AMAdam : Improved Adaptive Momentum Method

Hichame KABIRI[1*], Youssef GHANOU[1*], Hamid KHALIFI[2†] and Gabriella CASALINO[3†]

[1*]Laboratory IA, Moulay Ismail University of Meknes, Morocco.
[2]Department of Computer Science, Faculty of Sciences, Mohammed V University in Rabat, Morocco.
[3]Department of Computer Science, University of Bari, Italy.

*Corresponding author(s). E-mail(s): hichamme@outlook.fr; y.ghanou@umi.ac.ma;
Contributing authors: h.khalifi@gmail.com; gabriella.casalino@uniba.it;
†These authors contributed equally to this work.

Abstract

This paper presents a new gradient descent optimization method, AMAdam, designed to address the critical limitations of existing optimization techniques. AMAdam's innovation lies in its ability to dynamically adjust the learning rate by exploiting delicate gradient variations. This dynamic adaptation improves convergence speed while guaranteeing robust generalization, a balance that has proved difficult to achieve for many conventional algorithms. A key feature of AMAdam is its remarkable reduction in memory usage and hyperparameter complexity, compared to established methods. A comprehensive convergence analysis is carried out to establish the reliability of the algorithm. To empirically evaluate AMAdam, extensive experiments are carried out on different datasets, including MNIST, IMDB movie reviews, CIFAR-10, and CIFAR-100 datasets. Comparisons reveal AMAdam's consistent superiority over widely adopted optimizers such as SGD(M), Adam, Adamax, RMSProp, Adagrad, AdaDelta, AdamW, and Radam. Our results demonstrate AMAdam's efficacy in optimization tasks while enhancing computational efficiency. Code is available at https://github.com/thchi/AMad.

Keywords: Stochastic Gradient Descent, Adaptive Momentum, Learning Rate, Deep Neural Networks

1 Introduction

In recent years, the use of artificial neural networks (ANNs) has increasingly grown, due to their outstanding results in tasks ranging from image recognition to natural language processing [1–7]. ANNs operate on the fundamental premise of using loss scores to fine-tune their parameters, through optimization algorithms, often referred to as optimizers, being the workhorses driving this fine-tuning process [8, 9]. Consequently, every deep learning framework offers a variety of gradient descent optimization algorithms [10–12].

However, as machine learning methods have rapidly advanced, new challenges have emerged. These include achieving robust performance in the face of limited data, dealing with non-convex optimization landscapes, and addressing the phenomenon of slow convergence in deep networks [13–16]. To tackle these challenges, numerous gradient descent optimization methods have been devised, each striking a unique balance between computational efficiency, convergence speed, and parameter update characteristics [17–20]. Choosing the right optimizer hinges on the specific dataset and learning task at hand [21].

This paper introduces an innovative optimizer, Absolute Momentum Adam (AMAdam), a variant of the widely-used Adam algorithm. AMAdam addresses the sensitivity of Adam's convergence behavior to the choice of its second moment estimate, which plays a crucial role in learning rate adaptation for each parameter [22, 23]. Our solution replaces the second moment estimate with the absolute value of the first moment estimate, effectively moderating the learning rate throughout training. Through extensive experimentation across diverse machine learning tasks and datasets, including image classification and text categorization, we empirically demonstrate that AMAdam consistently outperforms not only the Adam algorithm but also other prominent optimization methods such as SGD, Adam, AdamW, and Radam [24, 25].

To facilitate a more nuanced understanding of these optimization methods, including AMAdam, and their inherent trade-offs, we present Table 1 2.4. This table acts as a compass for making informed decisions about the optimization strategy best suited to specific machine learning tasks, showcasing the respective strengths and weaknesses of each method.

Our contributions to this work can be summarized as follows:

- **Innovative Variant**: We introduce a novel variant of the Adam algorithm by replacing the second moment estimate with the absolute value of the first moment estimate.

- **Hyperparameter Reduction**: We streamline the hyperparameters, making the optimization process more efficient for various models.

- **Improved Convergence**: We enhance the convergence properties and generalization performance across a range of machine learning tasks.

- **Empirical Validation**: We provide a thorough experimental evaluation, substantiating our approach's superiority over several popular optimization methods.

The rest of this paper is organized as follows: Section 2 provides an overview of the existing literature on the Adam algorithm and related optimization techniques. In Section 3, we detail our proposed AMAdam algorithm and its distinctive features. Section 4 presents the experimental results, illustrating the effectiveness of our approach. Section 5 delves into the implications of our work within the broader context of optimization methods in machine learning. Finally, Section 6 concludes our paper and outlines avenues for future research.

2 Related work

Notation: For all vector $a, b \in R^d$, with slight abuse of notation we denote a^2 as the element-wise square, the element-wise division as $\frac{a}{b}$, max(a, b) as the element-wise maximum, abs (a) as the element-wise absolute value, and we use $\sqrt{a}$ or $a^{1/2}$ for element-wise square root. For the weight vector $\theta_t \in R^d$ we denote $\theta_{t,i}$ its i-th coordinate where $i \in [d]$. The loss function is denoted as $f_t(\theta)$ (to be considered as the loss of the model with the values chosen in mini batch), and the gradient of the loss function $\nabla_\theta f(\theta_t) = g_t$, we denote $g_{t,i}$ as the i-th coordinate of gradient obtained in the time step t with $t \in [T]$.

In this section, we present an overview of different types of gradient descent algorithms used in machine learning problems.

2.1 Stochastic Gradient Descent with Momentum (SGD(M))

Stochastic Gradient Descent method is the first and most used optimization method that works by iteratively modifying the variables in the reverse direction of the gradients of the objective function. The traditional Stochastic Gradient Descent (SGD) method can get stuck in a local minimum, as it has difficulty traversing valleys with steep curves in one direction. Momentum (M) is a technique that helps SGD to overcome this limitation by adding a fraction γ of the previous update vector m_{t-1} to the current gradient update m_t. The learning rate α determines the step size in each iteration, while γ controls the momentum [26]. The parameters of the SGD(M) method are updated as follows:

$$m_t = \gamma m_{t-1} + g_t$$
$$\theta_t = \theta_{t-1} - \alpha\ m_t$$

2.2 Accelerated Nesterov Gradient

The main purpose of the Nesterov Accelerated Gradient (NAG) method [27] is to obtain a kind of projection into the future by computing a gradient that is not relative to the current parameters. The key update is described as follows:

$$m_t = \gamma m_{t-1} + \alpha \cdot \nabla_\theta f_t(\theta_{t-1} - \gamma m_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

2.3 Adagrad

In addition, Adagrad [26] is a gradient optimization approach that adjusts the learning rate by the sum of the squares of the gradients of each parameter, resulting in smaller fits for values associated with recurrent circumstances and larger fits for values with unusual features. As a result, it is best suited for handling small amounts of data. According to [28], Adagrad improves the robustness of SGD and has been used to train Google's large-scale neural networks, which have explored how to detect cats in YouTube videos. This update can be described as:

$$g_{t,i} = \nabla_\theta f_t(\theta_{t-1,i})$$
$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{G_{t,i}} + \epsilon}.g_{t,i}$$

$G_{t,i} = \sum_{i=0}^{i=t} g_{t,i}^2$: The sum squares of the past gradients and ϵ is a smoothing concept that prevents division by zero . By using a matrix-vector product $\odot$, we could vectorize the method. The resulting update is as follows :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \odot g_t$$

G_t is diagonal matrix in which each diagonal element represents the sum of the squares of gradient g_t.

2.4 Adam

The adaptive momentum estimation (Adam) method is widely used for deep learning. It is distinguished by its approach of determining adaptive learning rates for each of the parameters. To do this, it stores the quadratic gradient priors as exponentially decreasing averages, which is similar to the approaches of RMSprop [29] and Adadelta [30]. Parameter updates are performed according to the formula described below. In its original formulation, Adam employs default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$, along with a small ϵ value of 10^{-8}. The authors showed that Adam outperforms other adaptive learning algorithms in practice.

$$
\begin{aligned}
g_t &= \nabla_\theta f_t(\theta_{t-1}) \\
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

Moreover, as shown in the work of Reddi [35], Adam may not converge to an optimal solution even in cases of simple one-dimensional convex configurations. These examples of non-convergence contradict the convergence claim of Kingma and Ba [31], and they raise the crucial issue of the quantity $\Omega_t = \frac{v_t}{\alpha_t} - \frac{v_{t-1}}{\alpha_{t-1}}$ which must be positive to guarantee convergence. Whereas for Amsgrad method, the second momentum is

Table 1 Overview of the key optimization methods.

Method	Advantages	Shortcomings
AMAdam	- Improved generalization - Reduced hyperparameters - Faster convergence - Robust performance	- Sensitive to hyperparameters - May not universally outperform other methods - Risk of overshooting in non-convex landscapes
Adam [31]	- Effective for diverse tasks - Adaptive learning rates - Converges quickly	- Non-monotonic convergence - Sensitivity to hyperparameters
AdamW [32]	- Weight decay regularization - Mitigates weight updates	- Requires additional hyperparameter tuning
Adagrad [26]	- Adaptive learning rates based on past gradients - Effective for sparse data	- Rapid decay of learning rates - Inefficient on non-sparse data
AdaDelta [30]	- No learning rate hyperparameter - Effective on some non-convex landscapes	- Slow convergence in some scenarios - Sensitivity to learning rate parameters
Radam [33]	- Adaptive learning rates based on moving averages of gradients	- May not consistently outperform other methods
NAG [15]	- Simplicity and low memory usage - Can escape local minima	- Slow convergence - Susceptible to local minima
AdaBelieve [34]	- Improved generalization - Adaptive learning rates	- Prone to overfitting - Computationally expensive for larger networks
Adamax [31]	- Effective for noisy gradients - Simpler than Adam	- Robustness issues in extreme gradient scenarios
RMSProp [29]	- Adaptive learning rates with gradient moving averages	- Non-monotonic convergence - Oscillation around minima

computed as the maximum of the past square gradients, using an additional variable $\hat{v}$ to store these maximums, which allows for more efficient updating of weights and subsequently guarantees that the quantity Ω_t is positive. On the other hand, the Adadelta and RMSprop approaches also have their drawbacks, such as the potential looping update around a late local minimum in the learning phase [36].

