Phase-wise Parameter Aggregation For Improving SGD Optimization

Takumi Kobayashi
National Institute of Advanced Industrial Science and Technology
1-1-1 Umezono, Tsukuba, Japan
takumi.kobayashi@aist.go.jp

Abstract

*Stochastic gradient descent (*SGD*) is successfully applied to train deep convolutional neural networks (CNNs) on various computer vision tasks. Since fixed step-size* SGD *converges to so-called error plateau, it is applied in combination with decaying learning rate to reach a favorable optimum. In this paper, we propose a simple yet effective optimization method to improve* SGD *with a phase-wise decay of learning rate. Through analyzing both a loss surface around the error plateau and a structure of the* SGD *optimization process, the proposed method is formulated to improve convergence as well as initialization at each training phase by efficiently aggregating the CNN parameters along the optimization sequence. The method keeps the simplicity of* SGD *while touching the* SGD *procedure only a few times during training. The experimental results on image classification tasks thoroughly validate the effectiveness of the proposed method in comparison to the other methods.*

1. Introduction

As deep convolutional neural networks (CNNs) produce promising performance on variety of computer vision tasks, optimization of the deep models is also an active research topic in this literature. Stochastic gradient descent (SGD) [26, 23] is arguably one of the most successful approaches by effectively leveraging large-scale training dataset via mini-batches to train the huge number of parameters with favorable stochasticity in a simple formulation [38, 35]. In recent years, much research effort has been made to improve SGD especially in terms of adaptive scaling [34, 15, 25, 20]. Though the adaptive method, such as ADAM [15], contributes to rapid training time, the generalization performance is still unclear and in some practical cases is even inferior to SGD. Some variants of ADAM [25, 20] are proposed to alleviate the shortcomings toward filling the performance gap against SGD.

Thus, SGD is still a competitive optimization method to train deep CNNs, and optimization behavior of SGD is actively analyzed in the framework of deep learning [32, 2, 7, 6]. On a strongly convex loss function, SGD equipped with a fixed learning rate exhibits linear convergence up to a radius around the optimal solution [1, 21]. In the case of training a deep CNN which imposes a non-convex loss, the fixed step-size SGD (with momentum [23]) also converges to non-optimal loss, so-called *error plateau* [17, 10]. As a common practice toward better optimization, we usually decay (decrease) the learning rate so that SGD proceeds beyond the error plateau.

In this study, we consider a training *phase*, a period of constant learning rate as shown in Fig. 1a, and focus on the error plateau to which SGD locally converges at each training phase. Through analyzing a loss surface around the error plateau and a phase-wise structure of SGD optimization, the proposed method aggregates (averages) the model parameters along the optimization sequence *within* a phase for effectively enhancing the local convergence of SGD as well as providing the favorable initialization for the next phase. Thereby, the proposed method improves the whole SGD optimization process through repeated phase-wise modification while keeping almost the same computation cost as the original SGD. The contributions of this paper are summarized as follows; (1) through the phase-wise analysis of SGD optimization process, we propose a simple yet effective practical method to improve SGD by means of phase-wise parameter averaging and (2) in the experiments using various datasets/CNNs on image classification, the method is thoroughly evaluated to exhibit favorable performance in comparison with the other methods.

1.1. Related works

We briefly review the related methods which aggregate CNN models in a way of *external* or *internal* fusion. Ensemble of CNN models independently trained with different random seeds is used to externally fuse their outputs for final classification [17] and recently, those models are sampled along the optimization sequence [6] without repeating whole training process. Model ensemble, however, requires large amount of computation cost and mem-

ory consumption in accordance with the number of models to use. On the other hand, from optimization viewpoint, internally averaging model parameters along the optimization sequence has been shown to be effective for convergence by Polyak-averaging [24], while averaging parallel training [8] demands extra computation resources. Based on [6], stochastic weight averaging [13] aggregates parameters of models which are diversely trained through fine-tuning. These internal methods work rather *after* training, thus being complementary to the following methods and ours that operate on the optimization process itself *during* training. In [27], the aggregation weights are adaptively learned, though requiring k-times extra-memory to store k multiple model parameters. Toward faster convergence, the *Lookahead* method [36] efficiently applies model aggregation every k updates by means of moving average which requires a buffer only for single model parameter. The proposed method is connected to the above-mentioned internal parameter fusion to improve optimization process *during* training. While the works [6, 13] focus on the behavior around the optimum based on *pre-trained* models, our first analysis (Sec. 2.1.1) reveals that the similar behavior can be found in *early* phase of training, which is distinctively necessary for constructing our repeated procedure (Algorithm 1). Through phase-wise analyses of SGD process, we formulate an efficient optimization method to improve performance with almost the same amount of computation as the original SGD, in contrast to the previous methods [27, 36]. It might be noteworthy that those external and internal fusion techniques are also applied in the framework of semi-supervised learning [18, 30].

2. Optimization by SGD

Stochastic gradient descent (SGD) [26, 23] is widely applied to train neural networks by efficiently minimizing an objective loss, such as a softmax cross-entropy loss for classification. For optimizing a deep CNN on large-scale training data, SGD iteratively updates the model parameters $\boldsymbol{\theta}$ by means of gradients of the loss on a *mini-batch* with a learning rate λ:

$$\text{Mini-batch wise: } \boldsymbol{\theta}_{[b]} = \boldsymbol{\theta}_{[b-1]} + \lambda \boldsymbol{\delta}_{[b]}, \tag{1}$$

$$\text{Epoch-wise: } \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \lambda \sum_{b=1}^{B} \boldsymbol{\delta}_{t,[b]}, \tag{2}$$

where b indicates the mini-batch index and $\boldsymbol{\delta}_{[b]}$ is composed of a (negative) gradient of the loss w.r.t $\boldsymbol{\theta}$ and its momentum computed over the training samples in the b-th mini-batch. B mini-batches are drawn so as to visit all the training samples in a cycle, so-called *epoch*, and we also show in (2) an epoch-wise update for $\boldsymbol{\theta}_t$ by aggregating $\boldsymbol{\delta}_{t,[b]}$ at the t-th epoch.

