ADAPTIVE GRADIENT METHOD WITH RESILIENCE AND MOMENTUM

Anonymous authors
Paper under double-blind review

ABSTRACT

Several variants of stochastic gradient descent (SGD) have been proposed to improve the learning effectiveness and efficiency when training deep neural networks, among which some recent influential attempts would like to adaptively control the parameter-wise learning rate (*e.g.*, Adam and RMSProp). Although they show a large improvement in convergence speed, most adaptive learning rate methods suffer from compromised generalization compared with SGD. In this paper, we proposed an Adaptive Gradient Method with Resilience and Momentum (AdaRem), motivated by the observation that the oscillations of network parameters slow the training, and give a theoretical proof of convergence. For each parameter, AdaRem adjusts the parameter-wise learning rate according to whether the direction of one parameter changes in the past is aligned with the direction of the current gradient, and thus encourages long-term consistent parameter updating with much fewer oscillations. Comprehensive experiments have been conducted to verify the effectiveness of AdaRem when training various models on a large-scale image recognition dataset, *i.e.*, ImageNet, which also demonstrate that our method outperforms previous adaptive learning rate-based algorithms in terms of the training speed and the test error, respectively.

1 INTRODUCTION

Stochastic gradient descent based optimization methods, *e.g.*, SGD (Robbins & Monro, 1951), have become the most popular algorithms to train deep neural networks, especially used in high-level vision tasks such as image recognition (Hu et al., 2018), object detection (Song et al., 2020), instance segmentation (Dai et al., 2016) and etc.

However, as stated in Luo et al. (2019), one limitation of SGD is that it uniformly scales each element in the gradient of one network parameter. Recent efforts have discovered a variety of adaptive methods that rescale the gradient based on the element-wise statistics. Referred as adaptive learning rate family, these methods include Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012), Adam (Kingma & Ba, 2014), NAdam (Dozat, 2016) and RMSProp (Tieleman & Hinton, 2012). In particular, Adagrad, which firstly proposes to adaptively modify the learning rates, was later found to have poor performance because of the rapid decay of the learning rate. Many variants of Adagrad, such as RMSProp, Adam, Adadelta, Nadam were proposed to solve this problem by adopting an exponential moving average. Among these adaptive optimization methods, Adam has been widely used in the community due to its fast convergence. Despite its popularity, Wilson et al. (2017) recently found that the generalization performance of Adam and its variants is worse than their non-adaptive counterpart: SGD with momentum (SGDM) and weight decay (Krogh & Hertz, 1992), even when better memorization on the train set is observed.

From a different viewpoint towards the adaptive learning rate methods, we investigate the trajectories of model parameters in the training process, in which a lot of oscillations are observed for each parameter. It seriously hinders the network training. Oscillations might come from the evaluation at random subsamples (mini-batches) of data points, or arise from the inherent function that changes dramatically locally. To address this issue, we propose Adaptive Gradient Methods with REsilience and Momentum (AdaRem), a new adaptive optimization method that reduces useless oscillations by introducing damping. For each parameter, AdaRem adjusts the learning rate according to whether the direction of the current gradient is the same as the direction of parameter change in the past.

Furthermore, we find that weight decay affects the magnitude of the gradient, thus affecting the estimation of the update direction of the parameter. Inspired by Li & Arora (2019), we also propose AdaRem-S, a variant of AdaRem that constrains the optimization of the neural network to a sphere space so as to eliminate the influence of weight decay.

Adaptive learning rate methods are usually hard to tune in practical scenarios. However, we find that AdaRem and AdaRem-S usually perform well by simply borrowing SGDM's hyper-parameters, which tremendously reduces the burden of hyper-parameter tuning. Furthermore, adaptive methods suffer "the small learning rate dilemma" (Chen & Gu, 2018), which means adaptive gradient methods have to choose a very small base learning rate to allieviate over-large learning rates on some coordinates. However, after several rounds of decaying, the learning rates are too small to make any significant progress in the training process (Chen & Gu, 2018). With a learning rate as large as that used in SGD, our methods avoid this dilemma.