Table 2.4 summarises the adopted algorithms by highlighting their advantages and shortcomings, together with the proposed new optimization algorithm, detailed in the following paragraph.

3 Improved Adaptive Momentum Method - AMAdam

We propose a new optimization algorithm called AMAdam that aims to improve the convergence properties of the classical Adam algorithm and addresses some problems with current optimization methods. Our main idea is to replace the second moment estimate used to update the parameters with the absolute value of the first moment estimate[37].

Our Stochastic Gradient Descent optimization method employs an adaptive decay rate rather than a fixed and pre-configured one. The suggested optimization strategy is based on changes in simple gradients and manages the learning rate based on the need for effective learning adjustment mechanism [38]. This suggests that our developed algorithm follows the rule that the parameter change ought to be smaller in low gradient changing regions and larger in high gradient changing regions. The method significantly reduces the number of hyper-parameters that needed to be modified and adjusted when necessary [39].

The key steps in our algorithm are as follows:

1. Compute the first momentum estimate m_t as it is in Adam algorithm .
2. Use the absolute estimate of m_t to regulate the rate at which the parameters are changed: we compute the absolute estimate of m_t using the absolute function, as shown below:
$$v_t = abs(m_t) \tag{1}$$
3. Re-initialize $\hat{v}$: to avoid problems due to excessive decay, we use re-initialization of $\hat{v}$ to maintain some stability in updating the parameters. Moreover we guarantee that the quantity
$$\Omega_t = \frac{v_t}{\alpha_t} - \frac{v_{t-1}}{\alpha_{t-1}}$$
is positive to ensure convergence as mentioned in [40] . This is accomplished by using the max function to compare $\hat{v}$ to v_t, as shown below:
$$\hat{v_t} = max(\hat{v}_{t-1}, v_t) \tag{2}$$
With this update. The iteration using the stochastic gradient descent method is done as follows:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)} \tag{3}$$

Algorithm 1 summarises the requirements and the adopted steps.

Algorithm 1 Proposed Algorithm - AMAdam.

Require: $\alpha = 0.01$, $\beta_1 = 0.9$, and $\epsilon = 10^{-8}$. With β_1^t, denote β_1 to the power t.

$m_0 = 0$ and $v_0 = 0$
$t = 0$
while θ_t not converged **do**
    $t = t + 1$
    $g_t = \nabla_\theta f_t(\theta_{t-1})$
    $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $\hat{m}_t = \frac{m_t}{(1-\beta_1^t)}$
    $v_t = abs(\hat{m}_t)$
    $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$
    $\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$
end while
return θ_t

3.1 Mathematics intuition and Motivation behind the modification

To understand our modification, it is important to note that optimization algorithms such as Adam use two types of estimates to update the parameters: the first is the estimate of the first moment m_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t. \tag{4}$$

This gives an idea of the direction in which the parameters should be changed, and the second estimate of the moment, v_t, regulates the rate at which the parameters are changed. In the classical Adam algorithm, v_t is calculated as the exponential moving average of the squares of the gradients.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{5}$$

In the Adam's original algorithm, the rule for updating the θ parameters is as follows:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t} + \epsilon} m_t. \tag{6}$$

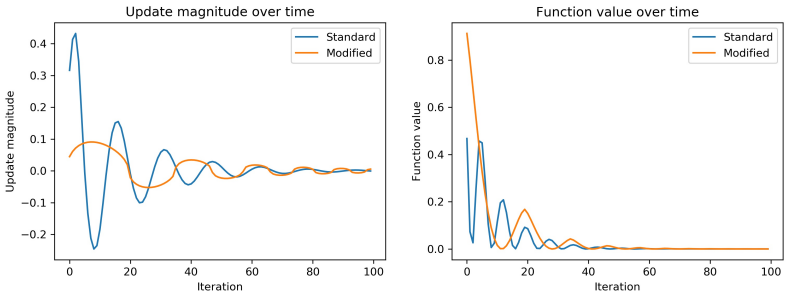

Fig. 1 Comparison the effect of update magnitude and the value of the function over time.

The proposed modification of the update rule is motivated by the need of preventing the value of v_t to be excessively small, which can cause instability in the algorithm. This phenomenon can occur when the gradient is consistently small, causing the value of v_t to approach zero. As a result, the scaling factor $\frac{1}{\sqrt{v_t}+\epsilon}$ in the update rule becomes very large, potentially leading to overshooting of the optimum solution. This overshooting can result in slow convergence or even divergence of the algorithm. Therefore, by using the absolute value of m_t instead of v_t in the denominator of the update rule, we ensure that the denominator is always positive, and the stability of the update rule is maintained. The updated equation then becomes:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{abs(m_t)} + \epsilon} m_t. \tag{7}$$

This modification does not change the direction of the update, but it alters the magnitude of the update by including the absolute value of m_t in the denominator of the update rule. Specifically, this modification changes the magnitude of the update and leads to faster optimization and better convergence properties.

(ie) In the case of the Adam optimization algorithm, the magnitude of the update is controlled by the denominator of the update rule, which involves the exponential moving average of the squares of the gradients (v_t) and the epsilon value for numerical stability. The larger the denominator value, the smaller the magnitude of the parameter update. By changing the denominator to include the absolute value of m_t rather than v_t, the magnitude of the update is changed, which leads to better convergence properties and faster optimization.

To further illustrate the effect of the modification on the magnitude of the updates, let's consider a simple example. We can compare the effect of the standard Adam optimizer with the modified version on a quadratic function defined by $f(x) = x^2$. Figure 1 shows the value of the function and the magnitude of the parameter updates over time for the two optimizers (Adam and AMAdam):

- The standard Adam optimizer tends to make large updates at the beginning of the optimization process and gradually reduces the magnitude of the updates as it

converges to the minimum. This can be observed in the figure where the magnitude of the parameter updates (blu line) starts high but gradually decreases over time.

- In contrast, the modified version of the Adam optimizer (orange line) starts with a small update magnitude and gradually decreases it, as it approaches the minimum. This helps to avoid overshooting the minimum and may result in faster convergence to the minimum.

In the case of the quadratic function $f(x) = x^2$, the standard Adam algorithm overshoot the minimum due to its large initial updates, while the modified optimizer converge faster due to its gradual decay in update magnitude.

In conclusion, the modified Adam algorithm changes the scaling factor of the update rule to be based on the gradient magnitude estimate $abs(m_t)$ rather than the second moment estimate of the gradient (v_t). This modification ensures that updates are always correctly adjusted for gradient magnitude, while avoiding the potential for vanishing scaling factors when the second moment estimate is close to zero.

Theorem 1. *(Convergence) Assume that the gradient* $g_t = \nabla_\theta f_t(\theta_{t-1})$ *is bounded for all* $t \in [T], ||abs(g_t)||_\infty \leq G_\infty$ *for all* $\theta \in R^d$ *and distance between any* θ_t *generated by our algorithm is bounded,* $||\theta_m - \theta_n||_\infty \leq D_\infty$ *for any n,m* $\in 1, ..., T$ *and* $0 \leq \beta_1 < 1$*.let* $\alpha_t = \frac{\alpha}{\sqrt{t}}$ *and* $\beta_{1,t} = \beta_1 \lambda^{t-1}$, $0 \leq \lambda < 1$*. Our algorithm achieves the following guarantee, for all* $T \geq 1$

$$R(T) \leq \frac{dG_\infty^{1/2} D_\infty^2}{2\alpha(1-\beta_1)}\sqrt{T} + \frac{\beta_1 d\ G_\infty^{1/2} D_\infty^2}{2\alpha(1-\beta_1)(1-\lambda)^2} + \frac{\alpha G_\infty^{3/2} d}{(1-\beta_1)}\sqrt{T}$$

A side-benefit of our work is adopting a similar demonstration approach in [23]. Eventually our analysis shows that the proposed algorithm achieves convergence rate of $O(1/\sqrt{T})$ (See Appendix A) .

3.2 Efficiency and Characteristics

In this part we discuss the main characteristics and efficiency of our proposed algorithm in relation to a set of algorithms well known in the literature for their performance and efficiency.

One key advantage of AMAdam is its ability to effectively adapt the learning rate for each parameter based on the historical gradient information. This allows AMAdam to avoid non-monotonic slowdowns in convergence and improve generalization performance. In addition, AMAdam requires less memory than other optimization algorithms that uses the gradient descent square and its derivatives, making it a more efficient and scalable option for large-scale machine learning tasks.

Table 2 Set of algorithms used in the literature and their differences in terms of several parameters such as 1^{st} Momentum, 2^{nd} Momentum, Update, and Hyper-parameter.

Algorithm	1^{st} Momentum	2^{nd} Momentum	Update	Hyper-parameter
SGD(M)	$m_t = \gamma m_{t-1} + g_t$	I	$\theta_{t+1} = \theta_t - \alpha m_t$	γ, α
Adagrad	g_t	G_t	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} g_t$	α, ϵ
AdaDelta	g_t	$E[g^2]_t$	$\theta_{t+1} = \theta_t + \Delta\theta_t$	β_1, ϵ
RMSProp	$\hat{m}_t$	$E[g^2]_t$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t$	$\alpha, \beta_1, \beta_2, \epsilon$
Adam	$\hat{m}_t$	$\hat{v}_t$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$	$\alpha, \beta_1, \beta_2, \epsilon$
Adamax	$\hat{m}_t$	u_t	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{u_t + \epsilon}} \hat{m}_t$	$\alpha, \beta_1, \beta_2, \epsilon$
Nadam	$m_t = \beta_1 \hat{m}_{t-1} + (1 - \beta_1) g_t$	$\hat{v}_t$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$	$\alpha, \beta_1, \beta_2, \epsilon$
Amsgrad	$\hat{m}_t$	$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$	$\alpha, \beta_1, \beta_2, \epsilon$
Adabelief	$\hat{m}_t$	$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(g_t - m_t)^2 + \epsilon$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$	$\alpha, \beta_1, \beta_2, \epsilon$
AdamW	$\hat{m}_t$	$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g^2$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t + \lambda\theta$	$\alpha, \beta_1, \beta_2, \lambda, \epsilon$
AMAdam	$\hat{m}_t$	$abs(\hat{m}_t)$	$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{abs(\hat{m}_t) + \epsilon}} \hat{m}_t$	$\alpha, \beta_1, \epsilon$

where: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$, $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$, $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$, $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$,
$u_t = \beta_2^\infty u_{t-1} + (1 - \beta_2^\infty) |g_t|^\infty = \max(\beta_2 \cdot u_{t-1}, |g_t|)$,
$E[g^2]_t$ is the decaying average over past squared gradients,
$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$, $G_t = \sum_{i=1}^{i=t} g_i^2$,
$\Delta\theta_t = -\frac{E[\Delta\theta]_{t-1}}{E[g]_t} g_t$.

