TRAINABLE WEIGHT AVERAGING: EFFICIENT TRAINING BY OPTIMIZING HISTORICAL SOLUTIONS

Tao Li[1], **Zhehao Huang**[1], **Qinghua Tao**[2], **Yingwen Wu**[1] **& Xiaolin Huang**[*1]
[1]Department of Automation, Shanghai Jiao Tong University
[2]ESAT-STADIUS, KU Leuven

ABSTRACT

Stochastic gradient descent (SGD) and its variants are considered as the de-facto methods to train deep neural networks (DNNs). While recent improvements to SGD mainly focus on the descent algorithm itself, few works pay attention to utilizing the historical solutions—as an iterative method, SGD has gone through substantial explorations before convergence. Recently, an interesting attempt is stochastic weight averaging (SWA), which significantly improves the generalization by simply averaging the solutions at the tail stage of training. In this paper, we realize that the averaging coefficients could be determined in a trainable manner and propose *Trainable Weight Averaging* (TWA), a novel optimization method in the reduced subspace spanned by historical solutions. TWA has much greater flexibility and can be applied to the head stage of training to achieve training efficiency while preserving good generalization capability. Further, we propose a distributed training scheme to resolve the memory burden of large-scale training with efficient parallel computation. In the extensive numerical experiments, (i) TWA achieves consistent improvements over SWA with less sensitivity to learning rate; (ii) applying TWA in the head stage of training largely speeds up the convergence, resulting in over 40% time saving on CIFAR and 30% on ImageNet with improved generalization compared with regular training. The code of implementation is available `https://github.com/nblt/TWA`.

1 INTRODUCTION

Training deep neural networks (DNNs) usually requires a large amount of time. As the sizes of models and datasets grow larger, more efficient optimization methods together with better performance are increasingly demanded. In the existing works, great efforts have been made to improve the efficiency of stochastic gradient descent (SGD) and its variants, which mainly focus on adaptive learning rates (Duchi et al., 2011; Zeiler, 2012; Kingma & Ba, 2015; Loshchilov & Hutter, 2019; Yao et al., 2021; Heo et al., 2021) or accelerated schemes (Polyak, 1964; Nesterov, 1983; 1988; 2003). As an iterative descent method, SGD generates a series of solutions during optimization. These historical solutions provide dynamic information about the training and have brought many interesting perspectives, e.g., trajectory (Li et al., 2022), landscape (Garipov et al., 2018), to name a few. In fact, they can also be utilized to improve the training performance, resulting in the so-called stochastic weight averaging (SWA) (Izmailov et al., 2018), which shows significantly better generalization by simply averaging the tail stage explorations of SGD. A similar idea could be found in Szegedy et al. (2016), which designs an exponential moving average (EMA) and considers that a heuristic strategy could be better than equivalently averaging. The success of SWA and EMA encourages more in-depth investigations on the roles of historical solutions obtained during the training (Athiwaratkun et al., 2018; Nikishin et al., 2018; Yang et al., 2019).

In this paper, our main purpose is to utilize historical solutions by optimizing them, rather than using fixed averaging (e.g., SWA) or a heuristic combination (e.g., EMA). With such an optimized averaging scheme, we can achieve higher accuracy using only the solutions in a relatively early state (i.e. the head stage). In other words, we speed up the training and meanwhile improve the performance. The idea of utilizing these early solutions in DNNs' training is mainly inspired by two facts. On the

[*]Corresponding author (xiaolinhuang@sjtu.edu.cn)

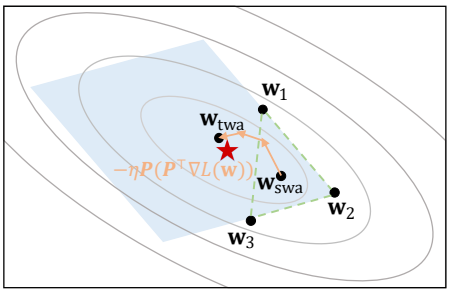


Figure 1: TWA intuition.

Input: Sampled weights $\{\boldsymbol{w}_i\}_{i=1}^n$, Batch size b, Loss function $L : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, Learning rate η.
Output: Model trained with TWA
Orthogonalize $\{\boldsymbol{w}_i\}_{i=1}^n$ to $\{\boldsymbol{e}_i\}_{i=1}^n$;
Initialize $\boldsymbol{w}_{\text{twa}}^{(0)}$, $t = 0$, $\boldsymbol{P} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n]$;
while *not converged* **do**
  Sample batch data: $\mathcal{B} = \{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^b$;
  Compute gradient: $\boldsymbol{g} = \nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w}_{\text{twa}}^{(t)})$;
  Update weights: $\boldsymbol{w}_{\text{twa}}^{(t+1)} = \boldsymbol{w}_{\text{twa}}^{(t)} - \eta \boldsymbol{P}(\boldsymbol{P}^\top \boldsymbol{g})$;
  $t = t + 1$;
end while
Return $\boldsymbol{w}_{\text{twa}}^{(t)}$

Algorithm 1: TWA algorithm.

one hand, high test accuracy commonly starts appearing at an early stage. For example, a PreAct ResNet-164 model (He et al., 2016) achieves over 80% test accuracy within 10 training epochs on CIFAR-10 (Krizhevsky & Hinton, 2009), while requiring to complete the whole 200 epochs to reach its final 95% accuracy. This observation also coincides with the recent findings that the key connectivity patterns of DNNs emerge early in training (You et al., 2020; Frankle et al., 2020), indicating a well-explored solution space formed. On the other hand, simply averaging the solutions collected at the SWA stage immediately provides a huge accuracy improvement, e.g. over 16% on CIFAR-100 with Wide ResNet-28-10 (Zagoruyko & Komodakis, 2016) than before averaging (Izmailov et al., 2018). These facts point out a promising direction that sufficiently utilizing these early explorations may be capable of quickly composing the final solution while obtaining good accuracy.

As the model parameters go through a rapid evolution at the early stage of training, a simple averaging strategy with fixed weighting coefficients as in SWA and EMA can result in large estimation errors. We introduce a *Trainable Weight Averaging* (TWA), which allows explicit adjustments for the averaging coefficients in a trainable manner. Specifically, we construct a subspace that contains all sampled solutions during the training and then conduct efficient optimization therein. As optimization in such a subspace takes into account all possible averaging choices, we are able to adaptively search for a good set of averaging coefficients regardless of the quality of sampled solutions and largely reduce the estimation errors. The proposed optimization scheme is essentially the gradient projection onto a tiny subspace. Hence, the degree of freedom for training is substantially reduced from the original millions to dozens or hundreds (equal to the dimension of the subspace), making TWA enjoy fast convergence and meanwhile immune to overfitting, the latter explains that better training accuracy of TWA over SWA or EMA can lead to better test accuracy. In extensive experiments with various network architectures on different tasks, we reach superior performance with TWA applied to the head stage of training. For instance, we attain $1.5 \sim 2.2\%$ accuracy improvement on CIFAR-100 and 0.1% on ImageNet with over 40% and 30% training epochs reduced, respectively, compared with the regular training.