Finally, we evaluate AdaRem and AdaRem-S by training classifiers on the ImageNet (Deng et al., 2009) dataset. Most of the previous works about optimizers only evaluate their methods on small datasets such as CIFAR-10 (Krizhevsky et al., 2009). We argue that these datasets are not enough representative to fairly and comprehensively measure the performance of optimizers for nowadays vision tasks. Therefore, we conduct all experiments directly on ImageNet with various models. Experimental results show that AdaRem (including its spherical version) has higher training speed and at the meantime leads to improved performance on the test datasets compared to existing popular adaptive methods. We show that AdaRem and AdaRem-S close the performance gap between adaptive gradient methods and SGDM empirically. Furthermore, AdaRem-S can bring considerable improvement over SGDM in terms of the final performance, especially on small networks.

In summary, our main contributions are the following:

- We propose AdaRem, a novel and simple adaptive optimization method that accelerates the training process by reducing useless oscillations, which enjoys a fast convergence speed and performs as well as SGDM on the unseen data.
- We improve our method by constraining the optimization on a sphere space. The resulted variant: AdaRem-S shows significant improvement on top-1 accuracy for MobileNetV2 (Sandler et al., 2018) and ShuffleNetV2 (Ma et al., 2018) on ImageNet.

2 Related work

Optimization methods directly related to AdaRem are Rprop and Momentum. Their similarities and differences are discussed below. Other adaptive optimization methods mainly include Adam and its variants. Kingma & Ba (2014) proposes Adam which is particularly popular on vision tasks. Dozat (2016) increases the performance by combining Adam and the Nesterov accelerated gradient. AdamW (Loshchilov & Hutter, 2017) attempts to recover the original formulation of weight decay regularization by decoupling the weight decay from the gradient updates and thus substantially improves Adam's generalization performance. Recently, Reddi et al. (2019) observes that Adam does not converge in some settings due to the "short memory" problem of the exponential moving average. They fix this problem by endowing Adam with "long-term memory" of past gradients and propose Amsgrad optimizer. Based on Amsgrad, a series of modified Adam optimization methods emerge, including PAdam (Chen & Gu, 2018), AdaShift (Zhou et al., 2018), AdaBound (Luo et al., 2019), NosAdam (Huang et al., 2018).

RProp A closely related adaptive optimization method is Rprop (Riedmiller & Braun, 1992). There are a few important differences between Rprop and AdaRem: Rprop increases or decreases each learning rate η_i according to whether the gradient concerning w_i has changed sign between two iterations or not, whereas AdaRem adjusts η_i according to whether a running average of the history gradient has an opposite sign with the gradient of current iteration or not. What's more, AdaRem changes the learning rate softly while Rprop changes the learning rate multiplicatively (e.g. times 1.2). Furthermore, Rprop does not work with mini-batches (Tieleman & Hinton, 2012).

Momentum Momentum (Sutskever et al., 2013) helps accelerate SGD in the relevant direction and dampens oscillations. However, it directly uses the momentum term as the damping, without

considering whether the damping term is in the same direction as the current gradient and adaptively adjust the learning rate.

3 METHOD

In this section, we introduce our AdaRem and AdaRem-S methods. Firstly, we offer the notations and preliminaries. Secondly, we explain the motivations of our approach. Then we introduce our momentum guided adaptive gradient method in detail. Finally, the spherical version of our method called AdaRem-S is presented.

3.1 NOTATIONS AND PRELIMINARIES

Given two vectors $v, v \in \mathbb{R}^d$, we use $\langle v, v \rangle$ for inner product, $v \odot v$ for element-wise product and v/v to denote element-wise division. For the set of all positive definite $d \times d$ matrices. We use $\mathcal{S}_+^d$ to denote it. For a vector $a \in \mathbb{R}^d$ and a positive definite matrix $A \in \mathbb{R}^{d \times d}$, we use a/A to denote $A^{-1}a$ and $\sqrt{A}$ to denote $A^{1/2}$. The projection operation $\Pi_{\mathcal{F},A}(y)$ for $A \in \mathcal{S}_+^d$ is defined as $\arg\min_{x \in \mathcal{F}} \|A^{1/2}(x-y)\|$ for $y \in \mathbb{R}^d$. Furthermore, we say $\mathcal{F}$ has bounded diameter D_∞ if $\|x-y\|_\infty \leq D_\infty$ for all $x, y \in \mathcal{F}$.