Another main advantage of AMAdam is that it has fewer hyperparameters than Adam, which may make it easier to configure for different models. Furthermore, AMAdam is designed to reduce the rate of learning in the last iterations, which may contribute in avoiding poor generalization performance caused by a non-linear rate of learning.

As we will see in the experimental part, our algorithm at the beginning of the learning process makes iterations with very little progress compared to the other algorithms that use the second decay rate and its variants. What explains the greater progress of the other adaptive methods compared to ours is that they use the second momentum v_t (square gradient and its variants), while our algorithm just uses the first average exponential decay of the gradient with its absolute value. However, during the training, our algorithm adapts well and manages to have the same behavior as the other methods. Furthermore, it succeeds in outperforming most of the widely used algorithms.

Note that the attributes presented in Table 2 are a summary of the characteristics of each algorithm and how they relate to our proposed method, to support our claims about the effectiveness of our method.

To recapitulate, in our proposed stochastic gradient descent optimization method, we make sure that the learning rate decreases to avoid any dispersion or poor generalization performance caused by a non-monotonic slowdown of the learning rate. We then check that the learning rate decreases more and more even in the last iterations

until our optimal solution is reached. Another advantage of our strategy is that the algorithm requires less memory than other methods that use the gradient descent square and its different derivatives. In addition, we decreased the hyper-parameters in order to obtain efficiently various learning rates for different models. The proposed algorithm is represented in Table 1 (We set our default hyper-parameterization as it is introduced in the Adam and Amsgrad algorithms [31][30]).

4 Experiments

The following sections provide a detailed description of the experimental setup, including the datasets and tasks used, the model architectures, and the evaluation metrics. We then present the results of our experiments, comparing AMAdam to the standard Adam algorithm on a variety of tasks and datasets.

4.1 Small Data and Architecture

We present the datasets and network architecture used in many works in the literature. The results of our classification algorithms are then analyzed. The comparisons are conducted on a local computer with a 1.40 GHz AMD E1-2500 APU and 4 GB of Random Access Memory (RAM).

Our algorithm was implemented in python 3.6 using Tensorflow 2.x. We held our approach to the test on two datasets using different Recurrent Neural Network architectures. To minimize the Cross-Entropy loss target between network performance and labels, we train our neural networks in a supervised manner using SGD(M), Adam, Adamax, Nadam RMSProp, Adagrad,and AdaDelta. For all algorithms, we use the hyper-parameter defaults introduced in the keras library [11]. However, we used the fixed learning $\alpha = 0.01$, $\beta_1 = 0.9$, and $\epsilon = 10^{-8}$ for the hyper-parameters of our algorithm.

4.1.1 Classification of Handwritten Digits

In our first set of experiments, we use the MNIST handwritten digit classification task to train a neural network. We trained our model with 500 hidden units in the first layer, 250 hidden units in the second layer, and the final softmax output layer on top for an easy comparison with SOTA algorithms. All models were trained over 50 epochs using mini-batches of 128 frames each.

The comparison involves seven different algorithms: SGD(M), Adam, Adamax, Nadam, RMSProp, Adagrad and AdaDelta with our algorithm. Figure 2 shows different experiments in which we used 20% of the data as test set and performed real-time training on the remaining 80%. From the results in Figure 2 and Table 3, it is clear that the adaptive algorithms are clearly superior to the non-adaptive algorithm in terms of speed and accuracy. However, the closer we get to the optimal solution, the

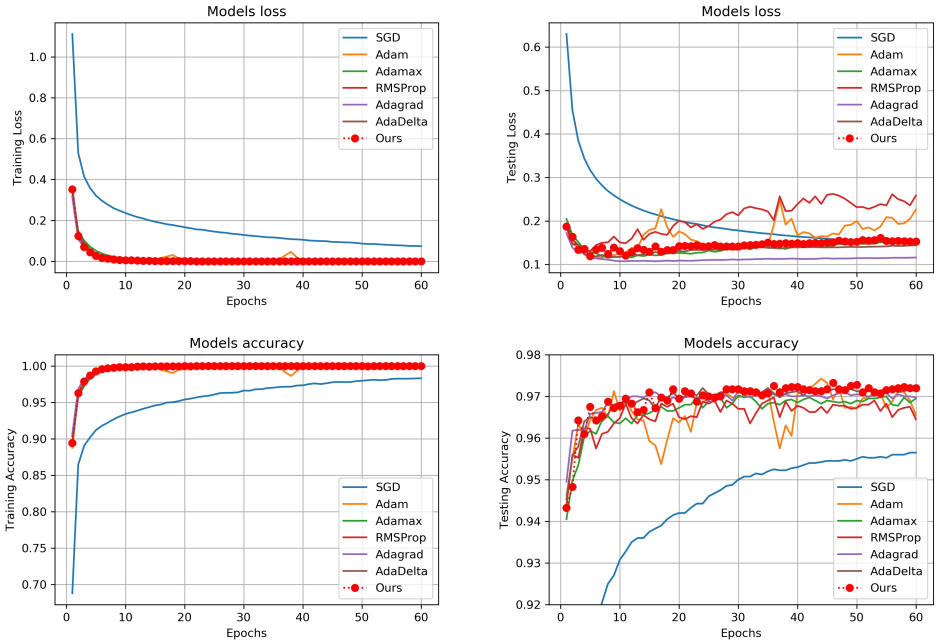

Fig. 2 Classification of MNIST Digit.

Table 3 Classification result of MNIST Dataset (Results are averaged across five repetitions of the experiment).

Algorithm	Training accuracy	Test accuracy
SGD(M)	0.9778	0.9548
Adam	0.9937	0.9702
Adamax	0.9931	0.9652
Nadam	0.9941	0.9704
RMSProp	0.9932	0.9688
Adagrad	0.9935	0.9683
AdaDelta	0.9944	0.9710
Ours	**0.9954**	**0.9731**

more robust our method becomes and does not suffer from oscillations. In the training phase, we achieved an accuracy rate of 99.54% as the highest achieved compared to all known adaptive algorithms in the Keras library. In the test part, our approach performs admirably and achieves a higher accuracy than all the established algorithms, See Table 3.

4.1.2 Classification of IMDB movie reviews

In our second set of tests, we use the IMDB movie analysis classification task to train the models. 128 hidden units were used in the first layer for training, 0.3 dropout in the second layer, and 1 unit and the sigmoid function in the final output layer.

We have matched the architecture to models that were trained using SGD, Adam, Adamax, Nadam, RMSProp, Adagrad and AdaDelta. We used 80% of data for training and 20% for testing.

As it can be seen in the experiments shown in Figure 3, our algorithm advances very slowly at the beginning of the learning process compared to adaptive algorithms using the secondary decay rate and its variants such as Adam, Adamax, Nadam, RMSProp, Adagrad and AdaDelta, but with much higher accuracy and speed than the non-adaptive SGD(M) algorithm. However, the closer we get to the optimal solution, the more stable and oscillation-free our algorithm becomes. Table 4 reports the classification results on Training and Test sets, with all the considered algorithms. It can be observed that we achieved 100% accuracy for all known adaptive algorithms in the literature in the training phase (including ours). For the tests, our algorithm performs very well compared to all previous algorithms with a very high accuracy, demonstrating its high efficiency and effectiveness [41]. Moreover, the fine-tuning of our method allows it to achieve higher levels of performance than any of the other adaptive algorithms tested, without suffering from the instability and oscillations encountered by some of them as they approach the optimal solution [42][43].
Briefly, we compared the performance of AMAdam to the standard Adam algorithm and several other popular optimization algorithms. The results of our experiments demonstrate that AMAdam consistently outperforms the standard Adam algorithm

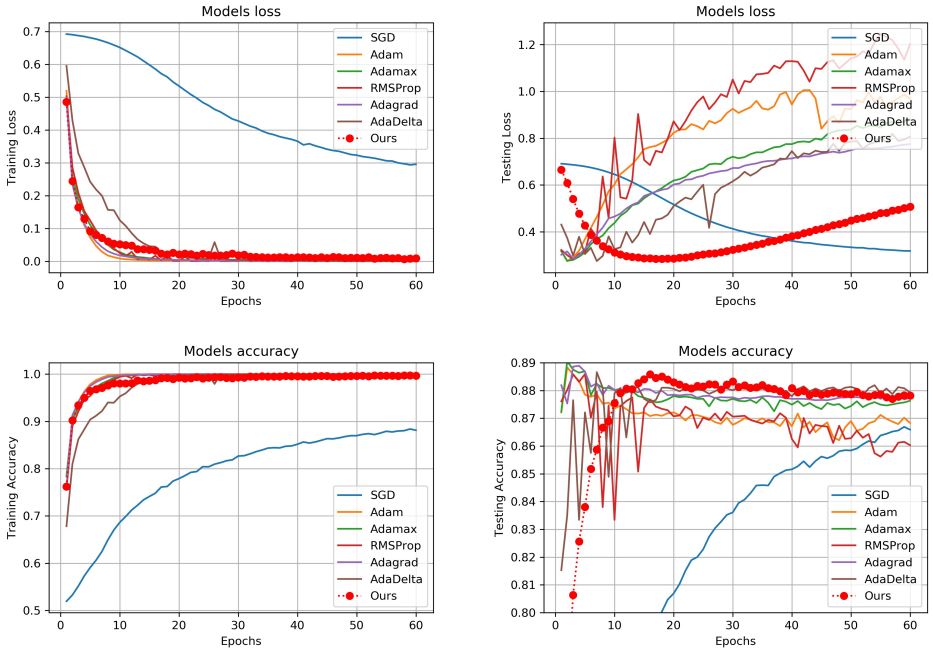

Fig. 3 Comparison on IMDB movie reviews.

Table 4 Classification result IMDB movie reviews (Results are averaged across five repetitions of the experiment).

