ADAS: ADAPTIVE SCHEDULING OF STOCHASTIC GRADIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

The choice of learning rate has been explored in many stochastic optimization frameworks to adaptively tune the step-size of gradients in iterative training of deep neural networks. While adaptive optimizers (e.g. Adam, AdaGrad, RMSProp, AdaBound) offer fast convergence, they exhibit poor generalization characteristics. To achieve better performance, the manual scheduling of learning rates (e.g. step-decaying, cyclical-learning, warmup) is often used but requires expert domain knowledge. It provides limited insight into the nature of the updating rules and recent studies show that different generalization characteristics are observed with different experimental setups. In this paper, rather than raw statistic measurements from gradients (which many adaptive optimizers undertake), we explore the useful information carried between gradient updates. We measure the energy norm of the low-rank factorization of convolution weights in a convolution neural network to define two probing metrics; *"knowledge gain"* and *"mapping condition"*. By means of these metrics, we provide empirical insight into the different generalization characteristics of adaptive optimizers. Further, we propose a new optimizer—*AdaS*—to adaptively regulate the learning rate by tracking the rate of change in knowledge gain. Experimentation in several setups reveals that AdaS exhibits faster convergence and superior generalization over existing adaptive learning methods.

1 INTRODUCTION

Stochastic Gradient Descent (SGD), a first-order optimization method Robbins & Monro (1951); Bottou (2010; 2012), has become the mainstream method for training over-parametrized Deep Neural Network (DNN) models LeCun et al. (2015); Goodfellow et al. (2016). Attempting to augment this method, mSGD (SGD with momentum) Polyak (1964); Sutskever et al. (2013); Yuan et al. (2016); Loizou & Richtárik (2020) accumulates the historically aligned gradients which helps in navigating past ravines and towards a more optimal solution. It is less sensitive toward falling in local-minimas for optimization. However, as the step-size (aka global learning rate) is mainly fixed for mSGD, it blindly follows these past gradients by overshooting an optimum with oscillatory behavior and leads to poorer convergence Bengio (2012); Schaul et al. (2013).

A handful of methods have been introduced over the past decade to solve the latter issues based on the adaptive gradient methods [refer to Section 1.1 for list of references]. These methods can be represented in the general form of

$$\boldsymbol{w}^k \longleftarrow \boldsymbol{w}^{k-1} - \frac{\eta_k}{\psi(\boldsymbol{g}_1, \cdots, \boldsymbol{g}_k)} \phi(\boldsymbol{g}_1, \cdots, \boldsymbol{g}_k), \tag{1}$$

where, for some kth iteration, $\boldsymbol{g}_i$ is the stochastic gradient obtained at the ith iteration, $\phi(g_1, \cdots, g_k)$ is a modifier for gradient updating, and $\eta_k/\psi(\boldsymbol{g}_1, \cdots, \boldsymbol{g}_k)$ is the adaptive learning function e.g. first and second order statistics measure in Adam Kingma & Ba (2014). Each update therefore is not solely reliant on the current gradients, but also depends on the historical aggregation of the past gradients. The adaptive learning rate is also selected via different choice of $\psi(\cdot)$ to tune the gradient magnitudes for updating.

The Adam optimizer, as well as its other variants, has attracted many practitioners in deep learning for two main reasons: (1) it provides an efficient convergence optimization framework; and (2) it

requires minimal hyper-parameter tuning effort. Despite the ease of implementation of such optimizers, there is a growing concern about their poor *"generalization"* characteristics. They perform well on the given-samples i.e. training data, but perform poorly on the out-of-samples i.e. test/evaluation data (Keskar et al., 2017; Wilson et al., 2017; Li et al., 2019; Yu & Zhu, 2020). In retrospect, the non-adaptive SGD based optimizers (such as scheduled learning by Warmup Techniques Loshchilov & Hutter (2017), Cyclical-Learning Smith (2017); Smith & Topin (2019), and Step-Decaying Goodfellow et al. (2016)) are still dominantly used in training CNNs to achieve high performance at the price of either more epochs for training and/or costly tuning for optimal hyper-parameter configurations; given different datasets and models.

Unfortunately, such scheduling techniques are human-intuitive-based approaches. For instance, in step-decaying method, the learning rate is suggested to start from a high value and then drop in several epoch steps Goodfellow et al. (2016); Loshchilov & Hutter (2017), or it is suggested to start from small value and increase as the epoch training proceeds to drop down again to finalize training Smith (2017); Smith & Topin (2019). Unfortunately, there is little understood from such methods why they should really work? It really becomes more like an *alchemy* rather than analytical/empirical reasoning. Recent investigations reveal that these two different scheduling scenario actually cause different generalization characteristics Li et al. (2019). This simply creates a high threshold for ML practitioners.

Our goal in this paper is twofold: (1) we introduce new probing metrics that enable the monitoring of the well-posedness of learning layers in Convolution Neural Network (CNN); and (2) we propose a new learning function ψ and accordingly tailor an adaptive optimization algorithm. Unlike the general trend where the raw gradients are used in $\psi(\cdot)$ for adapting the step-size (by some sort of statistics measure), we avoid direct measuring of the gradients and associate this learning function with one of the proposed metrics called the *"knowledge gain"*. This gain is measured by the energy norm of low-rank factorization of convolution weights which basically defines *"the useful knowledge carried over gradient updates"*. Our main contributions are as follows.

Probing metrics. The concepts of *"Knowledge Gain - ($\mathcal{G}$)"* and *"Mapping Condition - ($\mathcal{C}$)"* are introduced to quantify the well-posedness of convolutional layers. The gain $\mathcal{G}$ encodes the useful information carried over each convolution layer and the condition $\mathcal{C}$ encodes the numerical stability of feature mapping from input to the output layer. Using the probing metrics, we provide empirical evidences on the generalization performance of different stochastic optimizers and explain how different layers in CNN can function as a better autoencodeer over epoch training.

AdaS. A new adaptive optimizer is introduced by defining the learning function (that is independent to each network layer) inversely proportional to the difference of knowledge gain over consecutive epochs i.e. $\psi(\cdot) \propto 1/[\mathcal{G}(\boldsymbol{w}^{t+1}) - \mathcal{G}(\boldsymbol{w}^{t})]$. We associate the final learning rate with a *"gain factor - (β)"* by exponential decaying average of learning functions over epoch training. The factor trades-off between fast convergence and optimum performance which we leave it up to the user as a preference choice to either save training time or reach to a maximum possible accuracy.

Experimentation. Thorough experiments are conducted in the context of image classification problem using various dataset and CNN models. The empirical results reveal that AdaS converges much faster while maintaining superior generalization ability.

1.1 Related Works

The first adaptive optimizer was introduced in Duchi et al. (2011) (AdaGrad) by regulating the update size with the accumulated second order statistical measures of gradients. The issue of vanishing learning rate caused by equally weighted accumulation of gradients is the main drawback of AdaGrad that was raised in Tieleman & Hinton (2012) (RMSProp), which utilizes the exponential decaying average of gradients instead of accumulation. A variant of first-order gradient measures was also introduced in Zeiler (2012) (AdaDelta), which solves the decaying learning rate problem using an accumulation window. The adaptive moment estimation in Kingma & Ba (2014) (Adam) was introduced later to leverage both first and second moment measures of gradients. Adam solves the vanishing learning rate problem and offers a more optimal adaptive learning rate to improve in rapid convergence and generalization capabilities. Further improvements were made on Adam using Nesterov Momentum Dozat (2016), long-term memory of past gradients Reddi et al. (2018), rectified estimations Liu et al. (2020), dynamic bound of learning rate Luo et al. (2019), hyper-gradient

descent method Baydin et al. (2018), and loss-based step-size Rolinek & Martius (2018). Methods based on line-search techniques Vaswani et al. (2019) and coin betting Orabona & Tommasi (2017) are also introduced to avoid bottlenecks caused by the hyper-parameter tuning issues.

2 On New Metric Measure Tools

In this section we develop new metrics that can be probed in intermediate layers of CNN to measure the well-posedness of each convolution layer. We first adopt the low-rank factorization for CNN.

2.1 Low-Rank Factorization of Convolution Weights

Consider the convolutional weights of a CNN layer defined by a four-way array (aka fourth-order tensor) $\mathbf{W} \in \mathbb{R}^{N_1 \times N_2 \times N_3 \times N_4}$, where N_1 and N_2 are the height and width of the kernels, and N_3 and N_4 the input and output channel sizes, respectively. The feature mapping under this convolution operates by $\mathbf{F}^O_{:,:,i_4} = \sum_{i_3=1}^{N_3} \mathbf{F}^I_{:,:,i_3} * \mathbf{W}_{:,:,i_3,i_4}$, where $\mathbf{F}^I$ and $\mathbf{F}^O$ are the input and output feature maps (stacked in 3D volumes), and $i_4 \in \{1, \ldots, N_4\}$ is the output index. The well-posedness of this feature mapping can be studied by the generalized spectral decomposition (i.e. SVD) of tensor arrays using the Tucker model Kolda & Bader (2009); Sidiropoulos et al. (2017) $\mathbf{W} = \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \sum_{i_3=1}^{N_3} \sum_{i_4=1}^{N_4} \mathbf{G}_{i_1,i_2,i_3,i_4} \boldsymbol{u}_{i_1} \circledcirc \boldsymbol{u}_{i_2} \circledcirc \boldsymbol{u}_{i_3} \circledcirc \boldsymbol{u}_{i_4}$, where, the core $\mathbf{G}$ (containing singular values) is called a (N_1, N_2, N_3, N_4)-tensor, $\boldsymbol{u}_{i_k} \in \mathbb{R}^{N_k}$ is the factor basis for decomposition, and $\circledcirc$ is outer product operation.

The weight structure $\mathbf{W}$ is initialized at random in CNN training where new structure will be learned throughout iteration training lying in the tensor space; presumably in a mixture of a low-rank manifold and perturbing noise. In fact, this is the main rationale behind many CNN compression methods (such as in Lebedev et al. (2015); Tai et al. (2016); Kim et al. (2016); Yu et al. (2017)). This is mainly done by factorizing the observing weights $\mathbf{W} = \widehat{\mathbf{W}} + \mathbf{E}$, where, $\widehat{\mathbf{W}}$ corresponds to the low-rank tensor pertinent to the small-core tonsor. A handful of techniques are deployed to approximate the small core tensor by minimizing the error residues $\mathbf{E} = ||\mathbf{W} - \widehat{\mathbf{W}}||_{\mathrm{F}}^2$ Kolda & Bader (2009); Oseledets (2011); Grasedyck et al. (2013); Sidiropoulos et al. (2017). Most solutions, however, cast iterative algorithms and create computation burden.

Here, we consider vector form $\mathrm{vec}\,(\mathbf{W}) = (\boldsymbol{U}_1 \otimes \boldsymbol{U}_2 \otimes \boldsymbol{U}_3 \otimes \boldsymbol{U}_4)\,\mathrm{vec}\,(\mathbf{G})$, where $\mathrm{vec}\,(\cdot)$ is a vector obtained by stacking all tensor elements column-wise, $\otimes$ is the Kronecker product, and $\boldsymbol{U}_i$ is a factor matrix containing all bases $\boldsymbol{u}_{i_k}$ stacked in column. To decompose input and output channels of convolution weights, we use mode-3 and mode-4 vector expressions

$$\boldsymbol{W}_3 = (\boldsymbol{U}_1 \otimes \boldsymbol{U}_2 \otimes \boldsymbol{U}_4)\,\boldsymbol{G}_3 \mathbf{U}_3^T \quad \text{and} \quad \boldsymbol{W}_4 = (\mathbf{U}_1 \otimes \mathbf{U}_2 \otimes \mathbf{U}_3)\,\boldsymbol{G}_4 \mathbf{U}_4^T, \tag{2}$$

where, $\boldsymbol{W}_3 \in \mathbb{R}^{N_1 N_2 N_4 \times N_3}$, $\boldsymbol{W}_4 \in \mathbb{R}^{N_1 N_2 N_3 \times N_4}$, and $\boldsymbol{G}_3$ and $\boldsymbol{G}_4$ are likewise reshaped forms of the core tensor $\mathbf{G}$. The vector decomposition in 2 is a two-way array (aka matrix) representation e.g. $\boldsymbol{W}_3 = \boldsymbol{U}\Lambda\boldsymbol{V}^T$ where $\boldsymbol{U} \equiv \boldsymbol{U}_1 \otimes \boldsymbol{U}_2 \otimes \boldsymbol{U}_4$, $\boldsymbol{V} \equiv \boldsymbol{U}_3$, and $\Lambda \equiv \boldsymbol{G}_3$ Sidiropoulos et al. (2017). In other words, to decompose a tensor on a given mode, the tensor is unfolded first and a decomposition of interest is applied (such as SVD) i.e. $\mathbf{W} \xrightarrow{\texttt{unfold}} \boldsymbol{W}_3 \xrightarrow{\texttt{decompose}} \boldsymbol{U}\Lambda\boldsymbol{V}^T$. The noise presence, however, prevents better understanding of the latter reshaped forms. Similar to Lebedev et al. (2015), we revise our goal into low-rank matrix factorizations of $\boldsymbol{W}_3 = \widehat{\boldsymbol{W}}_3 + \boldsymbol{E}_3$ and $\boldsymbol{W}_4 = \widehat{\boldsymbol{W}}_4 + \boldsymbol{E}_4$. The global analytical solution is given by the Variational Baysian Matrix Factorization (VBMF) technique in Nakajima et al. (2013) as a re-weighted SVD of the observation matrix. This method avoids unnecessary implementing an iterative algorithm. Figures 4.2 and 1(b) demonstrate the singular values obtained by decomposing multiple layers of VGG16 network.