Scalars and vectors are denoted in lowercase and bold lowercase, respectively. Our goal is to solve the optimization problem: $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^n} f(\boldsymbol{\theta})$. We denote the gradient with $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ at timestep t. We use $g_{t,i}$ to represent the i^{th} component of vector $\boldsymbol{g_t}$ and $\|\boldsymbol{\theta}_t\|$ to represent the length of vector $\boldsymbol{\theta}_t$. Consider the general formula of adaptive optimization methods: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \boldsymbol{a}_t \odot \boldsymbol{g}_t$, where η_t is the learning rate at timestep t and $\odot$ is elementwise multiplication. We use $\boldsymbol{a}_t$ to adjust η_t adaptively.

Following (Reddi et al., 2019), we use online convex programming framework to analyze our optimization methods. It can be formulated as a repeated game between a player and an adversary. At iteration t, the player chooses $\boldsymbol{\theta}_t$ from convex set $\mathcal{F}$ as learned parameters of the model. Then the adversary chooses a convex loss function f_t which can be seemed as the loss of the model with the chosen parameters and the next minibatch's data. The method's regret at the end of T iterations of this process is given by $R_T = \sum_{t=1}^{T} f_t(\boldsymbol{\theta}_t) - \min_{\boldsymbol{\theta} \in \mathcal{F}} \sum_{t=1}^{T} f_t(\boldsymbol{\theta})$, where the former term is the total loss and the latter term is the smallest total loss of any fixed parameters. Throughout this paper, we assume that the feasible set $\mathcal{F}$ has bounded diameter and $\|\nabla f_t(\boldsymbol{\theta})\|_\infty$ is bounded for all $t \in [T]$ and $\theta \in \mathcal{F}$. Our aim is to devise an algorithm that ensures $R_T = o(T)$, which implies that on average, the model's performance converges to the optimal one.

3.2 MOTIVATION

3.2.1 KEY OBSERVATION

As a powerful technique to reduce oscillations during training, momentum has become the standard configuration of SGD. However, as shown in Figure 1, we observe that the oscillation is still severe for each parameter. We argue that directly using the momentum instead of the gradient to reduce the oscillation in SGDM does not give full play to the effect of momentum.

3.2.2 THE QUANTITATIVE INDEX OF OSCILLATIONS

To better understand the oscillation of various optimization methods in the training process, we propose a quantitative metric to evaluate it. As seen in Figure 1, we record the path length and displacement from the starting point to the ending point for each parameter and define "the path length per displacement": $q = l/d$, where path length l is the total distance a parameter travels from a starting point and displacement d is the shortest distance between the ending point and starting point of a parameter. Assuming the number of parameters is n, We define the average value of q among all parameters:

$$Q = \frac{1}{n} \sum_{i=1}^{n} q_i$$

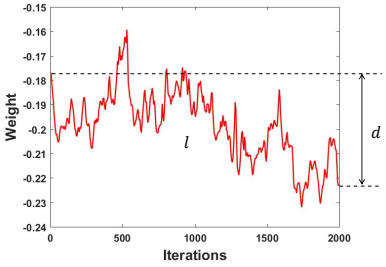


Figure 1: The parameter changes as training is going on. From Iteration 0 to Iteration 2k, the effective displacement of this parameter is d(about 0.04), but the path length l is observably much larger than d.

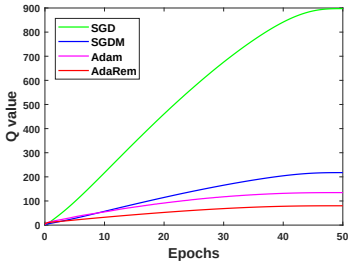


Figure 2: The Q value of AdaRem, Adam and SGDM for ResNet-18 on ImageNet. Higher Q value means more oscillations and useless updates. Compared with SGDM and Adam, AdaRem reduces oscillation more effectively.

to measure the degree of oscillation in the training process. As seen in Figure 2, the Q value of SGDM is significantly smaller than SGD verifying the effect of momentum on oscillation reduction. Furthermore, Adam has a smaller Q than SGDM indicating an adaptively changing learning rate may be useful to suppress oscillations.