Algorithm	Training accuracy	Test accuracy
SGD	0.8762	0.8566
Adam	1.000	0.8658
Adamax	1.000	0.8759
Nadam	1.000	0.8710
RMSProp	1.000	0.8702
Adagrad	1.000	0.8795
AdaDelta	1.000	0.8789
Ours	**1.000**	**0.8801**

and other popular optimization algorithms, achieving higher accuracy and faster convergence on a variety of tasks and datasets. In particular, AMAdam was able to achieve state-of-the-art performance on the MNIST and IMDB datasets.

4.2 Large Data and Architecture

In this part, we explore the details of our experimental setup, particularly focusing on training large-scale deep learning models on CIFAR-10 and CIFAR-100 datasets. We outline the consistent conditions under which our experiments were conducted and the strategies employed to enhance model learning capabilities, including learning rate scheduling. Furthermore, we provide a comprehensive analysis of our optimization algorithm's performance across different architectures, namely AlexNet, VGG19-bn, and ResNet-18, while comparing it with a selection of widely used optimization methods. Additionally, we explore the impact of varying learning rates on model performance, with a particular focus on our AMAdam algorithm's behavior under different learning rate regimes.

To ensure fair and unbiased evaluation in our experiments, we conducted experiments under consistent conditions. The model was trained for 250 epochs using a batch size of 128. Additionally, we incorporated a learning rate scheduling strategy, where the learning rate was multiplied by 0.1 at epoch 80 and again at epoch 160. This approach aimed to enhance the model's learning capability and enable it to adapt to the complex underlying patterns in the data [44]. We trained AlexNet, VGG19-bn, ResNet-18 models on the CIFAR-10 and CIFAR-100 datasets using 8 different optimization algorithms: Adam, Adamax, Adadelta, SGD, RMSProp, Radam, AdamW, and our proposed method. For AlexNet, we used a learning rate of 0.001. For VGG19-bn, we used a learning rate of 0.01. For ResNet-18, we used a learning rate of 0.01.

The results of our experiments provide strong evidence supporting the superiority of our proposed optimization method. When compared to other optimizers, our algorithm consistently outperformed them for both the AlexNet, VGG19-bn and ResNet-18 models. As it can be seen in Figures 4, 5, and 6, our method achieved higher accuracy at a faster rate while maintaining stability throughout the training process. These results are further reinforced by the summarized findings in Tables 5, where

Table 5 Classification results on Cifar 10 (Results are averaged across five repetitions of the experiment).

Architecture	Algorithm	Training accuracy	Test accuracy
AlexNet	SGD	0.8445	0.7127
AlexNet	AdaDelta	0.9197	0.7761
AlexNet	Adamax	0.9148	0.7588
AlexNet	RMSProp	0.9235	0.7531
AlexNet	Adagrad	0.9596	0.7623
AlexNet	Adam	0.8783	0.7638
AlexNet	AdamW	0.9713	0.7709
AlexNet	Radam	0.9268	0.7736
AlexNet	**AMAdam (Ours)**	**0.9663**	**0.7781**
VGG19-bn	SGD	0.9995	0.9307
VGG19-bn	AdaDelta	0.9997	0.9165
VGG19-bn	Adamax	0.9995	0.9279
VGG19-bn	RMSProp	0.9996	0.9276
VGG19-bn	Adagrad	0.9997	0.9201
VGG19-bn	Adam	0.9996	0.9288
VGG19-bn	AdamW	0.9996	0.9303
VGG19-bn	Radam	0.9998	0.9297
VGG19-bn	**AMAdam (Ours)**	**0.9997**	**0.9329**
ResNet-18	SGD	0.9595	0.8207
ResNet-18	AdaDelta	0.9897	0.8975
ResNet-18	Adamax	0.9196	0.8794
ResNet-18	RMSProp	0.9045	0.8475
ResNet-18	Adagrad	0.9267	0.8502
ResNet-18	Adam	0.9383	0.8538
ResNet-18	AdamW	0.9867	0.8997
ResNet-18	Radam	0.9783	0.9038
ResNet-18	**AMAdam (Ours)**	**0.9797**	**0.9042**

Table 6 Classification results on Cifar 100 (Results are averaged across three repetitions of the experiment).

Architecture	Algorithm	Training accuracy	Test accuracy
ResNet-18	SGD	0.8784	0.6846
ResNet-18	AdaDelta	0.8465	0.6392
ResNet-18	Adamax	0.7686	0.6186
ResNet-18	RMSProp	0.8416	0.6296
ResNet-18	Adagrad	0.7406	0.6097
ResNet-18	Adam	0.8417	0.6598
ResNet-18	AdamW	0.8391	0.6436
ResNet-18	Radam	0.8429	0.6312
ResNet-18	**AMAdam (Ours)**	**0.9012**	**0.6631**

our approach clearly surpassed other optimizers in terms of accuracy by a significant margin.

In addition, we carefully trained ResNet-18 on the challenging Cifar100 dataset, Figure 7 and Table 4.2, while systematically varying the learning rate. This rigorous approach enabled us to explore the impact of different learning rates on model performance. We closely monitored training and test accuracy, as well as loss, as a function

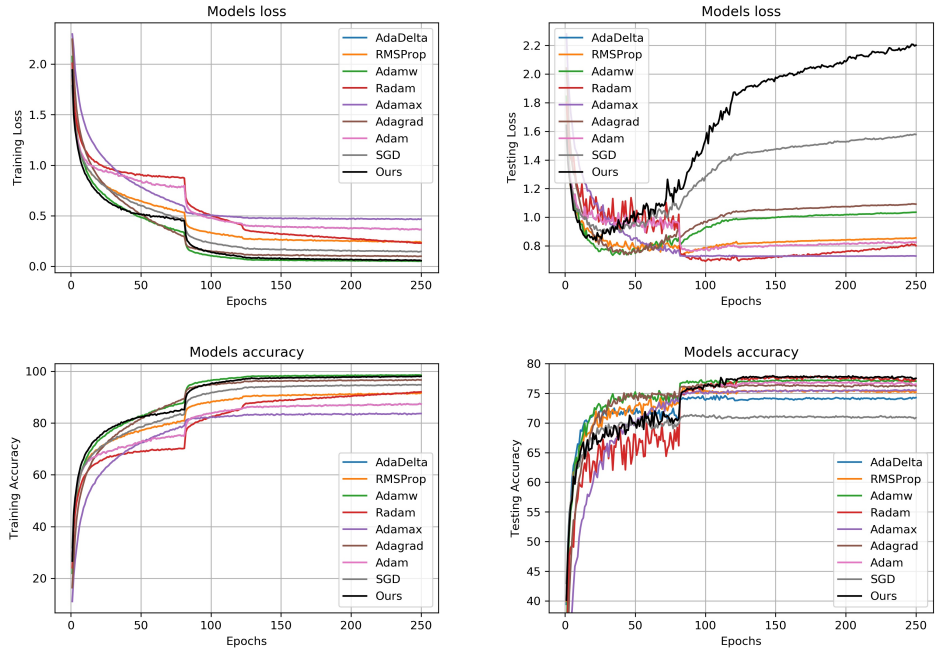

Fig. 4 Comparison on CIFAR-10 with Alexnet

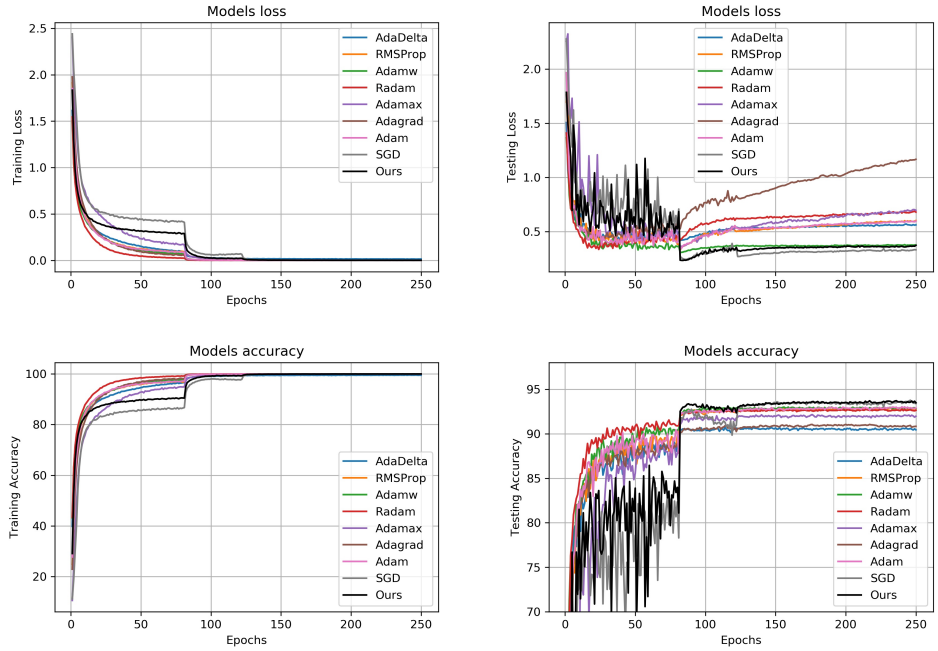

Fig. 5 Comparison on CIFAR-10 with VGG19-bn.

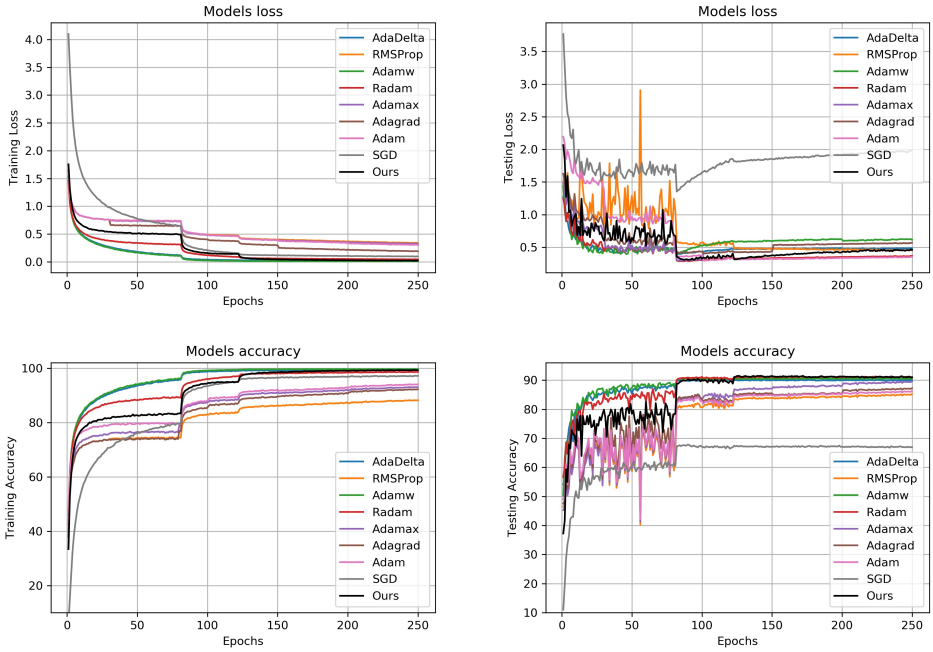

Fig. 6 Comparison on CIFAR-10 with ResNet-18.

of these different learning rates, to better understand how this crucial hyperparameter influences the network's training dynamics and its ability to generalize efficiently on the complex Cifar100 dataset. However AMadam generates higher test accuracy curve, but unfortunately it is not robust to the change of learning rate (see Figure 8).