To produce favorably optimal CNNs, we can employ the standard training practice regarding a learning rate as follows. The learning rate λ is initially set to small value, e.g., $\lambda = 0.1$, and then is decayed by a certain factor, e.g., $\lambda = 0.1 \rightarrow 0.01$, after passing through some epochs, as shown in Fig. 6. According to this standard practice, we partition the whole SGD training process into several training *phases*. The training phase is a period of constant learning rate and is of the higher level than *epoch* (Fig. 1a).

2.1. Proposed method

We analyze the SGD optimization process in a phase-wise manner from the following two perspectives, error plateau (Sec. 2.1.1) and repeated structure (Sec. 2.1.2), toward improving SGD.

2.1.1 Error plateau

In the case of strongly convex loss functions, SGD with *properly* fixed learning rate λ renders linear convergence up to a radius around the optimal solution [1, 21], depending on the variances of stochastic gradients; toward faster convergence, some variants of SGD are proposed to reduce the gradient variance, such as in SVRG [14] and SAGA [3], though being less effective for training deep neural networks [4]. We here practically analyze the SGD optimization on each training phase (fixed λ) for a CNN which poses a non-convex loss.

Fig. 1 shows how the training of WideResNet-28-10 [33] on Cifar-100 [16] dataset proceeds by SGD (momentum=0.9) over 3 phases of the learning rates $\lambda \in \{0.1, 0.01, 0.001\}$. The training loss in Fig. 1a converges to non-zero value, *error plateau* [10], in each phase. To monitor the model parameters in the 1st training phase, Fig. 1b shows 2-D PCA subspace of the last FC classifier weights in the CNN which approximately represent the status of CNN parameters; the classifier weight matrix of 640×100 is flattened into a 64K-dimensional vector. In addition, similarly to an attracter analysis [29], the time-delayed representations of the parameters are also shown in Fig. 1 whose coordinates are described by Euclidean distances between adjacent parameters $(\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\|_2, \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-2}\|_2)$. Those results demonstrate that the fixed step-size SGD actually converges around the optimum while exhibiting a certain curvature of the optimization sequence.

We then explore the loss surface on which the SGD sequence traverses. As in the model connectivity [6], we measure the training/test losses on a (line) segment connecting two parameters of the SGD sequence. Fig. 2 shows losses on four segments connecting five anchor points of $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_{20}, \boldsymbol{\theta}_{40}, \boldsymbol{\theta}_{60}, \boldsymbol{\theta}_{80}\}$ in the 1st phase. During early training ($\boldsymbol{\theta}_1 \sim \boldsymbol{\theta}_{20}$), there contains no barrier but a simply decreasing surface, which is also accordance with the anal-

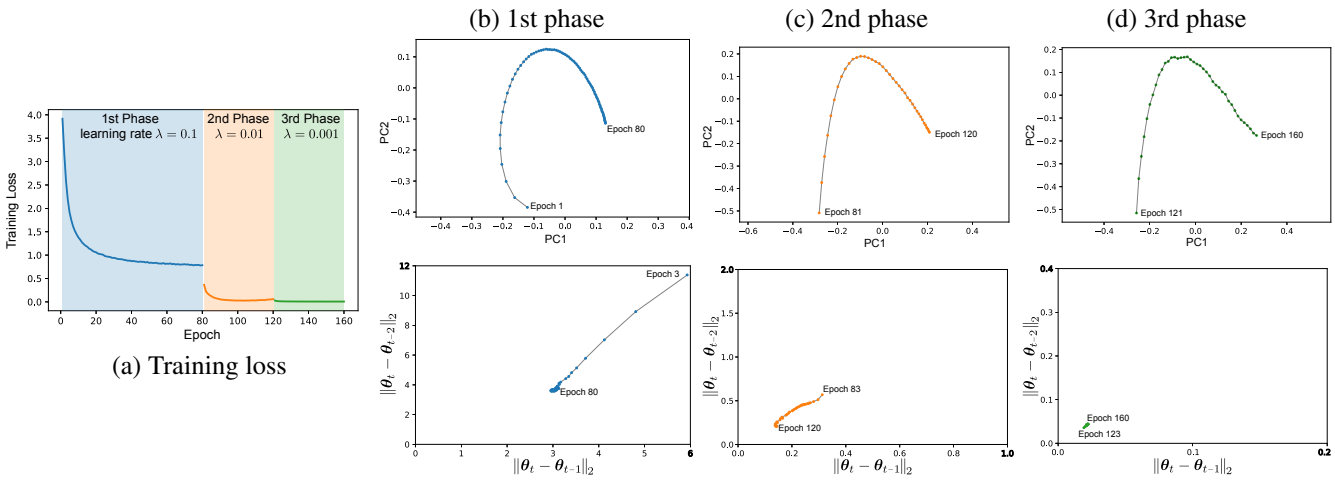

Figure 1. Some visualization of training CNN. We apply momentum-SGD [23] to train Wide-ResNet-28-10 [33] on Cifar-100 dataset [16] over three training phases of $\lambda \in \{0.1, 0.01, 0.001\}$ (a). Each phase is detailed by PCA and time-delay (TD) plot in (b,c,d).