2.2 Well-Posed Structure of Convolution Weights

At the core of our metric definition is the *"well-posed"* structure of convolution weights in CNN and how they effect the casecade mapping of the features along the network layers. Recall the low-rank factorization of a tensor weight to be decomposed by the following procedure

$$\mathbf{W}\ (\text{Tensor}) \xrightarrow{\texttt{unfold}} \boldsymbol{W}_d\ (\text{2D-Matrix}) \xrightarrow{\texttt{Factorize}} \widehat{\boldsymbol{W}}_d\ (\text{Low-Rank}) + \boldsymbol{E}_d.$$

We importantly factorize $\boldsymbol{W}_d = \widehat{\boldsymbol{W}}_d + \boldsymbol{E}_d$ where $\boldsymbol{E}_d$ is some perturbing noise that we subsequently ignore and $\widehat{\boldsymbol{W}}_d$ is a low-rank factorized matrix of $\boldsymbol{W}_d$. In this formulation, the randomness of initialization is captured by $\boldsymbol{E}_d$ and thus filtered when ignore it, and we apply SVD on the low-rank factorized matrix $\widehat{\boldsymbol{W}}_d = \widehat{\boldsymbol{U}}_d \widehat{\Lambda}_d \widehat{\boldsymbol{V}}_d^T$ to determine the structural characteristics of the weights. We refer to $\widehat{\Lambda}_d$ as being the singular values of the low-rank matrix. We now define the "well-posedness" by

1. the summation of singular values $\sum \widehat{\Lambda}_d$ should be high to guarantee the space-span (under the convolution layer mapping) is mainly encoded by the low-rank structure, but not the noise perturbation (aka the low-rank energy is dominant $\|\widehat{\boldsymbol{W}}_d\| \gg \|\boldsymbol{E}_d\|$)
2. the relative ratio of the highest to the lowest singular values $\widehat{\Lambda}_d^{\max}/\widehat{\Lambda}_d^{\min}$ should be low to guarantee the perturbation of feature-space (under the convolution layer mapping) is mainly related to the noise sampling but not the low-rank structure

Under the above assumptions, one can claim that the feature mapping by the convolution layer can pose a meaningful structure for feature decomposition i.e. layers can better fit the training data.

2.3 Defining Metrics

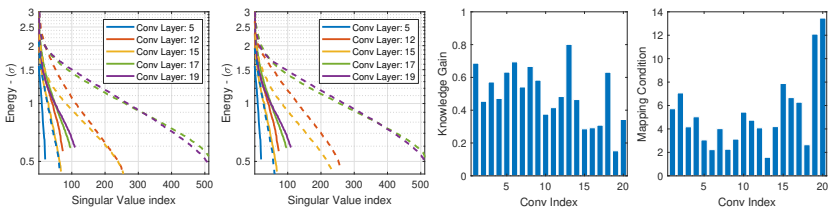
Following the conditions of well-posed structure, we introduce the following two definitions.

Definition 1. *(Knowledge Gain–$\mathcal{G}$). The knowledge gain across a particular channel (dth dimension) of a layer in CNN is defined by*

$$\mathcal{G}_d^p(\mathbf{W}) = \frac{1}{N_d\ \sigma_1^p(\widehat{\boldsymbol{W}}_d)} \sum_{k=1}^{N'_d} \sigma_k^p(\widehat{\boldsymbol{W}}_d), \tag{3}$$

where, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{N'_d}$ are the low-rank singular values of a single-channel convolutional weight in descending order, $N'_d = \operatorname{rank} \widehat{\boldsymbol{W}}_d$, and $p \in \{1, 2\}$.

The remark of the Definition 1 is given in subsection A.3. The knowledge gain is a direct measure of the normalized energy of low-rank structure of convolution weights. The division factor N_d in (3) normalizes the gain $\mathcal{G}^p \in [0, 1]$ as a fraction of channel capacity. The definition in Equation 3 is closely related to the stable-rank defined in Rudelson & Vershynin (2007). Note that we apply $\mathcal{G}_d$ on channels $d = \{3, 4\}$ (but not kernel size i.e. $d = \{1, 2\}$) to measure the gain through input/output feature mapping. Figure 1(c) demonstrates the distribution of knowledge gain cross all ResNet18 convolution layers trained on ImageNet. Throughout many experimental setup, we noticed a consistent behavior where the knowledge gain of early layers are statistically significant compared to the proceeding layers.



(a) Mode-3 Decomposition (b) Mode-4 Decomposition (c) $\mathcal{G}_4^1(\mathbf{W}^{\texttt{ResNet18}})$ (d) $\mathcal{C}_4(\mathbf{W}^{\texttt{ResNet18}})$

Figure 1: Metric measurements using low-rank factorization of convolution weights from ResNet18 trained on ImageNet. Distribution of singular values are shown for (a) input channel and (b) output channel; on multiple convolution layers of ResNet18. The solid and dashed lines correspond to with and without low-rank factorization. The metrics of knowledge gain and mapping condition are also shown in (c) and (d) cross all conv layers of ResNet18, respectively.

Definition 2. *(Mapping Condition–$\mathcal{C}$). The mapping condition across a particular channel (dth dimension) of a layer in CNN is defined by*

$$\mathcal{C}_d(\mathbf{W}) = \sigma_1(\widehat{\boldsymbol{W}}_d)/\sigma_{N'_d}(\widehat{\boldsymbol{W}}_d), \tag{4}$$

where, σ_1 and $\sigma_{N'_d}$ are the maximum and minimum low-rank singular values of a single-channel convolutional weight, respectively.

The notion of mapping condition in Definition 2 is adopted from the matrix analysis theory Horn & Johnson (2012) where it measures the relative ratio of maximum to minimum low-rank singular values. The number indicates the sensitivity of convolution mapping with respect to minor input perturbations. High condition number pertains to poor stability and vice versa. Figure 1(d) demonstrates the distribution of mapping condition cross all ResNet18 convolution layers trained on ImageNet.

3 Learning Function $\psi(\cdot)$ by Knowledge Gain

Our goal here is to motivate and introduce a new learning function $\psi(\cdot)$ for SGD update in (1). Recall the convolution weight training in deep CNN Bottou (2010; 2012); Loizou & Richtárik (2020), where the objective is to minimize an associated loss function given a train dataset $f(\boldsymbol{W}; (X)^{train})$. The update rule for SGD minimization therefore is given by

$$\boldsymbol{W}^k \longleftarrow \boldsymbol{W}^{k-1} - \eta_k \overline{\nabla f}_k(\boldsymbol{W}^{k-1}) \quad \text{for} \quad k \in \{(t-1)K+1, \cdots, tK\}, \tag{5}$$

where, t and K correspond to epoch index and number of mini-batches, respectively, $\overline{\nabla f}_k(\boldsymbol{W}^{k-1}) = 1/|\Omega_k| \sum_{i \in \Omega_k} \nabla f_i(\boldsymbol{W}^{k-1})$ is the average stochastic gradients on kth mini-batch that are randomly selected from a batch of n-samples $\Omega_k \subset \{1, \cdots, n\}$, and η_k defines the step-size taken toward the opposite direction of average gradients.

We aim to update the learning function once at each epoch and therefore the step-size will be a function of epoch index i.e. $\eta_k \equiv \eta(t)$. We now setup our problem by accumulating all observed gradients throughout K mini-batch updates fit in one epoch training

$$\boldsymbol{W}^t = \boldsymbol{W}^{t-1} - \eta(t-1) \overline{\nabla f}_t, \tag{6}$$

where, $\overline{\nabla f}_t = \sum_{k=(t-1)K+1}^{tK} \overline{\nabla f}_k(\boldsymbol{W}^{k-1})$ corresponds to the total accumulated gradients in one epoch training. The idea here is to select a learning rate $\eta(t)$ such that knowledge gain from one epoch training to another is increased i.e. $\psi = \{\eta(t) : \mathcal{G}(\boldsymbol{W}^t) \geq \mathcal{G}(\boldsymbol{W}^{t-1})\}$. *Why is that?* From the definition of knowledge gain in previous section, higher $\mathcal{G}(\boldsymbol{W})$ corresponds to a better encoder such that the index of separability will be increased for better space spanning via convolution mapping. Here we provide a satisfying conditions on the step-size.

Theorem 1. *(Increasing Knowledge Gain for Vanilla SGD). Let the knowledge gain to be defined by Equation 3. Starting with an initial learning rate $\eta(0) > 0$ and by setting the step-size of vanilla Stochastic Gradient Descent (SGD) proportionate to*

$$\eta(t) \triangleq \zeta \left[\mathcal{G}(\boldsymbol{W}^t) - \mathcal{G}(\boldsymbol{W}^{t-1}) \right] \tag{7}$$

will guarantee the monotonic increase of the knowledge gain over consecutive epoch updates i.e. $\mathcal{G}(\boldsymbol{W}^{t+1}) \geq \mathcal{G}(\boldsymbol{W}^t)$ for some existing lower bound $\eta(t) \geq \eta_0$ and $\zeta \geq 0$.

The proof of Theorem 1 is provided in the Appendix A.

Note that with the start of a positive initial learning rate $\eta(0) > 0$ and following the update rule in (7), Theorem 1 guarantees the increase of the knowledge gain for the next epoch update. Accordingly, the positivity of learning rates using (7) will be maintained throughout consecutive epoch updates.

4 AdaS Algorithm

In this section, we introduce our scheduling algorithm AdaS for adaptive adjustment of learning-rate adopted within the mSGD framework.

4.1 Learning Rate by Exponential Decay Averaging (Gain Factor β)

The step-size defined in 7 for vanilla SGD is only measured from two consecutive epochs i.e. $t-1$ and t. Due to the stochastic nature of gradients, the increase of knowledge gain over two epochs

can fluctuate. To adapt a smooth change of learning-rate, we employ a momentum algorithm for historic accumulation of step-sizes over epoch updates. The concept is similar to AdaM but with only the difference where the updates are done at every epoch (not every mini-batch). It also worth noting that this is different from the momentum update in SGD (mSGD) where the gradients are accumulated over mini-batch updates i.e. k.