In conclusion, our proposed optimization method shows great potential for improving deep neural network training on complex datasets such as CIFAR-10, and CIFAR-100 outperforming Adam, Adamax, Adadelta, SGD, RMSProp, AdamW, and Radam in terms of accuracy and efficiency.

5 Discussion

The proposed AMAdam algorithm undoubtedly brings advances in gradient descent optimization. However, like any methodology, it has certain limitations that deserve to be examined and discussed in depth.

Hyperparameter Sensitivity: while AMAdam reduces the number of hyperparameters compared to some traditional optimization algorithms, it still retains sensitivity to hyperparameter settings. The performance of AMAdam can be influenced by choices like learning rate, decay rates, and initialization values. Fine-tuning these hyperparameters for optimal results remains a non-trivial task and can be

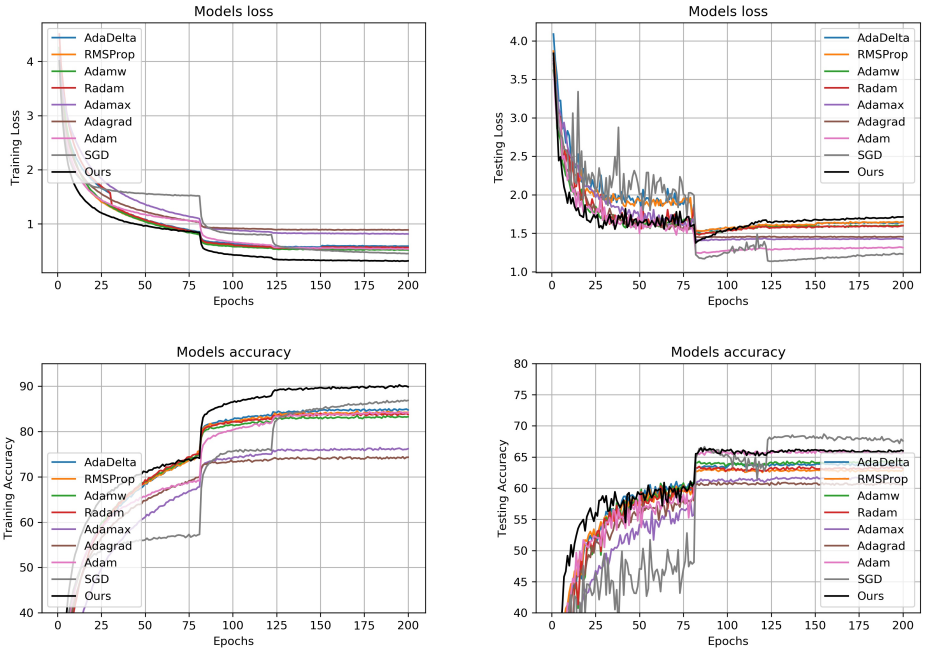

Fig. 7 Comparison on CIFAR-100 with ResNet-18.

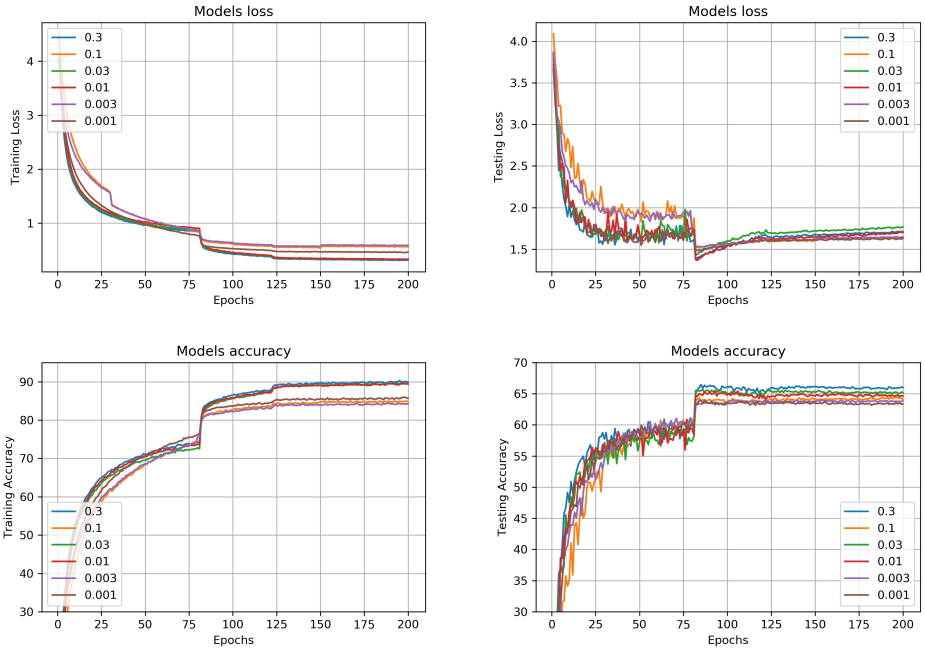

Fig. 8 Study of the impact of different learning rates on CIFAR-100 data with ResNet-18.

computationally demanding, especially in complex neural network architectures.

Limited Performance on Certain Architectures: although AMAdam demonstrates significant improvements in various scenarios, its performance might not be universally superior across all neural network architectures and datasets. Its effectiveness could potentially be contingent on factors like network depth, width, and problem complexity. Identifying the specific conditions under which AMAdam excels or underperforms requires further investigation .

Sparse Gradient Tasks: the algorithm's response to tasks characterized by sparse gradients could be an area of concern. Since AMAdam incorporates the absolute value of gradients, it might not handle sparse gradients as effectively as other optimization methods specifically designed for such scenarios. Careful exploration is necessary to gauge its performance in such contexts.

Risk of Overshooting in Non-Convex Surfaces: while AMAdam's adjustment of learning rates is designed to facilitate convergence, there exists a possibility of overshooting or oscillations in non-convex optimization landscapes. This risk becomes more pronounced in intricate loss surfaces with numerous local optima. Analyzing the algorithm's behavior in these situations and implementing countermeasures would be valuable.

Generalization Across Diverse Datasets: while the algorithm's success across image and text categorization tasks is promising, its generalizability across a broader spectrum of tasks and datasets needs validation. Extending the experimentation to encompass diverse domains, such as audio or video data, could provide a more comprehensive understanding of its applicability.

In addition, there are many other optimization algorithms that can be used in machine learning and deep learning, here are a few examples :

- Newton's Method is an optimization algorithm for finding the minimum of a function using the first and second derivatives of the function.
- L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) is a quasi-Newton method that approximates the BFGS algorithm using a limited amount of computer memory.
- L-BFGS-B (Limited-memory BFGS with Bounds) is an optimization algorithm that combines L-BFGS with bound constraints on the variables.
- Conjugate Gradient is an optimization algorithm for finding the minimum of a function using the first derivatives of the function.
- Trust Region is an optimization algorithm that seeks to find the minimum of a function by iteratively approximating the function using a quadratic model.

Each algorithm has its own advantages and disadvantages and it's up to the researcher to decide which algorithm to use based on their specific problem and dataset. It's important to carefully evaluate the different options and choose the one

that is best suited to the task at hand [45][46][47][48]. Table 7 compares various optimization algorithms and highlights some of their differences:

Table 7 Comparison of Optimization Algorithms.

Algorithm	Approach	Memory	Convergence Rate
SGD(M)	Stochastic	Low	SC , FNC
Adagrad	Adaptive Learning Rate	Low	SC , FNC
RMSProp	Adaptive Learning Rate	Low	SC , FNC
Adam	Adaptive Moments	Low	FC , FNC
L-BFGS	Quasi-Newton	High	FC
Newton's Method	Second-order	High	FC
Conjugate Gradient	First-order	Low	FC
Trust Region	Iterative	Medium	FNC

SC : Slow for Convex functions, FC : Faster for Convex functions and FNC : Faster for Non-Convex functions.

Overall, the AMAdam method offers a promising alternative to traditional optimization algorithms, especially for tasks involving deep learning models. Its ability to adjust the learning rate for each parameter based on the absolute value of the first estimate may result in improved convergence and generalization properties. Further research is needed to fully understand the behavior of the AMAdam algorithm and determine its limitations and potential uses.

6 Conclusion and Future work

The research presented in this paper has provided a solution to the pervasive challenge of poor generalization in adaptive gradient algorithms by introducing the novel AMAdam optimization method. Through extensive experimentation in the domains of image and text categorization tasks using recurrent neural networks, we have demonstrated that AMAdam outperforms existing optimization methods. AMAdam exhibits several key advantages, including superior accuracy, reduced hyperparameter dependence, faster convergence, and robust generalization. This contribution significantly advances the field of machine learning optimization, and we anticipate its adoption as a prominent method for enhancing the performance of deep learning and other machine learning tasks.

In light of these achievements, we now turn our attention to the promising avenues for future research. We recognize the need for further empirical studies to comprehensively assess AMAdam's performance across a wide spectrum of tasks and datasets. Specifically, we plan to investigate its efficacy in scenarios involving sparse gradients, which pose unique challenges. Additionally, exploring AMAdam's compatibility with various regularization techniques is of paramount importance to adapt it to a broader range of applications.