3.2.3 ADAPTIVE GRADIENT WITH RESILIENCE AND MOMENTUM

Consider an elastic ball rolling down a very rough road from a high place, with a lot of oscillations going back and forth as it falls. If the ball can remember the update direction of each coordinate and change the learning rate based on whether the current update direction is consistent with "memory", it will descend more smoothly and fastly. We propose a similar adaptive way to employ momentum with resilience, which adjusts the learning rate according to whether the direction of momentum is the same as the direction of the current gradient for each parameter. As shown in Figure 2, our method AdaRem has the smallest Q value which means AdaRem can accelerate SGD, dampen oscillations and reduce useless updates during training more efficiently than SGDM and Adam.

3.3 ADAREM

In this section, we describe the AdaRem algorithm and discuss its properties. The algorithm maintains an exponential moving average(EMA) of the gradient($\boldsymbol{m}_t$) where the hyper-parameter $\beta \in [0, 1)$ control the exponential decay rate. The moving average is an estimation of the mean of the gradient, we call it momentum, whose i^{th} component($m_{t,i}$) can represent the update trend(increase or decrease) of the corresponding parameter of neural network in the past. An important property of AdaRem's update rule is its special adjustment rule of learning rate:

$$b_{t,i} = \frac{g_{t,i} \times m_{t,i}}{|g_{t,i}| \max |\boldsymbol{m}_t| + \epsilon}, \tag{1}$$

$$\boldsymbol{a}_t = 1 + \lambda^t \boldsymbol{b}_t, \tag{2}$$

where ϵ is a term added to the denominator to improve numerical stability. The greater the component of the momentum $\boldsymbol{m}_t$, the greater the damping of this term. The adjustment coefficient $\boldsymbol{a_t}$ is bounded: $b_{t,i} \in [-1, 1]$, and then $a_{t,i} \in [1 - \lambda, 1 + \lambda]$, which can obviate very large learning rates on some coordinates and escape from "the small learning rate dilemma" (Chen & Gu, 2018). For each parameter, AdaRem adjusts the learning rate according to whether the direction of the current gradient is the same as the direction of parameter change in the past. There are two cases:

(1) $g_{t,i} \times m_{t,i} \geq 0, b_{t,i} \in [0, 1], a_{t,i} \in [1, 1 + \lambda]$. The direction of the current gradient is the same as the direction of parameter change in the past. Therefore, the current update should be encouraged. We use $|g_{t,i}| \max |\boldsymbol{m}_t|$ to normalize $b_{t,i} \in [-1, 1]$;

(2) $g_{t,i} \times m_{t,i} < 0, b_{t,i} \in [-1, 0), a_{t,i} \in [1 - \lambda, 1)$. The direction of the current gradient is opposite to the direction of parameter change in the past. Therefore, the current update should be suppressed.

Algorithm 1 AdaRem Algorithm

Require: learning rate $\hat{\eta}_t$ at each iteration, momentum parameter β, iteration number T, weight decay factor γ, $m_0 = 0$, λ.
for $t = 0$ to T **do**
$\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_t)$
$\eta_{t,i} = \left(1 + \lambda^t \frac{g_{t,i} \times m_{t,i}}{|g_{t,i}| \max|\boldsymbol{m}_t| + \epsilon}\right) \hat{\eta}_t$
$\boldsymbol{\theta}_{t+1} = \Pi_{\mathcal{F}, \mathrm{diag}(\eta_t^{-1})}(\boldsymbol{\theta}_t - \boldsymbol{\eta}_t \odot \boldsymbol{g}_t - \hat{\eta}_t \gamma \boldsymbol{\theta}_t)$
$\boldsymbol{m}_{t+1} = \beta \boldsymbol{m}_t + (1 - \beta) \boldsymbol{g}_t$
end for

Table 1: Final test accuracy of various networks on the ImageNet dataset. The bold number indicates the best result.

Model	Top-1 Accuracy(%)	
	SGDM	AdaRem-S
ResNet50	**76.18**	76.10
ResNet18	70.67	**70.89**
MobileNetV2-1.0	70.71	**71.71**
MobileNetV2-0.5	62.99	**64.01**
ShuffleNetV2-1.0	67.37	**68.33**
ShuffleNetV2-0.5	57.75	**60.15**

An important property of AdaRem is that the gradient does not change the sign in spite of a large momentum with opposite sign. This means AdaRem is friendly to the situation where the gradient changes dramatically. Following Reddi et al. (2019) and Luo et al. (2019), we analyze the convergence of AdaRem using the online convex programming framework. We prove the following key result for AdaRem.

