IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 31, 2021, accepted July 5, 2021, date of publication July 13, 2021, date of current version July 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3096976

Adadb: Adaptive Diff-Batch Optimization Technique for Gradient Descent

MUHAMMAD U. S. KHAN[1], (Member, IEEE), MUHAMMAD JAWAD[2], AND SAMEE U. KHAN[3], (Senior Member, IEEE)

[1]Department of Computer Science, COMSATS University Islamabad, Abbottabad Campus, Abbottabad 22060, Pakistan
[2]Department of Electrical Engineering, COMSATS University Islamabad, Lahore Campus, Lahore 54000, Pakistan
[3]Electrical and Computer Engineering, Mississippi State University, Starkville, MS 39762, USA

Corresponding author: Muhammad U. S. Khan (ushahid@cuiatd.edu.pk)

ABSTRACT Gradient descent is the workhorse of deep neural networks. Gradient descent has the disadvantage of slow convergence. The famous way to overcome slow convergence is to use momentum. Momentum effectively increases the learning factor of gradient descent. Recently, many approaches have been proposed to control the momentum for better optimization towards global minima, such as Adam, diffGrad, and AdaBelief. Adam decreases the momentum by dividing it with square root of moving averages of squared past gradients or second moment. The sudden decrease in the second moment often results in the overshoot of the gradient from the minima and then settle at the closest minima. DiffGrad decreases this problem by using a friction constant based on the difference of current gradient and immediate past gradient in Adam. The friction constant further decreases the momentum and results in slow convergence. AdaBelief adapts the step size according to the belief in the current gradient direction. Another famous way of fast convergence is to increase the batch size adaptively. This paper proposes a new optimization technique named adaptive diff-batch or adadb that removes the problem of overshooting gradient in Adam, slow convergence in diffGrad, and combines the methods with adaptive batch size for further increase in convergence rate. The proposed technique uses the friction constant based on the past three differences of gradients rather than one as in diffGrad and a condition to decide the use of friction constant. The proposed technique has outperformed the Adam, diffGrad, and AdaBelief optimizers on synthetic complex non-convex functions and real-world datasets.

INDEX TERMS Machine learning, gradient descent, optimization, image classification.

I. INTRODUCTION

In recent times, neural network-based algorithms are gaining popularity due to the availability of big data and large computing power in the form of GPUs. With the help of these two factors, neural networks have achieved high accuracy in solving problems in various fields, such as computer vision, signal processing, human activity recognition, and natural language processing.

Despite recent popularity and achievements, the neural networks still dependent on a gradient descent algorithm that was developed in 1847 [1]. All the variants of gradient descent (Batch, mini-batch, and stochastic) inherit the disadvantage of slow convergence towards global minima. Recently, many attempts have been made to optimize the convergence of gradient descent to get the true benefits of big data and large computing power with neural networks.

The most famous way of increasing the convergence rate of gradient descent is the use of momentum. It is reported that momentum effectively increases the convergence rate by a factor of 2 [2]. However, fast convergence often forces the gradient descent to settle at the local minima instead of the global minima.

There are two renowned methods of controlling the convergence rate: reduce the momentum and increase the batch size. Adam [3], diffGrad [4], and AdaBelief [5] optimization techniques reduce the momentum whereas adabatch technique [6] increases the batch size for better and fast optimization towards the global minima. Adam and AdaBelief optimizer techniques often overshoot the global minima whereas diffGrad suffers from the slow convergence of the solution. Similarly, adabatch is dependent on the use

The associate editor coordinating the review of this manuscript and approving it for publication was Haiyong Zheng.

of convergence technique used with the adabatch. In this paper, we use both methods, control of convergence rate and increase in batch size. We remove the problem of slow convergence in diffGrad by replacing the friction constant with a sum of past three gradient differences rather than one and combines the approach with adaptive batch size increment. Combining adaptive batch size with convergence technique is based on our previous work that has shown success in improving the convergence rate [7]. The list of contributions of this paper is as follows:

- A modified compact friction constant is defined that prevents the solution from overshooting the global minima
- A conditional constraint is defined to select an appropriate friction constant that can increase or decrease the momentum rather than only decreasing
- A batch-size increment technique is proposed to increase the convergence of the optimizer

The rest of the paper is organized as follows: Section 2 describes the basics of the gradient descent. Related work is presented in Section 3. The proposed technique is explained in Section 4 and its convergence analysis is covered in Section 6. Section 6 covers the empirical analysis and experimental setup is explained in Section 7. Results and discussion is covered in Section 8. The paper is concluded in Section 9.