In summary, we make the following contributions:

- We propose Trainable Weight Averaging (TWA) that allows the averaging coefficients determined in a trainable manner instead of a pre-defined strategy. It brings consistent improvements over SWA or EMA with reduced estimation error.
- We successfully apply TWA to the head stage of training, resulting in a great time saving (e.g. over 40% on CIFAR and 30% on ImageNet) compared to regular training along with improved performance and reduced generalization gap.
- Our TWA is easy to implement and can be flexibly plugged into different stages of training to bring consistent improvements. It provides a new scheme for achieving efficient DNNs' training by sufficiently utilizing historical explorations.

2 METHOD

In this section, we first formulate the optimization target of TWA. Then a detailed training algorithm is introduced, which consists of two phases: Schmidt orthogonalization and projected optimization.

Note in this paper, the model's weights are aligned as a vector, i.e., $\boldsymbol{w} \in \mathbb{R}^D$, where D denotes the number of parameters.

2.1 Optimization Target

In SWA, weight averaging is simply given by $\boldsymbol{w}_{\text{swa}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{w}_i$, where n solutions of the network collected at the tail of training are equally weighted. Such an averaging strategy has been proven quite effective with improved generalization ability. However, equally averaging could not always be a perfect solution, which motivates some heuristic modifications on weighting strategy, e.g., EMA. Both SWA and EMA are fixed averaging strategies, which may not adequately adapt to the head stage of training and would result in estimation errors, due to the fact that early historical solutions have not stepped into a stationary distribution.

In this paper, we propose to optimize the averaging coefficients of different weights with the hope of reducing the corresponding estimation error. Specifically, the set of possible TWA solutions considered, i.e., $\boldsymbol{w}_{\text{twa}}$, can be represented as follows:

$$A = \{\alpha_1 \boldsymbol{w}_1 + \alpha_2 \boldsymbol{w}_2 + \cdots + \alpha_n \boldsymbol{w}_n \mid \alpha_i \in \mathbb{R}\}. \tag{1}$$

The weight vectors between consecutive solutions could have a high cosine similarity. To decouple them and for better optimization, we will further orthogonalize $\{\boldsymbol{w}_i\}_{i=1}^n$ and find a set of orthogonal bases $\{\boldsymbol{e}_i\}_{i=1}^n$ to support the solution space, i.e., $A = \{\beta_1 \boldsymbol{e}_1 + \beta_2 \boldsymbol{e}_2 + \cdots + \beta_n \boldsymbol{e}_n \mid \beta_i \in \mathbb{R}\}$. Then we search for a good solution $\boldsymbol{w}_{\text{twa}}$ in A by optimizing the following problem,

$$\begin{aligned} \min_{\beta_1,\beta_2,\cdots,\beta_n} \quad & \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\mathcal{L}\left(f\left(\boldsymbol{w}_{\text{twa}};\boldsymbol{x}\right),\boldsymbol{y}\right)\right] + \frac{\lambda}{2}\sum_{i=1}^{n}\beta_i^2, \\ \text{s.t.} \quad & \boldsymbol{w}_{\text{twa}} = \beta_1 \boldsymbol{e}_1 + \beta_2 \boldsymbol{e}_2 + \cdots + \beta_n \boldsymbol{e}_n, \end{aligned} \tag{2}$$

where $\mathcal{L}(\cdot,\cdot)$ is the loss function as in regular training and the second term is a regularization coefficient $\lambda > 0$. Note that both SWA and EMA are special solutions of (2) without optimization.

Optimizing over β_i brings benefits in the view of training loss, and a good generalization ability could also be expected: in regular training, the number of optimization variables is D, which is very large, but in (2), there are only n averaging coefficients $\{\beta_i\}_{i=1}^n$ to be optimized. The significant dimensionality reduction could benefit better generalization.

2.2 Training Algorithm

Instead of directly optimizing β_i, we note that there exists a bijection between the coefficient space $\{\beta_i\}_{i=1}^n \in \mathbb{R}^n$ and the parameter space $\mathbb{R}^D$, i.e., each set of the coefficients is uniquely mapped to one point in the parameter space, which forms a complete subspace (with dimensionality n). We could alternatively optimize these coefficients in such a subspace.

We first focus on finding a set of orthogonal bases $\{\boldsymbol{e}_i\}_{i=1}^n$ to span the subspace that covers $\{\boldsymbol{w}_i\}_{i=1}^n$. This is a standard Schmidt orthogonalization, which sequentially takes the following steps:

$$\begin{cases} \boldsymbol{e}_k = \boldsymbol{w}_k - \left(\boldsymbol{w}_k{}^\top \boldsymbol{e}_1\right)\boldsymbol{e}_1 - \left(\boldsymbol{w}_k{}^\top \boldsymbol{e}_2\right)\boldsymbol{e}_2 - \cdots - \left(\boldsymbol{w}_k{}^\top \boldsymbol{e}_{k-1}\right)\boldsymbol{e}_{k-1}, \\ \boldsymbol{e}_k = \boldsymbol{e}_k / \|\boldsymbol{e}_k\|_2. \end{cases} \tag{3}$$

We then initialize the $\boldsymbol{w}_{\text{twa}}^{(0)}$ as one point in the subspace (e.g. $\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{w}_i$), and optimize the network's parameters therein. Let $\boldsymbol{P} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n]$, such optimization can be easily achieved by projecting the gradient onto the subspace via projection matrix $\boldsymbol{P}\boldsymbol{P}^\top$. We summarize the detailed training procedures of the proposed TWA in Algorithm 1 with an intuitive illustration in Figure 1. The detailed implementation is described in Appendix B.

3 Optimization Provides Better Flexibility

The key difference between SWA and TWA is that the averaging coefficients in TWA are determined in a trainable manner, or more precisely, are data-dependent. This potentially enables more precise estimation for the center minima and better tolerance for the outliers that are not aware by SWA.

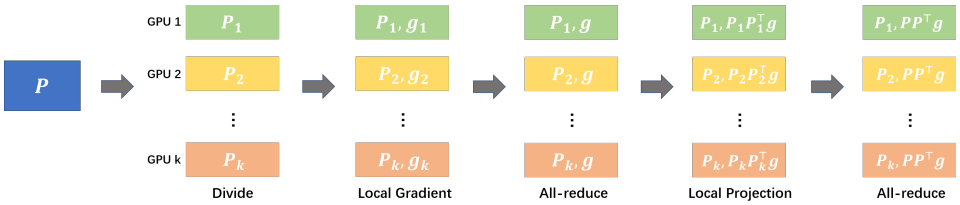


Figure 2: An efficient parallel scheme for distributed training.