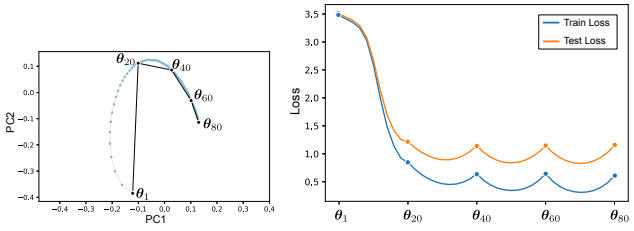

Figure 2. Training and test losses on the line segments connecting five parameter points in the 1st training phase ($\lambda = 0.1$).

ysis of [7]. This indicates that SGD effectively proceeds at the early training epochs by simply decreasing the loss. On the other hand, the loss surfaces on the segments at the later training epochs are concave, thereby demonstrating that SGD walks around the optimal point with a certain radius while being trapped in an error plateau. According to the standard practice of training CNNs, a learning rate is decayed so as to go beyond the error plateau and further decrease the loss by SGD.

The analysis based on Fig. 2 also brings us possibility to construct a better parameter by simply aggregating (averaging) the parameters on the *later* sequence of a phase to approach the mode of the concave loss surface. For that purpose, we analyze the granularity of the SGD sequence points to be averaged. From a statistical viewpoint, it is desirable to exploit samples $\boldsymbol{\theta}$ of high *diversity* around the error plateau for effectively estimating the better parameter via averaging that produces a smaller loss. In this case, there are several levels of granularity for measuring the diversity from mini-batch (1) to epoch (2). Fig. 3 shows losses of the two-point average $\frac{1}{2}(\boldsymbol{\theta}_{80} + \boldsymbol{\theta})$ at various temporal intervals between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_{80}$. While the granularity of mini-batch level is too small to sufficiently decrease the loss due to less diversity, the interval greater than one epoch contributes to favorably reducing the loss; we thus employ the epoch-wise parameter representation $\boldsymbol{\theta}_t$ for averaging. It should be noted that it is desirable to consider favorably smaller interval so that we can exploit samples as many as possible for averaging.

Fig. 3 also demonstrates that the point far from the error plateau less contributes to decreasing loss. Considering hardness to properly detect the period of an error plateau, we apply the following exponential averaging to pay more attention to the later training epochs at which the parameter probably reaches the error plateau:

$$\bar{\boldsymbol{\theta}} = \frac{1-\gamma}{1-\gamma^T} \sum_{t=1}^{T} \gamma^{T-t} \boldsymbol{\theta}_t \;\Leftrightarrow\; \bar{\boldsymbol{\theta}} \leftarrow \gamma \bar{\boldsymbol{\theta}} + (1-\gamma)\boldsymbol{\theta}_t, \quad (3)$$

where T is the number of epochs in the training phase, γ is a (pre-fixed) decaying factor; the averaging is efficiently computed by applying an iterative update based on the learnt parameter $\boldsymbol{\theta}_t$ at the t-th epoch. In the case that CNNs contain batch normalization [12], after averaging parameters in (3), we additionally run only forward-pass of the network to calibrate the local statistics at the batch normalization layers as in [13, 6].

The number of training epoch T is a hyper-parameter to be tuned by users and thus is varied across phases, CNN model and tasks; for example, we use $T_1 = 80$ epochs to reach the plateau in the first phase and $T_2 = 40$ epochs in the second phase in the case of Fig. 1a. It is necessary to robustly cope with the training phase duration T of various length. To this end, we define the decaying factor γ according to T as $\gamma = \gamma_0^{\frac{80}{T}}$ where γ_0 is a pre-fixed base factor; in this study, we set $\gamma_0 = 0.9$ as a *rule of thumb* based on the ablation experiments in Sec. 3. Thereby, the weight γ^{T-t} for the parameter $\boldsymbol{\theta}_t$ in (3) is adaptively changed according to the relative position of the current epoch t by

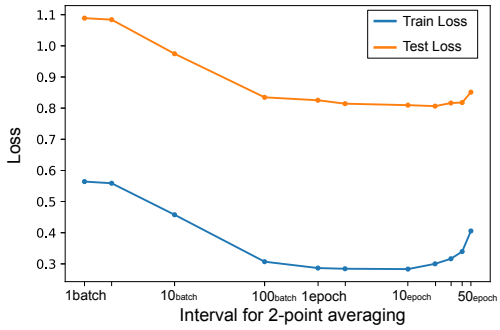

Figure 3. Losses by two-point average $\frac{1}{2}(\boldsymbol{\theta}_{80}+\boldsymbol{\theta})$ at various temporal intervals.

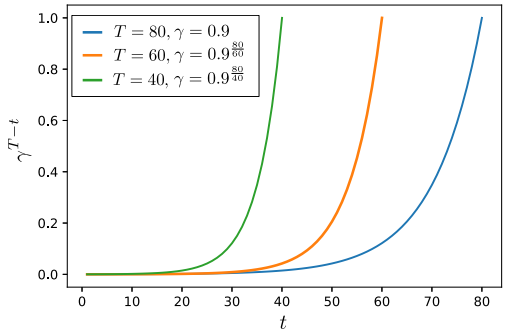

Figure 4. Exponential weight for averaging parameters $\{\boldsymbol{\theta}_t\}_{t=1}^{T}$ on various T; we set the base factor as $\gamma_0 = 0.9$.

$\gamma^{T-t} = \gamma_0^{\frac{T-t}{T}80}$, as shown in Fig. 4.