Theorem 1. *Let $\{\theta_t\}$ be sequences obtained from Algorithm 1, $\eta_{t,i} = \frac{\eta}{\sqrt{t}} \left(1 + \lambda^t \frac{g_{t,i} \times m_{t,i}}{|g_{t,i}| \max|\boldsymbol{m}_t| + \epsilon}\right)$ and $\gamma = 0$. Assume that $\|\boldsymbol{x} - \boldsymbol{y}\|_\infty \leq D_\infty$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{F}$ and $\|\nabla f_t(\boldsymbol{x})\| \leq G_2$ for all $t \in [T]$ and $\boldsymbol{x} \in \mathcal{F}$. For θ_t generated using the AdaRem algorithm, we have the following bound on the regret*

$$R_T \leq \frac{D_\infty^2 d}{\eta(1-\lambda)^3} \left[(5 - 4\lambda)\sqrt{T} + 2\lambda - 1 \right] + \frac{D_\infty^2 d}{2\eta(1-\lambda)} + G_2^2 d\eta(2\sqrt{T} - 1).$$

It is easy to see that the regret of AdaRem is upper bounded by $O(\sqrt{T})$. Please see Appendix for details of the proof of convergence.

We end this section with a comparison to the previous work. Using momentum to reduce oscillations is also found in Adam-like algorithms and SGD with Momentum. These methods directly use momentum to replace the gradient while our method considering momentum as a representation of the update trend of parameters. AdaRem inspects whether the momentum is in the same direction with the current gradient for each parameter, so as to carefully adjust the learning rate.

3.4 ADAREM-S

For networks with Batch Normalization layer or BN (Ioffe & Szegedy, 2015), all the parameters before BN layer satisfy the property of **Scale Invariance** (Li & Arora, 2019): If for any $c \in \mathbb{R}^+$, $L(\boldsymbol{\theta}) = L(c\boldsymbol{\theta})$, then

(1) $\langle \nabla_{\boldsymbol{\theta}} L, \boldsymbol{\theta} \rangle = 0$
(2) $\nabla_{\boldsymbol{\theta}} L|_{\boldsymbol{\theta} = \boldsymbol{\theta}_0} = c \nabla_{\boldsymbol{\theta}} L|_{\boldsymbol{\theta} = c\boldsymbol{\theta}_0}$ for any $c > 0$

We use the moving average of the gradient to represent the update trend. However, the length of the parameter vector changes while training due to weight decay. In terms of the network containing the BN layer, changing the length of the parameters vector will affect the length of the gradient vector due to the scale invariance, hence makes $\boldsymbol{m}_t$ not a good representative of the update trend in the past. Based on scale invariance, Li & Arora (2019) proves that weight decay can be seen as an exponentially increasing learning rate schedule. This equivalence holds for BN, which is ubiquitous and provides benefits in optimization and generalization across all standard architectures. This means that weight decay is redundant in training, and thus we can use a learning rate schedule to achieve the same effect. Furthermore, we can fix the length of the parameter vector during the training and optimize the neural network on the sphere, eliminating the influence of the change of the parameter vector's length on the estimation of the update trend. See algorithm 2 for the pseudo-code of our proposed spherical AdaRem algorithm(AdaRem-S).

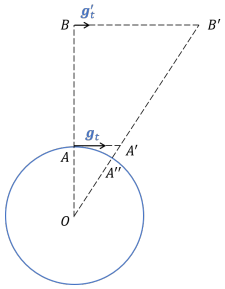


Figure 3: Schematic diagram of spherically constrained optimization. The length of the solid arrows shows the magnitude of the gradient vectors. According to **Scale Invariance**, network at A and network at B are equivalent because they have the same output for any input. Similarly, network at $A^{'}$ and network at $B^{'}$ are equivalent. In addition, $\boldsymbol{g}_t$ is larger than $\boldsymbol{g}_t^{'}$ and they are both perpendicular to OB. After the update, $A^{'}$ is projected onto the point $A^{''}$ on the sphere.

