Applied Intelligence (2023) 53:14383–14399
https://doi.org/10.1007/s10489-022-04205-9

WSAGrad: a novel adaptive gradient based method

Krutika Verma[1] · **Abyayananda Maiti**[1]

Accepted: 25 September 2022 / Published online: 26 October 2022

Abstract

The vanishing gradient problem under nonconvexity is an important issue when training a deep neural network. The problem becomes prominent in the presence of sigmoid activation. It stops the learning task, which prevents further improvement in the performance of an algorithm. SGD and $ADAM$ are two popular methods frequently used to train deep networks. However, their performance deteriorates in the presence of the vanishing gradient problem. In this paper, we present $WSAGrad$(*An* ***A****daptive* ***Grad****ient method with* ***W****eighted* ***S****ine based step size*). It uses a trigonometric function over the exponential moving average of the weight parameters to compute the step size. This is the first work to use a trigonometric function in the calculation of step size. The trigonometric function combined with weight parameters, has the following significance: (1) it will not fade away when the gradient vanishes because the moving average carries information from the past iterations, and (2) as the weight parameters are tuned with each iteration, it approximately shapes the step size based on the geometric properties of the data. Additionally, two new parameters are incorporated to control the stability and convergence of the proposed algorithm. Moreover, we provide a convergence rate under mild assumptions. In the experimental section, we have shown that our proposed step size performs better than the existing baseline on $FMNIST$, $CIFAR-10$, and $CIFAR-100$ for the classification task.

Keywords Nonconvex problems · SGD · ADAM · Deep network

1 Introduction

The learning efficiency of deep networks [1–3] allowed them to be developed rapidly, and they have been successfully used in various applications, such as machine vision [4], machine translation [5], sound recognition [6], charge prediction in lithium-ion batteries [7, 8] and many more. The deep network-based approach ($FF-LSTM$) in [8] establishes the groundwork for long-term state prediction of lithium-ion batteries with increasing energy management and safety. The extension of deep networks to such vivid applications shows their robustness. In addition to its practical success theoretical findings show its general competence [9, 10]. However, training deep learning models for nonconvex problems is strenuous because finding a global minimum is intractable. In the early 1990s, the authors of [11], showed that training a multilayer perceptron(MLP) is indeed NP-hard, which significantly contributed to the shrinkage of this field. While recent practical successes have revived the area, we still need to know the best minima that can be reached for deep models when trained via different optimizers.

Despite being proposed in the last century, the stochastic gradient descent (SGD) algorithm remains one of the most effective algorithms for training deep neural networks Its simplicity and efficiency make it suitable for almost all applications. However, SGD has the disadvantage of scaling the gradient uniformly in all directions. This might result in poor performance and limited training speed when data are sparse. Several variants of SGD have been proposed [12–14] to speed up the training process and improve performance. These variants perform well when the momentum parameter is significant. However, a large momentum induces staleness [15], implying a preference for past gradients over the current gradient. Consequently, it comes with a cost of reduced generalization capacity. Authors of [16] showed the instability of Nesterov's accelerated method

✉ Krutika Verma
1821cs10@iitp.ac.in

Abyayananda Maiti
abyaym@iitp.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology Patna, Patna, Bihar, India

with respect to initialization and exponentially fast deterioration in the performance with the number of gradient steps. SGD's dependency on the learning rate and the need for proper initialization for ill-conditioned problems necessitate a paradigm shift towards adaptive algorithms with provable guarantees.

A number of adaptive gradient algorithms have been proposed to address these problems. Usually, these algorithms alter the learning rate for each gradient coordinate based on the objective function's current geometry curvature. For example, $Adagrad$ [17] computes the adaptive learning rate by dividing gradients by a denominator equal to the root mean square of the previous gradients. A sparse gradient was reported to lead to a quicker convergence of $Adagrad$ compared to the vanilla SGD [17]. However, $Adagrad$ has a limited ability to generalize on unseen data [18]. In the presence of a sizeable dense gradient, the mean in the denominator makes the update small, leading to the failure of Adagrad.

Some adaptive algorithms have been proposed to address this issue by replacing the denominator with the square root of the exponential moving average of the past gradient squares, i.e., $RMSProp$ [19], $AdaDelta$ [20], and $ADAM$ [21]. $ADAM$ has become the most popular among all the variants due to its faster convergence and computational efficiency. Regardless of its faster convergence, recent theoretical studies show its instability and weaker convergence guarantee [18, 22]. Experiments conducted in [23] show that when the decay rate of the second moment accumulator is too slow, larger-than-desired updates might be provided. The generation of extreme learning rates during training is reported in [24]. The parameter ϵ is introduced in the denominator of the update rule of $ADAM$ [21] to prevent the denominator from becoming zero. Its influence on the performance of $ADAM$ was later discovered in [25].

To address the challenges of $ADAM$ and SGD, we introduce *An **A**daptive **Grad**ient method with **W**eighted **S**ine based step size*($WSAGrad$). Sine-and cosine-based algorithms are generally used with derivative–free algorithms [26–28]. This is the first work to use a sine function to compute the step size for a gradient–based method. This method captures the geometric properties of the data through weight parameters. The intuition of using a weight vector is linked with the primary idea of preserving the metric structure of the data via a deep network initialized with random Gaussian weights, as mentioned in [29]. These structures propagate along the layer, enabling robust recovery of the original data from the features computed by the network. In other words, training a deep network preserves the angle between intraclass and interclass points, which implies that every layer preserves the important information about the data [29]. Hence, using the weight vector to compute the step size of $WSAGrad$ will incorporate this information, which is further tuned with each set of iterations. $WSAGrad$ shows a significant improvement in performance for different deep learning tasks over other state-of-the-art techniques. A theoretical guarantee is proposed with some mild assumptions. The main highlights of this paper are as follows:

- The paper contributes by proposing a novel adaptive step size based gradient descent algorithm that inherits important capabilities of both approaches such as the adaptability and generalization ability of $ADAM$ and SGD.
- The step size is constructed using a trigonometric function over the exponential moving average of the norm of the weight parameters. It is designed to operate on the vanishing gradient problem. Instead of stopping the learning task, it minutely increases the objective value so that the coming iteration can reach a better minimum. Additionally, two controlling parameters are used to nullify the scope of divergence and assure stability inside a certain region. To the best of our knowledge, this is the first work to use a network's weight parameters in the step size.
- The step size is computationally efficient due to the small memory requirement per iteration with only one extra set of auxiliary variables. Furthermore, it reduces human intervention and hyperparameter tuning, making it more robust to different optimization problems.
- Under a minimum assumption of smoothness and level bound, we prove the convergence of our algorithm.
- We validate the performance of our algorithm through extensive experiments. We tested the performance of our algorithm compared to that of the baseline on the task of image classification on real-world data $FMNIST$, $CIFAR-10$ and $CIFAR-100$. The experiments span basic to advanced architectures of different batch sizes with various activations types.

It is worth noting that our algorithm's performance significantly improves with the number of iterations when most of the existing methods saturate. We also show the stability of our algorithm in all sets of experiments. For each set of experiments, we have reported the best results, the mean and the standard deviation of our algorithm's performance. The remained of this paper is organized as follows: we discuss the literature in Section 2. Section 3 outlines the problem definition in detail, followed by the proposed methodology and a theoretical guarantee under mild assumptions. Next, we describe the dataset, experimental setup, and baselines in Section 4, followed by results and analysis under the same heading. Lastly, Section 5 concludes the paper with some directions for related future works.

2 Background and notation

2.1 Background

Training a deep neural network (DNN) for a large-scale machine learning problem is a complex task. Recently, adaptive gradient methods, a class of optimization tools, have drawn the community's attention due to their faster convergence on such tasks. The first popular algorithm in this line of research is $Adagrad$ [17, 30], which achieves significantly better performance than the vanilla SGD when gradients are sparse or generally small. Update equation of $Adagrad$ is as follows:

$$x_{t+1} = x_t - \eta_t \frac{g_t}{\sqrt{(v_t)}}, \tag{1}$$

where $g_t = \nabla f(x_t; \xi t)$, and $v_t = \frac{1}{t}\sum_{j=1}^{t} g_j^2$, and $\eta_t = \frac{\eta}{\sqrt{(t)}}$ where $\eta > 0$ is the step size. The above equation shows the learning rate for each dimension. Although $Adagrad$ works well for sparse settings, its performance has been observed to degrade in conditions where loss functions are nonconvex and gradients are dense. The quick decline of the learning rate in these settings makes it unattractive because it uses all the previous gradients in the update. This issue is compounded even more in high-dimensional deep learning problems.

To mitigate this problem, several variants of $Adagrad$, such as $RMSProp$ [19], $ADAM$ [21], $AdaDelta$ [20], and $NADAM$ [31] have been proposed to handle the rapid decay of the learning rate by using the exponential moving averages of past squared gradients. Essentially, these strategies limit the update to only the past few gradients. Of these variants, $ADAM$ is the default method of choice for training DNNs. The updated equation adopts the following form:

$$\begin{aligned} m_t &= \alpha_1 m_{t-1} + (1-\alpha_1)\nabla f(x_t; \xi t), \hat{m}_t = \frac{m_t}{1-\alpha_1^t} \\ v_t &= \alpha_2 v_{t-1} + (1-\alpha_2)(\nabla f(x_t; \xi t))^2, \hat{v}_t = \frac{v_t}{1-\alpha_2^t}, \end{aligned} \tag{2}$$

$$x_{t+1} = x_t - \frac{\eta_t \hat{m}_t}{\sqrt{(\hat{v}_t)} + \epsilon}, \forall t \geq 1, \tag{3}$$

where $\alpha_1, \alpha_2 \in (0, 1)$, and $\epsilon > 0$ and $\eta_t = \frac{\eta}{\sqrt{(t)}}$, $\eta > 0$. However, to our knowledge, $ADAM$ still has no convergence proof. The proof in the original paper [17] was shown wrong in [32, 33]. A counterexample was introduced in [18], and an improved form of (3) was introduced and named $AMSGrad$. It follows:

$$\hat{v}_t = \max(\hat{v_{t-1}}, \hat{v}_t); x_{t+1} = x_t - \frac{\eta_t \hat{m}_t}{\sqrt{(\hat{v}_t)}}. \tag{4}$$