2.1.2 Repeated optimization structure

After reaching the error plateau, the learning rate λ is decreased to further proceed the SGD optimization in the next training phase (Fig. 1a). This procedure is *repeated* several times to finally produce the optimal CNN model parameter. Therefore, we regard such an SGD optimization not as a whole but rather as a composite of several training phases. Actually, as shown in Fig. 1b-d, the optimization in each training phase proceeds similarly from the initial point to the convergence even under the different learning rate. We here focus on the *initial* parameter at the training phase. In the first ($p = 1$) phase, some techniques such as [9] are applied to initialize $\boldsymbol{\theta} = \boldsymbol{\theta}_0^{(1)}$, while the second and later ($p > 1$) phases receive as an initial point the last estimation $\boldsymbol{\theta}_{T_{p-1}}^{(p-1)}$ that the previous $(p-1)$ phase finally produces. From this viewpoint of phase-wise initialization, as the better initialization can induce the better solution, we leverage the averaged parameter $\bar{\boldsymbol{\theta}}^{(p-1)}$ via (3) at the $p-1$-th phase to the initialization of the next p-th phase, instead of the last estimate $\boldsymbol{\theta}_{T_{p-1}}^{(p-1)}$. Namely, we improve the SGD optimization simply yet effectively by injecting the parameter aggregation (3) into the convergence at each training phase,

Algorithm 1 Proposed Optimizer

Input: $\{T_p, \lambda_p\}_{p=1}^{P}$: Number of epochs T_p and learning rate λ_p at the p-th training phase,
$\boldsymbol{\theta}$: Model parameters,
$\gamma_0 = 0.9$: Base decaying factor for averaging.

1: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$: *Parameter initialization*
2: **for** $p = 1$ to P **do**
3: $\quad\bar{\boldsymbol{\theta}} \leftarrow \mathbf{0}$, $\gamma = \gamma_0^{\frac{80}{T_p}}$
4: $\quad$**for** $t = 1$ to T_p **do**
5: $\quad\quad$**for** $b = 1$ to B **do**
6: $\quad\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda_p \boldsymbol{\delta}_{t,[b]}$: *Mini-batch based update by* SGD
7: $\quad\quad$**end for**
8: $\quad\quad\bar{\boldsymbol{\theta}} \leftarrow \gamma\bar{\boldsymbol{\theta}} + (1-\gamma)\boldsymbol{\theta}$: *Exponential moving average*
9: $\quad$**end for**
10: $\quad\boldsymbol{\theta} \leftarrow \frac{1}{1-\gamma^T}\bar{\boldsymbol{\theta}}$: *Initialization for the next training phase*
11: **end for**

Output: $\boldsymbol{\theta}$: Optimized model parameters

which simultaneously provides the better initialization for the next phase.

Based on these two analyzes (Sec. 2.1.1&2.1.2), the proposed method is formulated to improve SGD optimization, as summarized in Algorithm 1. It is so simple that the SGD optimization procedure is modified only once at each training phase (line 8 in Algorithm 1).

2.2. Discussion

We discuss connection between aggregating parameters in (3) and decaying learning rate, and the computation cost of the proposed method (Algorithm 1), and convergence analysis.

Connection to Decayed Learning Rate: Given the epoch-wise update formula in (2), the parameter averaging (3) with the weight $\alpha_t = \frac{1-\gamma}{1-\gamma^T}\gamma^{T-t}$ can be unfolded to

$$\bar{\boldsymbol{\theta}} = \sum_{t=1}^{T}\alpha_t\boldsymbol{\theta}_t = \boldsymbol{\theta}_0 + \sum_{t=1}^{T}\alpha_t(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) \tag{4}$$

$$= \boldsymbol{\theta}_0 + \sum_{t=1}^{T}\alpha_t\sum_{\tau=1}^{t}\lambda\boldsymbol{\Delta}_\tau = \boldsymbol{\theta}_0 + \sum_{\tau=1}^{T}\left[\sum_{t=\tau}^{T}\alpha_t\right]\lambda\boldsymbol{\Delta}_\tau \tag{5}$$

$$= \boldsymbol{\theta}_0 + \sum_{\tau=1}^{T}\frac{1-\gamma^{T-\tau+1}}{1-\gamma^T}\lambda\boldsymbol{\Delta}_\tau, \tag{6}$$

where $\boldsymbol{\theta}_0$ is an initial parameter and $\boldsymbol{\Delta}_t$ is the update at the t-th epoch summing up the mini-batch wise updates, $\boldsymbol{\Delta}_t = \sum_{b=1}^{B}\boldsymbol{\delta}_{t,[b]}$ in (2). This implies that averaging parameters (3) is related to updating $\boldsymbol{\theta}$ with the *decayed* learning rate $\frac{1-\gamma^{T-t+1}}{1-\gamma^T}\lambda$ (Fig. 6). It, however, should be noted that in (6) the update $\boldsymbol{\Delta}_t$ is computed at the parameters $\boldsymbol{\theta}_t$ sequentially trained with the *constant* learning rate λ. Thus,

the averaging in (3), equivalently (6), can be regarded as *a posteriori* decay of learning rate in contrast to the *a priori* approach that updates parameters $\boldsymbol{\theta}$ with the decayed learning rate $\frac{1-\gamma^{T-t+1}}{1-\gamma^T}\lambda$. These two approaches are empirically compared in Sec. 3.