Algorithm 2 AdaRem-S Algorithm

Require: learning rate η_t at each iteration, momentum parameter β, iteration number T, initial parameters $\boldsymbol{\theta}_0$, sphere radius R, weight decay factor γ, $m_0 = 0$.
$\widehat{\boldsymbol{\theta}}_0 = \frac{\boldsymbol{\theta}_0}{\|\boldsymbol{\theta}_0\|} R$
for $t = 0$ to T **do**
$\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} f_t\left(\hat{\boldsymbol{\theta}}_t\right)$
$\eta_{t,i} = \left(1 + \lambda^t \frac{g_{t,i} \times m_{t,i}}{|g_{t,i}| \max|\boldsymbol{m}_t| + \epsilon}\right) \hat{\eta}_t$
$\boldsymbol{\theta}_{t+1} = \Pi_{\mathcal{F}, \mathrm{diag}(\eta_t^{-1})}(\hat{\boldsymbol{\theta}}_t - \boldsymbol{\eta}_t \odot \boldsymbol{g}_t - \hat{\eta}_t \gamma \hat{\boldsymbol{\theta}}_t)$
$\boldsymbol{m}_{t+1} = \beta \boldsymbol{m}_t + (1 - \beta) \boldsymbol{g}_t$
$\hat{\boldsymbol{\theta}}_{t+1} = \frac{\boldsymbol{\theta}_{t+1}}{\|\boldsymbol{\theta}_{t+1}\|} R$
end for

For optimization algorithms, the learning rate is a very critical hyper-parameter. Then what kinds of learning rate scheduler should be used on the sphere? First of all, we introduce the equivalent learning rate for spherical stochastic gradient descent: (1) We use the exponential learning rate from Li & Arora (2019) to replace the weight decay; (2) The equivalent learning rate on the sphere is obtained by using scale invariance.

3.4.1 THE APPROPRIATE LEARNING RATE ON THE SPHERE

After constraining the optimization of the neural network to the sphere, a simple method called SLR(sphere learning rate) is proposed to find the equivalent learning rate on the sphere for SGD. The parameter vector of the network corresponds to a point in the high dimensional space. Therefore, the term of point in high dimensional space appearing in the following text is a neural network. As shown in Figure 3, suppose that there is a network at point B and the equivalent network of B on the sphere is at point A. According to **Scale Invariance**, If $OB = \alpha_t OA$, then $\boldsymbol{g}_t = \alpha_t \boldsymbol{g}_t'$.

To make the network on the sphere and the network in Euclidean space equivalent everywhere in the training process, it is required that the corresponding points of the two networks are on the same ray from the origin after updating, that is, the network at point B should move to B', and the network at point A should move to A'. According to the similar triangle theorem,

$$BB' = \alpha_t AA'.$$

Assuming the learning rates of networks at point A and point B are η_t and η_t', respectively, then

$$\eta_t' \boldsymbol{g}_t' = \alpha_t \eta_t \boldsymbol{g}_t.$$

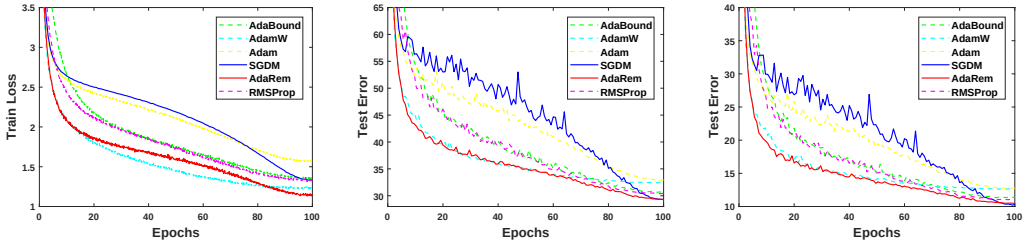
We obtain the equivalent learning rate on the sphere: $\eta_t = \frac{1}{\alpha_t^2} \eta_t'$. Using the similar triangle theorem, α_t can be got sequentially as the training goes on. Empirically, we found that it is also working to use the SLR algorithm to obtain an appropriate learning rate on the sphere for AdaRem. Therefore, for our AdaRem-S method, we simply use SLR to obtain the sphere learning rate while training.

4 EXPERIMENTS

4.1 CONVOLUTIONAL NEURAL NETWORK

4.1.1 DATASET AND HYPER-PARAMETERS TUNING

We test our optimizers on ImageNet dataset which contains roughly 1.28 million training images and 50000 validation images with 1000 categories. To our knowledge, it is particularly challenging for



(a) Train Loss for ResNet18 (b) Top-1 Error for ResNet18 (c) Top-5 Error for ResNet18

Figure 4: Train loss and test error of ResNet18 on ImageNet. Adam, AdamW, AdaBound and RMSProp have fast progress in the early stages, but their performance quickly enters a period of slow growth. AdaRem achieves the fastest training speed among all methods and performs as well as SGDM on the test set.