However, in the experiments, $AMSGrad$ does not show improvements [34, 35]. In contrast, it sometimes results in a worse accuracy than $ADAM$. For $ADAM$-type optimizers, convergence is studied in [35] for nonconvex problems, and the authors reported that a minimum could be achieved with no guarantee of staying there. Another work [36] used a dynamical system viewpoint to interpret the $ADAM$ optimizer. This approach uses a non-autonomous ordinary differential equation without the Lipschitz condition. Subsequently, to improve the generalization performance of $ADAM$, another variant with a convergence guarantee, namely, $AdamW$, was introduced [37]. It is defined as follows:

$$\begin{aligned} g_t &= \nabla f(x_t; \xi t) + \lambda x_t, m_t = \alpha_1 m_{t-1} + (1-\alpha_1) g_t, \\ \hat{m}_t &= \frac{m_t}{1-\alpha_1^t}, v_t = \alpha_2 v_{t-1} + (1-\alpha_2)(g_t)^2, \hat{v}_t = \frac{v_t}{1-\alpha_2^t}, \end{aligned} \tag{5}$$

$$x_{t+1} = x_t - \eta_t(\alpha \hat{m}_t / (\sqrt{(\hat{v}_t)} + \epsilon) + \lambda x_t), \tag{6}$$

where, $\alpha_1, \alpha_2 \in (0, 1)$, $\alpha > 0, \lambda > 0$ and $\epsilon > 0$. In [38], under a convex setting, the authors proposed stabilizing the coordinatewise weighting factors to ensure convergence. $PADAM$ [39] has been introduced with a partial adaptive parameter for a better generalization, which changes the coordinatewise weighting factor. The convergence rates of the original $ADAM$ and $RMSprop$ under the full-batch (deterministic) setting are provided in [40, 41]. $AdaBelief$ [42] has been proposed to obtain a good generalization by adopting the step size according to the *'-belief-'* in the current gradient direction. All the equations are similar to $ADAM$ except for v_t, which is defined as follows:

$$v_t = \alpha_2 v_{t-1} + (1-\alpha_2)(\nabla f(x_t; \xi t) - m_t)^2 + \epsilon. \tag{7}$$

More recently, some accelerated adaptive gradient methods [43, 44] have been proposed based on variance-reduced techniques. In particular, the authors of [44], proposed $SUPER-ADAM$, a faster and universal adaptive gradient framework based on a universal adaptive matrix. Recently, an adaptive variant of Nesterov accelerated gradient descent was introduced in [45], which replaces the constant momentum with an adaptive momentum. To assure stability, the momentum resets to zero according to a scheduler. In Table 1, we list some of the most recent algorithms and their convergence rates.

This paper attempts to design an algorithm that works well with a minimum set of resources and models. Rather than focusing on the moving average of the gradient as most of the previous algorithms have done, our work depends on the weight parameters. The idea is that with each iteration, we obtain weights that are more tuned than the previous weights. An adjusted weight with a step size as defined in [46] works towards decreasing the objective function with better smoothness. Next, we present the assumptions used for the theoretical analysis. In the following section we define the problem and introduce the proposed algorithm and convergence proof.

Table 1 Convergence rates of different algorithms: here, T denotes the number of iterations, and b denotes the mini-batch size

Assumptions	Methods	Convergence rate
Smoothness or component-wise smoothness of the objective function, Bounded stochastic and true gradient.	Generalized ADAM[35]	$O\left(\frac{\sqrt{log(T)}}{T^{1/4}}\right)$
	ADAM/ YOGI [25]	$O\left(\frac{1}{\sqrt{T}}+\frac{1}{\sqrt{b}}\right)$
	PADAM [39]	$O\left(\frac{1}{\sqrt{T}}+\frac{1}{\sqrt{T^{1/4}}}\right)$
	AdaBelief [42]	$O\left(\frac{\sqrt{log(T)}}{T^{1/4}}\right)$
Smoothness of the objective function and true gradient	Ada-Norm-SGD[47]	$O\left(\frac{1}{T^{2/7}}\right)$
	STORM [43]	$O\left(\frac{(ln(T)^{3/4})}{\sqrt{T}}+\frac{\sqrt{(ln(T))}}{T^{1/3}}\right)$
Lipschitz continuous and Smooth objective	Adaptive SGD [48]	$O\left(\frac{\sqrt{ln(T)}}{\sqrt{T}}+\frac{\sqrt{(ln(T))}}{T^{1/4}}\right)$
Smooth objective and Bounded gradient	AdaGrad-Norm [49]	$O\left(\frac{\sqrt{log(T)}}{T^{1/4}}\right)$
Smooth or Component wise smooth	SUPER-ADAM [44]	$O\left(\frac{\sqrt{ln(T)}}{\sqrt{T}}+\frac{\sqrt{(ln(T))}}{T^{1/4}}\right)$
Lipschitz continuous, Smooth objective and bounded gradient	WSAGrad (Our method)	$O\left(\frac{1}{\sqrt{T}}\right)$

2.2 Assumptions and notations

In general, we consider a finite sum problem of the following form:

$$\min_{x\in\mathbb{R}^d} f(x) = \frac{1}{n}\sum_{i=1}^{n} m_i(x), \tag{8}$$

where each $m_i : \mathbb{R}^d \to \mathbb{R}$ is a smooth and nonconvex function. Prior to the analysis, we state the mild assumptions and notations required for the proof. We work with the Euclidean spaces, $\mathbb{R}^n$ and $\mathbb{R}^d$, which are equipped with the standard inner product $\langle . \rangle$ and Euclidean norm $\| \ . \ \|$. For any function f, $dom(f) := \{x \in \mathbb{R}^d | f(x) < +\infty\}$ denotes the effective domain of f. If f is continuously differentiable, then ∇f denotes its gradient. Furthermore, if f is twice continuously differentiable, then $\nabla^2 f$ denotes its Hessian. Bold upper-case letters represent matrices **A,B**, normal upper-case letters represent scalars X, Y, and lower-case letters denote vectors x, y. x_t represents the value of x at time step t.

Assumption 1 A function $f(x)$ is at least once differentiable and is L-smooth, i.e., for any $x, y \in \mathbb{R}^d$ we have the following:

$$f(x) \leq f(y) + \langle \nabla f(y), x-y \rangle + \frac{L}{2} \| x-y \|^2, \forall x, y \in \mathbb{R}^d. \tag{9}$$

Assumption 2 The level set S_0 is bounded as follows:

$$S_0 = \{x \in \mathbb{R}^d | f(x) \leq f(x_0)\}, \tag{10}$$

i.e., there exists a constant $R > 0$ such that, $\| x \| \leq R, \forall x \in S_0$, where $R << d$.

Assumption 3 An obvious conclusion of the above two assumptions is that the gradient of the function and the stochastic gradient are bounded, i.e., as follows:

$$\| \nabla m_i(x) \| <= C, \| \nabla m_i(x : \xi) \| <= C, \tag{11}$$

where ξ is a random variable, which implies, the following:

$$\| \nabla f(x) \| <= \frac{1}{n}\sum_{i=1}^{n} \| \nabla m_i(x) \| <= C. \tag{12}$$

Definition 1 A point x is said to be ϵ- **First order stationary point** (FSP) for a function $f : \mathbb{R}^d \to \mathbb{R}$, if

$$\| \nabla f(x) \| \leq \epsilon. \tag{13}$$

3 Methodology

Let $f(x)$ be a noisy objective function, i.e., a differentiable stochastic scalar function with respect to parameters x. We are interested in minimizing the expected value of this function, $E[f(x)]$ w.r.t. its parameters x. Let, $f_1(x), \ldots, , f_T(x)$ denote the realizations of the stochastic function at subsequent time steps 1, ..., T. The stochasticity may result from the assessment of the data points in random subsamples (minibatches) or from intrinsic function noise.

Let, $\nabla_x f_t(x)$ denote the gradient, i.e., the vector of partial derivatives of f_t, w.r.t x evaluated at timestep t

3.1 Proposed method

Require: $x_0, \beta, \gamma_1, \gamma_2$
1: Initialize $g_{-1} \leftarrow 0$
2: **for** $t = 1$ **to** n **do**
3: $\quad g_t = \beta * g_{t-1} + (1-\beta) * \| x_t \|$
4: $\quad x_{t+1} \leftarrow x_t - (\gamma_1 * \sin(\frac{\pi * g_t}{2 * g_t + 1}) + \gamma_2) \nabla f_t(x_t)$
5: **end for**
6: **return** x_n

Algorithm 1 WSAGrad($x_0, \beta, \gamma_1, \gamma_2$).

Our proposed Algorithm 1 inherits the qualities of both the paradigms of the first order method. The equation

$$x_{t+1} \leftarrow x_t - (\gamma_1 * \sin(\frac{\pi * g_t}{2 * g_t + 1}) + \gamma_2) \nabla f_t(x_t), \tag{14}$$

relies on the following four important parameters: x_t (weight parameter), g_t (exponential moving average over x_t), β, which lies in an interval (0, 1) that decides the rate of the moving average, and γ_1, which represents the weights given to the calculated step size. γ_2 is an additive lower limit, which prevents zero in the step size. To understand its connection with the existing algorithm and the overall improvement in the proposed algorithm, we simplified (14) by assigning different values to the parameters. Let us consider that the value of γ_1 equals zero; thus, (14) can be written as follows:

$$x_{t+1} \leftarrow x_t - \gamma_2 \nabla f_t(x_t). \tag{15}$$

The above equation is equivalent to the equation of SGD, which signifies how the inherent quality from SGD affects the generalization capability of our algorithm. To check its connection with the adaptive method, we consider γ_2 equal to zero. Thus, (14) can be written as follows:

$$x_{t+1} \leftarrow x_t - \left(\gamma_1 * \sin\left(\frac{\pi * g_t}{2 * g_t + 1}\right)\right) \nabla f_t(x_t). \tag{16}$$

We can observe the value of the sine function, which depends on g_t, and as g_t is tuned with every iteration, the value of the sine function changes for each iteration. The adaptability of our algorithm helps to obtain quicker convergence to better minima. Using a weight vector in the calculation of the step size gives significant advantages over gradient information, as it includes more stable information about the geometry of data. Random Gaussian weights with a deep neural network preserve local structures in the manifold, as proven in [29]. Weight parameters learn decision boundaries for the given data through extensive training. The best decision boundaries have an optimal distance from the points that need to be classified or predicted. In addition, the angle between intraclass points and interclass points is preserved while training the network. Thus, it was observed in [29], that every layer keeps the important information about the data, which is a characteristic very desirable for the classification task. Hence, weigh parameters represent the geometric information of the data across the loss landscape. However, the gradient alone is insufficient to carry such information because it is dependent on the loss surface, which makes it fluctuate heavily. Hence, we consider a weighted average of the weight parameters' magnitude to compute our step size. Through experiments, we find that it outperforms the baselines. Moreover, for the proposed algorithm, we observed that the values of γ_1, γ_2 depend greatly on the architecture we use for our tasks.