We formulate the update rule for AdaS using the mSGD framework as follows

$$\begin{aligned} &\textit{-update learning rate with momentum: } \eta(t,\ell) \leftarrow \beta\eta(t-1,\ell) + \zeta[\overline{\mathcal{G}}(t,\ell) - \overline{\mathcal{G}}(t-1,\ell)], \\ &\textit{-update gradients with momentum: } \boldsymbol{v}_\ell^k \leftarrow \alpha \boldsymbol{v}_\ell^{k-1} - \eta(t,\ell)\boldsymbol{g}_\ell^k, \\ &\textit{-weights update: } \boldsymbol{w}_\ell^k \leftarrow \boldsymbol{w}_\ell^{k-1} + v_\ell^k, \end{aligned} \tag{8}$$

where, k is the current mini-batch iteration, t is the current epoch, ℓ is the conv block index, $\overline{\mathcal{G}}(\cdot)$ is the average knowledge gain obtained from both input and output channels of convolution layers, $\boldsymbol{v}$ is the velocity term, and $\boldsymbol{w}$ are the learnable parameters. Notice there are two associated momentum parameters: gradient momentum adopted from mSGD fixed at default rate $\alpha = 0.9$, and learning rate momentum β which we call it the *"gain factor"*. Setting this parameter trades-off between faster convergence and increased performance. This is up to the user to select and it will impact the final performance and training time e.g. $\beta = 0.8$ takes ≈ 50 epochs (4.86 hours) for Tiny-ImageNet to train and converge, while for $\beta = 0.9$ this consumes twice the number of epochs ≈ 100 to converge with higher performance. An ablative study on the effect of this parameter is provided in Experiment Section 5.1 as well as in the Appendix-BD. Note that we fix $\zeta = 1$ for all experiments in this paper as it becomes a redundant parameter since the relative ratio of learning rate is now directly controlled by β.

Algorithm 1: Adaptive Scheduling (AdaS) for mSGD

Require : batch size n, # of epochs T, # of conv blocks L, initial step-sizes $\{\eta(0,\ell)\}_{\ell=1}^{L}$, initial momentum vectors $\{v_\ell^0\}_{\ell=1}^{L}$, initial parameter vectors $\{\boldsymbol{w}_\ell^0\}_{\ell=1}^{L}$, mSGD momentum rate $\alpha = 0.9$, AdaS gain factor $\beta \in [0, 1)$.

for $t = 1 : T$ **do**
  // Phase-I: mSGD optimization by adaptive learning rates
  randomly shuffle dataset, generate K mini-batches $\{\Omega_k \subset \{1, \cdots, n\}\}_{k=1}^{K}$
  for $k = (t-1)K + 1 : tK$ **do**
    1. compute gradient: $\boldsymbol{g}_\ell^k \leftarrow 1/|\Omega_k| \sum_{i\in\Omega_k} \nabla f_i(\boldsymbol{w}_\ell^{k-1}), \quad \ell \in \{1, \cdots, L\}$
    2. compute the velocity term: $\boldsymbol{v}_\ell^k \leftarrow \alpha \boldsymbol{v}_\ell^{k-1} - \eta(t,\ell)\boldsymbol{g}_\ell^k$
    3. apply update: $\boldsymbol{w}_\ell^k \leftarrow \boldsymbol{w}_\ell^{k-1} + \boldsymbol{v}_\ell^k$
  end
  // Phase-II: adaptive computation of learning rates
  for $\ell = 1 : L$ **do**
    1. unfold tensors using (2): $\boldsymbol{W}_d \leftarrow$ mode-d $\left(\mathbf{W}_\ell^t\right)$ for $d = \{3, 4\}$
    2. factorize low-ranks Nakajima et al. (2013): $\widehat{\boldsymbol{W}}_d \leftarrow \texttt{EVBMF}\left(\boldsymbol{W}_d\right)$ for $d = \{3, 4\}$
    3. compute average knowledge gain using (3): $\overline{\mathcal{G}}(t,\ell) \leftarrow [\mathcal{G}_3^1(\mathbf{W}) + \mathcal{G}_4^1(\mathbf{W})]/2$
    4. compute learning rate momentum:
     $\eta(t+1,\ell) \leftarrow \beta\eta(t-1,\ell) + [\overline{\mathcal{G}}(t,\ell) - \overline{\mathcal{G}}(t-1,\ell)]$
    5. lower bound the learning rate: $\eta(t+1,\ell) \leftarrow \max\left(\eta(t+1,\ell), 0\right)$
  end
end

4.2 ALGORITHM

The pseudo-code for our proposed algorithm *AdaS* is presented in Algorithm 1. Each convolution block in CNN is assigned with an index $\{\ell\}_{\ell=1}^{L}$ where all learnable parameters (e.g. conv, biases, batch-norms, etc) are called using this index. The goal in AdaS is firstly to callback the convolution weights of each layer, secondly compute the overall knowledge gain $\overline{\mathcal{G}}(t,\ell)$ of each layer using (3), thirdly compute the difference gain using (7), and finally accumulate this difference value in exponential decay averaging (as discussed in previous section) to compute the associated learning rate of current epoch t to plug in the mSGD optimization framework. We note here that the initial knowl-

edge gain yields zero $\overline{\mathcal{G}}(0,\ell)=0$ for all conv layers due to random initialization of weights. Figure 2 (left plot) demonstrates an evolution example of the learning rate adapted by AdaS algorithm. Interestingly, the behavior of learning rate coincides with cyclical learning scheduling in Smith (2017); Smith & Topin (2019). We believe our method explains the intuition behind such scheduling technique that why the rate is suggested to start with small rate $\rightarrow$ increase with proceeding epochs $\rightarrow$ and then decrease to finalize training.

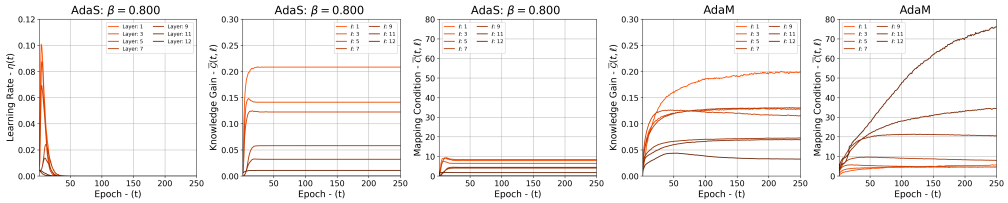


Figure 2: Metric evaluation of two different optimizers: AdaS$^{\beta=0.8}$ and AdaM. Both average knowledge gain $\overline{\mathcal{G}}(t,\ell)$ and average mapping condition $\overline{\mathcal{C}}(t,\ell)$ are monitored for duration of all epoch training of VGG16 on CIFAR10. The evolution of learning rate $\eta(t)$ are also shown in left. The plots with different color shades correspond to multiple layers of the network.

Here we provide the convergence rate for AdaS.

Theorem 2. *(AdaS convergance [adopted from Loizou & Richtárik (2020)]). Let $\{\boldsymbol{w}^k\}_{\infty}^{k=0}$ be the random sequence generated by AdaS algorithm in 1, and $\boldsymbol{w}^*$ be the converging point. Assume $0 \leq \eta(t) < 2$ and $\alpha \geq 0$, and the expressions $a_1(t) \triangleq 1+3\alpha+2\alpha^2-(\eta(t)(2-\eta(t))+\alpha\eta(t))\,\lambda^{+}_{\min}$ and $a_2(t) \triangleq \alpha+2\alpha^2+\alpha\eta(t)\lambda_{\max}$ satisfy $a_1(t)+a_2(t)<1$. Then the convergence rate of the generated sequences is bounded by*

$$\mathbb{E}\left[\|\boldsymbol{w}^k-\boldsymbol{w}^*\|_B^2\right] \leq q(t)^k(1+\delta(t))\|\boldsymbol{w}^0-\boldsymbol{w}^*\|_B^2, \tag{9}$$

and the convergence rate of the loss function is bounded by

$$\mathbb{E}\left[f(\boldsymbol{w}^k)\right] \leq q(t)^k\tfrac{\lambda_{\max}}{2}(1+\delta(t))\|\boldsymbol{w}^0-\boldsymbol{w}^*\|_B^2, \tag{10}$$

where, $q(t)=(a_1(t)+\sqrt{a_1^2(t)+4a_2(t)})/2$ and $\delta(t)=q(t)-a_1(t)$. Moreover, $a_1(t)+a_2(t) \leq q(t)<1$.

Theorem 2 is mainly adopted from Loizou & Richtárik (2020) to analyze mSGD convergence. The update rule in AdaS in fact introduces minor adjustments to the proof which we describe in Appendix A.2. Following the convergence rate in (9), within one training epoch, AdaS enjoys the convergence rule of the fixed learning rate, $0<\eta<2$ in mSGD. Further, the learning rate under AdaS continues to decreases, contracting the convergence rate in (9) i.e. $q(t)^k(1+\delta(t))$ decays quicker as the learning rate $\eta(t)$ decreases over epoch progression and hence yields a faster speed for training.

Empirical Reasoning on Generalization Ability of Optimizers. In Figure 2, the performance of AdaS optimizer is compared to AdaM by monitoring the evolution of both metrics of knowledge gain and mapping condition on multiple layers of CNN. Both optimizers gain knowledge over epoch training where the gain for early layers of network yield relatively higher value compared to the proceeding layers. We relate this to better encoding capability of the network in early stage of conv layers. We also make an extra interesting observation where AdaS yields much lower mapping condition in different conv layers (shown in different colors) as opposed to AdaM. Particularly, the last layers of AdaM yield much higher condition number. We relate this phenomena to poor generalization of AdaM. For more experimentation, please refer to Appendix D.

AdaS Computational Overhead. Note that AdaS uses mSGD's framework for adapting the learing rates. The computational overhead introduces by AdaS is only related to the low-rank factorization per epoch update using `EVBMF` method introduced in phase-II of the algorithm 1. The factorization use quadratic minimization for matrix decomposition which is linear to the matrix size. AdaS easily scales to many CNN models for optimization. In comparison, the adaptive optimizers such as Adam use different statistical model per mini-batch training where the computational overhead is very well comparable with AdaS. For instance, all mSGD+StepLR, AdaS, AdaM, RMSProp, AdaBound, and AdaGrad consume $\sim$ 40 min per epoch to train ResNet34 on ImageNet using the experimental setup defined in Appendix B.

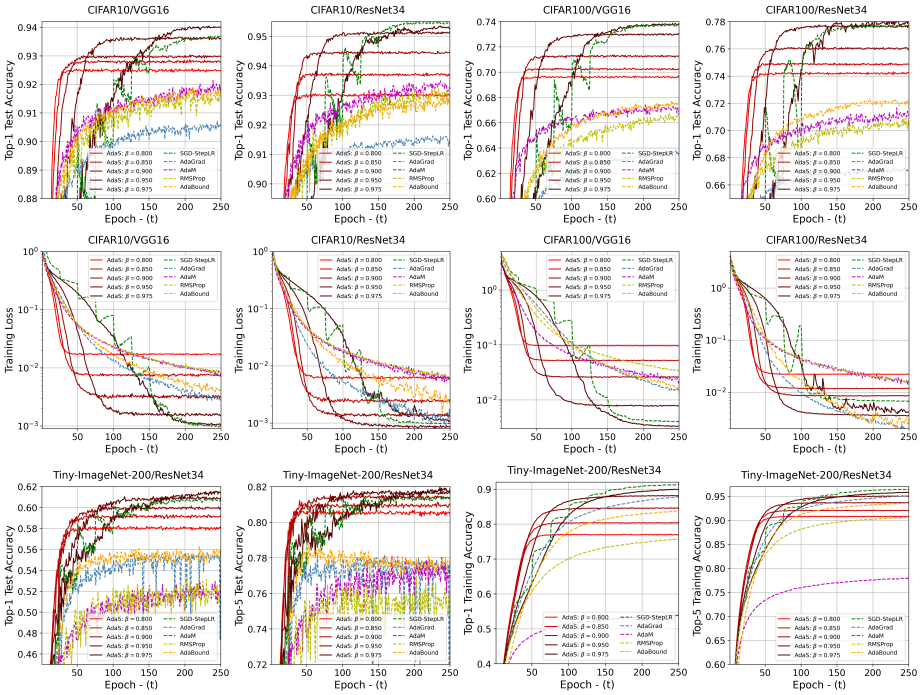


Figure 3: Training performance using different optimizers across three different datasets (i.e. CIFAR10, CIFAR100, Tiny-ImageNet-200) and two different CNNs (i.e. VGG16 and ResNet34).