II. BASICS OF GRADIENT DESCENT

The gradient descent presents the most basic approach for optimizing parameters in a neural network. Initially, random values are used for parameters that are used with input data to calculate the predicted values. A loss function defined for a specific problem is used to calculate the difference between predicted output values and original ones. Later, the gradient against each parameter is computed and helps in updating the respective parameter. The updated parameters are used to calculate new predicted values. The procedure is repeated until the convergence or for a certain number of epochs. The stochastic gradient descent uses the above procedure against each data sample. The direction of gradient in each iterative step oscillate and therefore, slows the convergence. The batch gradient descent uses the average of all the data samples at once and performs only one parameter update in each epoch. The batch process helps in reducing the oscillatory behavior but requires a large number of epochs for optimization. In practice, mini-batch descent algorithm is used that helps in reducing the path oscillation and creates less overhead as compared to the batch algorithm. However, the mini-batch gradient descent still inherits the problem of slow convergence due to different behavior among a large number of parameters used in the neural networks.

In gradient descent, all parameters of the model are updated on a same learning rate α_i in the ith iteration, such as:

$$\Phi_{i+1,j} = \Phi_{i,j} - \alpha_i \times g_{i,j}, \tag{1}$$

where $\Phi_{i+1,j}$ and $\Phi_{i,j}$ are the updated and previous values of the jth parameter with $j = 1, 2, 3, \ldots\ldots J$. Here, J represents a total number of parameters. The $g_{i,j}$ is the gradient of the loss function $\mathcal{L}$ with respect to the parameter Φ_{ij}. The mathematical representation of the gradient $g_{i,j}$ is

$$g_{i,j} = \frac{\partial(\mathcal{L}_{i,\Phi})}{\partial(\Phi_{i,j})}. \tag{2}$$

In Eqn 2, the $\mathcal{L}_{i,\Phi}$ is the loss function of the parameter Φ in the ith iteration. For image processing applications, this loss function is the cross-entropy loss that is defined as:

$$\mathcal{L}_{i,\Phi} = \frac{1}{K_B}\sum_{K}^{K_B}\mathcal{L}_{i,\Phi,K} + \sigma R_{i,\Phi}, \tag{3}$$

where K_B is the number of images in the Batch B, the $\mathcal{L}_{i,\Phi,K}$ is the cross entropy data loss for the kth training image in the ith iteration, $R_{i,\Phi}$ is the regularization loss for the ith iteration, and hyper-parameter of the regularization loss is denoted by σ. The $\mathcal{L}_{i,\Phi,K}$ for the Kth training sample is computed as:

$$\mathcal{L}_{i,\Phi,K} = -Log(\frac{e^{S_{o_K}}}{\sum_{n=1}^{N_c} e^{S_n}}). \tag{4}$$

In Eqn 4, the number of classes in the dataset is denoted by N_c, the o_K is the ground truth class for the Kth training image, and nth class score computed for Kth training image is denoted by S_n. Moreover, the regularization loss function $R_{i,\Phi}$ is computed as:

$$R_{i,\Phi} = \sum_{j=1}^{J}(\Phi_{i,j})^2. \tag{5}$$

III. RELATED WORKS

The SGD with momentum is the advancement in the gradient descent algorithms [8]. The concept of gradient is each dimension (parameter) is included in the stochastic gradient descent to enhance the momentum of the parameters having the consistent gradient. The moment gradient is computed as:

$$m_{i,j} = \gamma m_{i-1,j} + g_{i,j}, \tag{6}$$

where $m_{i,j}$ is the gained moment at the ith iteration for the jth parameter $\Phi_{i,j}$ with $m_{i,j} = 0$ for $i = 0$, and moment is controlled by the hyper-parameter γ. Therefore, Eqn 1 is modified as:

$$\Phi_{i+1,j} = \Phi_{i,j} - \alpha_i \times m_{i,j}. \tag{7}$$

In another technique AdaGrad [9], the learning rate for the gradient descent is normalized as:

$$\Phi_{i+1,j} = \Phi_{i,j} - \frac{\alpha_i \times g_{i,j}}{\sqrt{G_{i,j}} + \epsilon}. \tag{8}$$

In Eqn 8, the ϵ is a very small value close to zero (mostly selected as $1e^{-8}$) added to avoid division by zero and $G_{i,j}$ is the sum of squares of the gradients of t steps for the jth parameter computed as:

$$G_{i,j} = \sum_{i=1}^{i}(g_{i,j})^2, \tag{9}$$

where $g_{i,j}$ is computed as Eqn 2. However, the issue with the denominator of Eqn 8 is the sum of squares of the gradients may become very large over the t steps due to the positive accumulation of the squares of the gradients. The AdaDelta [10] and RMSProp [11] address this issue by reducing the impact of accumulating squares of gradients by using a decay rate parameter β. In RMSProp, the Eqn 9 is modified as:

$$G_{i,j} = \beta G_{i-1,j} + (1-\beta)(g_{i,j})^2. \tag{10}$$

The value of $G_{i,j} = 0$ for $i = 1$. The most renowned gradient descent optimizing technique is Adam [3]. In Adam, at each step, the learning rate is computed by using 1st and 2nd order moments also known as mean and variance, respectively. Both moments are defined recursively using gradient and squares of gradient, respectively, such as:

$$m_{i,j} = \beta_1 m_{i-1,j} + (1-\beta_1) g_{i,j} \tag{11}$$

$$v_{i,j} = \beta_2 v_{i-1,j} + (1-\beta_2)(g_{i,j})^2 \tag{12}$$

where β_1 and β_2 are the decay rates of mean and variance, respectively. The $m_{i-1,j}$ and $v_{i-1,j}$ are the mean and variance of the previous steps, respectively that are initialized with 0 for 1st iteration. The observation noted here is that initially the 1st moment is small and 2nd moment is very small that leads to a very large step size. This issue is resolved by introducing a bias correction in both moments, such as:

$$\hat{m_{i,j}} = \frac{m_{i,j}}{(1-\beta_1^i)} \; and \; \hat{v_{i,j}} = \frac{v_{i,j}}{(1-\beta_2^i)} \tag{13}$$

In Eqn. 13, the β_1^i is the β_1 with power i, β_2^i is the β_2 with power i, and $m_{i,j}$ and $v_{i,j}$ are the bias-corrected mean and variance, respectively. The good starting choice for the values of the β_1 and β_2 are selected as 0.9 and 0.999, respectively. The learning rate α is selected in the range $\alpha \in [10^{-2}, 10^{-4}]$. The modified parameter's update Equation (Eqn. 1) for Adam is defined as:

$$\Phi_{i+1,j} = \Phi_{i,j} - \frac{\alpha_i \times \hat{m}_{i,j}}{\sqrt{\hat{v}_{i,j}} + \epsilon}. \tag{14}$$

However, the problem that arises in Adam is due to the variance as it decreases fast. Therefore, the friction in the optimization landscape decreases with the small value of the 2nd moment that can lead to a point where the updating process can overshoot the optimized solution because of the very high learning rate and the solution will diverge. AMSGrad [12] tried to solve the problem. The AMSGrad picks the largest value of the 2nd moment from current and previous iterations. Basically, the AMSGrad normalizes the learning rate a_i with the maximum value $\hat{v}_{i,j}^{max}$ among all previous and current variance instead of $\hat{v}_{i,j}$. The AMSGrad stores the previous maximum value of the 2nd moment and give priority to the maximum value of the 2nd moment. The $\hat{v}_{i,j}^{max}$ is computed as:

$$\hat{v}_{i,j}^{max} = max(\hat{v}_{i,j}^{max}, \hat{v}_{i,j}). \tag{15}$$

where $\hat{v}_{i,j}^{max} = 0$ for $i = 1$. The modified parameter's update Equation (Eqn. 14) for AMSGRad is defined as:

$$\Phi_{i+1,j} = \Phi_{i,j} - \frac{\alpha_i \times \hat{m}_{i,j}}{\sqrt{\hat{v}_{i,j}^{max}} + \epsilon}. \tag{16}$$

Both Adam and AMSGrad face the problem of auto adjustment of the learning rate. The actual issue is to control the friction of the 1st moment to avoid slipping on the optimum solution. AdaBelief [5] algorithm addresses the issue by taking the difference of gradient and 1st order moments instead of a gradient in Eqn. 12. The ratio $\frac{1}{\sqrt{\hat{v}_{i,j}}}$ serves as a belief in the system ((Eqn. 14)) with a newly added difference. The step size increases with high belief if the gradient is closer to moments otherwise step size decreases. Alternatively, DiffGrad [4] algorithm addresses the issue of controlling the 1st moment present in Adam by adding friction constant. The diffGrad is built on modification in short-term gradients to regulate the learning rate dynamically. The base concept of diffGrad is that update in the parameter must be smaller in the region of low gradient change must be large in the region of high gradient change. The diffGrad computed the 1st and 2nd moments as computed in Adam (Eqn. 13). However, a new diffGrad Friction Coefficient (DFC) is added in the enumerator of Eqn. 16 to control the learning rate based on short-term gradient behavior. The DFC is denoted by ξ and defined as:

$$\xi_{i,j} = AbsSig(\triangle g_{i,j}) \tag{17}$$

where *AbsSig* is the absolute value for the non-linear sigmoid function (Sig) that will squash every value between 0.5 and 1. The mathematical expression is written as:

$$AbsSig(y) = \frac{1}{1+e^{-|y|}} \tag{18}$$

The $\triangle g_{i,j}$ is the change is the gradient between current and previous iteration, such as:

$$\triangle g_{i,j} = g_{i-1,j} - g_{i,j} \tag{19}$$

where $g_{i,j}$ is computed using Eqn. 2. The diffGrad reported that DFC provides more friction when gradient changes slowly and vise-versa. The modified diffGrad parameter's update Equation (Eqn. 14) for jth parameter in the ith iteration is defined as:

$$\Phi_{i+1,j} = \Phi_{i,j} - \frac{\alpha_i \times \xi_{i,j} \times \hat{m}_{i,j}}{\sqrt{\hat{v}_{i,j}} + \epsilon}. \tag{20}$$