Notice that in the view of coefficient optimization, there is no essential difference between SWA and EMA, which both provide specific and data-independent solutions. Thus, we in the following only compare TWA with SWA.

Mandt et al. (2017) demonstrated that under appropriate assumptions, running SGD with a constant learning rate is equivalent to sampling from a stationary Gaussian distribution, and the variance of the distribution is controlled by the learning rate. Accordingly, we assume the solutions at the tail stage of SGD training are sampled from a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ centered at the minimum $\boldsymbol{\mu}$ with covariance $\boldsymbol{\Sigma}$. Approximately, the sampled solutions $\{\boldsymbol{w}_i\}_{i=1}^n$ are independent random variables from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, as long as there are sufficient iterations between adjacent samplings. SWA and TWA provide two estimators for the minimum $\boldsymbol{\mu}$, i.e., $\boldsymbol{w}_{\text{swa}}$ and $\boldsymbol{w}_{\text{twa}}$. As an averaged solution, $\boldsymbol{w}_{\text{swa}}$ has statistically better estimation than any single solution due to the effect of variance reduction, while $\boldsymbol{w}_{\text{twa}}$ is approaching the center by minimizing the training loss. As long as the training loss serves as meaningful supervision (which holds under the typical assumption that $\boldsymbol{\mu}$ is the center minimum with lowest training loss Mandt et al. (2017); Izmailov et al. (2018)), $\boldsymbol{w}_{\text{twa}}$ could approach $\boldsymbol{\mu}$ better than $\boldsymbol{w}_{\text{swa}}$ with the posterior optimization for averaging coefficients. In this regard, $\boldsymbol{w}_{\text{twa}}$ could have a lower expected variance: $\mathbb{E}\left(\|\boldsymbol{w}_{\text{twa}} - \boldsymbol{\mu}\|_2^2\right) \leq \mathbb{E}\left(\|\boldsymbol{w}_{\text{swa}} - \boldsymbol{\mu}\|_2^2\right)$.

The advantages of optimizing averaging coefficients could be more prominent in the head stage of training, where the weights are going through a rapid evolution. A simple averaging strategy as SWA could introduce a large estimation error (as illustrated in Table 3 and 4), while TWA enables correcting it to a smaller estimation error via optimization. In fact, TWA provides much more flexibility to sufficiently utilize historical solutions and produces an optimized solution adaptively.

4 An Efficient Implementation for Distributed Training

The above discussion shows promising improvement by optimizing the historical solutions. The only issue one may worry about is the burden in storage (the additional time complexity is small as shown in Table 6). During optimization, TWA requires the projection matrix $\boldsymbol{P}$ involving dozens or hundreds of historical weights, which indeed poses a challenge for large models on storage burden. It is preferable to locate $\boldsymbol{P}$ in GPUs to enable efficient matrix operations. However, the size of $\boldsymbol{P}$ increases as the model becomes larger, making it prohibitive to store in a single GPU. To cope with this, we design an efficient scheme with parallel distributed training to enable a) partition of the memory burden of $\boldsymbol{P}$ into multiple GPUs and b) efficient parallel computation of gradient projection. As a result, we successfully optimize more than 900 historical solution coefficients for ResNet-50 on ImageNet task by 4 v100 GPUs. In our experiments, we use at most 300 historical solutions and there is still available space for larger tasks.

Suppose that there are k GPUs for parallel training. We first uniformly divide $\boldsymbol{P}$ into k sub-matrices as $\boldsymbol{P} = [\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_k]$, where each GPU stores a local sub-matrix $\boldsymbol{P}_i, i = 1, \cdots, k$. Recall that for an iteration in distributed training, each GPU computes a local gradient $\boldsymbol{g}_i$ and synchronizes it with other GPUs to obtain the global gradient through an efficient all-reduce operation (Rabenseifner, 2004). We mimic such a process for gradient projection: the local projected gradient $\boldsymbol{P}_i\boldsymbol{P}_i^\top \boldsymbol{g}$ is firstly computed in each card and then synchronized with others to obtain the global projected gradient with another all-reduce operation. We illustrate such a process in Figure 2.

For averaging n historical solutions with per size B, the memory burden for each GPU card is reduced to $\lceil n/k \rceil B$, while the computation of gradient projection is also reduced to $\mathcal{O}(\lceil n/k \rceil D)$ (D

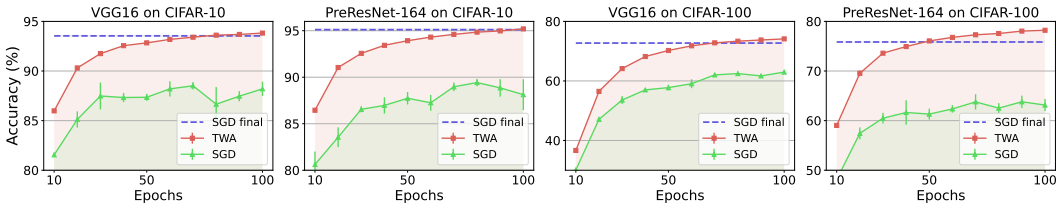


Figure 3: Performance comparisons on before and after TWA w.r.t. different epochs of weights used. "SGD final" indicates the accuracy reached by regular SGD training and "TWA" is the corresponding accuracy reached by Algorithm 1 with these epochs of weights. The final accuracy of SGD training is plotted for reference. TWA dramatically lifts the SGD accuracy and outperforms the final accuracy of SGD within 100 epochs. The experiments are repeated over 3 trials.

is the number of the parameters). Hence, we can achieve efficient TWA training by making full use of the remaining memory of each GPU aside the forward/backward training.

5 Experiments

In this section, a series of numerical experiments are conducted, demonstrating the effectiveness of our proposed TWA for fast convergence and better performance. First, we show that TWA improves SWA in the existing SWA settings, i.e., used in the tail stage of training. Second, we apply TWA to the head stage of training, which brings significant efficiency improvements together with better performance. Then, we visualize loss / accuracy surfaces to demonstrate the improvements of TWA.

5.1 Experimental settings

Datasets. We experiment over three benchmark image datasets, i.e., CIFAR-10, CIFAR-100 (Krizhevsky & Hinton, 2009), and ImageNet (Deng et al., 2009). Following prior works (Izmailov et al., 2018; Yang et al., 2019), we apply standard preprocessing for experiments on CIFAR datasets, and adopt the preprocessing and data augmentation procedures in the public Pytorch example on ImageNet (Paszke et al., 2017).

Architectures. We use two representative architectures, VGG-16 (Simonyan & Zisserman, 2014) and PreAct ResNet-164 (He et al., 2016) on CIFAR experiments. For ImageNet, we use ResNet-18 and ResNet-50 (He et al., 2016).