3.2 Step size analysis

The proposed step size consists of the following three important components: the values of γ_1, γ_2 and the sine function used over the first moment of the weight parameters. The values of γ_1 and γ_2 determine the range of fluctuation for the step size. These parameters depend on the model architecture. An architecture that smooths out the complex loss landscape requires a smaller value, while the value is high for the opposite case.

We know that $-1 \leq sin(x) \leq 1$. For the proposed step size, the parameter satisfies $0 \leq \frac{g_t}{2g_t+1} \leq 1$. Therefore, $0 \leq \frac{g_t \pi}{2g_t+1} \leq \pi$. Thus,

$$0 \leq \sin\left(\frac{g_t \pi}{2g_t + 1}\right) \leq 1, \tag{17}$$

$$0 \leq \sum_{k=0}^{T} \sin\left(\frac{g_k \pi}{2g_k + 1}\right) \leq T + 1. \tag{18}$$

The maximum value for the above sine function after the T iteration will be $T + 1$ and the minimum will be zero. Moreover, the sine function is non-monotone. To understand the behavior of the proposed step size, we categorize the value of $\| x_t \|$ into the following ranges: when the value of $\| x_T \| \leq 1$ and when the value of $\| x_T \| \geq 1$. Let us assume for consecutive T iterations, that the value of $0 \leq \| x_T \| \leq 1$; then, $\frac{g_T}{2g_T+1}$ lies near to zero. This implies

$$\sin(\| x_T \|) < \sin\left(\frac{g_T \pi}{2g_T + 1}\right). \tag{19}$$

This means that if the minima lie very near zero for a nonconvex problem, and initialization of weights is done

within a unit circle, then gradient information of the loss function with respect to weights will quickly vanish when the network size becomes large, as the initial layer of the network suffers from the vanishing gradient problem. The stated problem can be easily addressed by our proposed step size. The above equation always assures that the value of sine over the expected x_t is greater or equivalent to the sine over the original x_t. To control the stability of the sine function, we provide weight through γ_1. When the sine function becomes zero, then the variable γ_2 assures a decrease in the objective function. Now, when $\parallel x_t \parallel \geq 1$, then $\frac{g_T}{2g_T+1}$ lies near 1, which implies

$$\sin(\parallel x_T \parallel) \geq \sin\left(\frac{g_T \pi}{2g_T + 1}\right). \tag{20}$$

We can reasonably assume from the existing works that $\nabla f(x_t)$ is large when x is far from the minima of the loss function. Thus, the magnitude of the resultant vector will also be large. Therefore, the value of the step size will be closer to 1, which implies, that it provides an opportunity to reduce the value of the objective function more significantly in the initial iterations. In the initial iterations, the step size increases when the value of g is not very large and significantly reduces the objective value. However, when x is close to the minima of the loss function, $\nabla f(x)$ becomes small. Consequently, the value of g is close to the moving average of all the resultant vectors. The parameters $\{\beta, (1-\beta)\}$ give weights to the present and past magnitudes of the resultant vector. Therefore, the step length will not be minimal but will decrease and can help the optimizer surpass the suboptimal points.

3.3 Convergence analysis in a deterministic setting

Lemma 4 *(Necessary condition for convergence) For a positive series $\sum_{n=1}^{\infty} a_n$, with any natural number k , if the partial sum $S_k = \sum_{n=1}^{k} a_n$ has a constant upper bound C that is independent of k, then we have*

$$lim_{n\to\infty} a_n = 0. \tag{21}$$

Theorem 5 [50] *Let $\{x_n\}_{n=1}^{\infty}$ and $\{y_n\}_{n=1}^{\infty}$ be two convergent sequences of real numbers; then,*

$$if \quad lim_{n\to\infty} x_n = x, \quad and \quad lim_{n\to\infty} y_n = y, \quad then \quad lim_{n\to\infty} x_n y_n = xy. \tag{22}$$

Lemma 6 *For time sequence $t = 0, \ldots, T-1$, the recurrence relation is bounded as*

$$g_t = \beta * g_{t-1} + (1-\beta) * \parallel x_t \parallel \leq T. \tag{23}$$

Proof By iterating and expanding the above recurrence and then simplifying altogether and using assumptions (2), we have,

$$g_T = (1-\beta) \sum_{i=0}^{T-1} \beta^{n-1-i} \parallel x_i \parallel \leq R(1-\beta^T). \tag{24}$$

As the value of $\beta < 1$, for a large T,

$$R(1-\beta^T) \approx R \leq T. \tag{25}$$

□

Theorem 7 *Suppose $f(x)$ satisfy assumptions* (1), (2), (3)*, and WSAGrad runs under batch setting with a positive step size sequence and, $f^* = inf_x f(x) \geq -\infty$, then*

$$lim_{T\to\infty} \min_{t=0:T} \parallel \nabla f(x_t) \parallel = 0. \tag{26}$$

Proof From (9) , we have the following:

$$f(x_{t+1}) \leq f(x_t) + \langle \nabla f(x_t), x_{t+1} - x_t \rangle + \frac{L}{2} \parallel x_{t+1} - x_t \parallel^2 . \tag{27}$$

Using the update equation given in Algorithm 1, we have the following:

$$f(x_{t+1}) \leq f(x_t) + \left(\frac{L}{2}\eta_t - 1\right) \eta_t \parallel \nabla f(x_t) \parallel^2 . \tag{28}$$

Rearranging the above equation, we obtain the following:

$$\left(\frac{L}{2}\eta_t - 1\right) \eta_t \parallel \nabla f(x_t) \parallel^2 \leq f(x_{t+1}) - f(x_t). \tag{29}$$

After summing the values from $t = 0$ to $T-1$, using telescopic sum, we obtain

$$\sum_{t=0}^{T-1} \left(1 - \frac{L}{2}\eta_t\right) \eta_t \parallel \nabla f(x_t) \parallel^2 \leq f(x_0) - f(x^*). \tag{30}$$

Thus,

$$\begin{aligned} \min_{t=0:T} \parallel \nabla f(x_t) \parallel^2 \sum_{t=0}^{T-1} \left(1 - \frac{L}{2}\eta_t\right) \eta_t &\leq \sum_{t=0}^{T-1} \left(1 - \frac{L}{2}\eta_t\right) \eta_t \parallel \nabla f(x_t) \parallel^2 \\ &\leq f(x_0) - f(x^*), \end{aligned} \tag{31}$$

which is equivalent to the following:

$$\min_{t=0:T} \parallel \nabla f(x_t) \parallel^2 \leq \frac{f(x_0) - f(x^*)}{\sum_{t=0}^{T-1} (1 - \frac{L}{2}\eta_t)\eta_t}. \tag{32}$$

We can obtain an upper bound of the denominator of

$$\sum_{t=0}^{T-1} \left(1 - \frac{L}{2}\eta_t\right) \eta_t \leq T \left(1 - \frac{L}{2}\eta_z\right) \eta_z, \tag{33}$$

where $\eta_z = \max_t(\eta_t)$, $\max_t(\eta_t) = (\gamma_1 + \gamma_2)$. Rewriting the above equation provides a bound for the minimum square

norm of the gradient in T steps. Upon taking the limit on both sides of the above equation, we obtain

$$\min_{t=0:T} \| \nabla f(x_t) \|^2 \leq \frac{f(x_0) - f(x^*)}{T(1 - \frac{L}{2}\eta_z)\eta_z}. \quad (34)$$

□

4 Empirical study

We study the performance of our algorithm on the problems of multiclass classification. The proposed algorithm spans different architectures, and the complexity varies from basics to advance. We use SGD, $SGD - Nesterov$ and $ADAM$ as baselines on the $FMNIST$, $CIFAR - 10$ and $CIFAR - 100$ datasets. Each method's performance is evaluated with the best setting of manually tuned hyperparameters, and the performance is stated under multiple restarts.

4.1 Synthetic Experiment

To compare the performance of $WSAGrad$ with that of $ADAM$, we consider a simple convex function from [18].

$$f_t(x) = \begin{cases} 1010x & \text{with probability } 0.01 \\ -10x & otherwise, \end{cases} \quad (35)$$

with constraint set $F = [-1, 1]$. The optimal solution stated in [18] is, at $x = -1$. Thus, for convergence, we expect the algorithms to converge at $x = -1$. For this sequence of functions, we investigate the average regret calculated as in (36) and the value of iterate x_t for $ADAM$ and $WSAGrad$.

$$R(T) = \frac{1}{T}\sum_{t=1}^{T}[f_t(x_t) - f_t(x^*)] \quad (36)$$

To enable a fair comparison, we set $\beta_1 = 0.9$, and $\beta_2 = 0.99$, which are typical settings in $ADAM$. For $WSAGrad$, we consider $\beta = 0.99$. Figure 1 shows the average regret and value of x_t at the t iteration. From the two plots in Fig. 1, we can conclude that $WSAGrad$ converges at $x_t = -1$, which is the optimal value for the parameter, and $ADAM$ converges near zero. Therefore , the average regret for $WSAGrad$ is lower than that for $ADAM$.

4.2 Neural network

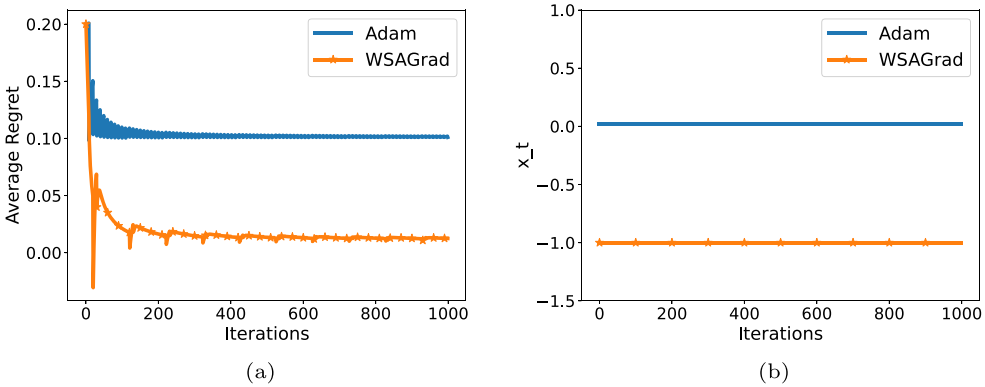
With nonconvex objective functions, multilayer neural networks, CNN, VGG, and $ResNet$ some powerful deep learning models. Although our convergence analysis may not be applicable up to a certain extent to nonconvex problems, we empirically observe that $WSAGrad$ outperforms other methods for the task of classification. We use MLP, CNN, VGG, and $ResNet$ for the classification task on three different datasets. An overview of our experiments is presented in Table 2.

