochastic gradient descent. In: 1999 Ninth international conference on artificial neural networks ICANN 99. (Conf. Publ. No. 470), vol 2, pp 569–574
49. Subramanian V (2018) Deep Learning with PyTorch: a practical approach to building neural network models using PyTorch. Packt Publishing Ltd
50. Sun R (2019) Optimization for deep learning: theory and algorithms. arXiv:1912.08957
51. Sutton RS (1992) Gain adaptation beats least squares. In: Proceedings of the 7th Yale workshop on adaptive and learning systems, vol 161168
52. Svozil D, Kvasnicka V, Pospichal J (1997) Introduction to multilayer feed-forward neural networks. Chemom Intell Lab Syst 39(1):43–62

53. Tieleman T, Hinton G (2012) Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Tech Rep, Technical report p 31
54. Wang G, Lu S, Cheng Q, Tu Ww, Zhang L (2019) Sadam: A variant of adam for strongly convex functions. In: International conference on learning representations
55. Wang M, Fu W, He X, Hao S, Wu X (2020) A survey on large-scale machine learning. IEEE Trans Knowl Data Eng:1–1
56. Wichrowska O, Maheswaranathan N, Hoffman MW, Colmenarejo SG, Denil M, de Freitas N, Sohl-Dickstein J (2017) Learned optimizers that scale and generalize. In: ICML, JMLR. org, pp 3751–3760
57. You Y, Gitman I, Ginsburg B (2017) Scaling sgd batch size to 32k for imagenet training. arXiv:1708.03888
58. You Y, Li J, Reddi S, Hseu J, Kumar S, Bhojanapalli S, Song X, Demmel J, Keutzer K, Hsieh CJ (2019) Large batch optimization for deep learning: training bert in 76 minutes. In: ICLR
59. Yu J, Aberdeen D, Schraudolph NN (2006) Fast online policy gradient learning with smd gain vector adaptation. In: NeurIPS, pp 1185–1192
60. Zaheer M, Reddi S, Sachan D, Kale S, Kumar S (2018) Adaptive methods for nonconvex optimization. Adv Neural Inf Process Syst 31
61. Zeiler MD (2012) Adadelta: an adaptive learning rate method. arXiv:1212.5701
62. Zhang M, Lucas J, Ba J, Hinton GE (2019) Lookahead optimizer: k steps forward, 1 step back. In: NeurIPS, pp 9593–9604
63. Zhang P, Lang H, Liu Q, Xiao L (2020) Statistical adaptive stochastic gradient methods. arXiv:2002.10597
64. Zhuang J, Tang T, Ding Y, Tatikonda SC, Dvornek N, Papademetris X, Duncan J (2020) Adabelief optimizer: adapting stepsizes by the belief in observed gradients. Adv Neural Inf Process Syst 33:18795–18806