Training. The main body of experiments contains two parts: (1) for the tail stage of training, we use the same hyper-parameters as in SWA (Izmailov et al., 2018) and then a larger tail learning rate is also tried. (2) for the head stage of training, we adopt the standard training protocol with a step-wise learning rate. For CIFAR, we run all experiments with 3 seeds and report the mean test accuracy. We use SGD optimizer with momentum 0.9, weight decay 10^{-4}, and batch size 128. We train the models for 200 epochs with an initial learning rate 0.1 and decay it by 10 at the 100th and the 150th epochs. For ImageNet, we follow official PyTorch implementation[1]. For TWA, we sample solutions once after each epoch training for CIFAR and uniformly sample 5 times per epoch for ImageNet. We use a scaled learning rate (Figure 4), which takes 10 epochs of training for CIFAR and 2 epochs for ImageNet for fast convergence. The regularization coefficient λ defaults to 10^{-5}. More details (including the number of the historical solutions used) could be found in Appendix A.

5.2 Improving SWA solutions

In this part, we compare SWA and TWA in the original SWA settings. Specifically, TWA and SWA use the *same* weights sampled from the tail stage of training. For CIFAR, we try two different tail learning rates: 0.05, the recommended one in (Izmailov et al., 2018), and 0.10, a larger one for the case with greater variance. The results in Table 1 show that TWA brings consistent improvements

[1]Available at `https://github.com/pytorch/examples/tree/main/imagenet`.

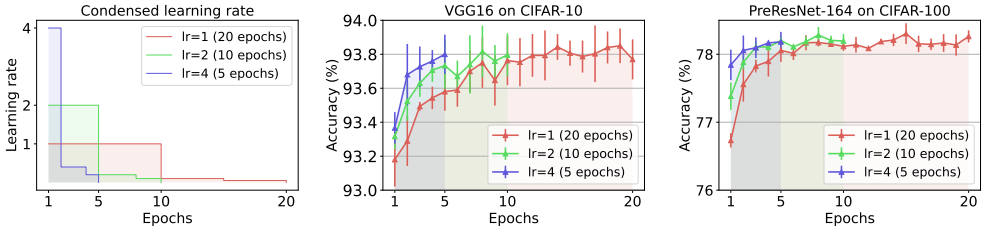


Figure 4: **Left**: Scaled learning rate schedules with different scaling factors; **Middle** and **Right**: Test accuracy curves of TWA w.r.t. to different schedules on CIFAR-10/100. Training in subspace shows high robustness to scaled learning rate, which enlarges the learning rate and reduces the corresponding training epochs. In this way, TWA achieves very fast convergence.

over SWA. Especially when the learning rate is not well-tuned, SWA's performance suffers a distinctive drop, but TWA is less sensitive since the estimation error could be well controlled through training. For instance, in the case of CIFAR-100 with VGG-16 and a tail learning rate of 0.10, the estimation error of SWA substantially increases while TWA achieves a significant accuracy improvement, i.e., 4.18%, over SWA. Note that for a fair comparison, TWA starts from the last sampled weights, not the averaged solution of SWA.

Table 1: Test accuracy ($\%$) on CIFAR-10/100 for tail training with different learning rates

		SWA_LR $= 0.05$		SWA_LR $= 0.10$	
DATASET	MODEL	SWA	TWA (+10)	SWA	TWA (+10)
CIFAR-10	VGG16	94.01 ± 0.04	94.16 ± 0.14	91.03 ± 0.14	92.41 ± 0.15
	PRERESNET-164	95.58 ± 0.09	95.65 ± 0.13	91.58 ± 0.45	92.61 ± 0.09
CIFAR-100	VGG16	74.71 ± 0.03	75.73 ± 0.18	65.52 ± 0.25	69.70 ± 0.45
	PRERESNET-164	80.20 ± 0.41	80.35 ± 0.27	78.14 ± 0.48	78.87 ± 0.27

On ImageNet, we experiment with ResNet-18/50 (He et al., 2016). Following (Izmailov et al., 2018; Yang et al., 2019), we start from pre-trained models in `torchvision.models` and collect model weights by running SGD optimizer up to 10 epochs (with a constant learning rate 0.005). In Table 2, we report the test accuracy and observe that with more sampling epochs, both TWA and SWA achieve better performance. Notably, TWA performs better than SWA by $0.1 \sim 0.3\%$, and such improvements are more obvious in the "5 EPOCHS" case. For example, using 5 epochs of sampled weights, TWA achieves 70.23% accuracy with ResNet-18 and 76.78% accuracy with ResNet-50, outperforming the SWA counterparts with 10 epochs. This indicates that TWA requires fewer historical solution samples to achieve a comparable or even better performance than SWA, due to its ability to reduce the estimation variance with optimized averaging coefficients.

Table 2: Top-1 accuracy ($\%$) on ImageNet for tail training with different averaging epochs

		5 EPOCHS		10 EPOCHS	
MODEL	PRETRAINED	SWA	TWA (+1)	SWA	TWA (+1)
RESNET-18	69.76	70.02	70.28	70.12	70.32
RESNET-50	76.13	76.62	76.78	76.74	76.93

5.3 EFFICIENT TRAINING AND BETTER GENERALIZATION

In the head stage of training, SWA usually fails due to the large estimation variance from fast-evolving solutions and large learning rate. Since TWA could reduce the variance and be less sensitive to the learning rate, it can also be expected to work well in this stage. If so, it is promising to simultaneously attain generalization improvements and training efficiency.

We first investigate the experiments on CIFAR datasets. The original training schedule contains 200 epochs and we take the first 100 epoch explorations for TWA. The results are given in Table 3. It can be observed that TWA achieves better performance compared to the regular SGD training with a significantly reduced generalization gap. For instance, we attain 1.52% accuracy improvement on CIFAR-100 with VGG-16 while the generalization gap is reduced by 9.56%. This suggests that a better solution could already be composed using these historical solutions without further training by more delicate learning rates, which instead may bring overfitting problems and harm the generalization. By comparisons, we also apply SWA to average these samples, which shows degraded performance due to the existence of estimation error. Apart from the good performance in accuracy, TWA also manifests its great potential in improving the training efficiency: we use only 10 epochs to complete the convergence, while the regular SGD needs 100 epochs. As TWA and SGD have nearly the same computation overhead per epoch, the time-saving is around 45% in TWA.