4.2.1 Multilayer perceptron

We use two configurations for our experiments: one with $sigmoid$ in all the layers and another with $SoftMax$ in the last layer and $sigmoid$ in all the rest. To measure the performance of our algorithm, we prefer $sigmoid$ over any other activation type. The reason behind using $sigmoid$ in all layers is that it creates a vanishing and exploding gradient problem in the initial and final layers, respectively, which can stop the learning task for all the algorithms.

FMNIST Fashion-MNIST($FMNIST$) [51] is a dataset consisting of a training set of 60000 examples images and a test set of 10000 examples. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels. This pixel-value is an integer between 0 and 255. For our experiments, we normalize the values. Our neural network contains two hidden layers of sizes 256 and 128 and an output layer of size 10. We use the cross-entropy function as a loss function with l_2 regularization and an accuracy metric to measure the performances of different algorithms.

We repeat the experiment multiple times with a batch size of 256 for $SoftMax$ and a size of 64 for all $sigmoids$



(a) (b)

Fig. 1 (a) Average regret w.r.t. iterations of $ADAM$ and $WSAGrad$. (b) Value of x_t w.r.t. iterations of $ADAM$ and $WSAGrad$

Table 2 Overview of experiments: our experiments span across classic datasets along with traditional and modern architectures

Dataset	Architecture	Batch size	Lr for ADAM	Lr for SGD, SGD-Nesterov	Value of γ_1, γ_2
FMNIST	3-Layer MLP	[256,64]	0.001	[0.8,0.8]	[0.9, 0.01]
FMNIST	ConvNet	[128]	0.001	[0.1,0.1]	[0.95, 0.01]
CIFAR-10	3-Layer MLP	[256]	0.001	[0.1,0.1]	[0.95, 0.01]
CIFAR-10	ConvNet	[128]	0.001	[0.1,0.1]	[0.95, 0.01]
CIFAR-10	ResNet18	[128,32]	Table 9	Table 9	[0.5,0.01]
CIFAR-10	VGG11	[128,32]	Table 7	Table 7	[0.3,0.1/0.01]
CIFAR-100	3-Layer MLP	[256]	0.001	[0.1,0.1]	[0.95, 0.01]
CIFAR-100	ConvNet	[128]	0.001	[0.1,0.1]	[0.95, 0.01]

The optimal learning rates for the baselines vary. A detailed explanation of the parameters is presented in the corresponding tables, e.g., TB-9 represents Table 9. The value of γ_1 in $WSAGrad$ varies for different architectures. The results are based on the optimal value of hyper-parameters tuned over multiple restarts with consistent configuration. Here, Lr stands for learning rate, and TB stands for table

with different seeds. For all the experiments, we decay the learning rate after every 1000^{th} iteration with a decay factor of 0.7. Then we plot the best results for all the algorithms under the consistent configuration. For SGD and its variant, we consider 0.1 as the suitable learning rate; for $ADAM$ and its variant, the suitable learning rate is 0.001. $\beta_1 = 0.9$ and β_2 are chosen from $\{0.99, 0.999\}$. We manually tune parameter β for our algorithm to obtain the best set of results for the defined models. $WSAGrad$ provides the best result for the value of $\beta = 0.5$.

CIFAR-10 The $CIFAR-10$ [52, 53] dataset consists of 60000 of $32x32$ color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Among them, the training batches contain exactly 5000 images from each class. The classes are completely mutually exclusive. For the experiments, normalization was performed. We use a neural network with hidden layers, of size 655, 256, and an output layer of size 10. We use the cross-entropy function as a loss function with l_2 regularization and accuracy metric to measure the performance of the different algorithms. We repeat the experiments multiple times with a batch size of 128 for both configurations with different seeds. The rest of the configurations are the same as those of the previous dataset $FMNIST$, and the best results are plotted. $WSAGrad$ gives the best result for the value of $\beta = 0.6$.

CIFAR-100 This dataset [52] is similar to $CIFAR-10$, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The reason for using $CIFAR-100$ is the increasing complexity since limited data are available per class to train. To test the authenticity of our algorithm, we use both the $sigmoid$ and $SoftMax$ configurations. The dataset is difficult, and using $sigmoid$ makes learning difficult. For this specific dataset, we use a dropout of 0.2 just after the first layer and 0.5 after the second layer. The rest of the configuration and preprocessing are similar to those of $CIFAR-10$. $WSAGrad$ gives the best result for the value of $\beta = 0.6$.

The average accuracy of MLP for different optimizers on the above dataset is shown in Table 3.

4.2.2 Deep networks

CNN Our CNN architecture has two alternating stages of 5×5 convolutional filters and 3×3 max pooling with stride two followed by a fully connected layer of 128 $sigmoid$ units. The channel sizes are 32, and 16 respectively. The batch size is 128. Cross entropy with l_2 regularization is used for loss. For $CIFAR-10$, we use the same configuration but with a few changes. For a set of experiments, we replace the last layer $sigmoid$ units with $SoftMax$, and in another set, we keep $sigmoid$ units in the last layer. In the third set, we use $ReLU$ for fully connected layers. For $CIFAR-100$, we use the architecture with all $sigmoids$.

VGG on CIFAR-10 $VGGNet$ [54] is another advanced architecture in the field of computer vision after CNN. The structure captures the depth of CNNs. We use the $VGG-11$ architecture for our experiment, as shown in Fig. 6. It consists of 11 weighted layers. The weights represent the strength of connections between units in adjacent network layers, and weights close to zero mean that changing the input will not change the output, or the change will be negligible. The primary reason for using $VGG-11$ in our experiments is the vanishing gradient problem that occurs

Table 3 Mean accuracy with standard deviation of the different optimizer's for MLP on different datasets

Data set	MLP with $sigmoid$	MLP with $SoftMax$	Methods
FMNIST	**0.88± 0.008**	**0.88± 0.01**	$WSAGrad$
CIFAR-10	**0.47± 0.03**	**0.51± 0.01**	$WSAGrad$
CIFAR-100	**0.20± 0.01**	0.07 ± 0.01	$WSAGrad$
FMNIST	0.83± 0.08	0.87± 0.01	ADAM
CIFAR-10	0.44± 0.01	**0.51± 0.01**	ADAM
CIFAR-100	0.15± 0.01	**0.08± 0.01**	ADAM
FMNIST	0.86± 0.05	0.87± 0.01	SGD Nesterov
CIFAR-10	0.36± 0.01	0.49±0.01	SGD Nesterov
CIFAR-100	0.12± 0.1	0.01 ± 0.01	SGD Nesterov
FMNIST	0.47± 0.07	0.75± 0.01	SGD
CIFAR-10	0.25± 0.01	0.35± 0.01	SGD
CIFAR-100	0.08± 0.009	0.01 ± 0.01	SGD

Bold fonts represents best mean accuracy

when the weights substantially reduce to zero, significantly impacting the learning task. The normal distribution is used to initialize the weights. An overview of the architecture is given below.

We conduct our experiments for the above architecture set with different variations tested on different batch sizes with fine-tuned learning rates. The variations in the architecture were created by replacing the activation function of the fully connected last layers. The reason for testing on different batch sizes is that a large batch induces smoothness, while a smaller batch does not. Therefore, it is more difficult to train on small batches than on larger batches.

ResNet18 on CIFAR-10 $ResNet$ [55] is an advanced architecture compared to VGG. It allows residual connection between layers, which helps it overcome the vanishing gradient problem. We modify the architecture by replacing the activation type of the last layer. The network in Table 8 is tested on $CIFAR-10$ with different batch sizes and finely tuned hyperparameters. Cross entropy with l_2 regularization is used for loss. The hyperparameters for the baselines and proposed method are shown in Table 2. The value of γ_1 is selected based on the stability and low error rate. It is determined through multiple sets of experiments on different values of γ_1, as shown in Fig. 8. The most suitable choice for γ_1 for the mentioned architecture was $\{0.3, 0.5\}$, which can be observed from Fig. 8. We select 0.5 over 0.3 due to its better stability and accuracy.

4.3 Results and discussion

For each set of experiments, we plot the best results and report the mean and the standard deviation of the algorithms' performances in Tables 3, 4, 5, 6, 7, 8 and 9. For MLP, the performance of $WSAGrad$ is equivalent to greater than existing state-of-the-art techniques.

From Figs. 2, and 3, we find that the performance of $WSAGrad$ is better than that of $ADAM$ and $AdamW$. Additionally, from Fig. 2 we observe that $ADAM$, $AdamW$, and $SGD-Nesterov$ converge in a nearby neighbourhood; nevertheless, the generalization capability of $SGD-Nesterov$ suffers. This is due to the continuous addition of bias in the update rule when the momentum parameter is large. A large momentum parameter prefers past updates instead of the current one. In theory, the performance of $SGD-Nesterov$ is independent of initialization, but in practice, we observe that the performance of SGD and $SGD-Nesterov$ heavily depend on the initialization and complexity of the model. Furthermore, we observe that the performance of $WSAGrad$ increases smoothly as reflected in Figs. 2 and 3, with successive iterations without affecting the model's generalization capability. We observe a similar performance of our algorithm for all the models and datasets used for empirical validation.