Another intriguing direction for future work involves the continuous refinement and enhancement of the AMAdam algorithm itself. This could encompass the exhaustive search for optimal hyperparameters that maximize its performance, although we

acknowledge the computational complexity and time constraints associated with this endeavor, particularly for intricate models characterized by numerous hyperparameters.

Furthermore, our research agenda extends to the development of hybrid optimization methods that capitalize on the strengths of multiple algorithms, including Adabelieve, AdamW, Radam, AdaBound and others. These hybrid approaches have the potential to deliver even better performance and accelerated convergence rates, opening new possibilities for addressing complex machine learning challenges.

Acknowledgment

I would like to express my heartfelt gratitude to everyone who contributed to this project. Your valuable input, feedback, and support have been instrumental in its success. To our diligent reviewers, thank you for your insightful critiques that have elevated the quality of our work. My appreciation also extends to colleagues, mentors, and collaborators who have provided invaluable support and encouragement throughout this journey.
G. Casalino acknowledges funding from the European Union PON project Ricerca e Innovazione 2014-2020, DM 1062/2021. G. Casalino is a member of the INdAM GNCS research group.

7 Appendix A: Convergence Proof

Definition 1. *Definition 1: A function* $f : R^d \to R$ *is convex if for all* $x,y \in R^d$, $f(\lambda y + (1-\lambda)x) \leq \lambda f(y) + (1-\lambda)f(x)$ *for all* $\lambda \in [0;1]$

lemma 1. *A function* $f : R^d \to R$ *is convex if for all* $x,y \in R^d$, *all* $\lambda \in [0;1]$

$$f(y) \geq f(x) + \lambda \nabla f(x)^T (y - x)$$

lemma 2. *For* $M \in \mathbb{R}$

$$\sum_{i=1}^{i=M} \frac{1}{\sqrt{i}} \leq 2\sqrt{M}$$

lemma 3. *For* $\beta \in \mathbb{R}$ *and* $0 \leq \beta \leq 1$

$$\sum_{t \geq 1} \beta^t = \frac{1}{1-\beta}$$

lemma 4. *For* $\beta \in \mathbb{R}$ *and* $0 \leq \beta \leq 1$

$$\sum_{t \geq 1} t\beta^{t-1} = \frac{1}{(1-\beta)^2}$$

By using the Lemma 1 above and the notation $\nabla_\theta f(\theta_t) = g_t$, we can get :

$$f_t(\theta_t) - f_t(\theta^*) \leq g_t^T(\theta_t - \theta^*) = \sum_{i=1}^{i=d} g_{t,i}(\theta_{t,i} - \theta^*_{,i})$$

From the update rule in our algorithm, Table**??**, we ignoring ϵ:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \alpha_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \\ &= \theta_t - \frac{\alpha_t}{1-\beta_1^t}\left(\beta_{1,t}\frac{m_{t-1}}{\sqrt{\hat{v}_t}} + (1-\beta_{1,t})\frac{g_t}{\sqrt{\hat{v}_t}}\right) \end{aligned} \tag{8}$$

This yields

$$(\theta_{t+1} - \theta^*_{,i})^2 = (\theta_t - \theta^*_{,i})^2 - \frac{2\alpha_t}{(1-\beta_1^t)}\left(\beta_{1,t}\frac{m_{t-1,i}}{\sqrt{\hat{v}_t}} + (1-\beta_{1,t})\frac{g_{t,i}}{\sqrt{\hat{v}_t}}\right)(\theta_t - \theta^*_{,i}) + \alpha_t^2\frac{m_t^2}{\hat{v}_t} \tag{9}$$

By reordering the above Equation 9, we obtain :

$$\begin{aligned} g_{t,i}(\theta_{t,i} - \theta^*_{,i}) = &\frac{(1-\beta_1^t)\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{1,t})}\left((\theta_{t,i} - \theta^*_{,i})^2 - (\theta_{t+1,i} - \theta^*_{,i})^2\right) \\ &+ \frac{\beta_{1,t}}{(1-\beta_{1,t})} m_{t-1,i}(\theta^*_{,i} - \theta_{t,i}) \\ &+ \frac{\alpha_t(1-\beta_1^t)}{2(1-\beta_{1,t})}\frac{m_{t,i}^2}{\sqrt{\hat{v}_{t,i}}} \end{aligned} \tag{10}$$

The regret bound is defined as follows :

$$\begin{aligned} R(T) = \sum_{t=1}^{t=T} f_t(\theta_t) - f_t(\theta^*) &\leq \sum_{i=1}^{i=T} g_t^T(\theta_t - \theta^*) \\ &= \sum_{t=1}^{t=T}\sum_{i=1}^{i=d} g_{t,i}.(\theta_{t,i} - \theta^*_{,i}) \end{aligned} \tag{11}$$

Therefore by combining the two Equations ,10 and 11, above:

We have

$$R(T) \leq \sum_{t=1}^{t=T}\sum_{i=1}^{i=d} g_{t,i}(\theta_{t,i} - \theta^*_{,i}) = \sum_{t=1}^{t=T}\sum_{i=1}^{i=d} \frac{(1-\beta_1^t)\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{1,t})}\left((\theta_{t,i} - \theta^*_{,i})^2 - (\theta_{t+1,i} - \theta^*_{,i})^2\right) + \sum_{t=1}^{t=T}\sum_{i=1}^{i=d} \frac{\beta_{1,t}}{(1-\beta_{1,t})} m_{t-1,i}(\theta^*_{,i} - \theta_{t,i}) + \sum_{t=1}^{t=T}\sum_{i=1}^{i=d} \frac{\alpha_t(1-\beta_1^t)}{2(1-\beta_{1,t})}\frac{m_{t,i}^2}{\sqrt{\hat{v}_{t,i}}} \tag{12}$$

We now use the standard approach of bounding the regret at each step by bounding each term of these three terms in Equation 12.

The first term in the Equation 12 is bounded as follows:

$$\sum_{i=1}^{i=d}\sum_{t=1}^{i=T} \frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{t,i})}\left((\theta_{t,i} - \theta^*_{,i})^2 - (\theta_{t+1,i} - \theta^*_{,i})^2\right) = \sum_{i=1}^{i=d} \frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_1(1-\beta_{1,t})}(\theta_{1,i} - \theta^*_{,i})^2 + \sum_{i=1}^{i=d}\sum_{t=2}^{t=T} \frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_t(1-\beta_{1,t})}(\theta_{t,i} - \theta^*_{,i})^2 - \sum_{i=1}^{i=d}\sum_{t=2}^{t=T} \frac{\sqrt{\hat{v}_{t-1,i}}}{2\alpha_{t-1}(1-\beta_{1,t-1})}(\theta_{t,i} - \theta^*_{,i})^2 - \sum_{i=1}^{i=d} \frac{\sqrt{\hat{v}_{T,i}}}{2\alpha_T(1-\beta_{1,T})}(\theta_{T+1,i} - \theta^*_{,i})^2 \tag{13}$$

Ignoring the term $\sum_{i=1}^{i=d} \frac{\sqrt{\hat{v}_{T,i}}}{2\alpha_T(1-\beta_{1,T})}(\theta_{T,i} - \theta^*_{,i})^2 \geq 0$ and Since $\beta_{1,t} \leq \beta_1 \quad (1 \leq t \leq T)$, we obtain

$$\sum_{i=1}^{i=d}\sum_{i=1}^{i=d} \frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{t,i})}((\theta_{t+1,i} - \theta^*_{,i})^2 - (\theta_{t,i} - \theta^*_{,i})^2) = \sum_{i=1}^{i=d} \frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_1(1-\beta_1)}(\theta_{1,i} - \theta^*_{,i})^2 + \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}(\theta_{t,i} - \theta^*_{,i})^2\left(\frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{1,t})} - \frac{\sqrt{\hat{v}_{t-1,i}}}{2\alpha_{t-1}(1-\beta_{1,t-1})}\right) \tag{14}$$

The second term in Equation 12 is bounded as follows :

$$\begin{aligned}(\theta^*_{,i} - \theta_{t+1,i})m_{t-1,i} =& (\theta^*_{,i} - \theta_{t+1,i})\frac{\hat{v}^{1/4}_{t-1,i}}{\sqrt{\alpha_{t-1}}}\frac{\sqrt{\alpha_{t-1}}}{\hat{v}^{1/4}_{t-1,i}}m_{t-1,i} \\ &\leq (\theta^*_{,i} - \theta_{t+1,i})^2\frac{\hat{v}^{1/2}_{t-1,i}}{2\alpha_{t-1}} + \frac{\alpha_{t-1}}{2{\hat{v}_{t-1,i}}^{1/2}}m^2_{t-1,i}\end{aligned} \tag{15}$$

The inequality above is from the fact that $ab \leq a^2/2 + b^2/2$ for any $a, b \in R$.