Table 3: Test accuracy ($\%$) and generalization gap ($\%$) on CIFAR-10/100 for the head training

		SGD (200 epochs)		SWA (100 epochs)	TWA ($100+10$ epochs)	
Dataset	Model	Accuracy	Gap	Accuracy	Accuracy	Gap
CIFAR-10	VGG16	93.54 ± 0.11	6.42	92.40 ± 0.08	93.79 ± 0.18	5.59 (↓ 0.83)
	PreResNet-164	95.11 ± 0.17	4.86	92.52 ± 0.04	95.19 ± 0.04	4.11 (↓ 0.75)
CIFAR-100	VGG16	72.72 ± 0.17	26.70	69.18 ± 0.24	74.24 ± 0.24	17.14 (↓ 9.56)
	PreResNet-164	75.85 ± 0.18	24.10	73.36 ± 0.34	78.11 ± 0.19	20.02 (↓ 4.08)

Generally, utilizing more epochs of explorations can provide a better estimation for the center minimum and hence lead to better performance. Then, we also study the impact of different averaging epochs, and the results are illustrated in Figure 3, where the final accuracy of SGD and the accuracy reached by SGD before averaging are also given for reference. It could be observed that the model's performance is consistently improved with more epochs of explorations. Notably, although each historical solution in a relatively short period of explorations is not good, satisfied solutions have already emerged in the subspace spanned by these solutions. Then through proper optimization in subspace, TWA could find them out, e.g., on CIFAR-100 with PreAct ResNet-164 model, averaging over 50 epochs via TWA has already matched the final performance of regular SGD training.

Table 4: Top-1 accuracy ($\%$) and generalization gap ($\%$) on ImageNet for head training

	SGD (90 epochs)		SWA (60 epochs)	TWA ($60+2$ epochs)	
Model	Accuracy	Gap	Accuracy	Accuracy	Gap
ResNet-18	69.82	-1.59	62.19	69.82	-2.36 (↓ 0.77)
ResNet-50	75.82	0.25	67.66	75.90	-0.68 (↓ 0.93)

For ImageNet, the efforts required for each epoch training are much greater, and hence efficient methods to reduce the training epochs are highly desirable. The comparison results of SGD/SWA/TWA are shown in Table 4. Besides the reduced generalization gap, TWA takes only 2 epochs to average the historical solutions of the first 60 epochs, reaching comparable or even better performance than regular SGD training with 90 epochs. For comparison, Lookahead (Zhang et al., 2019) is another advanced optimizer recently proposed for improving convergence and reported 75.49% accuracy at the 60th epoch (Table 2 in Zhang et al. (2019)) with an aggressive learning rate decay (i.e., the learning rate is decayed at the 30th, 48th, and 58th epochs), while our TWA reaches 75.70% with the same budget but simply using the conventional decay.

TWA is very flexible and can be readily applied to different training stages, and we also conduct an experiment by averaging the solutions of the final training period (i.e. 61-90 epochs) and simply performing TWA for one epoch training, as presented in Table 5. Such cheap training still shows to bring significant improvements (e.g. $+0.51\%$ on ResNet-50 for ImageNet). Thus, TWA can serve as an effective approach for composing a better final solution.

Table 5: Top-1 accuracy ($\%$) on ImageNet for tail training and short TWA

Model	SGD	TWA (+1 epoch)
ResNet-18	69.82	70.37
ResNet-50	75.82	76.34

Scaled learning rate The optimization of TWA is conducted in a very low-dimensional space, which also suppresses the sensitivity of the learning rate. In fact, we can allow a very large learning rate to accelerate the training. Thereby, we design a *scaled learning rate*, which linearly scales up the learning rate and reduces the training epochs accordingly, as shown in Figure 4. Within an appropriate range, scaling the learning rate largely speeds up the convergence without affecting the final performance. For example, with the learning rate of 4, TWA approaches the final accuracy with only 1 epoch and converges within 5 epochs.

Wall-clock time comparison In the above experiments, we report the number of training epochs, since the wall-clock time per epoch for SGD and TWA is similar. As an illustration, Table 6 provides the wall-clock time of training PreAct ResNet-164 for CIFAR-100 on one Nvidia Geforce GTX 2080 TI.

Table 6: Wall-clock time per epoch

OPTIMIZER	TIME PER EPOCH
SGD	59.04s
TWA	60.20s

Comparison with EMA EMA serves as an alternative to SWA, which averages the model weights along the training trajectory with exponential decay. It requires a hyper-parameter γ to control the averaging horizons. As a manually defined averaging strategy, EMA could be sensitive to learning rates, datasets, architectures, etc. Here we compare the performance of EMA with SWA and TWA in the head stage of training, where we try $\gamma = 0.99/0.999$. The results are illustrated in Figure 5. We observe that the performance of EMA varies significantly with different choices of γ. It could perform notably better than SWA in the early stage of training as more weight is paid to the latest solutions. Note that both EMA and SWA are fixed averaging strategies for adapting different training stages (essentially could be viewed as particular solutions of TWA). By optimizing the averaging coefficients, TWA could consistently achieve better performance.

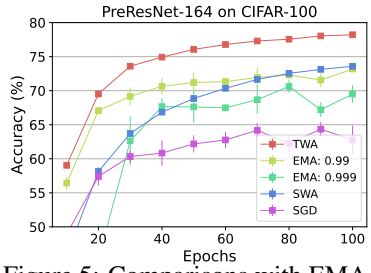


Figure 5: Comparisons with EMA.

Optimized Averaging Coefficients In Figure 6, we visualize the averaging coefficients α_i learned by TWA. Detailed derivation could be found in Appendix D. We observe all historical solutions could contribute to the final solutions. Solutions from the latter training stage are attached with more importance as expected. Different from the fix averaging strategies like SWA or EMA, such averaging coefficients enable to take full advantage of the historical solutions through delicate optimization and better adapt to the training dynamics.

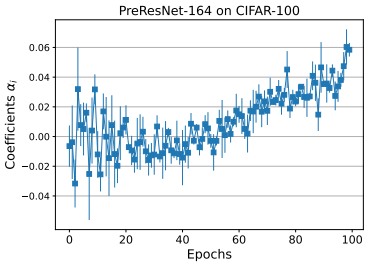


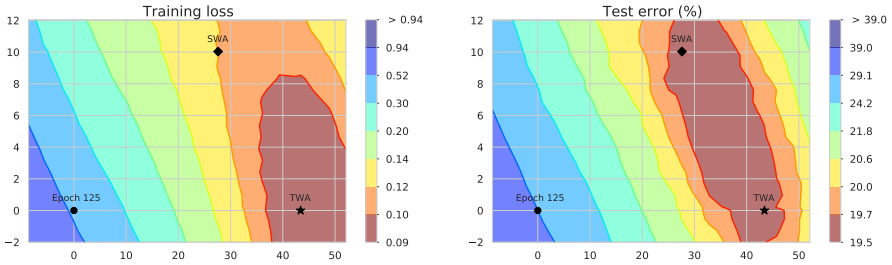
Figure 6: Averaging coefficients of TWA.