It is worth noting from Table 3, that the mean accuracy of our algorithm is higher than that of the top baselines, except for MLP with $SoftMax$, where $ADAM$ is equivalent to or greater than $WSAGrad$. A possible reason is that MLP with $SoftMax$ forces the structure of one winner, even when it has access to a limited set of features. For example, training a neural network creates a set of groups across the different layers and within the layers, as proved in [29]. Each set of groups carries information about a set of features; using $SoftMax$ in the last layer induces a one-winner relationship between groups. This relationship improves the generalization capability when the dataset

Table 4 Mean accuracy with standard deviation of the different optimizer's on Convolution net with $sigmoid$ and $SoftMax$ functions as the activation on different datasets

Data set	CNN with $sigmoid$	CNN with $SoftMax$	Methods
FMNIST	**0.88±0.007**	**0.88± 0.01**	$WSAGrad$
CIFAR-10	**0.66±0.04**	**0.63±0.017**	$WSAGrad$
CIFAR-100	**0.27±0.01**	**0.12±0.01**	$WSAGrad$
FMNIST	0.80±0.005	0.83±0.007	ADAM
CIFAR-10	0.52±0.01	0.55±0.01	ADAM
CIFAR-100	0.16±0.01	0.096 ± 0.01	ADAM
FMNIST	0.80±0.006	0.84±0.001	SGD Nesterov
CIFAR-10	0.53±0.01	0.58±0.01	SGD Nesterov
CIFAR-100	0.14±0.004	0.04 ± 0.01	SGD Nesterov
FMNIST	0.69±0.01	0.74±0.01	SGD
CIFAR-10	0.52±0.01	0.55±0.02	SGD
CIFAR-100	0.06±0.01	0.04 ± 0.01	SGD

Bold fonts represents best mean accuracy

Table 5 Mean accuracy with standard deviation of different optimizer's on Convolution net with $ReLU$ and $SoftMax$ functions as the activation function, on different dataset

Data set	CNN ReLU with SoftMax	CNN ReLU	Methods	learning rate
FMNIST	**0.89 ± 0.13**	**0.90±0.01**	WSAGrad	-
CIFAR-10	**0.66± 0.15**	**0.71±0.01**	WSAGrad	-
CIFAR-100	0.10± 0.01	**0.33± 0.01**	WSAGrad	-
FMNIST	0.87 ± 0.13	0.83±0.01	ADAM	0.001
CIFAR-10	0.53± 0.10	0.57±0.01	ADAM	0.0008
CIFAR-100	**0.11± 0.01**	0.22 ± 0.01	ADAM	0.0008
FMNIST	**0.89 ±0.09**	0.89 ± 0.01	SGD Nesterov	0.1
CIFAR-10	0.63±0.01	**0.71±0.014**	SGD Nesterov	0.08
CIFAR-100	0.09±0.01	**0.33±0.01**	SGD Nesterov	0.08
FMNIST	0.88 ±0.01	0.89 ± 0.001	SGD	[0.8.0.1]
CIFAR-10	0.61± 0.01	0.64±0.01	SGD	0.8
CIFAR-100	0.10±0.01	0.31±0.01	SGD	[0.08,0.8]

The value of $\{\gamma_1, \gamma_2\} = \{0.07, 0.01\}$, and $\beta = 0.1$. The batch size is similar to those of the previous experiments. Learning rate is not required for $WSAGrad$

Bold fonts represents best mean accuracy

Table 6 Mean accuracy with standard deviation and execution time on VGG-11 for CIFAR-10: the accuracy is averaged over multiple restarts with $linear$ activation in the last layer and a batch size of 128

Method	Accuracy	Method	Time in seconds
SGD	0.66 ± 0.01	SGD	1344.342
SGD -Nesterov	0.77 ± 0.01	SGD-Nesterov	1378.271
ADAM	0.74 ± 0.016	ADAM	1417.314
WSAGrad	**0.78 ± 0.01**	WSAGrad	1988.764

The execution time represents the time to finish the total number of epochs

Bold fonts represents best mean accuracy

Table 7 Mean accuracy with standard deviation on VGG-11 for CIFAR-10: The accuracy is averaged over multiple restarts with different batch sizes and activation types; for $SoftMax$ with batch size 32, we consider the value of $\{\gamma_1, \gamma_2\}$= {0.05,0.01}; for batch size 128, $\{\gamma_1, \gamma_2\}$= {0.1,0.01} and $\beta = 0.1$

Activation	Batch size	Method	Learning-rate	Accuracy
SoftMax	32	ADAM	0.0001	0.59 ± 0.01
SoftMax	32	SGD	0.2	0.62 ± 0.01
SoftMax	32	SGD- Nesterov	0.02	0.64 ± 0.01
SoftMax	32	WSAGrad	-	$\mathbf{0.69 \pm 0.01}$
SoftMax	128	ADAM	0.0009	0.58 ± 0.01
SoftMax	128	SGD	0.8	0.61 ± 0.01
SoftMax	128	SGD- Nesterov	0.1	0.66 ± 0.01
SoftMax	128	WSAGrad	-	$\mathbf{0.76 \pm 0.01}$

WSAGrad does not require a learning rate

Bold fonts represents best mean accuracy

Table 8 Architectures of ResNet-18 on CIFAR-10

Layer	$ResNet-18$	
	Output size	Layer parameters
conv-1	32×32×64	3×3,64,stride 1
conv-2	32×32×64	$\begin{bmatrix} 3\times 3,\ 64,\ stride1 \\ 3\times 3,\ 64,\ stride1 \end{bmatrix} \times 2$
conv-3	16×16×128	$\begin{bmatrix} 3\times 3,\ 128,\ stride2 \\ 3\times 3,\ 128,\ stride2 \end{bmatrix}, \begin{bmatrix} 3\times 3,\ 128,\ stride1 \\ 3\times 3,\ 128,\ stride1 \end{bmatrix}$
conv-4	8×8×256	$\begin{bmatrix} 3\times 3,\ 256,\ stride2 \\ 3\times 3,\ 256,\ stride2 \end{bmatrix}, \begin{bmatrix} 3\times 3,\ 256,\ stride1 \\ 3\times 3,\ 256,\ stride1 \end{bmatrix}$
conv-5	4×4×512	$\begin{bmatrix} 3\times 3,\ 512,\ stride2 \\ 3\times 3,\ 512,\ stride2 \end{bmatrix}, \begin{bmatrix} 3\times 3,\ 512,\ stride1 \\ 3\times 3,\ 512,\ stride1 \end{bmatrix}$
Average pooling	$1\times 1 \times 512$	4×4
Fully connected	10	512×10
Linear	10	

Table 9 Mean accuracy with standard deviation of ResNet-18 on CIFAR-10: the accuracy is averaged over multiple restarts with different batch sizes and activation types

Activation	Batch size	Method	Learning-rate	Accuracy
SoftMax	32	ADAM	0.008	0.57 ± 0.01
SoftMax	32	SGD	0.1	0.65 ± 0.01
SoftMax	32	SGD- Nesterov	0.01	0.74 ± 0.01
SoftMax	32	WSAGrad	-	$\mathbf{0.81 \pm 0.01}$
SoftMax	128	ADAM	0.008	0.70 ± 0.01
SoftMax	128	SGD	0.8	0.40 ± 0.01
SoftMax	128	SGD- Nesterov	0.8	0.75 ± 0.01
SoftMax	128	WSAGrad	-	$\mathbf{0.83 \pm 0.01}$
ReLU	32	ADAM	0.001	$\mathbf{0.86 \pm 0.01}$
ReLU	32	SGD	0.1	0.84 ± 0.01
ReLU	32	SGD- Nesterov	0.8	0.85 ± 0.01
ReLU	32	WSAGrad	-	0.82 ± 0.01
ReLU	128	ADAM	0.001	$\mathbf{0.86 \pm 0.01}$
ReLU	128	SGD	0.1	0.82 ± 0.01
ReLU	128	SGD- Nesterov	0.1	0.83 ± 0.01
ReLU	128	WSAGrad	-	0.85 ± 0.01

Bold fonts represents best mean accuracy

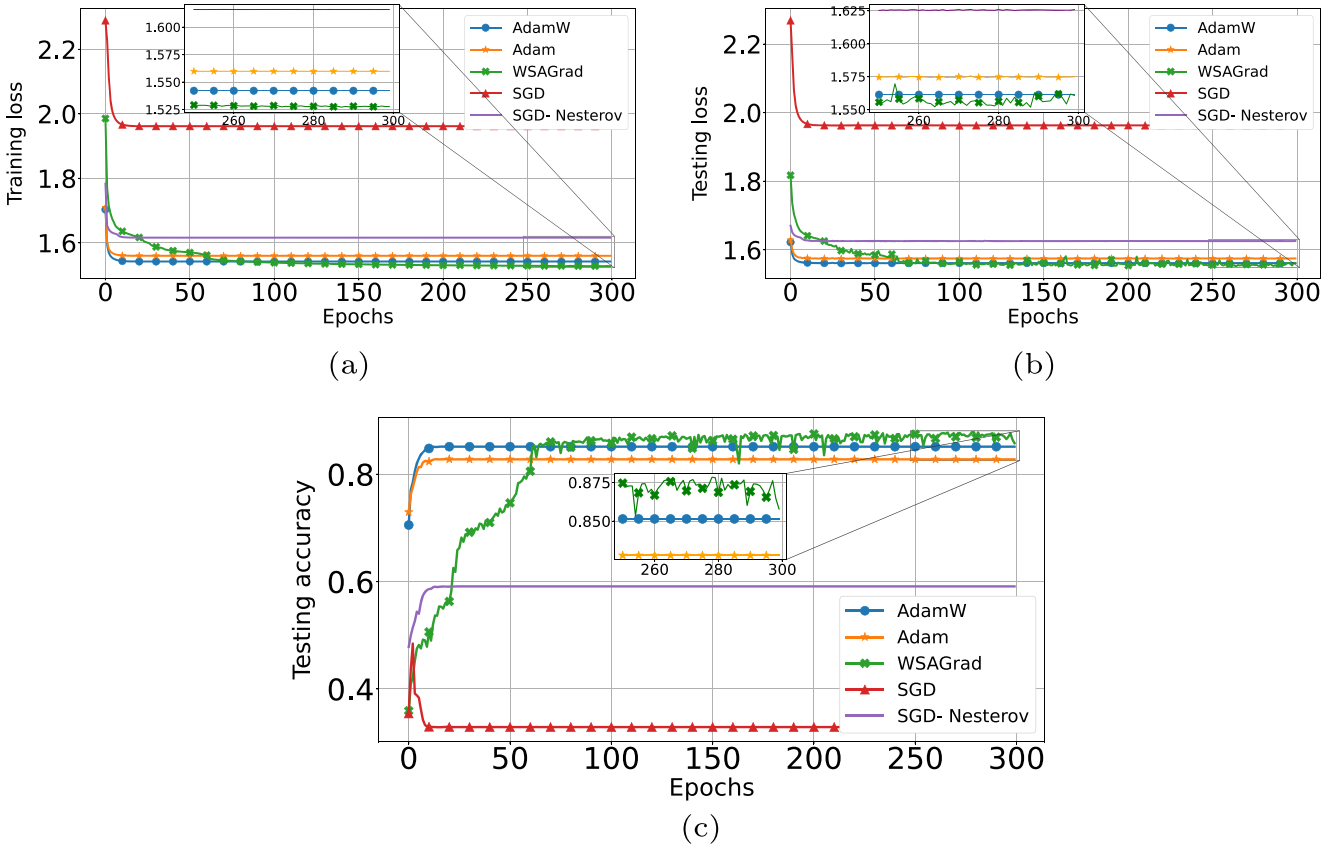
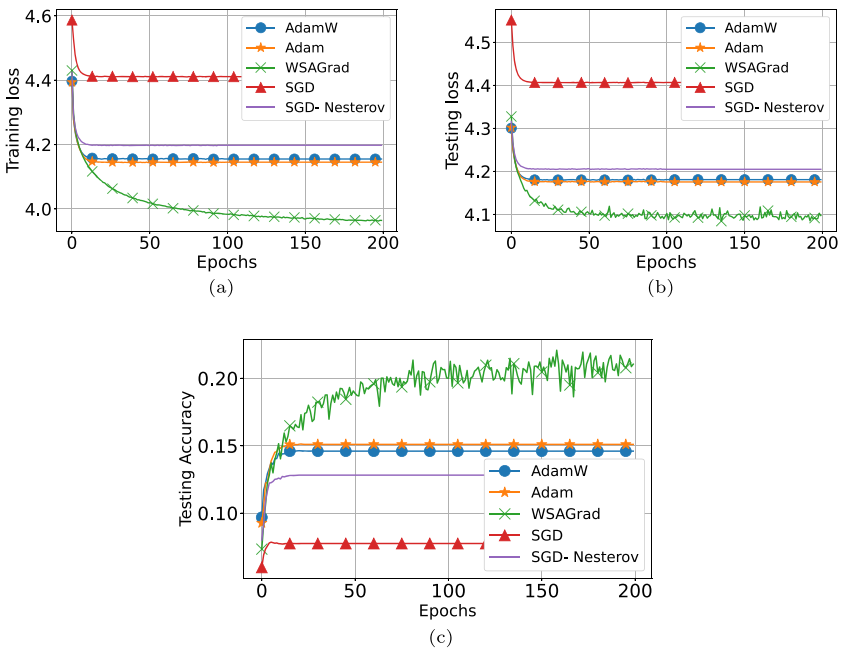