In diffGrad, the DFC is used to control the gradient oscillation/ frequency of fluctuation near the optimum. The difference of gradient taken in Eqn. 19 reduces the learning rate by controlling the moving average near the optimal point. The diffGrad algorithm claims that Adam has ignored the impact of 1st moment to control the learning rate over the entire optimization landscape and the inclusion of DFC provides high learning rate in large gradient change area and reduce the learning rate for low gradient change area.

IV. PROPOSED METHODOLOGY

The diffGrad only takes advantage of the gradient difference between the current and the previous iteration. We observe that the inclusion of the sum of the last three gradient differences with a decaying factor provides an extra zip of friction to control the learning rate faster. Therefore, the change is the gradient $\triangle g_{i,j}$ as written in Eqn. 19 is modified as:

$$\begin{aligned}\triangle g_{i,j} = \beta_3(g_{i-1,j} - g_{i,j}) \\ +\beta_3^2(g_{i-2,j} - g_{i-1,j}) \\ +\beta_3^3(g_{i-3,j} - g_{i-2,j})\end{aligned} \tag{21}$$

where $\beta_3 = 0.999$ as a decaying factor. Moreover, for a faster updating process for parameter optimization in both the large gradient change area and low gradient change, a condition is introduced to decide the DFC factor. The signum function is introduced to observe the sign change in the gradient, such as:

$$Sgn(x) = \begin{cases} -1 & \text{if x} < 0 \\ 0 & \text{if x} = 0 \\ +1 & \text{if x} > 0 \end{cases} \tag{22}$$

The condition is defined as:

1) If the sign of gradient value in the two consecutive iterations is changing, then the DFC ($\xi_{i,j}$) of Eqn. 17 will be computed using Eqn. (21), such as:

$$\begin{aligned}&if\,(sgn(g_{i,j}) \neq sgn(g_{i-1,j}))\quad then \\ &\xi_{i,j} = AbsSig(\triangle g_{i,j})\end{aligned} \tag{23}$$

2) If the sign of gradient value in the two consecutive iterations remains same positive or negative, then the new DFC is computed as:

$$\xi_{i,j} = 1 + r_i \times AbsSig(\triangle g_{i,j}) \tag{24}$$

where $r_i = 0.1$ as the percentage gradient difference and the $\triangle g_{i,j}$ is the same as computed using Eqn. 21. The Eqn. 24 makes the parameter updating process faster in the large gradient change area as the DFC will slightly increase the learning rate rather than always decreasing the learning rate. Therefore, the problem of slow convergence in diffGrad is resolved by the appropriate selection of the DFC. Furthermore, the researchers often ignore the choice of batch size for the optimization training process and select static batch for every iteration in the training process. The small batch size is required to produce convergence in fewer epochs. Conversely, large batch sizes often produce data-parallelism that can improve computational time and scalability. In diffGrad and Adam, the batch size is considered static during the training process. Therefore, to further improve the convergence time and accuracy, we introduce the trade-off between small batch size and large batch size by periodically update the batch size in the training process. The training is started with a small batch size to rapid the convergence in early epochs and then periodically increase the batch size to reduce convergence time.

V. CONVERGENCE ANALYSIS

The convergence analysis Adam and diffGrad is computed using an online learning framework proposed in [13]. We work on a similar footstep to observe the convergence of the proposed AdaDB. Let $f_1(\Phi)$, $f_2(\Phi)$, …, $f_N(\Phi)$ be the unfamiliar sequence of convex cost functions. The target is to predict the optimal value of the parameter Φ_i in ith iteration to compute the function $f_i(\Phi)$. In such a case, where the nature of sequence is unknown, the regret bound method is used to evaluate the optimization algorithm. The regret bound is defined as the sum of the difference between the past unknown guesses $f_i(\Phi_i)$ and the best fix point parameter $f_i(\Phi^*)$ in the practicable set of all prior iterations. The regret bound is defined as [13]:

$$R(N) = \sum_{i=1}^{N}[f_i(\Phi_i) - f_i(\Phi^*)] \tag{25}$$

where $\Phi^* = argmin_{\Phi \in X} \sum_{i=1}^{N} f_i(\Phi)$.