Since $m_0 = v_0 = 0$,we have

$$\begin{aligned}\sum_{i=1}^{i=d}\sum_{t=1}^{t=T}\frac{\beta_{1,t}}{(1-\beta_{1,t})}(\theta^*_{,i} - \theta_{t+1,i})m_{t-1,i} =& \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{(1-\beta_{1,t})}(\theta^*_{,i} - \theta_{t+1,i})m_{t-1,i} \\ \leq& \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{2(1-\beta_{1,t})}(\theta^*_{,i} - \theta_{t+1,i})^2\frac{\hat{v}^{1/2}_{t-1,i}}{\alpha_{t-1}} \\ &+ \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{2(1-\beta_{1,t})}\frac{\alpha_{t-1}}{\hat{v}^{1/2}_{t-1,i}}m^2_{t-1,i}\end{aligned} \tag{16}$$

For the first term in the inequality above, Eq.16 we have

$$\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\sqrt{\hat{v}_{t-1,i}}}{2\alpha_{t-1}(1-\beta_{1,t})}(\theta_{t,i} - \theta^*_{,i})^2 \leq \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\sqrt{\hat{v}_{t-1,i}}}{2\alpha_{t-1}(1-\beta_1)}(\theta_{t,i} - \theta^*_{,i})^2$$

since $\beta_{1,t} \leq \beta_1 \quad for\ (1 \leq t \leq T)$

Moreover,for the second term, Eq.16, we have :

$$\begin{aligned}\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{2(1-\beta_{1,t})}\frac{\alpha_{t-1}}{\hat{v}^{1/2}_{t-1,i}}m^2_{t-1,i} =& \sum_{i=1}^{i=d}\sum_{t=1}^{t=T-1}\frac{\beta_{1,t+1}}{2(1-\beta_{1,t+1})}\frac{\alpha_t}{\hat{v}^{1/2}_{t,i}}m^2_{t,i} \\ \leq& \sum_{i=1}^{i=d}\sum_{t=1}^{t=T}\frac{1}{2(1-\beta_1)}\frac{\alpha_t}{\hat{v}^{1/2}_{t,i}}m^2_{t,i}\end{aligned}$$

Adding this last term to Third term in the Equation 12 we obtain

$$\sum_{i=1}^{i=d}\sum_{t=1}^{t=T}\frac{1}{2(1-\beta_1)}\frac{\alpha_t}{\hat{v}^{1/2}_{t,i}}m^2_{t,i} + \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{2(1-\beta_{1,t})}\frac{\alpha_{t-1}}{\hat{v}^{1/2}_{t-1,i}}m^2_{t-1,i} \leq \sum_{i=1}^{i=d}\sum_{t=1}^{t=T}\frac{1}{(1-\beta_1)}\frac{\alpha_t}{\hat{v}^{1/2}_{t,i}}m^2_{t,i}$$

Since $\beta_{1,t} \leq \beta_1 \leq 1 \quad for\ (1 \leq t \leq T)$

Therefore, we return to our regret bound of Equation.12. Using the above inequalities we obtain

$$R(T) \leq \sum_{i=1}^{i=d} \frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_1(1-\beta_1)}(\theta_{1,i} - \theta^*_{,i})^2 + \sum_{i=1}^{i=d}\sum_{t=2}^{t=T}(\theta_{t,i} - \theta^*_{,i})^2 \left(\frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{1,t})} - \frac{\sqrt{\hat{v}_{t-1,i}}}{2\alpha_{t-1}(1-\beta_{1,t-1})} \right)$$
$$+ \sum_{i=1}^{i=d}\sum_{t=2}^{t=T} \frac{\beta_{1,t}}{2(1-\beta_{1,t})}(\theta^*_{,i} - \theta_{t+1,i})^2 \frac{\sqrt{\hat{v}_{t-1,i}}}{\alpha_{t-1}} + \sum_{i=1}^{i=d}\sum_{t=1}^{t=T} \frac{1}{(1-\beta_1)} \frac{\alpha_t}{\sqrt{\hat{v}_{t,i}}} m^2_{t,i} \tag{17}$$

lemma 5. *For the parameter configurations and conditions assumed in Theorem 1, we have*

$$\|v_t\| \leq G_\infty$$

In fact

$$\begin{aligned} v_t = abs(m_t) &= (1-\beta_1)abs(\sum_{i=1}^{i=t} \beta_1^{t-i} g_i) \\ &\leq (1-\beta_1)abs(\sum_{i=1}^{i=t} \beta_1^{t-i}(max(g_i))) \\ &\leq (1-\beta_1)(\sum_{i=1}^{i=t} \beta_1^{t-i}(abs(max(g_i))) \\ &\leq G_\infty(1-\beta_1)(\sum_{i=1}^{i=t} \beta_1^{t-i}) \\ &\leq G_\infty \end{aligned}$$

The fourth inequality is from Lemma 3

lemma 6. *For the parameter configurations and conditions assumed in Theorem 1, we have*

$$\frac{1}{(1-\beta_1)} \frac{\alpha_t}{\sqrt{\hat{v}_{t,i}}} m^2_{t,i} \leq G^{3/2}_\infty \frac{\alpha}{(1-\beta_1)} \frac{1}{\sqrt{t}}$$

In fact

$$\begin{aligned}
\frac{1}{(1-\beta_1)}\frac{\alpha_t}{\sqrt{\hat{v}_{t,i}}}m_{t,i}^2 =& \frac{\alpha}{(1-\beta_1)}\frac{m_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} \\
=& \frac{\alpha}{(1-\beta_1)}\frac{abs(m_{t,i})^2}{\sqrt{abs(m_{t,i})t}} \\
=& \frac{\alpha}{(1-\beta_1)}\frac{abs(m_{t,i})^{3/2}}{\sqrt{t}} \\
\leq& \frac{\alpha}{(1-\beta_1)}\frac{(1-\beta_1)^{3/2}(abs(\sum_{k=1}^{k=t}\beta_1^{t-k}max(g_{k,i})))^{3/2}}{\sqrt{t}} \\
\leq& \alpha(1-\beta_1)^{1/2}\frac{(\sum_{k=1}^{k=t}\beta_1^{t-k}abs(max(g_{k,i})))^{3/2}}{\sqrt{t}} \\
\leq& G_\infty^{3/2}\frac{\alpha(1-\beta_1)^{1/2}}{(1-\beta_1)^{3/2}}\frac{1}{\sqrt{t}} \\
\leq& G_\infty^{3/2}\frac{\alpha}{(1-\beta_1)}\frac{1}{\sqrt{t}}
\end{aligned}$$

Hence

$$\begin{aligned}
\sum_{i=1}^{i=d}\sum_{t=1}^{t=T}\frac{1}{(1-\beta_1)}\frac{\alpha_t}{\sqrt{\hat{v}_{t,i}}}m_{t,i}^2 \leq& \sum_{i=1}^{i=d}\sum_{t=1}^{t=T}G_\infty^{3/2}\frac{\alpha}{(1-\beta_1)}\frac{1}{\sqrt{t}} \\
\leq& G_\infty^{3/2}d\frac{\alpha}{(1-\beta_1)}\sum_{t=1}^{t=T}\frac{1}{\sqrt{t}} \\
\leq& G_\infty^{3/2}d\frac{\alpha}{(1-\beta_1)}2\sqrt{T}
\end{aligned}$$

where the first inequality is from Lemma 5 The last inequality is from Lemma 2

lemma 7. *For the parameter configurations and conditions assumed in Theorem 1, we have*

$$\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{2(1-\beta_1)}(\theta_{,i}^* - \theta_{t+1,i})^2\frac{\sqrt{\hat{v}_{t-1,i}}}{\alpha_{t-1}} \leq \frac{\beta_1 d\ G_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\frac{1}{(1-\lambda)^2}$$

Indeed

$$\begin{aligned}\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_{1,t}}{2(1-\beta_1)}(\theta_{,i}^*-\theta_{t+1,i})^2\frac{\sqrt{\hat{v}_{t-1,i}}}{\alpha_{t-1}} =&\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\frac{\beta_1\lambda^{t-1}}{2(1-\beta_1)}(\theta_{,i}^*-\theta_{t+1,i})^2\frac{\sqrt{t-1}\sqrt{\hat{v}_{t-1,i}}}{\alpha}\\ \leq&\frac{\beta_1 G_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\lambda^{t-1}\sqrt{t-1}\\ \leq&\frac{\beta_1 G_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}\lambda^{t-1}t\\ \leq&\frac{\beta_1 d\ G_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\frac{1}{(1-\lambda)^2}\end{aligned}$$

The first inequality is from Lemma 2 and the last inequality is from Lemma 4

lemma 8. *For the parameter configurations and conditions assumed in Theorem 1, we have*

$$\sum_{i=1}^{i=d}\sum_{i=1}^{i=d}\frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{t,i})}((\theta_{t+1,i}-\theta_{,i}^*)^2-(\theta_{t,i}-\theta_{,i}^*)^2)\leq\frac{dG_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\sqrt{T}$$

In fact

$$\begin{aligned}\sum_{i=1}^{i=d}\sum_{i=1}^{i=d}\frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{t,i})}((\theta_{t+1,i}-\theta_{,i}^*)^2-(\theta_{t,i}-\theta_{,i}^*)^2) =&\sum_{i=1}^{i=d}\frac{\sqrt{\hat{v}_{1,i}}}{2\alpha_t(1-\beta_{1,t})}(\theta_{1,i}-\theta_{,i}^*)^2\\ &+\sum_{i=1}^{i=d}\sum_{t=2}^{t=T}(\theta_{t,i}-\theta_{,i}^*)^2\left(\frac{\sqrt{\hat{v}_{t,i}}}{2\alpha_t(1-\beta_{1,t})}-\frac{\sqrt{\hat{v}_{t-1,i}}}{2\alpha_{t-1}(1-\beta_{1,t-1})}\right)\\ \leq&dD_\infty^2\frac{\sqrt{\hat{v}_{T,i}}}{2\alpha_T(1-\beta_1)}\\ \leq&\frac{dG_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\sqrt{T}\end{aligned}$$

In summary. Using Lemmas 6,7,8 we have

$$R(T)\leq\frac{dG_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)}\sqrt{T}+\frac{\beta_1 d\ G_\infty^{1/2}D_\infty^2}{2\alpha(1-\beta_1)(1-\lambda)^2}+\frac{\alpha G_\infty^{3/2}d}{(1-\beta_1)}\sqrt{T}\tag{18}$$

which ends the poof.