5.4 LANDSCAPE VISUALIZATION

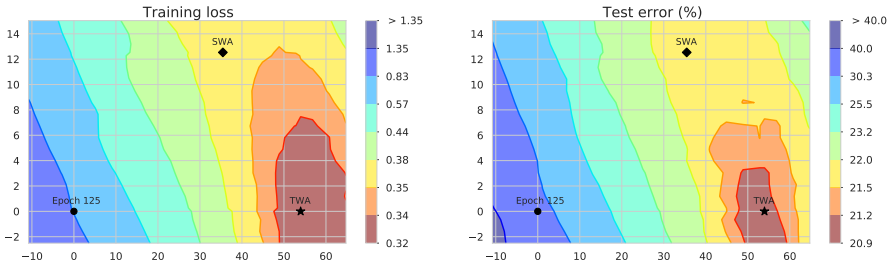
Following (Garipov et al., 2018; Izmailov et al., 2018), we visualize the training loss and test error surfaces of SWA and TWA in Figure 7 on CIFAR-100 with PreAct ResNet-164. We set the SGD solution after 125 training epochs as the origin and plot the TWA and SWA solutions on the plane. For the case with a default learning rate of 0.05, TWA achieves slightly better test accuracy with lower training loss. This shows that in the subspace, minimizing the training loss is meaningful and results in lower test errors. Especially for the case with a larger learning rate of 0.10, the superiority of TWA over SWA is more significant (over 0.7% improvement on test accuracy), since the variance grows larger and the variance reduction effect of TWA becomes more obvious.

6 RELATED WORK

Improving the model's generalization capability is of great importance and has received wide attention. The recent efforts mainly focus on two aspects: (1) proper regularization terms to search for more flat minimum (Keskar et al., 2016; Li et al., 2018), such as weight decay (Krogh & Hertz, 1991), dropout (Srivastava et al., 2014), label smoothing (Szegedy et al., 2016), Shake-Shake (Gastaldi,



(a) SWA_LR = 0.05



(b) SWA_LR = 0.10

Figure 7: Train loss and test error surface of TWA and SWA with different SWA_LR.

2017), MixUp (Zhang et al., 2018), SAM (Foret et al., 2020) and AMP (Zheng et al., 2021); (2) effective data augmentation to diversify the dataset, such as Cutout (DeVries & Taylor, 2017), AutoAugment (Cubuk et al., 2019) and RandAugment (Cubuk et al., 2020). Different from these techniques, we improve the generalization ability by constraining the training in a low-dimensional subspace spanned by historical explorations, which regularizes the model complexity. We note that TWA is orthogonal to these methods, and it is promising to combine them for boosted improvements.

A lot of efforts have been made to speed up the DNNs' training. Apart from the well-known methods on adaptive learning rates, e.g. Adam (Kingma & Ba, 2015) and accelerated schemes, e.g. Nesterov momentum Nesterov (1983), a new method is proposed in Zhang et al. (2019) proposed, where a look-ahead search direction generated by another "fast" optimizer is utilized, achieving faster convergence and better learning stability. Goyal et al. (2017) adapted a large mini-batch to speed up the training and introduced a scaling rule for adjusting the learning rates. In this paper, we realize training efficiency by sufficiently utilizing the historical solutions of DNNs' training and conducting training in a subspace with substantially reduced dimensions.

For utilizing historical explorations, SWA (Izmailov et al., 2018) adopts a simple averaging strategy at the tail of training. Cha et al. (2021) extended it to the domain generalization task with a dense and overfit-aware stochastic weight sampling strategy. We firstly propose to utilize the explorations at the head stage of training to achieve training efficiency. Exponentially decaying running average (Hunter, 1986; Szegedy et al., 2016) is a common technique adopted by practitioners. It requires a manually set averaging horizon and generally performs comparably as SGD (Izmailov et al., 2018). Another closely related work is model soups (Wortsman et al., 2022), which improves the model performance by averaging the weights from different fine-tuning configurations in a greedy order. We differ in that the historical solutions are from one single configuration. We mainly focus on improving training efficiency and optimizing the averaging coefficients in a trainable manner.

7 Conclusion

In this work, we propose TWA, a novel training algorithm that optimizes the averaging coefficients of historical solutions in DNNs' training to achieve efficiency and better performance. It differs from the manually set averaging strategies as SWA or EMA and manifests better adaptation to different stages of training. We further design a parallel framework for large-scale training with efficiency in memory and computation. Extensive experiments demonstrate the superior performance of TWA on benchmark computer vision tasks with various architectures.

ACKNOWLEDGEMENTS

We are very grateful for anonymous reviewers for the valuable feedback on the paper. We thank Minqi Chen at Huawei Technologies for the great support. The research leading to these results has received funding from National Natural Science Foundation of China (61977046), Shanghai Science and Technology Program (22511105600), and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

REFERENCES

Ben Athiwaratkun, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. There are many consistent explanations of unlabeled data: Why you should average. In *International Conference on Learning Representations (ICLR)*, 2018.

Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 113–123, 2019.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12(7), 2011.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations (ICLR)*, 2020.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning (ICML)*, 2020.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. In *Workshop Track Proceedings in International Conference on Learning Representations (ICLR)*, 2017.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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 (CVPR)*, pp. 770–778, 2016.

Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International Conference on Learning Representations (ICLR)*, 2021.

J Stuart Hunter. The exponentially weighted moving average. *Journal of Quality Technology*, 18(4): 203–210, 1986.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

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. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

Anders Krogh and John Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1991.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Tao Li, Lei Tan, Zhehao Huang, Qinghua Tao, Yipeng Liu, and Xiaolin Huang. Low dimensional trajectory hypothesis is true: Dnns can be trained in tiny subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research (JMLR)*, 18:1–35, 2017.

Yurii Nesterov. On an approach to the construction of optimal methods of minimization of smooth convex functions. *Ekonomika i Mateaticheskie Metody*, 24(3):509–517, 1988.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pp. 543–547, 1983.

Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in Artificial Intelligence Workshop on Uncertainty in Deep learning*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

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

Rolf Rabenseifner. Optimization of collective reduction operations. In *International Conference on Computational Science*, pp. 1–9. Springer, 2004.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning (ICML)*, 2022.

Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, and Chris De Sa. Swalp: Stochastic weight averaging in low precision training. In *International Conference on Machine Learning (ICML)*, pp. 7015–7024. PMLR, 2019.

Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W. Mahoney. ADAHESSIAN: an adaptive second order optimizer for machine learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, 2021.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith (eds.), *British Machine Vision Conference (BMVC)*, 2016.

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

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018.

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

Yaowei Zheng, Richong Zhang, and Yongyi Mao. Regularizing neural networks via adversarial model perturbation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8156–8165, 2021.

A Training Details

For SWA experiments, we replicate the SWA baseline by using the publicly released implementation of Izmailov et al. (2018). We use VGG-16 architecture with batch normalization for a unified learning rate setting as PreAct ResNet-164. For ImageNet experiments, we follow official PyTorch implementation. We use a scaled learning rate for TWA training with 20x and 30x factors on CIFAR and ImageNet, respectively (e.g. the original learning rate of 0.1 is scaled up to 2 on CIFAR). CIFAR experiments are performed on one Nvidia Geforce GTX 2080 TI GPU, while ImageNet experiments are on four NVIDIA Tesla A100. The number of historical solutions optimized by TWA is presented in Table A1.