5 EXPERIMENTS

We compare our AdaS algorithm to several adaptive optimizers in the context of image classification. In particular, we implement AdaGrad Duchi et al. (2011), RMSProp Goodfellow et al. (2016), AdaM Kingma & Ba (2014), AdaBound Luo et al. (2019), and SLS Vaswani et al. (2019) for comparison. We also implement mSGD-StepLR guided by scheduled dropping (step decaying) Goodfellow et al. (2016) to understand the baseline for achieving high accuracy that is extensively used in the literature for training CNNs. We further investigate the efficacy of AdaS with respect to variations in the number of deep layers using VGG16 Simonyan & Zisserman (2015), ResNet18 and ResNet34 He et al. (2016), ResNeXt50 Xie et al. (2017), and DenseNet121 Huang et al. (2017), and the number of class datasets using the standard CIFAR-10 and CIFAR-100 Krizhevsky et al. (2009), Tiny-ImageNet-200 (Li et al.), and ImageNet Russakovsky et al. (2015) for training. Please refer to Appendix B for details of experimental setup, Appendix C for additional experiments, as well as Appendix D for AdaS ablative studies.

5.1 RESULTS

Generalization Performance. The primary results of each experiment are visualized in Figure 3 (additional results are also shown in Figure 6 in Appendix D). The experiments demonstrate how well our optimizer AdaS is generalized across different dataset and network. AdaS, with low gain factors (e.g. $\beta = 0.8$), consistently outperforms all other adaptive optimizers in terms of convergence speed, as well as achieving superior test accuracy. For instance, AdaS$^{\beta=0.8}$ outperforms AdaM in CIFAR100/ResNet34 with $\sim 5.5\%$ improvement in test accuracy; within 50 epoch training. Also, both SGD-StepLR and AdaS (with higher β) eventually overtake the other methods; once enough epochs are trained. The overall generalization of SGD-StepLR however is inferior to AdaS (using high β) where AdaS yields faster convergence to achieve high accuracies; at times even better than SGD-StepLR e.g. Tiny-ImageNet-200/ResNet34, CIFAR10/VGG16, CIFAR100/VGG16, and CIFAR100/ResNet34 (see also Table 1 for tabulated results). All optimizers achieve low training losses/training accuracies, but do not exhibit equivalently to high test accuracies. Such inconsistency is more evident for adaptive optimizers i.e. AdaGrad, RMSProp, AdaM, and AdaBound. Such poor

generalization coincides with the reports made in Wilson et al. (2017); Li et al. (2019); Jastrzebski et al. (2020) where adaptive optimizers generalize worse compared to non-adaptive methods.

On Fixed Epoch-Budget Performance. We further provide quantitative results on the convergence of all optimizers trained on ResNet34 in Table 1 with fixed number of epochs. The rank consistency of AdaS (using low/high β) over other optimizers is evident. For instance, AdaS$^{\beta=0.80}$ gains 11.63% test accuracy improvement over AdaM optimizer on Tiny-ImageNet-200 trained withing 50 epochs.

Table 1: Image classification performance (test accuracy) of ResNet34 on CIFAR-10, CIFAR-100, and Tiny-ImageNet-200 with fixed budget epochs. Three adaptive (AdaBound, AdaM, AdaS) and one non-adaptive (SGD-StepLR) optimizers are deployed for comparison.

	Epoch	AdaBound	AdaM	SGD-StepLR	AdaS$^{\beta=0.8}$	AdaS$^{\beta=0.9}$	AdaS$^{\beta=0.975}$
CIFAR-10	50	$90.84_{\pm 0.17}$	$91.54_{\pm 0.35}$	$86.53_{\pm 1.67}$	$93.04_{\pm 0.13}$	$\mathbf{94.35_{\pm 0.22}}$	$88.91_{\pm 1.30}$
	100	$92.03_{\pm 0.44}$	$92.71_{\pm 0.27}$	$92.25_{\pm 0.29}$	$93.03_{\pm 0.16}$	$\mathbf{94.48_{\pm 0.13}}$	$93.09_{\pm 0.52}$
	150	$92.67_{\pm 0.14}$	$93.07_{\pm 0.33}$	$94.64_{\pm 0.09}$	$93.00_{\pm 0.17}$	$94.45_{\pm 0.11}$	$\mathbf{94.99_{\pm 0.23}}$
	200	$92.59_{\pm 0.13}$	$93.17_{\pm 0.30}$	$\mathbf{95.44_{\pm 0.10}}$	$93.02_{\pm 0.16}$	$94.47_{\pm 0.13}$	$95.14_{\pm 0.34}$
	250	$92.74_{\pm 0.16}$	$93.22_{\pm 0.36}$	$\mathbf{95.43_{\pm 0.08}}$	$93.02_{\pm 0.13}$	$94.46_{\pm 0.12}$	$95.24_{\pm 0.15}$
CIFAR-100	50	$68.02_{\pm 0.75}$	$68.66_{\pm 0.46}$	$62.17_{\pm 1.68}$	$74.18_{\pm 0.32}$	$\mathbf{75.64_{\pm 0.25}}$	$62.49_{\pm 1.60}$
	100	$70.57_{\pm 0.40}$	$69.78_{\pm 0.27}$	$68.78_{\pm 0.97}$	$74.21_{\pm 0.35}$	$\mathbf{76.02_{\pm 0.10}}$	$74.48_{\pm 0.53}$
	150	$71.67_{\pm 0.49}$	$70.45_{\pm 0.42}$	$77.40_{\pm 0.46}$	$74.22_{\pm 0.35}$	$76.00_{\pm 0.13}$	$\mathbf{77.81_{\pm 0.23}}$
	200	$72.08_{\pm 0.27}$	$70.61_{\pm 0.33}$	$77.63_{\pm 0.42}$	$74.19_{\pm 0.24}$	$76.05_{\pm 0.11}$	$\mathbf{77.76_{\pm 0.38}}$
	250	$71.94_{\pm 0.66}$	$71.11_{\pm 0.37}$	$77.65_{\pm 0.32}$	$74.21_{\pm 0.26}$	$75.99_{\pm 0.09}$	$\mathbf{78.00_{\pm 0.28}}$
Tiny-ImageNet	50	$54.15_{\pm 1.13}$	$47.43_{\pm 1.60}$	$53.03_{\pm 1.55}$	$57.85_{\pm 0.55}$	$\mathbf{58.95_{\pm 0.45}}$	$55.19_{\pm 0.74}$
	100	$55.37_{\pm 0.15}$	$48.97_{\pm 1.86}$	$57.99_{\pm 0.38}$	$58.06_{\pm 0.48}$	$\mathbf{59.97_{\pm 0.33}}$	$59.19_{\pm 0.49}$
	150	$55.00_{\pm 0.85}$	$52.22_{\pm 0.51}$	$59.99_{\pm 0.27}$	$57.99_{\pm 0.36}$	$59.97_{\pm 0.40}$	$\mathbf{60.22_{\pm 0.27}}$
	200	$55.05_{\pm 0.58}$	$50.14_{\pm 1.20}$	$60.59_{\pm 0.49}$	$58.16_{\pm 0.40}$	$60.01_{\pm 0.30}$	$\mathbf{61.13_{\pm 0.37}}$
	250	$55.48_{\pm 0.67}$	$52.13_{\pm 1.14}$	$60.65_{\pm 0.39}$	$57.98_{\pm 0.44}$	$59.91_{\pm 0.45}$	$\mathbf{61.44_{\pm 0.27}}$

Note that for a given fixed epoch budget, one might tune the initial learning rates, as well as the other hyper-parameters (such as weight-decay, batch-size, momentum, etc) to achieve optimum performance accuracy. However, this comes with a huge price of tuning effort that is required by the end-user. Therefore, providing a fast converging optimizer such as AdaS (with minimal tuning effort) is imperative to achieve high performing accuracy.

Is β a hyper-parameter to tune? β (gain factor) is not a hyper-parameter but a design choice for application. Considering the computational budget, the gain offers to either (a) tune for *"Speedy Convergence"* by setting it low e.g. $\beta = 0.8$, or (b) tune for *"Greedy Convergence"* by setting it high e.g. $\beta = 0.975$. The selection of different β directly translates to different number of epoch trains. The speedy convergence is set to train rapidly with minimal sacrifice on the accuracy–AdaS performs much better than all adaptive optimizers in terms of accuracy and speed which is highly desirable in many optimization methods such as neural architecture search. The greedy convergence case can also be used to achieve maximum possible accuracy, similar to mSGD-StepLR, assuming the computational budget is unlimited.

6 Conclusion

We proposed two new metrics to prob in the intermediate layers of CNN and define the well-posedness of convolution layers. The metrics can be used alone for empirical reasoning of network generalization performance. Provided by one of the metrics, we tailor a new adaptive optimizer *AdaS* to define learning rate of mSGD as a function of relative metric change over consecutive epoch training. The optimizer admits super convergence characteristics as well as high generalization performance compared to many adaptive and non-adaptive optimizers. We highlight that these improvements come through the application of AdaS to simple mSGD. We further identify that since AdaS adaptively tunes the learning rates, it can be deployed with optimizers such as AdaM, replacing the traditional scheduling techniques. Finally, this paper only studied optimization for CNNs and we will extend this to the general form of DNNs for future work.

References

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning*

Representations, 2018. URL https://openreview.net/forum?id=BkrsAzWAb.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pp. 437–478. Springer, 2012.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pp. 421–436. Springer, 2012.

Didier A Depireux, Jonathan Z Simon, David J Klein, and Shihab A Shamma. Spectro-temporal response field characterization with dynamic ripples in ferret primary auditory cortex. *Journal of neurophysiology*, 85(3):1220–1234, 2001.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho*, and Krzysztof Geras*. The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1g87C4KwB.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=H1oyRlYgg.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. ompression of deep convolutional neural networks for fast and low power mobile applications. jan 2016. 4th International Conference on Learning Representations, ICLR 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3): 455–500, 2009.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. jan 2015. 3rd International Conference on Learning Representations, ICLR 2015.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Fei-Fei Li, Andrej Karpathy, and Justin Johnson. URL https://tiny-imagenet.herokuapp.com/.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, pp. 11674–11685, 2019.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgz2aEKDr.

Nicolas Loizou and Peter Richtárik. Momentum and stochastic momentum for stochastic gradient, newton, proximal point and subspace descent methods. *Computational Optimization and Applications*, pp. 1–58, 2020.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. 2017. 5th International Conference on Learning Representations, ICLR 2017.

Liangchen Luo, Yuanhao Xiong, and Yan Liu. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg3g2R9FX.

Shinichi Nakajima, Masashi Sugiyama, S Derin Babacan, and Ryota Tomioka. Global analytic solution of fully-observed variational bayesian matrix factorization. *Journal of Machine Learning Research*, 14(Jan):1–37, 2013.

Francesco Orabona and Tatiana Tommasi. Training deep networks without learning rates through coin betting. In *Advances in Neural Information Processing Systems*, pp. 2160–2170, 2017.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964. doi: 10.1016/0041-5553(64)90137-5.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Michal Rolinek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. In *Advances in Neural Information Processing Systems*, pp. 6433–6443, 2018.

Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)*, 54(4):21–es, 2007.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pp. 343–351, 2013.

Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472. IEEE, 2017.

Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pp. 1100612. International Society for Optics and Photonics, 2019.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and E. Weinan. Convolutional neural networks with low-rank regularization. jan 2016. 4th International Conference on Learning Representations, ICLR 2016.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, pp. 3727–3740, 2019.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7370–7379, 2017.

Kun Yuan, Bicheng Ying, and Ali H Sayed. On the influence of momentum acceleration on online learning. *The Journal of Machine Learning Research*, 17(1):6602–6667, 2016.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

A Appendix-A: Proof of Theorems and Remarks

A.1 Proofs for Theorem 1

The following two proofs correspond to the proof of Theorem 1 for $p = 2$ and $p = 1$, respectively.

Proof. (Theorem 1 for $p = 2$) Using the Definition 1, the knowledge gain of matrix $\boldsymbol{W}^{t+1}$ (assumed to be a column matrix $N \leq M$) is expressed by

$$\mathcal{G}(\boldsymbol{W}^{t+1}) = \frac{1}{N\sigma_1^2(\boldsymbol{W}^{t+1})} \sum_{i=1}^{N'} \sigma_i^2(\boldsymbol{W}^{t+1}) = \frac{1}{N||\boldsymbol{W}^{t+1}||_2^2} \operatorname{tr} \boldsymbol{W}^{t+1^T} \boldsymbol{W}^{t+1}. \tag{11}$$

An upper-bound of first singular value can be calculated by first recalling its equivalence to ℓ_2-norm and then applying the Cauchy–Schwarz inequality

$$\sigma_1^2(\boldsymbol{W}^{t+1}) = ||\boldsymbol{W}^{t+1}||_2^2 = ||\boldsymbol{W}^t - \eta(t)\overline{\nabla f}_{t+1}||_2^2 \leq ||\boldsymbol{W}^t||_2^2 + \eta^2(t)||\overline{\nabla f}_{t+1}||_2^2 + 2\eta(t)||\boldsymbol{W}^t||_2||\overline{\nabla f}_{t+1}||_2. \tag{12}$$

Note that $\eta(t)$ is given by previous epoch update and considered to be positive (we start with an initial learning rate $\eta(0) > 0$). Therefore, the right-hand-side of the inequality in (12) will be positive and holds.