Table 2: Final test accuracy of all algorithms on the ImageNet dataset. The bold number indicates the best result.

Models	Test Accuracy(%)					
	SGDM	Adam	AdamW	AdaBound	RMSprop	AdaRem
ResNet-18 Top-1	**70.67**	67.07	67.55	69.3	69.63	**70.67**
ResNet-18 Top-5	**89.74**	87.24	87.37	88.7	88.94	89.51

adaptive optimizers to outperform SGD on this large dataset. We show that for large scale dataset, our proposed algorithms still enjoy a fast convergence rate, while the unseen data performance of AdaRem-S outperforms SGDM in small networks and much better than existing adaptive optimizers such as Adam, AdaBound and RMSProp.

A learning rate scheduler is crucial to the training. He et al. (2019) reports that the cosine learning rate outperforms the step decay learning rate for vision tasks. Therefore, we run all experiments with cosine learning rates without a warmup stage and train for 100 epochs with a minibatch size of 1024 on 16 GPUS. We set the base learning rate of 0.4 for SGD, AdaRem and AdaRem-S, 0.004 for Adam, AdamW and AdaBound, 0.0001 for RMSProp. Empirically we set λ of 0.999 for AdaRem and AdaRem-S and just multiplying λ once per epoch is enough. We perform grid searches to choose the best hyper-parameters for all algorithms, additional details can be seen in the Appendix.

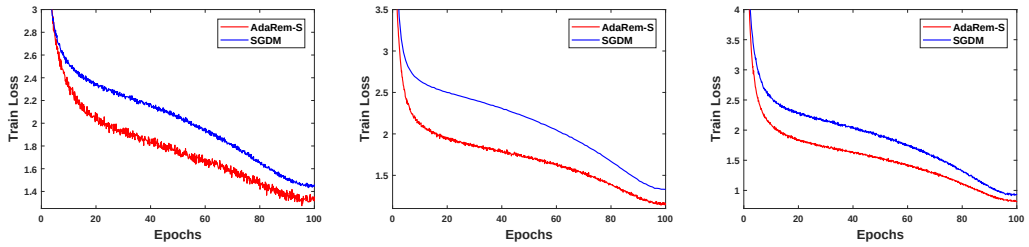
4.1.2 Adaptive optimizers' performance

We train a ResNet-18 (He et al., 2016) model on ImageNet with our AdaRem and several commonly used optimizers, including: (1) SGD with momentum(SGDM) (Sutskever et al., 2013), (2) Adam (Kingma & Ba, 2014), (3) AdamW (Loshchilov & Hutter, 2017), (4) AdaBound (Luo et al., 2019) and (5) RMSprop (Riedmiller & Braun, 1992). As seen in Figure 4 and Table 2, AdamW, AdaBound, RMSProp appear to perform better than SGDM early in training. But at the end of the training, they all have poorer performance on the test set than SGDM. As for our method, AdaRem converges almost fastest and performs as well as SGDM on the test set at the end of training while achieves a significantly minimum train loss.

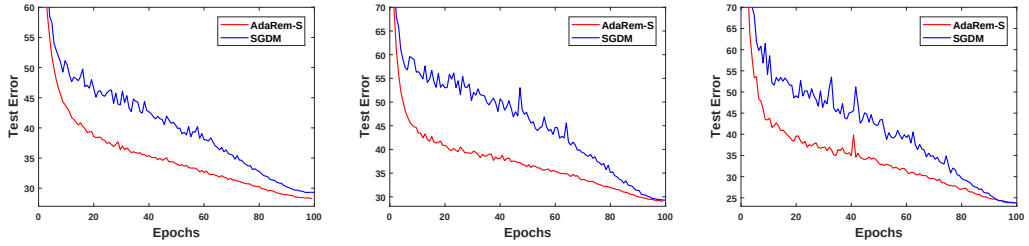
4.1.3 AdaRem-S vs. SGDM across various architectures

AdaRem-S eliminates the influence of the change of the parameter vector's length on the estimation of momentum. Here we compare AdaRem-S against SGDM on various networks, including ResNet-18, ResNet-50 (He et al., 2016), MobileNetV2 (Sandler et al., 2018) and ShuffleNetV2 (Ma et al., 2018).