On the other hand, it is also possible to embed the exponential moving average (EMA), $\bar{\boldsymbol{\theta}} \leftarrow \gamma\bar{\boldsymbol{\theta}} + (1-\gamma)\boldsymbol{\theta}_t$, into the parameter updating (2) during training in an *in-vivo* manner [27, 36]. In case of injecting EMA into every update, however, the parameter update formula is reduced into

$$\boldsymbol{\theta}_{t-1} \leftarrow \bar{\boldsymbol{\theta}},\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \lambda\boldsymbol{\Delta}_t,\ \bar{\boldsymbol{\theta}} \leftarrow \gamma\bar{\boldsymbol{\theta}} + (1-\gamma)\boldsymbol{\theta}_t \quad (7)$$

$$\Leftrightarrow\ \bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + (1-\gamma)\lambda\boldsymbol{\Delta}_t, \quad (8)$$

which corresponds to constantly decaying the learning rate by the factor of $1-\gamma$; this frequent injection is essentially equivalent to the original SGD. In [36], the issue is addressed by updating every k steps. In contrast, we disentangle the two effects of learning rate and EMA by injecting the averaged parameters into training only at the end of each training phase. Thus, the proposed method retains the favorable characteristics of SGD optimization at early training epochs (Fig. 2) while alleviating the convergence issue as well as improving the initialization at each phase.

Computation cost: The proposed method in Algorithm 1 requires almost the same computation cost as the standard SGD. The extra operation to update $\bar{\boldsymbol{\theta}}$ is performed only once per *epoch* and extra memory is used only for storing $\bar{\boldsymbol{\theta}}$, which can be done on CPU without consuming GPU memory. Transferring memory from CPU to GPU is only once at the end of each training phase to replace $\boldsymbol{\theta}$ with $\bar{\boldsymbol{\theta}}$.

Convergence analysis: We can simply apply the convergence analysis of SGD such as [1, 21, 37, 24] since the proposed method does not interfere the SGD optimization during each training phase. The method provides better initial point to the subsequent training phase as well as finally produces better convergent point [24]. We empirically show in Sec. 3 the effectiveness of the intervention in the initial points at respective training phases.

3. Experimental Results

We apply the proposed method to train various CNNs on image classification tasks as follows.

Dataset: We use Cifar-100/10 [16] and SVHN [22] of 32×32 pixel images as well as ImageNet [5] dataset of larger-sized images.

CNN: On Cifar-100/10 and SVHN, we apply the deep CNNs of VGG-like 13-layer network [30] with a weight decay of 0.0001 as well as ResNet34 [10], Wide-ResNet28-10 [33] and DenseNet121 [11] with a weight decay of 0.0005. The size of mini-batch is 128. For ImageNet, we train ResNet50 [10], ResNeXt50 [31] and VGG16 [28] with a weight decay of 0.0001 and a mini-batch size of 256.

Data augmentation: 32×32 pixel images are preprocessed by standardization (0-mean and 1-std) and are subject to random cropping through 4-pixel padding as data augmentation; on Cifar-100/10, we additionally apply random horizontal flipping. For ImageNet classification, we follow the standard procedure of data augmentation [10].

Evaluation: We report the average and standard deviation of error rates (%) over three runs with different random seeds on Cifar and SVHN, while measuring top-1 error rate (%) by single crop testing protocol [17] on ImageNet.

3.1. Ablation Study

We first analyze the proposed method from various aspects on Cifar-100 dataset [16]. According to the common practice, we apply the baseline SGD optimizer with a momentum of 0.9 and an initial learning rate of 0.1. The learning rate is then divide by 10 after the 80th and 120th training epochs to provide three training phases of $T \in \{80, 40, 40\}$ and $\lambda \in \{0.1, 0.01, 0.001\}$, as shown in Fig. 6.

Weight for averaging: In the exponential averaging (3), we apply the decaying factor $\gamma = \gamma_0^{\frac{80}{T}}$ where T denotes a period of a training phase (Sec. 2.1); thereby the parameters $\{\boldsymbol{\theta}_t\}_{t=1}^T$ are adaptively averaged even with various T (Fig. 4). We evaluate various base factor $\gamma_0 \in \{0.99, 0.95, 0.9, 0.8, 0.5\}$ and show the performance results in Table 1a. Too large γ_0 takes into account rather whole parameter points in a phase including ineffective ones far from the error plateau which degrade performance. On the other hand, the effect of averaging is impeded by smaller γ_0 which looks at only a few last samples near by $\boldsymbol{\theta}_T$. Thus, the moderately large γ is effective for the exponential averaging to provide the better convergent point $\bar{\boldsymbol{\theta}}$ for improving classification performance; based on Table 1a, we set $\gamma_0 = 0.9$, thereby $\gamma = 0.9^{\frac{80}{T}}$ in (3).

We then compare the exponential averaging with the method that learns weights for averaging [27]. For fair comparison, we replace the exponential weight $\alpha_t = \frac{1-\gamma}{1-\gamma^T}\gamma^{T-t}$ in (3) by the weights that the method [27] learns based on $\{\boldsymbol{\theta}_t\}_{t=1}^T$ in our optimization framework (Algorithm 1). The performance comparison in Table 1a shows that the exponential weighting by $\gamma = 0.9^{\frac{80}{T}}$ outperforms the learnt one. As shown in Fig. 5, the learnt weight is so long-tailed that irrelevant samples at the early training epochs contribute to averaging, while the simple exponential weighting is designed to focus on the more informative later epochs based on the analysis of error plateau in Sec. 2.1.

Training Epochs: We evaluate the robustness of the setting $\gamma = 0.9^{\frac{80}{T}}$ across various numbers of training epochs T per phase. Table 1b shows the performance results on shorter ($T_p \in \{60, 30, 30\}$), medium ($T_p \in \{80, 40, 40\}$) and longer ($T_p \in \{100, 50, 50\}$) training duration, demon-

Table 1. Ablation study on Cifar-100 dataset [16].