Fig. 2 Results for the FMNIST $MLP\ sigmoid$ configuration: (a) Training Loss w.r.t. epoch for baselines and $WSAGrad$. (b) Testing Loss w.r.t. epoch for baselines and $WSAGrad$. (c) Testing Accuracy w.r.t. epoch for baselines and $WSAGrad$

Fig. 3 Results for CIFAR-100 with the $MLP\ sigmoid$ configuration: (a) Training Loss w.r.t. epoch for baselines and $WSAGrad$. (b) Testing Loss w.r.t. epoch for baselines and $WSAGrad$. (c) Testing Accuracy w.r.t. epoch for baselines and $WSAGrad$



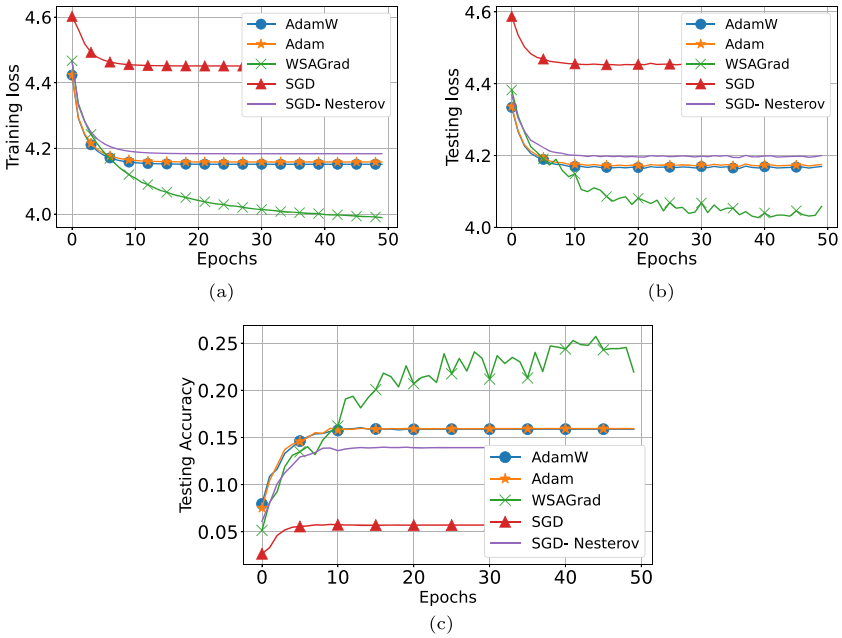


Fig. 4 Results for CIFAR-100 with the CNN $sigmoid$ configuration: (a) Training Loss w.r.t. epoch for baselines and $WSAGrad$. (b) Testing Loss w.r.t. epoch for baselines and $WSAGrad$. (c) Testing Accuracy w.r.t. epoch for baselines and $WSAGrad$

either has a higher number of variables or a lower number of instances. However, when we run $WSAGrad$ on MLP with $SoftMax$, there is a lack of a deep relation between the groups and the geometric information of the data carried by the neural network. This is not true when MLP is called with $sigmoid$, as it imbibes multiple winner structures over the neural network that carries enough geometric information for $WSAGrad$. For deep networks, we perform multiple sets of experiments with different architectures. The baselines and proposed method are tested with these architectures, and the results are reported here. For CNN, the results are stated in Tables 4 and 5. From the results, we conclude that $WSAGrad$ performs better for all activations types. The best performance of our proposed method is given under the variation of $ReLU$. Under this variation, for $CIFAR-10$ and $CIFAR-100$, the performance of $SGD-Nesterov$ is equivalent to ours, followed by SGD and $ADAM$. Apart from the best performer, we can observe that $WSAGrad$ is more robust to a model change compared to the baselines. This property makes it more reliable, as we can achieve significantly better accuracy under a basic set of models and activations. In

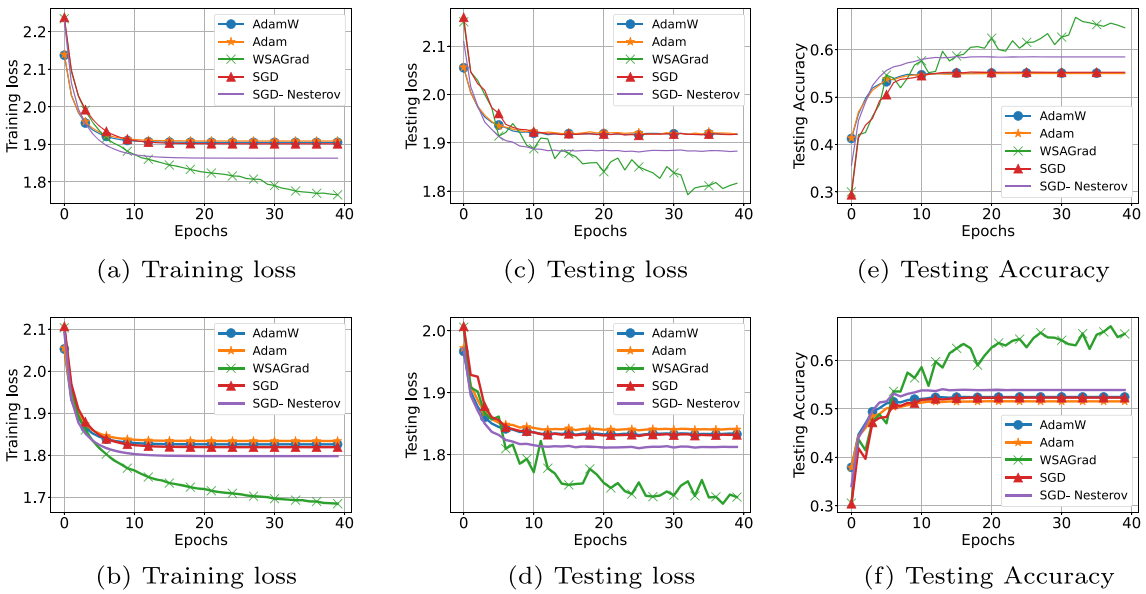


Fig. 5 Results for CIFAR-10 with CNN: the top figures show the evaluation of training loss, testing loss and testing accuracy with $SoftMax$ in the last layer on $WSAGrad$ with different baselines. The bottom figures show the evaluation of training loss, testing loss and testing accuracy with $sigmoid$ configuration on $WSAGrad$ with different baselines

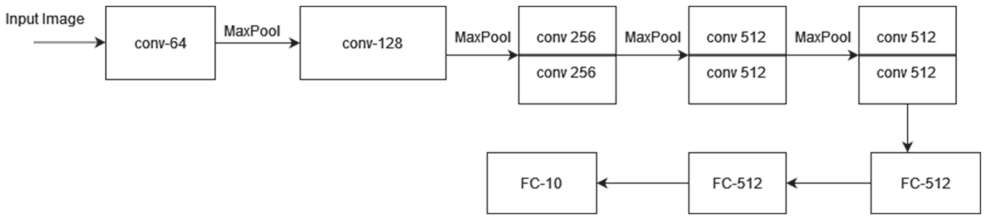


Fig. 6 Architecture of VGG-11 on CIFAR-10

other words, we can always trust $WSAGrad$ in regard to the challenging aspect of designing a better model, a loss function or a better optimizer. Figures 4 and 5 validate its ability to outperform other state-of-the-art methods. For advanced architectures such as $VGG-11$ (Fig. 6) and $ResNet-18$, we create different variations of architectures with different activations and test them on different batch sizes. We use the $VGG-11$ architecture with $SoftMax$ or a linear function as an activation function in the last layer. We can observe from Table 7 that $WSAGrad$ performs best under different variations and batch sizes. Moreover, we observe that $VGG-11$ with a small batch size is difficult to train, as it quickly leads to overfitting. For larger batch sizes, the performance depends on the activation type and hyperparameters. $WSAGrad$ is worth using for training, as the performance does not fluctuate much under different settings. The results and execution times are reported in Table 6. Figure 7 depicts our findings. It shows that the behavior of $WSAGrad$ is nonmonotonic as it oscillates within a certain range. This type of oscillation is required to find a better minimum. $WSAGrad$ forces the value of the objective function to increase so that it can reach better minima in near iterations. The minima of a loss landscape lie just below suboptimal points [56], and most of the minima are on the same level. The absence of a gradient in such a region stagnates the performance of the optimizers and leads it to a minimum that is near initialization point [57]. With $WSAGrad$, the exponential moving average of the weight parameters will not decrease quickly, even under small updates. Therefore, the value of the sine function will