By substituting (12) in (11) and expanding the terms in trace, a lower bound of $\boldsymbol{W}^{t+1}$ is given by

$$\mathcal{G}(\boldsymbol{W}^{t+1}) \geq \frac{1}{N\gamma}\left[\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t - 2\eta(t)\operatorname{tr} \boldsymbol{W}^{t^T}\overline{\nabla f}_{t+1} + \eta^2(t)\operatorname{tr} \overline{\nabla f}_{t+1}{}^T\overline{\nabla f}_{t+1}\right], \tag{13}$$

where, $\gamma = ||\boldsymbol{W}^t||_2^2 + \eta^2(t)||\overline{\nabla f}_{t+1}||_2^2 + 2\eta(t)||\boldsymbol{W}^t||_2||\overline{\nabla f}_{t+1}||_2$. The latter inequality can be revised to

$$\begin{aligned}
\mathcal{G}(\boldsymbol{W}^{t+1}) &\geq \tfrac{1}{N\gamma}\left[\left(1 - \tfrac{\gamma}{||\boldsymbol{W}^t||_2^2} + \tfrac{\gamma}{||\boldsymbol{W}^t||_2^2}\right)\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t \right. \\
&\qquad \left. -2\eta(t)\operatorname{tr} \boldsymbol{W}^{t^T}\overline{\nabla f}_{t+1} + \eta^2(t)\operatorname{tr} \overline{\nabla f}_{t+1}{}^T\overline{\nabla f}_{t+1}\right] \\
&= \tfrac{1}{N\gamma}\left[\tfrac{\gamma}{||\boldsymbol{W}^t||_2^2}\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t + \left(1 - \tfrac{\gamma}{||\boldsymbol{W}^t||_2^2}\right)\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t \right. \\
&\qquad \left. -2\eta(t)\operatorname{tr} \boldsymbol{W}^{t^T}\overline{\nabla f}_{t+1} + \eta^2(t)\operatorname{tr} \overline{\nabla f}_{t+1}{}^T\overline{\nabla f}_{t+1}\right] \\
&= \mathcal{G}(\boldsymbol{W}^t) + \tfrac{1}{N\gamma}\left[\underbrace{\left(1 - \frac{\gamma}{||\boldsymbol{W}^t||_2^2}\right)\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t - 2\eta(t)\operatorname{tr} \boldsymbol{W}^{t^T}\overline{\nabla f}_{t+1} + \eta^2(t)\operatorname{tr} \overline{\nabla f}_{t+1}{}^T\overline{\nabla f}_{t+1}}_{D}\right].
\end{aligned} \tag{14}$$

Therefore, the bound in (14) is revised to

$$\mathcal{G}(\boldsymbol{W}^{t+1}) - \mathcal{G}(\boldsymbol{W}^t) \geq \frac{1}{N\gamma}D. \tag{15}$$

Since $\gamma \geq 0$, the monotonicity of the Equation (15) is guaranteed if $D \geq 0$. The remaining term D can be expressed as a quadratic function of η

$$\begin{aligned}
D(\eta) = &\left[\operatorname{tr} \overline{\nabla f}_{t+1}{}^T\overline{\nabla f}_{t+1} - \tfrac{||\overline{\nabla f}_{t+1}||_2^2}{||\boldsymbol{W}^t||_2^2}\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t\right]\eta^2(t) \\
&- \left[2\operatorname{tr} \boldsymbol{W}^{t^T}\overline{\nabla f}_{t+1} + 2\tfrac{||\overline{\nabla f}_{t+1}||_2}{||\boldsymbol{W}^t||_2}\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t\right]\eta(t)
\end{aligned} \tag{16}$$

where, the condition for $D(\eta) \geq 0$ in (16) is

$$\eta \geq \max\left\{2\frac{\operatorname{tr} \boldsymbol{W}^{t^T}\overline{\nabla f}_{t+1} + \frac{||\overline{\nabla f}_{t+1}||_2}{||\boldsymbol{W}^t||_2}\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t}{\operatorname{tr} \overline{\nabla f}_{t+1}{}^T\overline{\nabla f}_{t+1} - \frac{||\overline{\nabla f}_{t+1}||_2^2}{||\boldsymbol{W}^t||_2^2}\operatorname{tr} \boldsymbol{W}^{t^T}\boldsymbol{W}^t}, 0\right\}. \tag{17}$$

The lower bound in (17) proves the existence of a lower bound for monotonicity condition.

Our final inspection is to check if the substitution of step-size (7) in (15) would still hold the inequality condition in (15). Followed by the substitution, the inequality should satisfy

$$\eta(t+1) \geq \zeta\frac{1}{N\gamma}D. \tag{18}$$

We have found that $D(\eta) \geq 0$ for some lower bound in (17), where the inequality in (18) also holds from some $\zeta \geq 0$ and the proof is done. □

Proof. (Theorem 1 for $p = 1$) The knowledge gain of matrix $\boldsymbol{W}^{t+1}$ is expressed by

$$\mathcal{G}(\boldsymbol{W}^{t+1}) = \frac{1}{N\sigma_1(\boldsymbol{W}^{t+1})}\sum_{i=1}^{N'}\sigma_i(\boldsymbol{W}^{t+1}). \tag{19}$$

By stacking all singular values in a vector form (and recall from ℓ_1 and ℓ_2 norms inequality)

$$\left[\sum_{i=1}^{N'} \sigma_i(\boldsymbol{W}^{t+1})\right]^2 = ||\underline{\sigma}(\boldsymbol{W}^{t+1})||_1^2 \geq ||\underline{\sigma}(\boldsymbol{W}^{t+1})||_2^2 = \sum_{i=1}^{N'} \sigma_i^2(\boldsymbol{W}^{t+1}) = \operatorname{tr} \boldsymbol{W}^{t+1^T} \boldsymbol{W}^{t+1},$$

and by substituting the matrix composition $\boldsymbol{W}^{t+1}$, the following inequality holds

$$\left[\sum_{i=1}^{N'} \sigma_i(\boldsymbol{W}^{t+1})\right]^2 \geq \operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t - 2\eta(t) \operatorname{tr} \boldsymbol{W}^{t^T} \overline{\nabla f}_{t+1} + \eta^2(t) \operatorname{tr} \overline{\nabla f}_{t+1}{}^T \overline{\nabla f}_{t+1}. \tag{20}$$

An upper-bound of first singular value can be calculated by recalling its equivalence to ℓ_2-norm and Cauchy–Schwarz inequality as follows

$$\sigma_1^2(\boldsymbol{W}^{t+1}) = ||\boldsymbol{W}^{t+1}||_2^2 = ||\boldsymbol{W}^t - \eta(t)\overline{\nabla f}_{t+1}||_2^2 \leq ||\boldsymbol{W}^t||_2^2 + 2\eta(t)||\boldsymbol{W}^t||_2||\overline{\nabla f}_{t+1}||_2 + \eta^2(t)||\overline{\nabla f}_{t+1}||_2^2. \tag{21}$$

Note that $\eta(t)$ is given by previous epoch update and considered to be positive (we start with an initial learning rate $\eta(0) > 0$). Therefore, the right-hand-side of the inequality in (21) will be positive and holds.

By substituting the lower-bound (20) and upper-bound (21) into (19), a lower bound of knowledge gain is given by

$$\mathcal{G}^2(\boldsymbol{W}^{t+1}) \geq \frac{1}{N^2\gamma}\left[\operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t - 2\eta(t) \operatorname{tr} \boldsymbol{W}^{t^T} \overline{\nabla f}_{t+1} + \eta^2(t) \operatorname{tr} \overline{\nabla f}_{t+1}{}^T \overline{\nabla f}_{t+1}\right],$$

where $\gamma = ||\boldsymbol{W}^t|_2^2 + 2\eta(t)||\boldsymbol{W}^t||_2||\overline{\nabla f}_{t+1}||_2 + \eta^2(t)||\overline{\nabla f}_{t+1}||_2^2$. The latter inequality can be revised to

$$\begin{aligned}\mathcal{G}^2(\boldsymbol{W}^{t+1}) \geq \tfrac{1}{N^2\gamma}[\tfrac{N''\gamma}{||\boldsymbol{W}^t||_2^2} \operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t + \\ \underbrace{(1 - \frac{N''\gamma}{||\boldsymbol{W}^t||_2^2}) \operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t - 2\eta(t) \operatorname{tr} \boldsymbol{W}^{t^T} \overline{\nabla f}_{t+1} + \eta^2(t) \operatorname{tr} \overline{\nabla f}_{t+1}{}^T \overline{\nabla f}_{t+1}}_{D}],\end{aligned} \tag{22}$$

where, the lower bound of the first summand term is given by

$$\begin{aligned}\tfrac{N''\gamma}{||\boldsymbol{W}^t||_2^2} \operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t &= \tfrac{N''\gamma}{||\boldsymbol{W}^t||_2^2} \textstyle\sum_{i=1}^{N''} \sigma_i^2(\boldsymbol{W}^t) \\ &= \tfrac{N''\gamma}{\sigma_1^2(\boldsymbol{W}^t)}||\underline{\sigma}(\boldsymbol{W}^t)||_2^2 \geq \tfrac{\gamma}{\sigma_1^2(\boldsymbol{W}^t)}||\underline{\sigma}(\boldsymbol{W}^t)||_1^2 = \gamma N^2 \mathcal{G}^2(\boldsymbol{W}^t).\end{aligned}$$

Therefore, the bound in (22) is revised to

$$\mathcal{G}^2(\boldsymbol{W}^{t+1}) \geq \mathcal{G}^2(\boldsymbol{W}^t) + \frac{1}{N^2\gamma} D. \tag{23}$$

Note that $\gamma \geq 0$ (previous step-size $\eta(t) \geq 0$ is always positive) and the only condition for the bound in (23) to hold is to $D \geq 0$. Here the remaining term D can be expressed as quadratic function of step-size i.e. $D(\eta) = a\eta^2 + b\eta + c$ where

$$\begin{aligned}&a = \operatorname{tr} \overline{\nabla f}_{t+1}{}^T \overline{\nabla f}_{t+1} - N'' \tfrac{||\overline{\nabla f}_{t+1}||_2^2}{||\boldsymbol{W}^t||_2^2} \operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t, \quad b = -2 \operatorname{tr} \boldsymbol{W}^{t^T} \overline{\nabla f}_{t+1} - N'' \tfrac{||\overline{\nabla f}_{t+1}||_2}{||\boldsymbol{W}^t||_2}, \\ &c = -(N'' - 1) \operatorname{tr} \boldsymbol{W}^{t^T} \boldsymbol{W}^t.\end{aligned}$$