It is observed that Adadb has $O\sqrt{N}$ regret bound. The proof of the convergence is computed similarly as computed in diffGrad and comparable to known convex online learning methods. The definitions are defined as: $g_{i,j}$ represents the gradient of the jth parameter in the ith iteration, $g_{1:i,j} = [g_{1,j}, g_{2,j}, \ldots, g_{i,j}] \in \Re^i$ represents the gradient vector in the jth dimensions over all iterations till i and $\gamma \triangleq \frac{\beta_1^2}{\beta_2}$.

Theorem: Let the function f_i has bounded gradients i.e. $||g_{i,\Phi}||_2 \leq G$ and $||g_{i,\Phi}||_\infty \leq G_\infty$ for all $\Phi \in \Re^J$. It is also assumed that Adadb will generate a bounded distance between any Φ_i, such as $||\Phi_n - \Phi_m||_2 \leq D$ and $||\Phi_n - \Phi_m||_\infty \leq D_\infty$ for any arbitrary n and $m \in \{1,\ldots,M\}$. Consider $\gamma \triangleq \frac{\beta_1^2}{\beta_2}$, where β_1 and $\beta_2 \in [0, 1)$ and satisfy $\frac{\beta_1^2}{\beta_2} < 1, \alpha_i = \frac{\alpha}{\sqrt{i}}$ and $\beta_{1,i} = \beta_1 \times \lambda^{i-1}, \lambda \in (0, 1)$; however, λ is normally close to 1 and here we selected it as $1 - \epsilon$, where $\epsilon = 1e^-8$. Using Lemma 10.2 and 10.4 of Adam and Theorem 2 from Eqn 27 to Eqn 35 of diffGrad, we obtained similar regret bound as:

$$\begin{aligned}R(N) \leq \frac{D^2}{2\alpha(1-\beta_1)} \sum_{j=1}^{J} \frac{\sqrt{N\hat{v}_{N,j}}}{\xi_{1,j}} \\ +\frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{(1-\beta_2)(1-\gamma)^2}} \sum_{j=1}^{J} ||g_{1:N,j}||_2 \\ +\sum_{j=1}^{J} \frac{D_\infty^2 G_\infty \sqrt{(1-\beta_2)}}{2\alpha(1-\beta_1(1-\gamma)^2)}\end{aligned} \tag{26}$$

For the first condition:

$$\begin{aligned}&if\,(sgn(g_{i,j}) \neq sgn(g_{i-1,j}))\quad then \\ &\xi_{1,j} = \frac{1}{1 + e^{-1|\beta_3(g_{1,j} - g_{0,j}) + \beta_3^2(g_{2,j} - g_{1,j}) + \beta_3^3(g_{3,j} - g_{2,j})}}\end{aligned} \tag{27}$$

as $g_{o,j} = g_{1,j} = g_{2,j} = 0$. Therefore, $N \geq 1$, the proposed adadb shows the following guarantee:

$$R(N) \leq \frac{D^2}{2\alpha(1-\beta_1)} \sum_{j=1}^{J} \sqrt{N\hat{v}_{N,j}}(1 + e^{-|\beta_3(g_{1,j})|}) + \frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{(1-\beta_2)(1-\gamma)^2}} \sum_{j=1}^{J} ||g_{1:N,j}||_2 + \sum_{j=1}^{J} \frac{D_\infty^2 G_\infty \sqrt{(1-\beta_2)}}{2\alpha(1-\beta_1(1-\gamma)^2)} \tag{28}$$

For the second condition:

$$\xi_{1,j} = 1 + r_1 \times \frac{1}{1 + e^{-1|\beta_3(g_{1,j}-g_{0,j})+\beta_3^2(g_{2,j}-g_{1,j})+\beta_3^3(g_{3,j}-g_{2,j})}} \tag{29}$$

Therefore, for all $N \geq 1$, the proposed adadb optimizer shows the following guarantee:

$$R(N) \leq \frac{D^2}{2\alpha(1-\beta_1)} \sum_{j=1}^{J} \sqrt{N\hat{v}_{N,j}}(\frac{1 + e^{-|\beta_3(g_{1,j})|}}{r_1}) + \frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{(1-\beta_2)(1-\gamma)^2}} \sum_{j=1}^{J} ||g_{1:N,j}||_2 + \sum_{j=1}^{J} \frac{D_\infty^2 G_\infty \sqrt{(1-\beta_2)}}{2\alpha(1-\beta_1(1-\gamma)^2)} \tag{30}$$