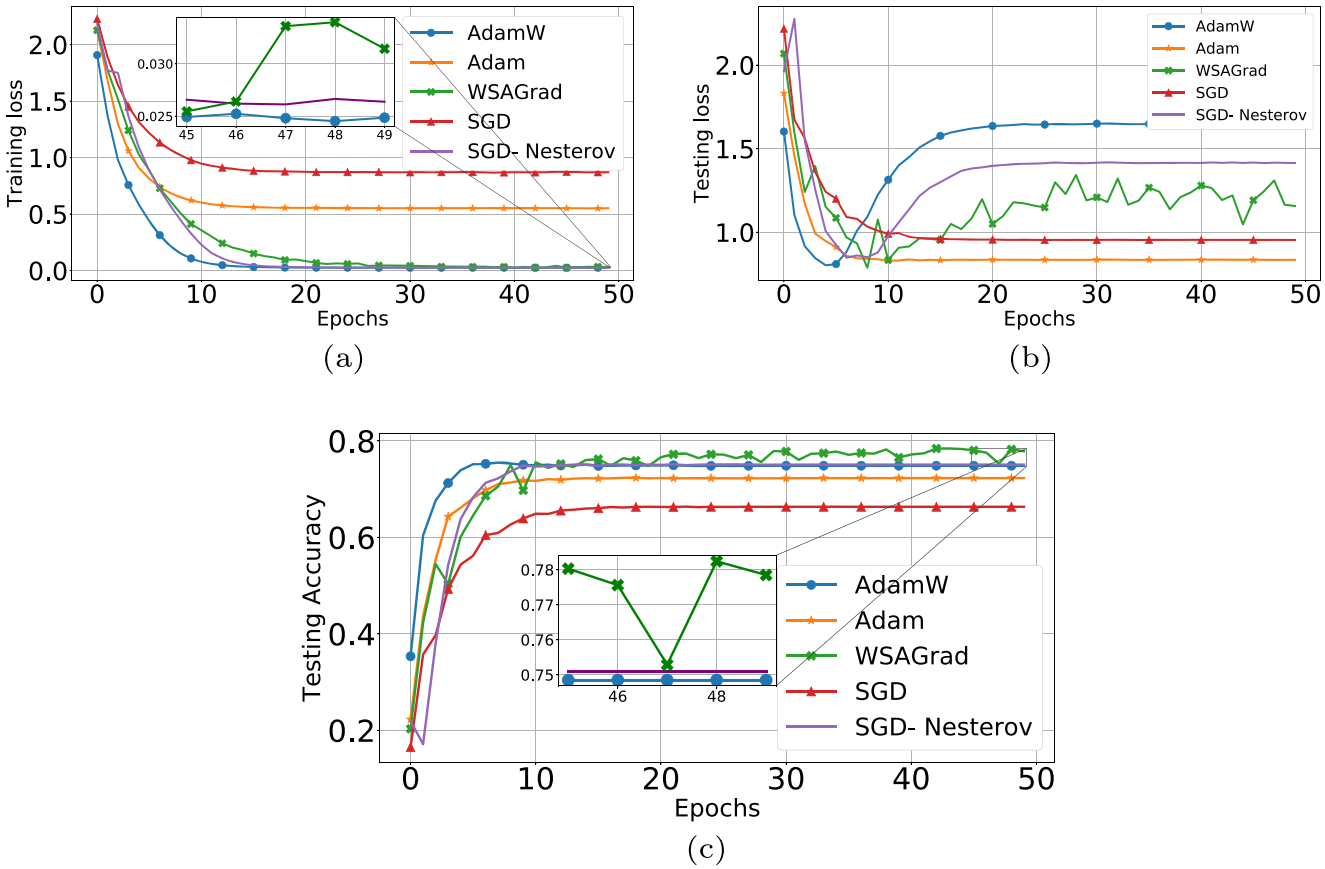


Fig. 7 Results for CIFAR-10 with VGG11: (a) Training Loss w.r.t. epoch for baselines and $WSAGrad$. (b) Testing Loss w.r.t. epoch for baselines and $WSAGrad$. (c) Testing Accuracy w.r.t. epoch for baselines and $WSAGrad$

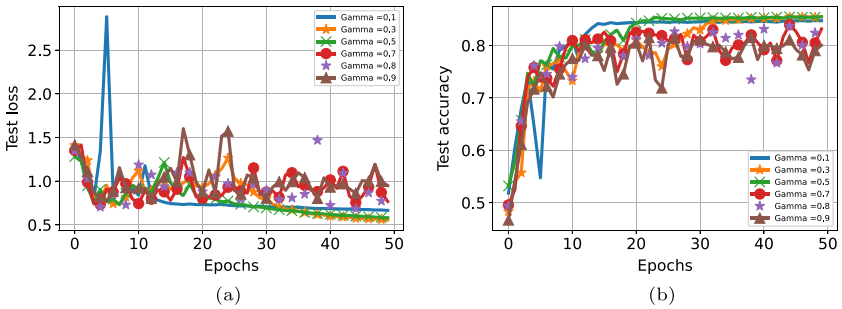


Fig. 8 Sensitivity of the hyperparameter for ResNet18: (a) Testing loss w.r.t. epoch and (b) testing accuracy w.r.t. epoch for different values of the hyperparameter

not be small, and it will maintain a moderate step size. A moderate step size increases the objective value to a certain extent such that for coming iterations, it can be decreased as needed. In addition, our step size is dependent on weight parameters, which are tuned at every iteration. Thus, with each iteration, the chance of reaching a better minimum increases. $ResNet$ is designed to handle the problems of $VGG-11$. The sensitivity of hyperparameters for ResNet-18 are tested and shown in Fig. 8, and results for different variations and batch sizes are reported in Table 9. Best results are shown in Fig. 9. We observe that $WSAGrad$ performs well when $SoftMax$ used in the final layer. When we switch to $ReLU$, $ADAM$ dominates over $WSAGrad$. From Fig. 9 we noted its nonmonotone behavior.

Overall, we observe that $WSAGrad$ outperforms the other algorithms in the classification task on different sets of architectures with varying batch sizes and activation types. $SGD-Nesterov$ performs better or equivalent to $ADAM$ for $FMNIST$ and $CIFAR-10$. Additionally, we emphasize that the test accuracy across the models and dataset for all the algorithms is relatively low compared to the accuracy achieved through different architectures. We restrict ourselves from using any other methods that can smooth out the loss landscape and undermine the

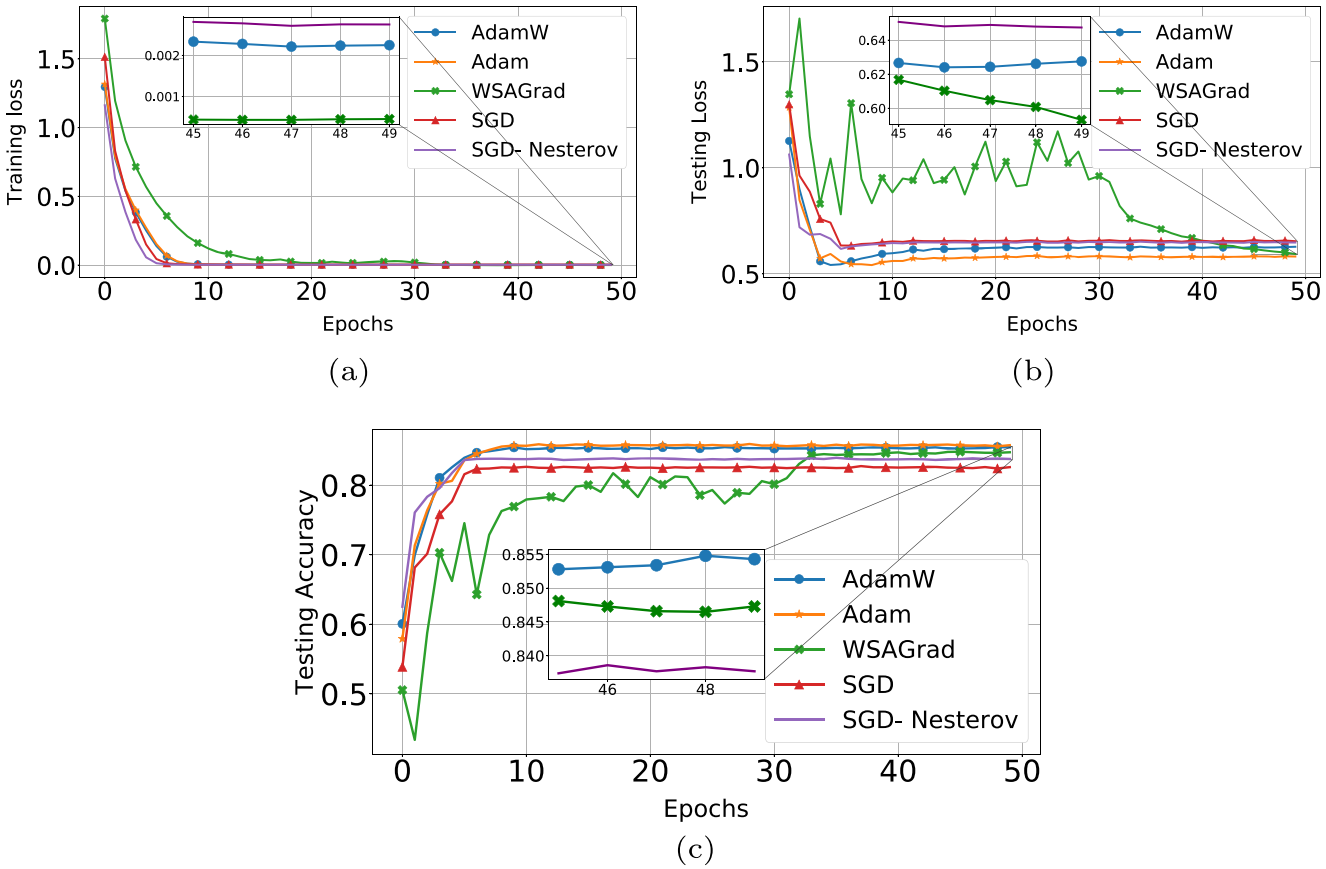


Fig. 9 Results for CIFAR-10 with ResNet18: (a) Training Loss w.r.t. epoch for baselines and $WSAGrad$. (b) Testing Loss w.r.t. epoch for baselines and $WSAGrad$. (c) Testing Accuracy w.r.t. epoch for baselines and $WSAGrad$

algorithms' actual performance. However, we still assert that $WSAGrad$ outperforms under proper initialization of the parameters. On $CIFAR-10$, the accuracy achieved by $WSAGrad$ is better than all the baselines under the settings of different activation types.