We now disclose the specific hyper-parameters in the following.

Table A1: The number of historical solutions optimized by TWA

DATASETS	SAMPLING EPOCHS	TIMES / EPOCH	#NUM
CIFAR	100	1	100
IMAGENET	60	5	300

A.1 SWA TRAINING

A.1.1 CIFAR

We use the same schedule and hyper-parameters as in Izmailov et al. (2018). For VGG-16, we use weight decay of 5×10^{-4} and train the model for 300 epochs with weight averaging at 161 to 300 epochs. For PreAct ResNet-164, we use weight decay of 3×10^{-4} and train the model for 225 epochs with weight averaging at 126 to 225 epochs.

For TWA training, we use the same weights as SWA and initialize $\boldsymbol{w}_{\rm twa}$ as the last checkpoint (i.e. 300 / 225 epochs). We train the models for 10 epochs with an initial learning rate of 2 and decay it by 10 at the 5th and 8th epochs. The regularization coefficient λ is set to 5×10^{-5}.

A.1.2 IMAGENET

Following Izmailov et al. (2018); Yang et al. (2019), we start from pre-trained models (they are from `torchvision.models`) and collect weights by running SGD optimizer up to 10 epochs (with a constant learning rate 0.005, weight decay 1×10^{-4}). We uniformly sample the solutions 5 times per epoch.

For TWA training, we use the same weights as SWA and initialize $\boldsymbol{w}_{\rm twa}$ as the pre-trained model. For ImageNet, there are many iterations in one epoch, and hence we conduct TWA training for one epoch, in which we linearly decay the learning rate from 0.03 to 0. The regularization coefficient λ is set to 1×10^{-5}.

A.2 REGULAR TRAINING

A.2.1 CIFAR

For regular training, we train the models for 200 epochs with an initial learning rate of 0.1 and decay it by 10 at the 100th and the 150th epoch. We use SGD optimizer with momentum 0.9, weight decay 1×10^{-4}, and batch size 128 by convention.

For TWA training, we initialize $\boldsymbol{w}_{\rm twa}$ as $\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{w}_i$, i.e., the center of sampled solutions. We train the models for 10 epochs with an initial learning rate of 2 and decay it by 10 at the 5th and 8th epochs. The regularization coefficient λ is set to 1×10^{-5}.

A.2.2 IMAGENET

We follow the training protocol described in He et al. (2016). Specifically, we train the models for 90 epochs with an initial learning rate of 0.1 and decay it by a factor of 10 every 30 epochs. We use SGD optimizer with momentum 0.9, weight decay 1×10^{-4}, and batch size 256.

For TWA training, we uniformly sample solutions 5 times per epoch and initialize $\boldsymbol{w}_{\rm twa}$ as $\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{w}_i$. We train the models for 2 epochs with a learning rate of 0.3 and 0.03, respectively. For the extra one epoch training, we use the same training protocol as in subsection A.1.2, i.e., linearly decaying the learning rate from 0.03 to 0. The regularization coefficient λ is set to 1×10^{-5}.

A.2.3 RESULTS ON ADAM OPTIMIZER

Adam (Kingma & Ba, 2015) is another mainstream optimizer with adaptive gradient descent, which enjoys fast convergence and insensitivity to the initial learning rate. Here, we apply TWA to the solutions generated by Adam optimizer and the results are in Table A2. The training settings are the same as in Table 3 with default $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for Adam. TWA is trained for 5 epochs

with a intial learning rate 3. We observe that it could similarly bring generalization improvement and training efficiency.

Table A2: Test accuracy (%) and generalization gap (%) on CIFAR-10/100 with Adam Optimizer

		ADAM (200 EPOCHS)		SWA (100 EPOCHS)	TWA ($100+5$ EPOCHS)	
DATASET	MODEL	ACCURACY	GAP	ACCURACY	ACCURACY	GAP
CIFAR-10	VGG16	93.60 ± 0.12	6.31	92.39 ± 0.07	93.63 ± 0.11	5.74 (↓ 0.57)
	PRERESNET-164	95.09 ± 0.11	4.89	92.65 ± 0.03	95.16 ± 0.09	3.85 (↓ 1.04)
CIFAR-100	VGG16	72.77 ± 0.13	26.67	68.74 ± 0.05	74.07 ± 0.07	17.35 (↓ 9.32)
	PRERESNET-164	76.23 ± 0.14	23.71	73.09 ± 0.13	77.96 ± 0.14	19.62 (↓ 4.09)

B IMPLEMENTATION

Let $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_n]^\top \in \mathbb{R}^n$, $\boldsymbol{P} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_n] \in \mathbb{R}^{D \times n}$, the optimization target for TWA can be formulated as,

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & L(\boldsymbol{\beta}) \triangleq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \left[\mathcal{L}\left(f\left(\boldsymbol{w}_{\text{twa}}; \boldsymbol{x}\right), \boldsymbol{y}\right)\right] + \frac{\lambda}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta}, \\ \text{s.t.} \quad & \boldsymbol{w}_{\text{twa}} = \boldsymbol{P}\boldsymbol{\beta}. \end{aligned} \tag{A.1}$$

We short the loss term $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \left[\mathcal{L}\left(f\left(\boldsymbol{w}; \boldsymbol{x}\right), \boldsymbol{y}\right)\right]$ to $\mathcal{L}(\boldsymbol{w})$ and the gradient w.r.t. $\boldsymbol{\beta}$ can be written as,

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \frac{\partial \boldsymbol{w}_{\text{twa}}}{\partial \boldsymbol{\beta}} \frac{\partial \mathcal{L}(\boldsymbol{w}_{\text{twa}})}{\partial \boldsymbol{w}_{\text{twa}}} + \lambda\boldsymbol{\beta} \tag{A.2}$$

$$= \boldsymbol{P}^\top \frac{\partial \mathcal{L}(\boldsymbol{w}_{\text{twa}})}{\partial \boldsymbol{w}_{\text{twa}}} + \lambda\boldsymbol{\beta}. \tag{A.3}$$

Let η be the learning rate. We could optimize (A.1) with the following gradient descent:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta \left(\boldsymbol{P}^\top \frac{\partial \mathcal{L}(\boldsymbol{w}_{\text{twa}}^{(t)})}{\partial \boldsymbol{w}_{\text{twa}}} + \lambda\boldsymbol{\beta} \right). \tag{A.4}$$