The quadratic function can be factorized $D(\eta(t)) = (\eta(t) - \eta_1)(\eta(t) - \eta_2)$ where the roots $\eta_1 = (-b + \sqrt{\Delta})/2a$ and $\eta_2 = (-b - \sqrt{\Delta})/2a$, and $\Delta = b^2 - 4ac$. Here $c \leq 0$ and assuming $a \geq 0$ then $\Delta \geq 0$. Accordingly, $\eta_1 \geq 0$ and $\eta_2 \leq 0$. For the function $D(\eta(t))$ to yield a positive value, both factorized elements should be either positive (i.e. $\eta(t) - \eta_1 \geq 0$ and $\eta(t) - \eta_2 \geq 0$) or negative (i.e. $\eta(t) - \eta_1 \leq 0$ and $\eta(t) - \eta_2 \leq 0$). Here, only the positive conditions hold which yield $\eta(t) \geq \eta_1$. The assumption $a \geq 0$ is equivalent to $\sum_{i=1}^{N'''} \sigma_i^2(\overline{\nabla f}_{t+1})/\sigma_1^2(\overline{\nabla f}_{t+1}) \geq N'' \sum_{i=1}^{N''} \sigma_i^2(\boldsymbol{W}^t)/\sigma_1^2(\boldsymbol{W}^t)$. The singular values $\sigma_i(\boldsymbol{W}^t)$ associated with low-rank weights are mainly sparse in starting epochs, where majority of update information are carried by gradients. Therefore, the condition $a \geq 0$ easily

holds for beginning epochs. As the training proceeds with more epoch trains, the information flow through gradient updates shrink, where little room is left to update the weights. In the case, the condition $a \geq 0$ might not hold and accordingly, the monotonicity of the knowledge gain could be violated. Nevertheless, with proceeding epoch train, the associated learning rates decreases where the optimizer reaches to a stabilizing phase. This phenomenon can be seen in the evolution of knowledge gain in ablation study in Figure 7. Notice how the monotonicity of knowledge gain gets violated in proceeding epochs–specifically with higher gain factors which consume more epoch for training. □

A.2 Proof for Theorem 2

Consider the AdaS update rule for Stochastic Gradient Descent with Momentum (mSGD)

$$\begin{aligned} \boldsymbol{v}_k &= \alpha v_{k-1} - \eta(t)\boldsymbol{g}_k, \\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \boldsymbol{v}_k. \end{aligned}$$

It is evident that $\boldsymbol{w}_k = \boldsymbol{w}_{k-1} + \boldsymbol{v}_{k-1}$ and thus, $\boldsymbol{v}_{k-1} = \boldsymbol{w}_k - \boldsymbol{w}_{k-1}$. We can then write our parameter update rule as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta(t)\boldsymbol{g}_k + \alpha(\boldsymbol{w}_k - \boldsymbol{w}_{k-1}).$$

Note how for AdaS, the parameter update rule is subject to the parametrization of η by t.

We highlight the per Theorem 1 introduced in Loizou & Richtárik (2020) (Page 17), $\mathcal{L}2$ convergence for mSGD is proven, using the update rule aforementioned above with fixed global learning rate. Particularly, Loizou & Richtárik (2020) study the convergence of $\mathbb{E}[\| \boldsymbol{w}_k - \boldsymbol{w}^* \|_{\mathbf{B}}^2]$ to zero. Loizou & Richtárik (2020), in Appendix 2: Proof of Theorem 1 (Page 46) show that

$$\mathbb{E}_{\Omega_k}[\| \boldsymbol{w}_{k+1} - \boldsymbol{w}^* \|_{\mathbf{B}}^2] \leq a_1 \| \boldsymbol{w}_k - \boldsymbol{w}^* \|_{\mathbf{B}}^2 + a_2 \| \boldsymbol{w}_{k-1} - \boldsymbol{w}^* \|_{\mathbf{B}}^2, \tag{24}$$

where, $a_1 = 1 + 3\alpha + 2\alpha^2 - [(\alpha+2)\eta - \eta^2]\lambda_{min}$, and $a_2 = \alpha + 2\alpha^2 + \eta\alpha\lambda_{max}$, where k is the current mini-batch iteration, $\boldsymbol{w}^*$ is the converging point of AdaS algorithm obtained by the linear projection of the starting point $\boldsymbol{w}^* = \Pi_{\mathcal{L}}^{\mathbf{B}}(\boldsymbol{w}^0)$ under AdaS updating rule, and λ_{min} and λ_{max} are the smallest and largest nonzero eigenvalues, respectively, of the Hessian of our objective function $f(\boldsymbol{w})$. Note that $0 \leq \lambda_{min} \leq \lambda_{max} \leq 1$ (Page 13 in Loizou & Richtárik (2020)).

Taking the expectation of the inequality in (24) with respect to $\boldsymbol{w}_k$, and letting

$$F_k := \mathbb{E}[\| \boldsymbol{w}_k - \boldsymbol{w}^* \|_{\mathbf{B}}^2],$$

we get

$$F_{k+1} \leq a_1 F_k + a_2 F_{k-1}. \tag{25}$$

Specifically, convergence rate is bounded by

$$\mathbb{E}[\| \boldsymbol{w}_{k+1} - \boldsymbol{w}^* \|_{\mathbf{B}}^2] \leq q^k(1+\delta) \| \boldsymbol{w}_0 - \boldsymbol{w}^* \|_{\mathbf{B}}^2, \tag{26}$$

where $q = (a_1 + \sqrt{a_1^2 + 4a_2})/2$ and $\delta = q - a_1$. From [Lemma 9, page 42-43, Loizou & Richtárik (2020)] the inequality $a_1 + a_2 \leq q < 1$ is proven to hold and under the assumptions $0 < \eta < 2$ and $\alpha \geq 0$, the convergence of generated sequences in (26) is guaranteed. Furthermore, from [Lemma 11, page 44, Equation (50), Loizou & Richtárik (2020)] it follows that the loss function is bounded by $f(\boldsymbol{w}^k) \leq \frac{\lambda_{\max}}{2} \| \boldsymbol{w}_0 - \boldsymbol{w}^* \|_{\mathbf{B}}^2$ which yields to

$$\mathbb{E}\left[f(\boldsymbol{w}^k)\right] \leq q^k \tfrac{\lambda_{\max}}{2}(1+\delta)\|\boldsymbol{w}^0 - \boldsymbol{w}^*\|_B^2. \tag{27}$$

Transitioning to AdaS, the only change to this formulation is the parametrization of the global learning rate: $\eta \to \eta(t(k))$. Note that the epoch index is parametrized by mini-batch iteration here i.e. where $\eta(k) = \eta(t)$ for all k mini-batches within the current tth epoch. Therefore, under AdaS, a_1 and a_2 are parametrized by t such that

$$\begin{aligned} a_1(t) &= 1 + 3\alpha + 2\alpha^2 - [(\alpha+2)\eta(t) - \eta^2(t)]\lambda_{min} \\ a_2(t) &= \alpha + 2\alpha^2 + \eta(t)\alpha\lambda_{max}. \end{aligned}$$

The parametrized inequality $a_1(t) + a_2(t) \leq q(t) < 1$ still holds under the condition $0 < \eta(t) \leq 2$ and $\alpha \geq 0$ across all epoch indices. Note that in AdaS, $0 < \eta(t) \leq 1$ and $\lim_{t\to\infty} \eta(t) \to 0$, which is shown empirically in Figure 2 as well as the ablative study in Figure 10.

A.3 On Energy of Singular Values of Low-Rank Factorized Weights

Remark 1. *Recall for $p = 2$ that the summation of squared singular values from Definition 1 is equivalent to the* Frobenius *(norm) i.e.* $||\widehat{\boldsymbol{W}}_d||_F^2 = \sum_{k=1}^{N'_d} \sigma_k^2(\widehat{\boldsymbol{W}}_d) = \text{tr}\, \widehat{\boldsymbol{W}}_d^T \widehat{\boldsymbol{W}}_d$ *Horn & Johnson (2012). Also, for $p = 1$ the summation is bounded by* $||\widehat{\boldsymbol{W}}_d||_F \leq \sum_{k=1}^{N'_d} \sigma_k(\widehat{\boldsymbol{W}}_d) \leq \sqrt{N'_d}||\widehat{\boldsymbol{W}}_d||_F$.

The energy here indicates the separability measure of convolution weights for space spanning throughout the input/output channel mapping (similar to the index of inseparability in neurophysiology Depireux et al. (2001)).

B Appendix-B: Experimental Setup

All experiments are run using an RTX2080Ti, 3 cores of an Intel Xeon Gold 6246 processor, and 64 gigabytes of RAM. The details of pre-processing steps, network implementation and training/testing frameworks are adopted from the CIFAR GitHub repository[1] using PyTorch. We set the initial learning rates of AdaGrad, RMSProp, AdaBound, and SLS to $\eta_0 = \{1\text{e-}2, 3\text{e-}4, 1\text{e-}3\}, 1$ per their suggested default values. We further followed the suggested tuning in Wilson et al. (2017) for AdaM ($\eta_0 = 3\text{e-}4$) and SGD-StepLR ($\eta_0 = 1\text{e-}1$ dropping half magnitude every 25 epochs for all CIFAR/ImageNet experiments and dropping half magnitude every 15epochs for ImageNet experiments) to achieve the best performance. We tested several initial learning rates for the above competing methods and found that the default rates are indeed the best performing parameters. It worth noting that from other papers like Luo et al. (2019) also use the same initial learning rates in their comparisons. To configure the best initial learning rate for AdaS, we performed a dense grid search and found the values for VGG16 and ResNet34 to be $\eta_0 = \{5\text{e-}3, 3\text{e-}2\}$. For other architectures, i.e. ResNet18, ResNetXt50, and DenseNet121, we used the same initial learning rate found for ResNet34. Despite the differences in optimal values that are independently obtained for each network, the optimizer performance is fairly robust relative to changes in these values. Each model is trained for 250 epochs in 5 independent runs and average test accuracy and training losses are reported. The mini-batch size is also set to $|\Omega_k| = 128$.

C Appendix-C: Additional Experimental Results

Additional training performance results are shown in Figure 4 and Figure 5 for CIFAR10, CIFAR100, and ImageNet. Notice the robust and superior performance of AdaS compared to all other adaptive optimizers.

Note that we omitted scheduled learning approach mSGD+StepLR on CIFAR experiments here to only compare among adaptive optimizers. We understand that it is a common practice nowadays (even using adaptive optimizers such as Adam and Adabound), that practitioners deploy scheduled drop of learning rate to achieve higher accuracies. To avoid complications and for fair comparison, we believe all adaptive optimizers should be compared to each other without scheduled dropping. And since mSGD-StepLR is out of context for comparison here, we have omitted it. We highlight the robustness of AdaS and its superior performance compared to other adaptive optimizers in both Train/Test accuracies.

D Appendix-D: AdaS Ablation Study

The ablative analysis of AdaS optimizer is studied here with respect to different parameter settings. Figure 6 demonstrates the AdaS performance with respect to different range of gain-factor β. Figure 7 demonstrates the knowledge gain of different dataset and network with respect to different gain-factor settings over successive epochs. Similarly, Figure 8 also demonstrates the rank gain (aka the ratio of non-zero singular values of low-rank structure with respect to channel size) over successive epochs. Mapping conditions are shown in Figure 9 and Figure 10 demonstrates the learning rate approximation through AdaS algorithm over successive epoch training.

[1] https://github.com/kuangliu/pytorch-cifar

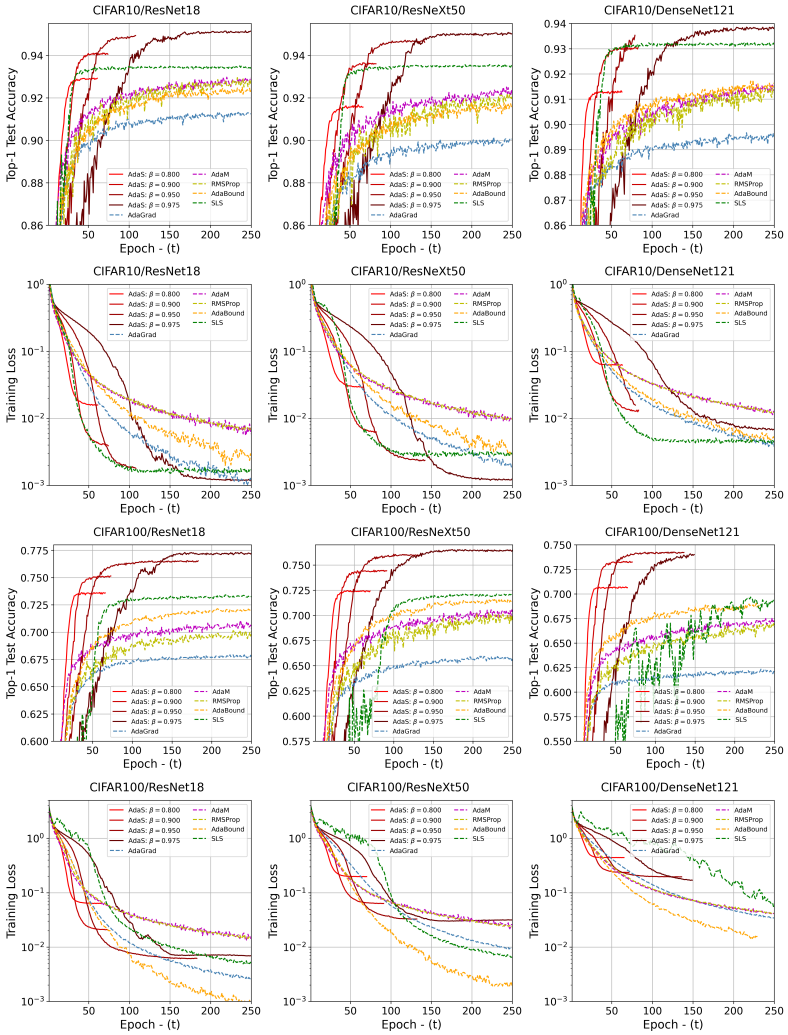


Figure 4: Additional experiments on training performance using different optimizers across two different datasets (i.e. CIFAR10, CIFAR100) and three different CNNs (i.e. ResNet18, ResNeXt50, DenseNet121).