It is observable that the sum of terms over the dimension J can be very small compared to its upper bound, i.e. $\sum_{j=1}^{J} ||g_{1:N,J}||_2 \ll dG_\infty\sqrt{N}$ and $\sum_{j=1}^{J} \sqrt{N\hat{v_{N,j}}}(1 + e^{-|\beta_3(g_{1,j})|}) \ll d(1 + E_\infty)G_\infty\sqrt{N}$, where E_∞ is the upper bound over the exponential function and $E_\infty \gg \sum_{j=1}^{J} e^{-|g_{1,j}|}$. Therefore, considering these bounds and the bounded gradients i.e., $||g_{i,\Phi}||_2 \leq G$ and $||g_{i,\Phi}||_\infty \leq G_\infty$ for all $\Phi \in \Re^J$.It is assumed that adadb will generate a bounded distance between any Φ_i, such as $||\Phi_n - \Phi_m||_2 \leq ||\Phi_n - \Phi_m||_\infty \leq D_\infty$ for any arbitrary n and $m \in \{1, 2, \ldots .N\}$.The proposed adadb optimization algorithm follow the following:

$$\frac{R(N)}{N} = O(\frac{1}{\sqrt{N}}), \tag{31}$$

where $\lim_{N\to\infty} \frac{R(N)}{N} = 0$.

VI. EMPIRICAL ANALYSIS

We perform the empirical analysis to justify the importance of a proposed methodology. We compare the proposed methodology with Adam, diffGrad, and AdaBelief optimization techniques over the following three different complex non-convex functions previously used in [4].

$$F_1(x) = \begin{cases} (x+0.3)^2 & \text{for } x \leq 0 \\ (x-0.2)^2 + 0.05 & \text{for } x > 0 \end{cases} \tag{32}$$

$$F_2(x) = \begin{cases} (-40x - 35.15) & \text{for } x \leq -0.9 \\ (x^3 + xsin(8x) + 0.85 & \text{for } x > -0.9 \end{cases} \tag{33}$$

$$F_3(x) = \begin{cases} x^2 & \text{for } x \leq -0.5 \\ 0.75 + x & \text{for } -0.5 < x \leq -0.4 \\ -7x/8 & \text{for } -0.4 < x \leq 0 \\ 7x/8 & \text{for } 0 < x \leq 0.4 \\ 0.75 - x & \text{for } 0.4 < x \leq 0.5 \\ x^2 & \text{for } 0.5 < x \end{cases} \tag{34}$$

In the above-mentioned equations, F_1, F_2, and F_3 are the non-convex functions and x presents the input to the functions with $-\infty < x < +\infty$. The function F_1 has one global and one local minima (Figure 1) whereas other two functions F_2 and F_3 have two global minima and one global maxima (Figure 2 and Figure 3). Following hyper-parameters are used in the experiments for adadb, diffGrad, AdaBelief, and Adam: decay rate for 1st moment β_1 is 0.95, decay rate for second moment β_2 is 0.999, and learning rate α is 0.5. The decay rate β_3 of adadb is set at 0.999. The parameter Φ or x in these equations is initialized with -1 to demonstrate the effectiveness of adadb optimizing technique over the Adam, AdaBelief, and diffGrad. In the experiments, we have not used the batch size increment during the operation of the adadb technique. The results of the empirical analysis are presented in Figure 4, Figure 5, and Figure 6.

Figure 4 demonstrates the problem in slow momentum of the diffGrad as compared to Adam, AdaBelief, and adadb. Adadb, AdaBelief, and Adam have faster momentum than the diffGrad and therefore, managed to move out of the local minima present at value 0.2 after touching it. Adadb converges to a global minimum earlier than both Adam and AdaBelief. The same scenario is presented in Figure 5 where diffGrad is stuck at local minima value 0.0. At the slow learning rates, diffGrad stuck at the nearest minima whereas Adam and adadb can leave the nearest minima but didn't get the time to come back if the next minimum is not the global minima as demonstrated by authors in [4]. However, there is no guarantee in the practical scenarios that the nearest one will always be the global minima. AdaBelief fluctuates twice over the different minima before behaving like Adam and adadb.

The problem in fast convergence of Adam and AdaBelief is presented in Figure 6. After visiting all the minima, Adam quickly settles at one minimum point (local in this case instead of global) and AdaBelief settles between global and local minimum values. Alternatively, adadb and diffGrad kept fluctuating between global and local minima. After around 230 epochs, adadb settles near the global minima.

The empirical analysis has demonstrated that adadb performs better than the slow diffGrad, and manages to find global minima better than the fast Adam and intermediate AdaBelief. In order to make the adadb faster than the Adam, we increases the batch size in different epochs, discussed in the next section.

VII. EXPERIMENTAL SETUP

For image categorization experiments, we uses the CIFAR10 and CIFAR100 datasets. The CIFAR10 dataset

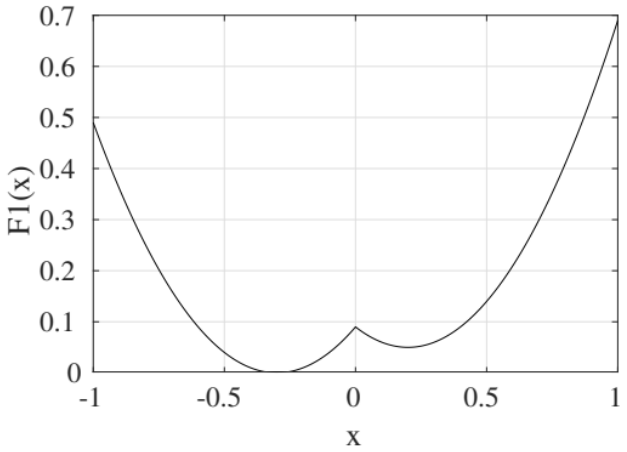

FIGURE 1. Graph for F_1 equation.