Since $\boldsymbol{w}_{\text{twa}} = \boldsymbol{P}\boldsymbol{\beta}$, we have the corresponding update in the parameter space:

$$\boldsymbol{w}_{\text{twa}}^{(t+1)} = \boldsymbol{w}_{\text{twa}}^{(t)} - \eta \left(\boldsymbol{P}\boldsymbol{P}^\top \frac{\partial \mathcal{L}(\boldsymbol{w}_{\text{twa}}^{(t)})}{\partial \boldsymbol{w}_{\text{twa}}} + \lambda\boldsymbol{P}\boldsymbol{\beta} \right) \tag{A.5}$$

$$= (1 - \eta\lambda)\, \boldsymbol{w}_{\text{twa}}^{(t)} - \eta\boldsymbol{P}\boldsymbol{P}^\top \frac{\partial \mathcal{L}(\boldsymbol{w}_{\text{twa}}^{(t)})}{\partial \boldsymbol{w}_{\text{twa}}}. \tag{A.6}$$

As $\boldsymbol{\beta}$ does not explicitly appear in (A.6), we could treat the coefficient $\boldsymbol{\beta}$ as an implicit variable. In practice, we optimize (A.1) by directly updating the model weights $\boldsymbol{w}_{\text{twa}}$ with weight decay λ, which is an optimization in the reduced subspace with projection matrix $\boldsymbol{P}\boldsymbol{P}^\top$.

C SWA WITH DIFFERENT STARTING EPOCHS

We test the performance of SWA with different starting epochs to average on ImageNet with the ResNet-50 model. We observe that the performance of SWA gradually becomes better with the relatively latter stage of solutions averaged, showing that SWA could not adapt well to the head stage of training where the solutions are fast-evolving. Hence, a good solution for SWA may require manually selecting which period to average. We also notice that TWA (with 0-60 epoch solutions) consistently outperforms the SWA wherever the averaging begins, confirming that TWA could automatically find a good set of averaging coefficients and provide better performance.

SWA Epoch 0-60	SWA Epoch 10-60	SWA Epoch 20-60	SWA Epoch 30-60	SWA Epoch 40-60	SWA Epoch 50-60	TWA Epoch 0-60
67.66	70.50	72.12	74.14	75.08	75.34	75.90

Table C3: SWA with different starting epochs.

D Discussion on the Sum-one Constraint

Let $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_n]^\top \in \mathbb{R}^n$, $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_n] \in \mathbb{R}^{D\times n}$, we have $\boldsymbol{w}_{\text{twa}} = \boldsymbol{W}\boldsymbol{\alpha}$. We multiply $\boldsymbol{W}^\top$ on the both sides, i.e., $\boldsymbol{W}^\top \boldsymbol{w}_{\text{twa}} = \boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{\alpha}$, and could obtain $\boldsymbol{\alpha} = \left(\boldsymbol{W}^\top \boldsymbol{W}\right)^{-1} \boldsymbol{W}^\top \boldsymbol{w}_{\text{twa}}$. Further, we could establish the relation between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$: $\boldsymbol{\alpha} = \left(\boldsymbol{W}^\top \boldsymbol{W}\right)^{-1} \boldsymbol{W}^\top \boldsymbol{P}\beta$, since they are the coordinates of $\boldsymbol{w}_{\text{twa}}$ under two different set of bases.

In the solution set $A = \{\alpha_1 \boldsymbol{w}_1 + \alpha_2 \boldsymbol{w}_2 + \cdots + \alpha_n \boldsymbol{w}_n \mid \alpha_i \in \mathbb{R}\}$, we do not explicitly require that the sum of averaging coefficients $\sum_{i=1}^n \alpha_i$ to be 1 since the network's performance would be sensitive to a direct scaling over all parameters (i.e. $k\boldsymbol{w}$), that is, a good solution in A will inherently have a coefficient sum very close to 1. We verify the sum of the averaging coefficients of the attained solution $\boldsymbol{w}_{\text{twa}}$ on CIFAR-100 with PreResNet-164 model and observe that $\sum_{i=1}^n \alpha_i = 1.02_{\pm<0.01}$.

E Numerical Discussion for DDP Training

We numerically measure the averaged epoch training time and memory burden for SGD and TWA in the DDP training setting. Specifically, we experiment with the ResNet-50 model on ImageNet and use 1, 2, and 4 GPUs with a batch size of 256 per GPU and a total of 300 historical solutions. The experiments are conducted on NVIDIA Tesla A100 40G GPUs. From the results reported in Table E4, we observe that TWA brings minor additional costs, e.g. +2.8% on time cost and +2.9% on memory burden with 4 GPUs, compared with regular SGD training. The additional memory burden becomes even minor with more GPUs. This shows that TWA could provide efficient and scalable averaging for large-scale problems.

Table E4: Time and memory comparisons of SGD and TWA with DDP training.

	Time		Memory	
#GPUs	SGD	TWA	SGD	TWA
1	1638s	1692s (+3.3%)	28286.5 MB	31432.5 MB (+11.1%)
2	824s	862s (+4.6%)	28382.5 MB	30092.5 MB (+6.0%)
4	420s	432s (+2.8%)	28874.5 MB	29718.5 MB (+2.9%)

F Ablation Study

We conduct an ablation study in Table F5 to analyze the impact of the regularization coefficient λ. We observe that such regularization brings improvements but is not significant. This is because the main regularization effects come from the significant decrease of training variables, i.e., regular training has D variables but TWA contains only n. Since such regularization is easy to implement and virtually brings little training cost, we include it in our method.

Table F5: Ablation studies on the regularization coefficient λ.

Datasets	Model	TWA	TWA ($\lambda = 0$)
CIFAR-10	VGG16	93.79 ± 0.18	93.78 ± 0.07
	PreActResNet-164	95.11 ± 0.04	95.03 ± 0.09
CIFAR-100	VGG16	74.24 ± 0.24	74.02 ± 0.16
	PreActResNet-164	78.11 ± 0.19	78.01 ± 0.22

G Additional Results on NLP Tasks

For NLP datasets, we try a finetune task with pre-trained models and compare the performance of SWA and TWA. Specifically, we experiment with The Corpus of Linguistic Acceptability (CoLA), a text classification task in the General Language Understanding Evaluation (GLUE, Wang et al. (2018)) benchmark. In experiment, we use a pre-trained BERT (Devlin et al., 2018) model (`bert-base-uncased`) from Hugging Face community[2]. We fine-tune BERT on CoLA for 3 epochs with AdamW optimizer Loshchilov & Hutter (2017), learning rate 2e-5, and weight decay 0.0. The model weights at the end of these epochs are collected for SWA and TWA. In TWA, we train the fine-tuned model for 1 epoch with a learning rate of 0.5 and regularization coefficients $\lambda = 0.001$. From the results below, we observe that TWA could achieve better performance than the competing methods. This further demonstrates the broad application of TWA.

Fine-tune	SWA	TWA
56.81	59.73	60.36

Table G6: Fine-tune results on CoLA.

[2]Available at `https://huggingface.co/`