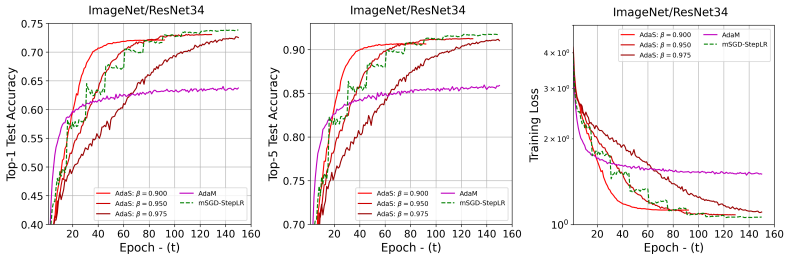
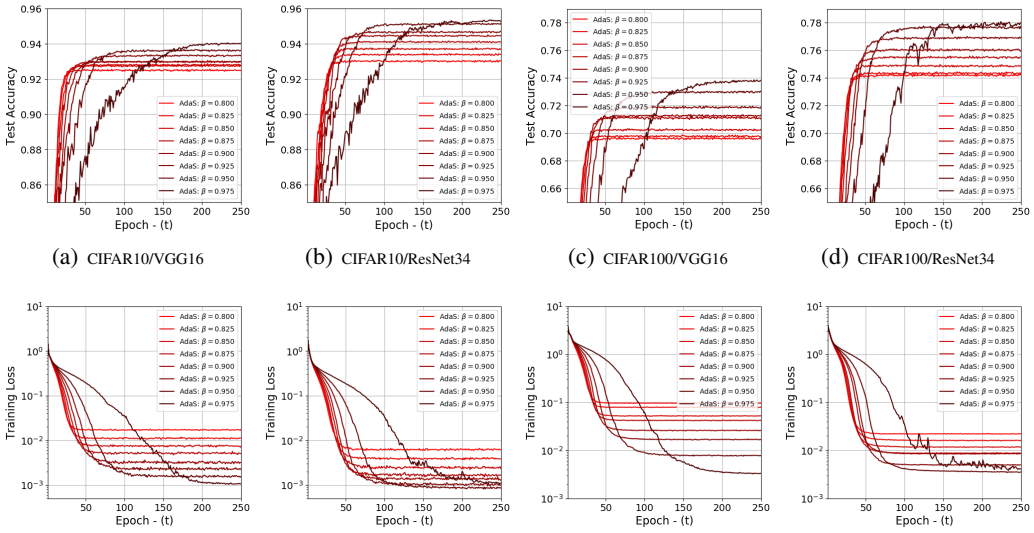


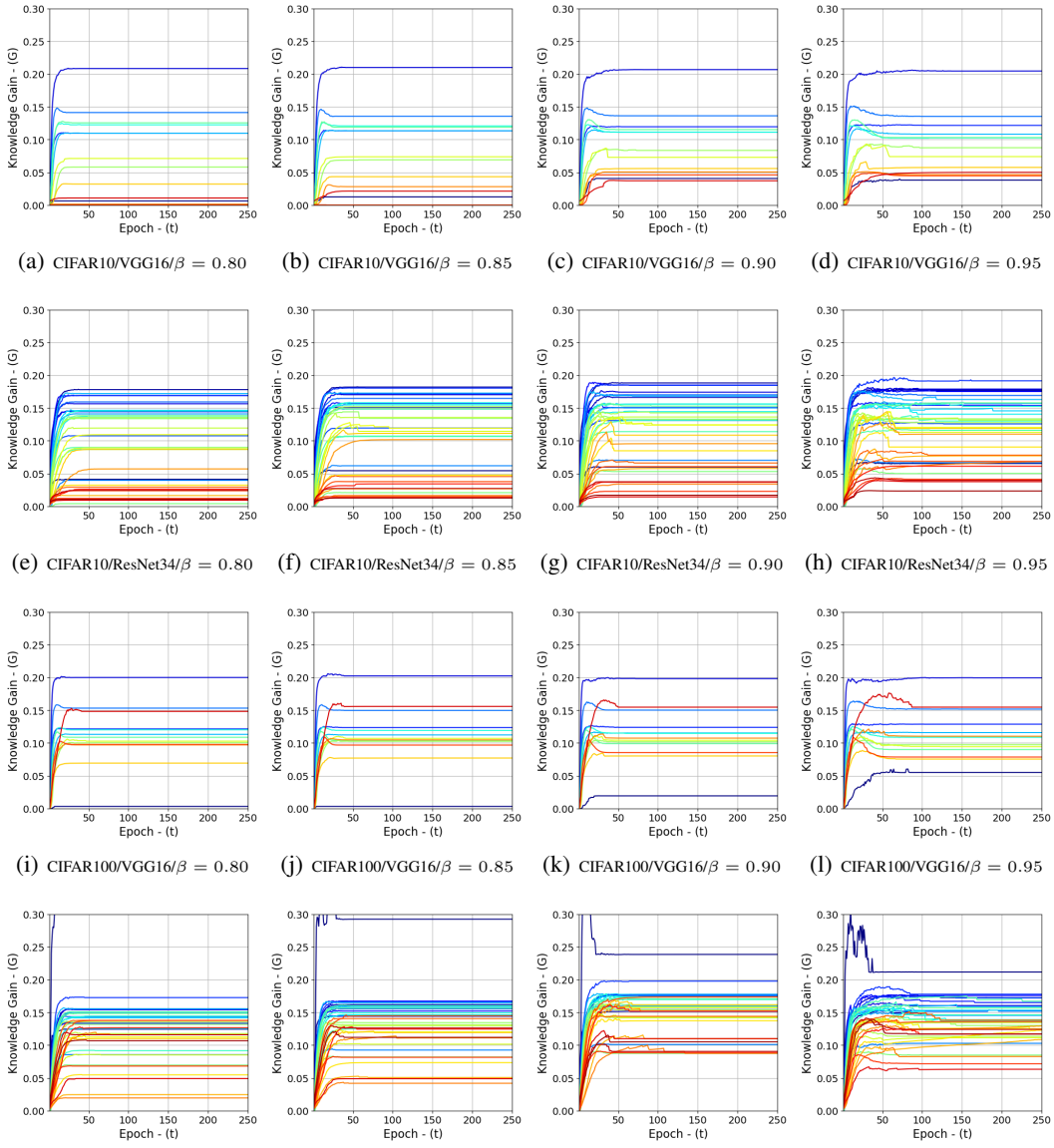
Figure 5: Additional experiments ImageNet/ResNet34 training performance using different optimizers.



(a) CIFAR10/VGG16 (b) CIFAR10/ResNet34 (c) CIFAR100/VGG16 (d) CIFAR100/ResNet34

(e) CIFAR10/VGG16 (f) CIFAR10/ResNet34 (g) CIFAR100/VGG16 (h) CIFAR100/ResNet34

Figure 6: Ablative study of AdaS gain factor over two different datasets (i.e. CIFAR10 and CIFAR100) and two CNNs (i.e. VGG16 and ResNet34). Top row corresponds to test-accuracies and bottom row to training-losses.



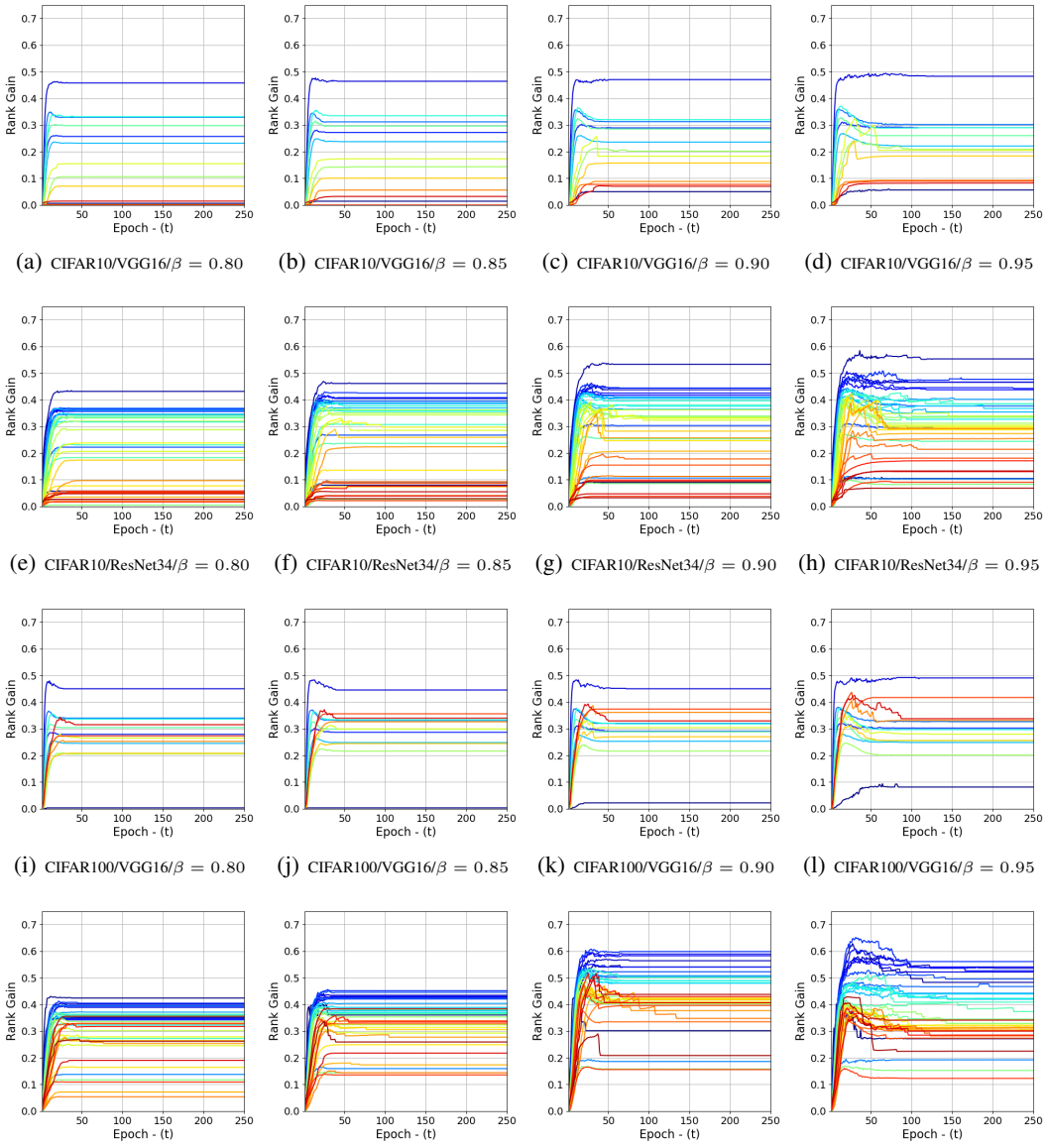
(a) CIFAR10/VGG16/$\beta = 0.80$ (b) CIFAR10/VGG16/$\beta = 0.85$ (c) CIFAR10/VGG16/$\beta = 0.90$ (d) CIFAR10/VGG16/$\beta = 0.95$

(e) CIFAR10/ResNet34/$\beta = 0.80$ (f) CIFAR10/ResNet34/$\beta = 0.85$ (g) CIFAR10/ResNet34/$\beta = 0.90$ (h) CIFAR10/ResNet34/$\beta = 0.95$

(i) CIFAR100/VGG16/$\beta = 0.80$ (j) CIFAR100/VGG16/$\beta = 0.85$ (k) CIFAR100/VGG16/$\beta = 0.90$ (l) CIFAR100/VGG16/$\beta = 0.95$

(m) CIFAR100/ResNet34/$\beta = 0.80$ (n) CIFAR100/ResNet34/$\beta = 0.85$ (o) CIFAR100/ResNet34/$\beta = 0.90$ (p) CIFAR100/ResNet34/$\beta = 0.95$

Figure 7: Ablative study of AdaS gain factor β versus knowledge gain G over two different datasets (i.e. CIFAR10 and CIFAR100) and two CNNs (i.e. VGG16 and ResNet34). The transition in color shades from light to dark lines correspond to first to the end of convolution layers in each network.