ResNet Results for this experiment is shown in Figure 5. As expected, AdaRem-S makes rapid progress lowing train loss at the early stage of the training process and finally performs as well as SGDM for ResNet-18 and ResNet-50. From Table 2 and Table 1, we can see that AdaRem-S performance better than AdaRem in terms of the test accuracy for ResNet-18.



(a) Train Loss for MobileNetV2 (b) Train Loss for ResNet18 (c) Train Loss for ResNet50



(d) Top-1 Error for MobileNetV2 (e) Top-1 Error for ResNet18 (f) Top-1 Error for ResNet50

Figure 5: Train loss and test error of three networks on ImageNet. Compared with SGDM, AdaRem-S significantly reduces the training loss across all three networks. It generalizes as well as SGDM on ResNet18 and ResNet50 and brings considerable improvement over SGDM on MobileNetV2.

Table 3: Test perplexity of LSTM models on the Penn Treebank dataset. A lower value is better.

Optimizers	SGDM	Adam	AdaRem
Perplexity	71.56	70.16	**69.13**

MobileNetV2 and ShuffleNetV2 As we can see in Figure 5 and Table 1, while enjoying a fast convergence rate, AdaRem-S improves the top-1 accuracy by 1.0% for MobileNetV2 and ShuffleNetV2-1.0. What's more, ShuffleNetV2-0.5 gets a surprising large gain (**2.4%**) when using AdaRem-S. Note that AdaRem-S achieves more improvement for smaller models(e.g. MobileNetV2, ShuffleNetV2). This is because the smaller models are more under fitted, and AdaRem-S significantly improves their fitting ability.

4.2 RECURRENT NEURAL NETWORK

Finally, to verify the generalization of our optimizer, we trained a Long Short-Term Memory (LSTM) network (Hochreiter & Schmidhuber, 1997) for language modeling task on the Penn Treebank dataset. As BN is not available on a recurrent neural network, we just conduct experiments with AdaRem. We followed the model setup of Merity et al. (2017) and made use of their publicly available code in our experiments. We trained all models for 100 epochs and divided the learning rate by 10 in 50th epoch. For hyper-parameters such as learning rate η and parameter β_1 and β_2, we performed grid searches to choose the best one. As shown in Table 3, our method AdaRem has the smallest perplexity value.

5 CONCLUSION

In this paper, we propose AdaRem and its spherical version called AdaRem-S. By changing the learning rate according to whether the direction of the current gradient is the same as the direction of parameter change in the past, these algorithms can accelerate SGD and dampen oscillations more efficiently than SGDM. The experiments show that AdaRem and AdaRem-S can maintain a fast convergence rate while performing as well as SGDM on the unseen data. In particular, AdaRem-S achieves better test performance than SGDM on MobileNetV2 and ShuffleNetV2.

REFERENCES

Jinghui Chen and Quanquan Gu. Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint arXiv:1806.06763*, 2018.

Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pp. 534–549. Springer, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

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

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 558–567, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Haiwen Huang, Chang Wang, and Bin Dong. Nostalgic adam: Weighting more of the past gradients when designing the adaptive learning rate. *arXiv preprint arXiv:1805.07557*, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

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

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*, 2019.

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

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. *arXiv preprint arXiv:1002.4908*, 2010.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Martin Riedmiller and Heinrich Braun. Rprop-a fast adaptive learning algorithm. In *Proc. of ISCIS VII), Universitat*. Citeseer, 1992.

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

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Guanglu Song, Yu Liu, and Xiaogang Wang. Revisiting the sibling head in object detector. *arXiv preprint arXiv:2003.07540*, 2020.

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

Tijmen Tieleman and G Hinton. Divide the gradient by a running average of its recent magnitude. coursera neural netw. *Mach. Learn*, 6, 2012.

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

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

Zhiming Zhou, Qingru Zhang, Guansong Lu, Hongwei Wang, Weinan Zhang, and Yong Yu. Adashift: Decorrelation and convergence of adaptive learning rate methods. *arXiv preprint arXiv:1810.00143*, 2018.