(a) Weights for averaging

CNN	γ_0 = 0.99	0.95	0.9	0.8	0.5	Learnt [27]
WRN28-10	18.58±0.12	18.25±0.10	18.34±0.05	18.38±0.22	18.79±0.14	19.45±0.18
ResNet34	22.49±0.38	21.78±0.34	21.72±0.29	22.07±0.57	21.99±0.39	22.92±0.35
DenseNet121	20.24±0.22	20.11±0.37	20.14±0.37	20.31±0.39	20.49±0.41	20.75±0.44
13-layer	26.47±0.13	26.12±0.04	26.05±0.08	26.20±0.14	26.57±0.09	26.65±0.19

(b) Training epochs (T_p)

CNN	$T_p \in \{60, 30, 30\}$ SGD	Ours	$T_p \in \{80, 40, 40\}$ SGD	Ours	$T_p \in \{100, 50, 50\}$ SGD	Ours
WRN28-10	19.47±0.11	18.84±0.19	19.02±0.11	18.34±0.05	18.73±0.24	18.15±0.10
ResNet34	23.05±0.24	22.31±0.20	22.62±0.33	21.72±0.29	22.09±0.35	21.34±0.26
DenseNet121	20.88±0.18	20.37±0.13	20.68±0.43	20.14±0.37	20.25±0.24	19.51±0.35
13-layer	26.69±0.18	25.88±0.22	26.52±0.16	26.05±0.08	26.02±0.15	25.15±0.07

(c) Frequency of averaging (d) Learning rate

CNN	No (SGD)	At the end [24]	Phase-wise (**ours**)	Every k steps [36]	$\frac{1-\gamma^{T-t+1}}{1-\gamma^T}\lambda$	cos [19]
WRN28-10	19.02±0.11	18.94±0.09	18.34±0.05	19.12±0.09	18.93±0.32	18.64±0.12
ResNet34	22.62±0.33	22.37±0.34	21.72±0.29	22.12±0.66	21.86±0.38	22.04±0.40
DenseNet121	20.68±0.43	20.78±0.41	20.14±0.37	20.25±0.33	20.29±0.22	20.20±0.27
13-layer	26.52±0.16	26.58±0.12	26.05±0.08	26.64±0.23	26.75±0.13	26.74±0.14

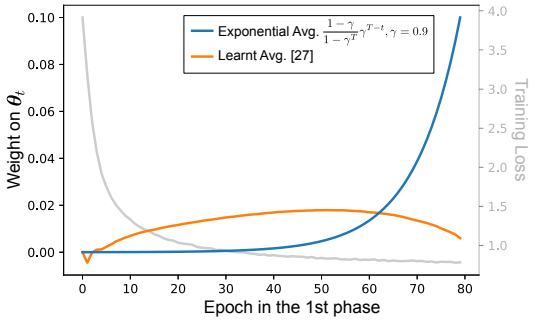

Figure 5. Weights for averaging $\{\boldsymbol{\theta}_t\}_{t=1}^{80}$ in the first training phase.

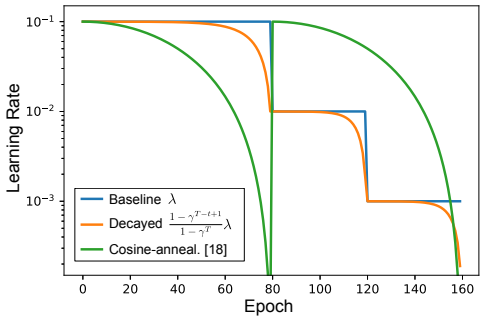

Figure 6. Various types of schedules for decaying learning rate.

strating that the proposed method with $\gamma = 0.9^{\frac{80}{T}}$ robustly contributes performance improvement. Thus, we can apply $\gamma = 0.9^{\frac{80}{T}}$ as a *rule of thumb* to various training settings.

Injection Timing: The proposed method triggers averaging parameters at the end of each training phase. According to the discussion in Sec. 2.2, we here compare the timing to inject the averaging into training process. The method of least frequent timing aggregates parameters at the end of whole training as in Polyak averaging [24], while a frequent injection is effectively performed by the *LookAhead* (LA) method [36] which replaces parameters with the moving averaging one every k steps; $k = 5$ as suggested in [36]. Table 1c shows the performance comparison of those methods, demonstrating that our approach outperforms the others. As shown in (8), frequently injecting the parameter averaging into training seems to be less contributing to improve the SGD optimization itself, being rather connected to decaying the learning rate. On the other hand, parameter aggregation at the end of training does not affect the optimization process of SGD while slightly improving the last convergence. The proposed method favorably modifies SGD by improving convergent point as well as initialization at each training phase. It is noteworthy that in this case the SGD optimization is modified only three times by the proposed method.

Decayed Learning Rate: As shown in (6), the exponential

Table 2. Performance comparison on various datasets.