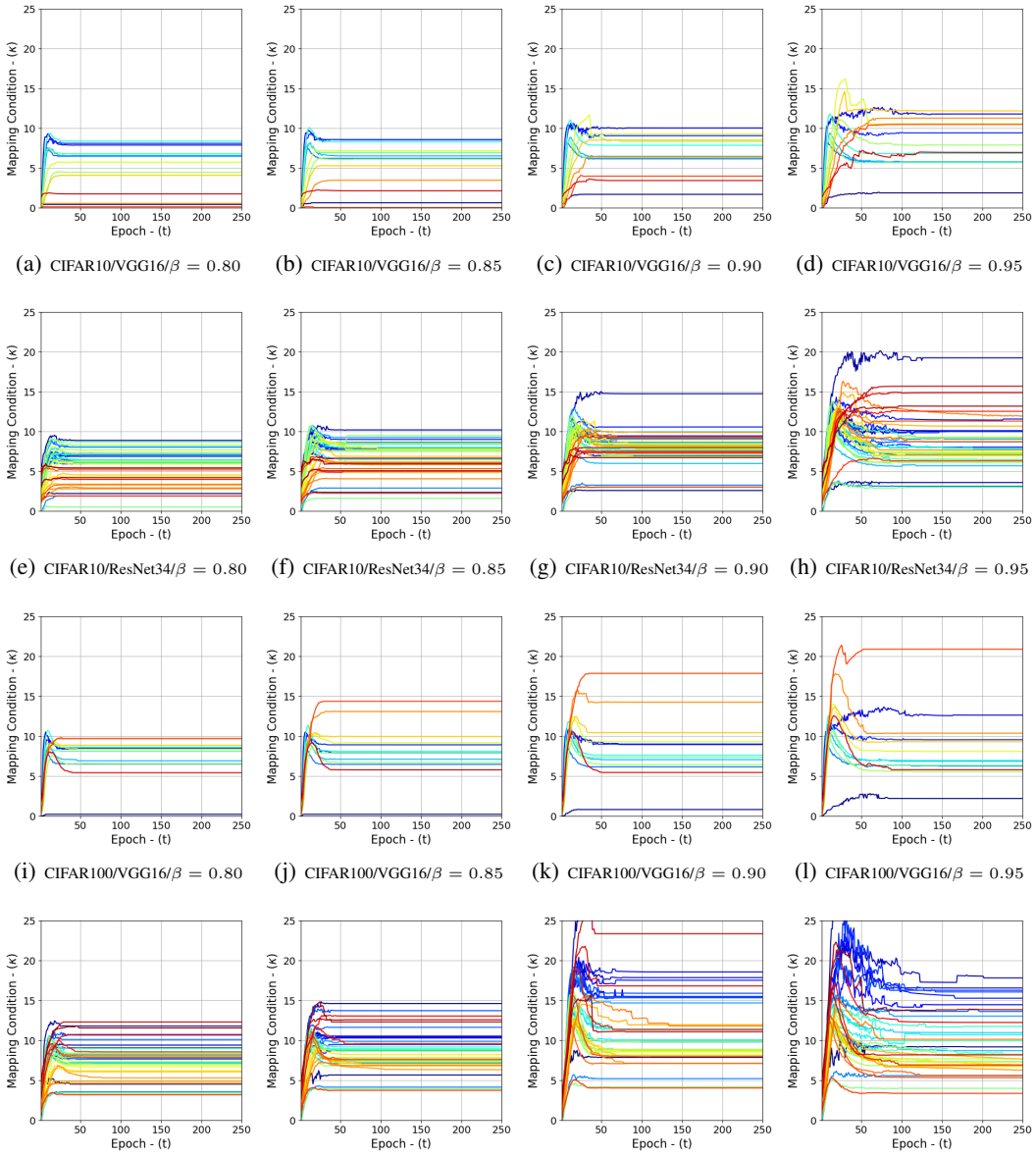
(a) CIFAR10/VGG16/$\beta = 0.80$ (b) CIFAR10/VGG16/$\beta = 0.85$ (c) CIFAR10/VGG16/$\beta = 0.90$ (d) CIFAR10/VGG16/$\beta = 0.95$

(e) CIFAR10/ResNet34/$\beta = 0.80$ (f) CIFAR10/ResNet34/$\beta = 0.85$ (g) CIFAR10/ResNet34/$\beta = 0.90$ (h) CIFAR10/ResNet34/$\beta = 0.95$

(i) CIFAR100/VGG16/$\beta = 0.80$ (j) CIFAR100/VGG16/$\beta = 0.85$ (k) CIFAR100/VGG16/$\beta = 0.90$ (l) CIFAR100/VGG16/$\beta = 0.95$

(m) CIFAR100/ResNet34/$\beta = 0.80$ (n) CIFAR100/ResNet34/$\beta = 0.85$ (o) CIFAR100/ResNet34/$\beta = 0.90$ (p) CIFAR100/ResNet34/$\beta = 0.95$

Figure 8: Ablative study of AdaS gain factor β versus rank gain $\text{rank}\,\hat{\mathbf{\Phi}}$ over two different datasets (i.e. CIFAR10 and CIFAR100) and two CNNs (i.e. VGG16 and ResNet34). The transition in color shades from light to dark lines correspond to first to the end of convolution layers in each network.



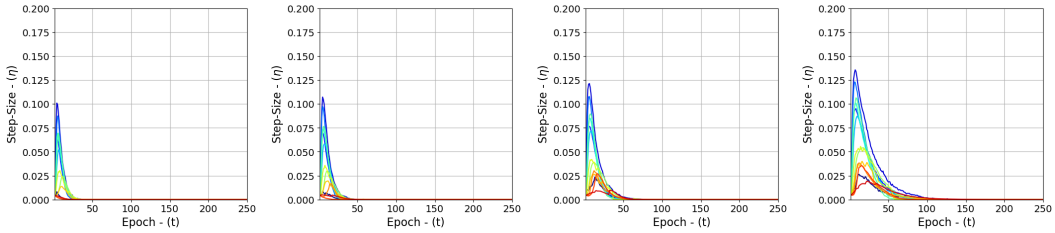
(a) CIFAR10/VGG16/$\beta = 0.80$ (b) CIFAR10/VGG16/$\beta = 0.85$ (c) CIFAR10/VGG16/$\beta = 0.90$ (d) CIFAR10/VGG16/$\beta = 0.95$

(e) CIFAR10/ResNet34/$\beta = 0.80$ (f) CIFAR10/ResNet34/$\beta = 0.85$ (g) CIFAR10/ResNet34/$\beta = 0.90$ (h) CIFAR10/ResNet34/$\beta = 0.95$

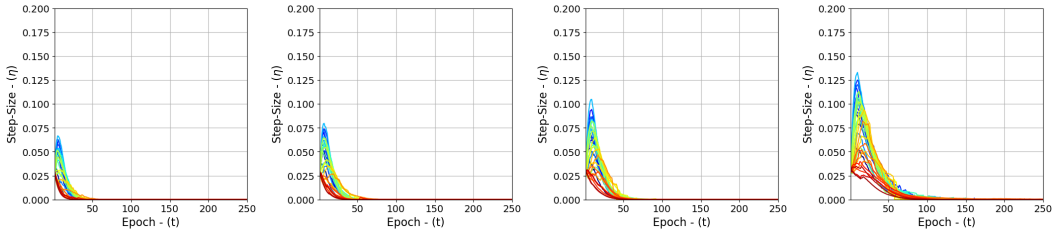
(i) CIFAR100/VGG16/$\beta = 0.80$ (j) CIFAR100/VGG16/$\beta = 0.85$ (k) CIFAR100/VGG16/$\beta = 0.90$ (l) CIFAR100/VGG16/$\beta = 0.95$

(m) CIFAR100/ResNet34/$\beta = 0.80$ (n) CIFAR100/ResNet34/$\beta = 0.85$ (o) CIFAR100/ResNet34/$\beta = 0.90$ (p) CIFAR100/ResNet34/$\beta = 0.95$

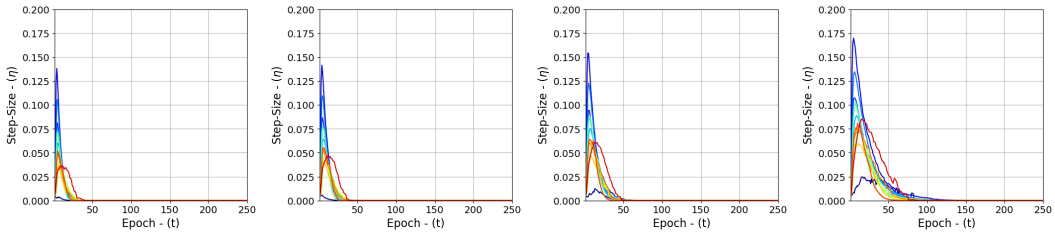
Figure 9: Ablative study of AdaS gain factor β versus mapping condition κ over two different datasets (i.e. CIFAR10 and CIFAR100) and two CNNs (i.e. VGG16 and ResNet34). The transition in color shades from light to dark lines correspond to first to the end of convolution layers in each network.



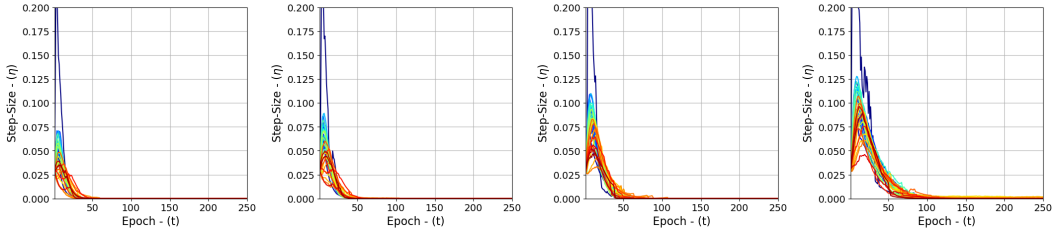
(a) CIFAR10/VGG16/$\beta = 0.80$ (b) CIFAR10/VGG16/$\beta = 0.85$ (c) CIFAR10/VGG16/$\beta = 0.90$ (d) CIFAR10/VGG16/$\beta = 0.95$



(e) CIFAR10/ResNet34/$\beta = 0.80$ (f) CIFAR10/ResNet34/$\beta = 0.85$ (g) CIFAR10/ResNet34/$\beta = 0.90$ (h) CIFAR10/ResNet34/$\beta = 0.95$



(i) CIFAR100/VGG16/$\beta = 0.80$ (j) CIFAR100/VGG16/$\beta = 0.85$ (k) CIFAR100/VGG16/$\beta = 0.90$ (l) CIFAR100/VGG16/$\beta = 0.95$



(m) CIFAR100/ResNet34/$\beta = 0.80$ (n) CIFAR100/ResNet34/$\beta = 0.85$ (o) CIFAR100/ResNet34/$\beta = 0.90$ (p) CIFAR100/ResNet34/$\beta = 0.95$

Figure 10: Ablative study of the evolution of learning rate in AdaS using different gain factor β for two different datasets (i.e. CIFAR10 and CIFAR100) and two CNNs (i.e. VGG16 and ResNet34). The transition in color shades from light to dark lines correspond to first to the end of convolution layers in each network.