References

[1] Tian, Y.: Artificial Intelligence Image Recognition Method Based on Convolutional Neural Network Algorithm. IEEE Access (2020) https://doi.org/10.1109/ACCESS.2020.3006097

[2] Wang, L., Ma, W., Fan, Y., Zuo, Z.: Trip chain extraction using smartphone-collected trajectory data. Transp. B (2019) https://doi.org/10.1080/21680566.2017.1386599

[3] Yazdizadeh, A., Patterson, Z., Farooq, B.: An automated approach from GPS traces to complete trip information. Int. J. Transp. Sci. Technol. **8**(1), 82–100 (2019) https://doi.org/10.1016/j.ijtst.2018.08.003

[4] Nabipour, M., Nayyeri, P., Jabani, H., Mosavi, A., Salwana, E., Shahab, S.: Deep learning for stock market prediction. Entropy (2020) https://doi.org/10.3390/E22080840

[5] Kabiri, H., Ghanou, Y.: Predicting the mode of transport from gps trajectories, 194–207 (2022) https://doi.org/10.1007/978-3-031-07969-6_15

[6] Khalifi, H., Elqadi, A., Ghanou, Y.: Support vector machines for a new hybrid information retrieval system. In: Procedia Comput. Sci. (2018). https://doi.org/10.1016/j.procs.2018.01.108

[7] Kaczmarek-Majer, K., Casalino, G., Castellano, G., Dominiak, M., Hryniewicz, O., Kamińska, O., Vessio, G., Díaz-Rodríguez, N.: Plenary: Explaining black-box models in natural language through fuzzy linguistic summaries. Information Sciences **614**, 374–399 (2022)

[8] Robbins, H., Monro, S.: A Stochastic Approximation Method. Ann. Math. Stat. (1951) https://doi.org/10.1214/aoms/1177729586

[9] Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Natl. Bur. Stand. (1934). (1952) https://doi.org/10.6028/jres.049.044

[10] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A system for large-scale machine learning. In: Proc. 12th USENIX Symp. Oper. Syst. Des. Implementation, OSDI 2016 (2016)

[11] Chollet, F.: Keras: The Python Deep Learning library. Keras.Io (2015)

[12] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala,

S.: Pytorch: An imperative style, high-performance deep learning library (2019) arXiv:1912.01703 [cs.LG]

[13] Singh, A., Plumbley, M.D.: Efficient cnns via passive filter pruning (2023) arXiv:2304.02319 [cs.LG]

[14] Hosseini, S., Akilan, T.: Advanced deep regression models for forecasting time series oil production (2023) arXiv:2308.16105 [cs.LG]

[15] Nesterov, Y., Spokoiny, V.: Random Gradient-Free Minimization of Convex Functions. Found. Comput. Math. **17**(2), 527–566 (2017) https://doi.org/10.1007/s10208-015-9296-2

[16] Ronald, N., Arentze, T., Timmermans, H.: Towards process validation for complex transport models: A sensitivity analysis of a social network-enhanced activity-travel model. Comput. Environ. Urban Syst. (2016) https://doi.org/10.1016/j.compenvurbsys.2015.09.005

[17] Yang, X., Stewart, K., Tang, L., Xie, Z., Li, Q.: A review of GPS trajectories classification based on transportation mode. Sensors (Switzerland) **18**(11), 1–20 (2018) https://doi.org/10.3390/s18113741

[18] Hu, J., Doshi, V., Eun, D.Y.: Efficiency ordering of stochastic gradient descent (2022) arXiv:2209.07446 [cs.LG]

[19] An, J., Lu, J.: Convergence of stochastic gradient descent under a local lajasiewicz condition for deep neural networks (2023) arXiv:2304.09221 [cs.LG]

[20] Koloskova, A., Doikov, N., Stich, S.U., Jaggi, M.: Shuffle sgd is always better than sgd: Improved analysis of sgd with arbitrary data orders (2023) arXiv:2305.19259 [cs.LG]

[21] Huang, H., Wang, C., Dong, B.: Nostalgic ADAM: Weighting more of the past gradients when designing the adaptive learning rate. IJCAI Int. Jt. Conf. Artif. Intell. **2019-Augus**, 2556–2562 (2019) https://doi.org/10.24963/ijcai.2019/355 arXiv:1805.07557

[22] Abbe, E., Boix-Adsera, E., Misiakiewicz, T.: Sgd learning on neural networks: leap complexity and saddle-to-saddle dynamics (2023) arXiv:2302.11055 [cs.LG]

[23] Tran, P.T., Phong, L.T.: On the Convergence Proof of AMSGrad and a New Version. IEEE Access (2019) https://doi.org/10.1109/ACCESS.2019.2916341 arXiv:1904.03590

[24] Defossez, A., Bottou, L., Bach, F., Usunier, N.: On the convergence of adam and adagrad. arXiv (2020) arXiv:2003.02395

[25] Frangella, Z., Rathore, P., Zhao, S., Udell, M.: Sketchysgd: Reliable stochastic optimization via randomized curvature estimates (2023) arXiv:2211.08597 [math.OC]

[26] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. (2011)

[27] Nesterov, Y., Spokoiny, V.: Random Gradient-Free Minimization of Convex Functions. Found. Comput. Math. **17**(2), 527–566 (2017) https://doi.org/10.1007/s10208-015-9296-2

[28] Luo, L., Xiong, Y., Liu, Y., Sun, X.: Adaptive gradient methods with dynamic bound of learning rate. arXiv (2018), 1–19 (2019) arXiv:1902.09843

[29] Tieleman, T., Hinton, G.: Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Tech. Rep. (2012)

[30] Zeiler, M.D.: ADADELTA: An Adaptive Learning Rate Method (2012) arXiv:1212.5701

[31] Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., 1–15 (2015) arXiv:1412.6980

[32] Loshchilov, I., Hutter, F.: Fixing weight decay regularization in adam. CoRR **abs/1711.05101** (2017) 1711.05101

[33] Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., Han, J.: On the variance of the adaptive learning rate and beyond. In: Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020) (2020)

[34] Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., Dvornek, N., Papademetris, X., Duncan, J.S.: AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. cite arxiv:2010.07468 (2020). http://arxiv.org/abs/2010.07468

[35] Reddi, S.J., Kale, S., Kumar, S.: On the convergence of Adam and beyond (2018) arXiv:1904.09237

[36] Dubey, S.R., Chakraborty, S., Roy, S.K., Mukherjee, S., Singh, S.K., Chaudhuri, B.B.: Diffgrad: An optimization method for convolutional neural networks (2019)

[37] Darken, C., Moody, J.E.: Note on Learning Rate Schedules for Stochastic Optimization. Adv. Neural Inf. Process. Syst. (1989)

[38] Gowgi, P., Garani, S.S.: Hessian-based Bounds on Learning Rate for Gradient Descent Algorithms (2020) https://doi.org/10.1109/IJCNN48605.2020.9207074

[39] Zhang, J., Hu, F., Li, L., Xu, X., Yang, Z., Chen, Y.: An adaptive mechanism

to achieve learning rate dynamically. Neural Comput. Appl. **31**(10), 6685–6698 (2019) https://doi.org/10.1007/s00521-018-3495-0

[40] Reddi, S.J., Kale, S., Kumar, S.: On the convergence of Adam and beyond. In: 6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc. (2018)

[41] Sharma, A.: Guided stochastic gradient descent algorithm for inconsistent datasets. Applied Soft Computing **73**, 1068–1080 (2018) https://doi.org/10.1016/j.asoc.2018.09.038

[42] Moradian, H., Kia, S.: On the positive effect of delay on the rate of convergence of a class of linear time-delayed systems. IEEE Transactions on Automatic Control **PP**, 1–1 (2019) https://doi.org/10.1109/TAC.2019.2961348

[43] Smith, S.L., Kindermans, P.-J., Ying, C., Le, Q.V.: Don't decay the learning rate, increase the batch size (2018) arXiv:1711.00489 [cs.LG]

[44] Wan, Y., Yao, C., Song, M., Zhang, L.: Non-stationary online convex optimization with arbitrary delays (2023) arXiv:2305.12131 [cs.LG]

[45] Ruder, S.: An overview of gradient descent optimization algorithms. CoRR **abs/1609.04747** (2016) 1609.04747

[46] Nocedal, J., Wright, S.J.: Numerical Optimization (2006)

[47] Liu, D.C.: On the limited memory bfgs method for large scale optimization. CoRR (1989)

[48] Bottou, L.: Large-scale machine learning with stochastic gradient descent, 177–187 (2010)

[49] LeCun, Y., Cortes, C.: MNIST handwritten digit database. AT&T Labs [Online]. Available http//yann. lecun. com/exdb/mnist (2010)

[50] Lakshmipathi, N.: IMDB Dataset of 50K Movie Reviews (2019)

[51] Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)

[52] Krizhevsky, A., Nair, V., Hinton, G.: Cifar-100 (canadian institute for advanced research)

[53] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks **25** (2012)

[54] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014) https://doi.org/10.48550/ARXIV.1409.1556

[55] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

8 Data and Architectures

MNIST: The images in the MNIST database were produced in 1998 and contain numbers from zeros to nine written by high school students and employees. This is a large database of handwritten digits that is widely used for training various image processing algorithms and commonly used for training and testing in the field of machine learning. The MNIST database contains 60,000 training images and 10,000 test images [49].

IMDB: This dataset has 25000 high polarity movie reviews for training and 25000 for prospecting, for a total number of 50000 reviews with positivity and negativity labels[50].

CIFAR-10 is an image (32x32 pixels) dataset used for deep learning. It contains 10 different classes, each with 5,000 training images and 1,000 test images. The classes are: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks. The images are in color with a depth of 3 (red, green and blue) [51].

CIFAR-100 is a collection of 60,000 32x32 pixel images, grouped into 100 different classes, each containing 600 images. These 100 classes are divided into 20 superclasses, each of which includes 5 subclasses. This hierarchical structure adds an extra layer of complexity compared to CIFAR-10. The dataset covers a wide range of object categories, making it suitable for evaluating the ability of models to recognize various objects and scenes. Some example classes include "apple," "beaver," "sunflower," "worm," and "cloud." [52].

AlexNet is a deep convolutional neural network designed for deep learning. It was created in 2012 by the research team of Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton. AlexNet is the first deep convolution neural network that has achieved state-of-the-art performance in image recognition using massive training data. The AlexNet model contains 8 layers, including five convolution layers followed by pooling layers and two fully connected layers. The model also uses optimization techniques such as oversampling, pooling layer regularization, batch normalization, and stochastic gradient descent method [53].

VGG19-bn is another deep convolutional neural network model designed for deep learning. It was created in 2014 by Karen Simonyan and Andrew Zisserman. The model is similar to AlexNet in structure, but uses smaller filters in the convolution layers. The VGG19-bn model contains 19 layers, including 16 convolution layers followed by pooling layers and three fully connected type layers. The model also uses optimization techniques such as batch normalization and stochastic gradient descent

method. The "bn" in the model name stands for "batch normalization", which is an optimization technique that normalizes the inputs to each layer to facilitate deep learning convergence [54].

ResNet ResNet's structure is quite different from traditional models like VGG19-bn or AlexNet. The core innovation of ResNet is the use of residual blocks. These blocks introduce a shortcut connection or skip connection that skips one or more layers, allowing the network to learn residual functions instead of directly fitting the desired underlying mapping. This ingenious approach makes it easier for the network to capture fine-grained details and learn deeper features.The ResNet architecture comes in various depths, such as ResNet-18, ResNet-50, ResNet-101, and even deeper variants. Deeper networks have been shown to perform exceptionally well in image classification and other computer vision tasks [55].