5 Conclusion and future work

The present work introduces a novel step size to operate on the vanishing gradient problem under extreme nonconvexity, to train deep neural networks. The experimental findings under various conditions show that the proposed model is considerably better than the existing baselines for the classification task. Even for a simple convex problem, we have shown its preeminence. The beauty of our algorithm lies in its relationship with the geometry of the data and loss function. The proposed step size carries the geometric information and maps it as required, nullifying the divergence scope. The effectiveness of the proposed model can be observed, with an overall average performance gain of $3-4\%$ achieved on the standard metric (Accuracy) for the classification task. With respect to sigmoid activation the average performance gain of $WSAGrad$ for the basic architecture i.e., MLP and CNN is nearly $5-6\%$. For advanced architectures such as VGG and $ResNet$, we obtained an average performance gain of $1-2\%$ for SoftMax activation. Nevertheless, the proposed model is expected to perform well on similar objective tasks such as language modeling, neural translation, image reconstruction and many other applications in various domains. We plan to conduct the experiments on many diversified datasets in those domains as our future work. Additionally, it is essential to rigorously understand the behavior and be aware of potential pitfalls while using these methods in practice. We believe this paper is the first step in this direction and suggests a good design for faster and better optimization.

References

1. Punn NS, Agarwal S (2021) Automated diagnosis of covid-19 with limited posteroanterior chest x-ray images using fine-tuned deep neural networks. Appl Intell 51(5):2689–2702
2. Gao J, Murphey YL, Zhu H (2018) Multivariate time series prediction of lane changing behavior using deep neural network. Appl Intell 48(10):3523–3537
3. Mukherjee H, Ghosh S, Dhar A, Obaidullah SM, Santosh K, Roy K (2021) Deep neural network to detect covid-19: one architecture for both ct scans and chest x-rays. Appl Intell 51(5):2777–2789
4. Smith ML, Smith LN, Hansen MF (2021) The quiet revolution in machine vision-a state-of-the-art survey paper, including historical review, perspectives, and future directions. Comput Ind 103472:130
5. Lopez A (2008) Statistical machine translation. ACM Comput Surveys (CSUR) 40(3):1–49
6. Brusa E, Delprete C, Di Maggio LG (2021) Deep transfer learning for machine diagnosis: from sound and music recognition to bearing fault detection. Appl Sci 11(24):11663
7. Wang S, Jin S, Bai D, Fan Y, Shi H, Fernandez C (2021) A critical review of improved deep learning methods for the remaining useful life prediction of lithium-ion batteries. Energy Rep 7:5562–5574
8. Wang S, Takyi-Aninakwa P, Jin S, Yu C, Fernandez C, Stroe D-I (2022) An improved feedforward-long short-term memory modeling method for the whole-life-cycle state of charge prediction of lithium-ion batteries considering current-voltage-temperature variation. Energy:124224
9. Livni R, Shalev-Shwartz S, Shamir O (2014) On the computational efficiency of training neural networks. Advances Neural Inf Process Syst, vol 27
10. Mhaskar H, Liao Q, Poggio T (2017) When and why are deep networks better than shallow ones? In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 31
11. Blum AL, Rivest RL (1992) Training a 3-node neural network is np-complete. Neural Netw 5(1):117–127
12. Polyak BT (1964) Some methods of speeding up the convergence of iteration methods. USSR Comput Math Math Phys 4(5):1–17
13. Nesterov Y (1983) A method for unconstrained convex minimization problem with the rate of convergence o $(1/k^2)$. In: Doklady an Ussr, vol 269, pp 543–547
14. Yuan W, Hu F, Lu L (2021) A new non-adaptive optimization method: stochastic gradient descent with momentum and difference. Appl Intell:1–15
15. Ma J, Yarats D (2018) Quasi-hyperbolic momentum and adam for deep learning. In: International conference on learning representations
16. Attia A, Koren T (2021) Algorithmic instabilities of accelerated gradient descent. Adv Neural Inf Process Syst:34
17. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learning Res, vol 12(7)
18. Reddi SJ, Kale S, Kumar S (2019) On the convergence of adam and beyond preprint at arXive:1904.09237
19. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw Mach Learning 4(2):26–31
20. Zeiler MD (2012) Adadelta: an adaptive learning rate method preprint at arXive:1212.5701
21. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization, arXive:1412.6980
22. Wilson AC, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning. Adv Neural Inf Process Syst, vol 30
23. Shazeer N, Stern M (2018) Adafactor: adaptive learning rates with sublinear memory cost. In: International conference on machine learning. PMLR, pp 4596–4604
24. Luo L, Xiong Y, Liu Y, Sun X (2019) Adaptive gradient methods with dynamic bound of learning rate preprint at, 1902.09843
25. Reddi S, Zaheer M, Sachan D, Kale S, Kumar S (2018) Adaptive methods for nonconvex optimization. In: Proceeding of 32nd conference on neural information processing systems (NIPS 2018)
26. Nenavath H, Jatoth RK, Das S (2018) A synergy of the sine-cosine algorithm and particle swarm optimizer for improved global optimization and object tracking. Swarm Evol Comput 43:1–30
27. Chen K, Zhou F, Yin L, Wang S, Wang Y, Wan F (2018) A hybrid particle swarm optimizer with sine cosine acceleration coefficients. Inform Sci 422:218–241

28. Gupta S, Deep K, Moayedi H, Foong LK, Assad A (2021) Sine cosine grey wolf optimizer to solve engineering design problems. Eng Comput 37(4):3123–3149
29. Giryes R, Sapiro G, Bronstein AM (2016) Deep neural networks with random gaussian weights: a universal classification strategy? IEEE Trans Signal Process 64(13):3444–3457
30. McMahan HB, Streeter M (2010) Adaptive bound optimization for online convex optimization preprint at, arXive:1002.4908
31. Dozat T (2016) Incorporating Nesterov momentum into Adam. In: Proc. Workshop Track (ICLR), pp 1–4
32. Rubio DM (2017) Convergence analysis of an adaptive method of gradient descent. University of Oxford, Oxford. M. Sc. thesis
33. Bock S, Goppold J, Weiß M (2018) An improvement of the convergence proof of the adam-optimizer. TAGUNGSBAND, pp 80
34. Keskar NS, Socher R (2017) Improving generalization performance by switching from adam to sgd preprint at arXive:1712.07628
35. Chen X, Liu S, Sun R, Hong M (2019) On the convergence of a class of adam-type algorithms for non-convex optimization. In: 7th international conference on learning representations, ICLR 2019
36. Barakat A, Bianchi P (2021) Convergence and dynamical behavior of the adam algorithm for nonconvex stochastic optimization. SIAM J Optim 31(1):244–274
37. Loshchilov I, Hutter F (2017) Decoupled weight decay regularization, arXive:1711.05101
38. Huang H, Wang C, Dong B (2019) Nostalgic adam: weighting more of the past gradients when designing the adaptive learning rate. In: Proceedings of the 28th international joint conference on artificial intelligence, pp 2556–2562
39. Chen J, Zhou D, Tang Y, Yang Z, Cao Y, Gu Q (2020) Closing the generalization gap of adaptive gradient methods in training deep neural networks. In: IJCAI
40. De S, Mukherjee A, Ullah E (2018) Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration, arXiv:1807.06766
41. Shi N, Li D (2021) Rmsprop converges with proper hyperparameter. In: International conference on learning representation
42. Zhuang J, Tang T, Ding Y, Tatikonda SC, Dvornek N, Papademetris X, Duncan J (2020) Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. Adv Neural Inf Process Syst 33:18795–18806
43. Cutkosky A, Orabona F (2019) Momentum-based variance reduction in non-convex sgd. Adv Neural Inf Process Syst, vol 32
44. Huang F, Li J, Huang H (2021) Super-adam: faster and universal framework of adaptive gradients. Adv Neural Inf Process Syst, vol 34
45. Wang B, Nguyen T, Sun T, Bertozzi AL, Baraniuk RG, Osher SJ (2022) Scheduled restart momentum for accelerated stochastic gradient descent. SIAM J Imaging Sci 15(2):738–761
46. Hafshejani SF, Gaur D, Hossain S, Benkoczi R (2021) Barzilai and borwein conjugate gradient method equipped with a non-monotone line search technique and its application on non-negative matrix factorization
47. Cutkosky A, Mehta H (2020) Momentum improves normalized sgd. In: International Conference on Machine Learning. PMLR, pp 2260–2268
48. Li X, Orabona F (2019) On the convergence of stochastic gradient descent with adaptive stepsizes. In: The 22nd international conference on artificial intelligence and statistics. PMLR, pp 983–992
49. Ward R, Wu X, Bottou L (2019) Adagrad stepsizes: sharp convergence over nonconvex landscapes. In: International conference on machine learning. PMLR, pp 6677–6686
50. Polansky AM (2011) Introduction to Statistical Limit Theory. Chapman and Hall/CRC, New York
51. Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms
52. Torralba A, Fergus R, Freeman WT (2008) 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE Trans Pattern Anal Mach Intell 30(11):1958–1970
53. Krizhevsky A, Hinton G. et al (2009) Learning multiple layers of features from tiny images
54. Simonyan K, Zisserman A (2018) Very deep convolutional networks for large-scale image recognition karen. Am J Health Pharm 75:398–406
55. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
56. Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y (2014) Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. Adv Neural Inf Process Syst, vol 27
57. Gupta C, Balakrishnan S, Ramdas A (2021) Path length bounds for gradient descent and flow. J Mach Learn Res 22(68):1–63

Krutika Verma received her M.Tech in Information Technology (with specialization in Banking Technology and Information Security), from Central University of Hyderabad in collaboration with IDRBT, Hyderabad, India in 2017. She is currently a Ph.D Scholar in the Department of Computer Science and Engineering at IIT Patna, India. Her current research interests include optimization, deep learning, machine learning, and algorithms.

Abyayananda Maiti received his Ph.D in Computer Science from University of Southern Denmark and MS in Computer Science and Engineering from IIT Kharagpur, India. Currently he is an Assistant Professor with the Department of Computer Science and Engineering at IIT Patna, India. Before joining IIT Patna, he was a Research Scientist at Essex Lake Group, Gurgaon, India. His research interests include Online Algorithms, Optimization, and Complex Networks.