CNN	ADAM	AMSGRAD	ADABOUND	SGD	LA	Ours
Cifar-100						
WRN28-10	29.16±0.24	28.47±0.55	22.63±0.21	19.02±0.11	19.12±0.09	18.34±0.05
ResNet34	28.17±0.39	27.11±0.18	24.68±0.24	22.62±0.33	22.12±0.66	21.72±0.29
DenseNet121	26.62±0.27	26.30±0.30	23.48±0.20	20.68±0.43	20.25±0.33	20.14±0.37
13-layer	27.73±0.15	28.02±0.08	30.02±0.30	26.52±0.16	26.64±0.23	26.05±0.08
Cifar-10						
WRN28-10	8.19±0.14	7.99±0.13	5.41±0.08	4.07±0.09	4.11±0.16	3.74±0.11
ResNet34	6.74±0.02	6.56±0.17	5.63±0.19	4.82±0.05	4.90±0.16	4.58±0.09
DenseNet121	7.10±0.16	6.42±0.08	5.37±0.04	4.79±0.03	4.56±0.05	4.49±0.03
13-layer	7.18±0.11	6.91±0.05	7.99±0.03	6.13±0.09	6.37±0.09	5.94±0.08
SVHN						
WRN28-10	3.73±0.11	3.66±0.08	2.97±0.08	2.64±0.05	2.63±0.05	2.51±0.02
ResNet34	3.59±0.07	3.64±0.05	3.00±0.08	2.81±0.05	2.86±0.14	2.65±0.03
DenseNet121	3.66±0.06	3.55±0.06	2.98±0.04	2.91±0.06	2.84±0.07	2.77±0.03
13-layer	3.64±0.13	3.59±0.01	3.97±0.11	3.37±0.02	3.43±0.06	3.29±0.05
ImageNet						
ResNet-50	33.79	31.15	24.72	23.98	23.48	23.16
ResNeXt-50	31.87	28.60	24.15	22.57	22.41	21.72
VGG16	36.34	30.37	25.77	25.41	24.89	24.67

averaging (3) is implicitly connected to decayed learning rate $\lambda_t = \frac{1-\gamma^{T-t+1}}{1-\gamma^T}\lambda$ where T and λ are the training duration and learning rate at the training phase. For comparison, we train CNNs by using so-decayed learning rate that decreases continuously and also apply the sophisticated learning rate of cosine-annealing [19]. These various types of learning rate are visualized in Fig. 6 and the performance results are shown in Table 1d. While the methods of decayed learning rates comparably perform slightly improving performance of the baseline SGD, the proposed method consistently outperforms them. It is generally hard to design the optimal schedule of decaying a learning rate in advance. In contrast, our method exploits fixed step-size SGD through aggregating parameters to improve convergence and initialization at each training phase without touching the learning rate.

3.2. Performance comparison

Finally, we compare the performance of various optimizers; in addition to LA method [36], we apply the adaptive methods of ADAM [15] AMSGRAD [25] and ADABOUND [20] by setting an initial learning rate to 0.001 and the other hyper-parameters to the values suggested in the respective papers. Those methods are evaluated not only on the datasets of 32×32 pixel images but also the large-scale ImageNet dataset [5]. On the ImageNet classification, we apply three training phases of $T_p \in \{60, 30, 30\}$ with $\lambda_p \in \{0.1, 0.01, 0.001\}$ for SGD, LA [36] and ours, while we changed the learning rate $\lambda_p \in \{0.001, 0.0001, 0.00001\}$ for the adaptive methods. Table 2 shows performance comparison on those datasets using diverse CNNs. The proposed method effectively improves SGD to favorably outperform the others, demonstrating the generality over various datasets and CNNs.

3.3. Application to Adaptive Optimization Methods

We have discussed so far the effect of the proposed method on SGD optimization. The method is so simple as to be applicable to the other types of optimization such as adaptive methods which are considered as comparison methods in the previous section; in Algorithm 1, the parameter updating by SGD in line 6 is simply replaced by the updating of adaptive methods, such as ADAM. One of the favorable characteristics of the adaptive methods is faster convergence compared to SGD, and thus we analyze such an aspect by applying the proposed method to them.

We train CNNs by the adaptive methods (ADAM, AMSGRAD and ADABOUND) over 80 epochs with a (constant) initial learning rate of 0.001 and then accordingly apply the proposed method with the decaying factor $\gamma = 0.9^{\frac{80}{80}} = 0.9$. For comparison, SGD is also applied with the same setting (80 epochs and constant learning rate of 0.1). Table 3 shows the performance results on Cifar-10/100 and SVHN datasets. The adaptive methods provides better performance, i.e., faster convergence, than SGD due to the adaptive scaling; the performance is effectively improved

Table 3. Performance by adaptive optimization methods on 80-epoch training.

	ADAM		AMSGRAD		ADABOUND		SGD	
CNN	Orig.	Ours	Orig.	Ours	Orig.	Ours	Orig.	Ours
Cifar-100								
WRN28-10	35.91±0.90	27.42±0.39	35.61±0.83	27.15±0.31	31.19±0.78	21.67±0.29	39.62±1.92	19.35±0.19
ResNet34	35.13±0.23	26.27±0.14	34.74±1.05	25.60±0.27	32.72±0.67	24.02±0.08	40.17±2.61	23.65±0.34
DenseNet121	32.64±0.86	24.85±0.32	31.87±0.71	24.68±0.22	31.55±0.95	22.19±0.23	39.37±1.13	21.86±0.14
13-layer	34.15±0.67	26.32±0.11	33.31±0.26	26.94±0.36	32.88±0.36	29.30±0.06	37.18±0.59	25.39±0.03
Cifar-10								
WRN28-10	12.83±0.40	7.81±0.32	11.84±0.59	7.91±0.34	9.66±0.69	5.19±0.13	13.61±0.80	4.78±0.06
ResNet34	11.20±0.52	6.85±0.02	10.48±0.35	6.23±0.09	8.78±0.15	5.39±0.08	16.12±0.75	6.01±0.10
DenseNet121	10.72±0.71	6.54±0.05	9.55±0.48	6.35±0.17	8.65±0.49	5.14±0.15	15.90±1.60	5.92±0.15
13-layer	10.22±0.36	6.49±0.14	9.95±0.01	6.45±0.08	10.18±0.34	7.76±0.11	10.74±0.03	5.77±0.03
SVHN								
WRN28-10	5.71±1.18	3.13±0.11	5.04±0.06	2.98±0.04	4.86±0.41	2.68±0.05	5.54±0.08	2.61±0.04
ResNet34	4.73±0.03	3.03±0.12	4.11±0.21	2.95±0.07	4.53±0.55	2.73±0.08	6.05±0.41	2.70±0.07
DenseNet121	4.43±0.35	2.98±0.02	4.52±0.31	2.98±0.04	4.66±0.25	2.74±0.02	5.99±0.19	2.82±0.06
13-layer	4.89±0.22	3.32±0.04	5.23±0.25	3.27±0.03	4.58±0.20	3.91±0.09	5.85±0.58	3.12±0.04