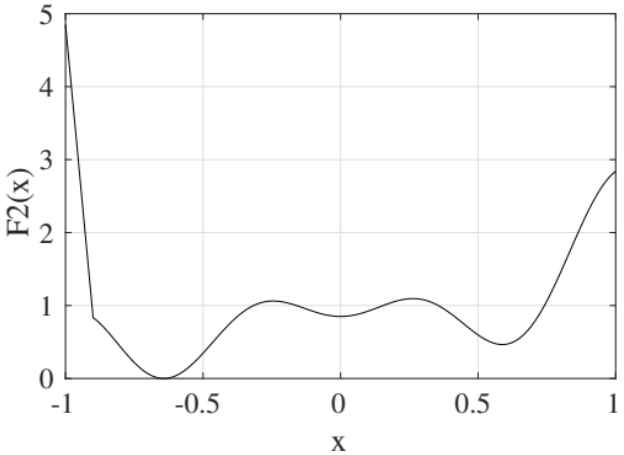

FIGURE 2. Graph for F_2 equation.

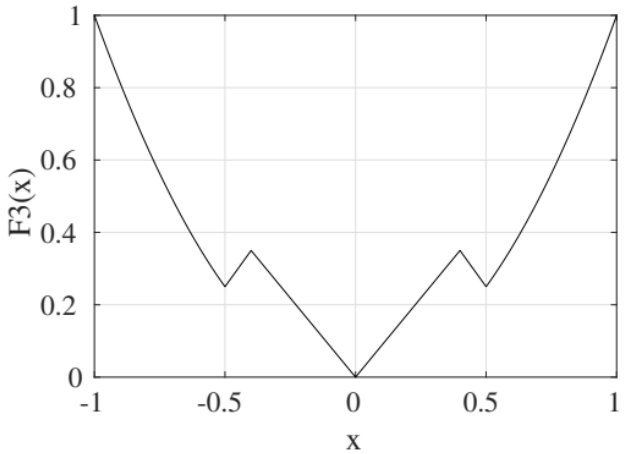

FIGURE 3. Graph for F_3 equation.

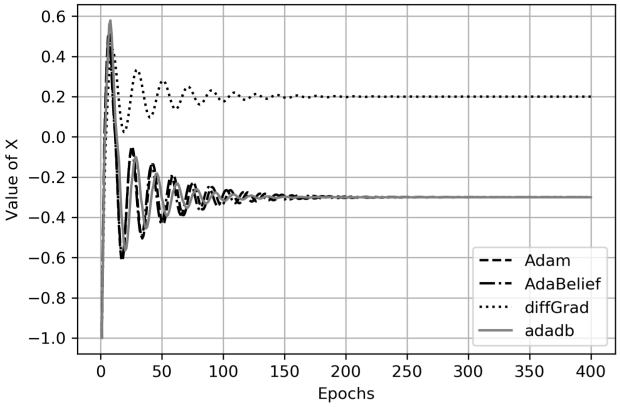

FIGURE 4. Empirical results over F_1 equation.

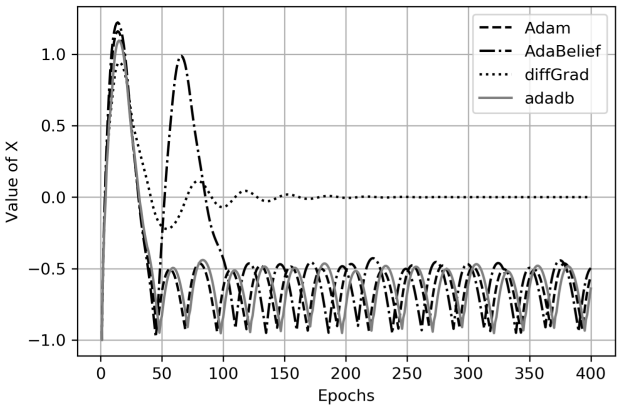

FIGURE 5. Empirical results over F_2 equation.