Table 4. Performance by adaptive optimization methods with the proposed method on (full) 160-epoch training.

CNN	ADAM	AMSGRAD	ADABOUND
Cifar-100			
WRN28-10	29.62±0.16	28.91±0.37	21.72±0.31
ResNet34	26.37±0.21	26.59±0.15	24.09±0.11
DenseNet121	26.38±0.36	26.04±0.19	22.43±0.18
13-layer	27.44±0.14	27.83±0.54	29.94±0.05
Cifar-10			
WRN28-10	8.26±0.14	8.10±0.04	5.22±0.06
ResNet34	6.90±0.07	6.18±0.07	5.39±0.10
DenseNet121	6.80±0.19	6.34±0.12	5.12±0.20
13-layer	6.80±0.14	6.63±0.13	7.86±0.10
SVHN			
WRN28-10	3.74±0.04	3.63±0.11	2.74±0.08
ResNet34	3.64±0.05	3.54±0.13	2.90±0.07
DenseNet121	3.59±0.12	3.58±0.06	2.82±0.03
13-layer	3.59±0.03	3.46±0.07	3.94±0.12

by ADABOUND [20] which is the SOTA variant of ADAM. By applying the proposed method, those performances are further improved, even surpassing the performance of full 160-epoch training (Table 2). Based on these results, we conjecture that the adaptive methods optimize CNN models as the ones closely around the (local) optimal point even by a constant initial learning rate and thus the proposed method effectively works to push the models toward the optimum via efficient averaging. Actually, the performance is not further improved even by further training with smaller learning rate (Table 4). Therefore, we could suggest to combine the adaptive methods equipped of a constant learning rater with the proposed averaging method; our future work includes the application of the adaptive methods to various tasks other than vision ones.

It is noteworthy that SGD equipped with the proposed method (Table 2) still outperforms those improved performance of the adaptive methods (Table 3); the SGD with our method is superior to the adaptive ones even on 80-epoch training (Table 3). This implies that SGD with a constant learning rate finds better local optima but cannot approach it due to the large (constant) learning rate. Thus, further training with smaller learning rate contributes to performance improvement as shown in Table 2.

4. Conclusion

We have proposed a simple yet effective optimization method to improve SGD. For training CNNs, fixed step-size SGD poses the issue regarding the error plateau while exhibiting favorable optimization property at the early training period. Through analyzing the error plateau and the repeated structure of SGD process with phase-wise decay of learning rate, the proposed method improves both the convergence and the initialization at each training phase in SGD optimization. The method is simple and requires almost the same computation cost as SGD. In the experiments on various datasets using diverse CNNs on image classification tasks, the proposed method produces favorable performance in comparison with the other methods.

References

[1] Francis Bach and Eric Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 451–459, 2011.

[2] Pratik Chaudhari and Stefano Soatto. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *arXiv,* 1710.11029 , 2017.

[3] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: a fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1646–1654, 2014.

[4] Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. *arXiv,* 1812.04529, 2018.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[6] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8789–8798, 2018.

[7] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In *International Conference on Learning Representations (ICLR)*, 2019.

[8] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck R. Cadambe. Local sgd with periodic averaging: Tighter analysis and adaptive synchronization. In *NeurIPS*, 2019.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, pages 2261–2269, 2017.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Journal of Machine Learning Research*, 37:448–456, 2015.

[13] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 876–885, 2018.

[14] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 315–323, 2013.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[16] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[18] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *ICLR*, 2017.

[19] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

[20] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In *ICLR*, 2019.

[21] Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1017–1025, 2014.

[22] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[23] Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[24] Boris T. Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–835, 1992.

[25] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *ICLR*, 2018.

[26] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[27] Damien Scieur, Edouard Oyallon, Alexandre d'Aspremont, and Francis Bach. Nonlinear acceleration of cnns. In *ICLR Workshop*, 2018.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[29] Floris Takens. Detecting strange attractors in turbulence. *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, 1981.

[30] A. Tarvainen and H. Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1195–1204, 2017.

[31] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 5987–5995, 2017.

[32] Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A walk with sgd. *arXiv,* 1802.08770, 2018.

[33] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[34] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR, abs/1212.5701*, 2012.

[35] Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Brando Miranda, Noah Golowich, and Tomaso Poggio. Theory of deep learning III: Generalization properties of sgd. CBMM Memo 67, 2017.

[36] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[37] Yi Zhou, Junjie Yang, Huishuai Zhang, Yingbin Liang, and Vahid Tarokh. Sgd converges to global minimum in deep learning via star-convex path. In *ICLR*, 2019.

[38] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. In *ICML*, 2019.