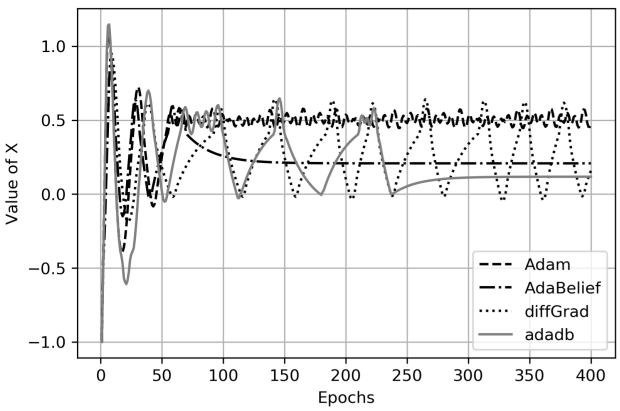

FIGURE 6. Empirical results over F_3 equation.

consists of 50K images for training and 10K images for testing. CIFAR100 dataset is similar to the CIFAR10 dataset. Howeover, it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. We use the optimization techniques (adadb, Adam, and diffGrad) in the PyTorch implementation of ResNet18 architecture. The experiments were performed on Google Colab. Following hyper-parameters are used in the experiments for adadb, diffGrad, and Adam: decay rate for 1st moment β_1 is 0.9, decay rate for second moment β_2 is 0.999, and learning rate α is 0.001. The decay rate β_3 of adadb is

set at 0.999. The number of maximum epochs is set to 100. The target is to find the technique with the highest accuracy in the first 100 epochs. The batch size is kept at 128 for all the techniques. Two different strategies are used for the comparison of techniques. In Strategy 1, the learning rate is kept constant throughout the 100 epochs. Strategy 2 is first search and then optimize strategy In Strategy 2, first 80 epochs are executed at a learning rate 0.001 (search) and the last 20 epochs are executed at 0.0001 (optimize). Each experiment was conducted 5 times and results are presented as best of 5 trials.

VIII. RESULTS AND DISCUSSION

The results of the experiments are presented in Table 1 and Table 2. In these experiments, adadb is also used without batch size increase and represented as adadb/wo.

The results have shown that all the optimization techniques improve the results with the Strategy 2. However, Adadb remains the fastest technique among the three techniques and provides the highest accuracy.

To make the optimization faster in the proposed technique, we incremented the batch size five percent after every 5 epochs. Adam and diffGrad are static techniques, therefore, this procedure is not repeated against these two methods. Table 3 and Table 4 present the results with different starting batch sizes.

Adadb is an upgrade of the previous technique adadiffgrad [7] that behaves exactly like diffgrad in the first 100 epochs and improves from diffgrad and Adam after around 250 epochs. Adadiffgrad is a combination of diffgrad with batch size adaption. Adadiffgrad doubles the batch size and decreases the step size at the same time after the first 100 epochs and repeats the step after every 50 epochs. However, if the methodology is applied earlier, the methodology tries to settle at the nearest available minima too quickly. For example, if the methodology is applied on the CIFAR10 dataset after every 25 epochs with starting batch size 32, the highest accuracy achieved in 100 epochs is 85.39. Alternatively, if the methodology is applied at every 50 epochs, the highest accuracy achieved in 100 epochs is 86.88. Therefore, adadiffgrad is not useful if we want high accuracy in the first 100 epochs. The proposed adadb uses batch size increment along with an improved friction constant that improves the algorithm in faster convergence than adadiffgrad.

One important observation has been made in this experiment that with a larger starting batch size, the testing accuracy does not improve considerably. However, with the small starting batch size, the testing accuracy increases. Using staring batch size equals 32, adadb gets better accuracy than the accuracy achieved by Adam, AdaBelief, and diffgrad methodologies. The reason for high accuracy with low batch size start is due to the high oscillation in the earlier epochs. This high oscillation helps the optimizing technique in searching a large number of minima and later with the increase in the size of the batch, the oscillation decreases and the technique

TABLE 1. Highest accuracies of three optimizers with Strategy 1 and Strategy 2 with CIFAR10 dataset.

OPTIMIZER	STRATEGY 1	STRATEGY 2
ADAM	90.66	90.71
ADABELIEF	86.24	86.76
DIFFGRAD	88.12	89.43
ADADB/WO	89.53	90.5

TABLE 2. Highest accuracies of three optimizers with Strategy 1 and Strategy 2 with CIFAR100 dataset.

OPTIMIZER	STRATEGY 1	STRATEGY 2
ADAM	63.99	64.46
ADABELIEF	54.44	57.31
DIFFGRAD	62.51	63.05
ADADB/WO	64.49	65.01

TABLE 3. Highest accuracies of adadb with different starting batch sizes for CIFAR10.

BATCH SIZE	ACCURACY
N=32	91.17
N=64	90.53
N=128	90.51

TABLE 4. Highest accuracies of adadb with different starting batch sizes for CIFAR100.

BATCH SIZE	ACCURACY
N=32	66.67
N=64	65.39
N=128	63.88

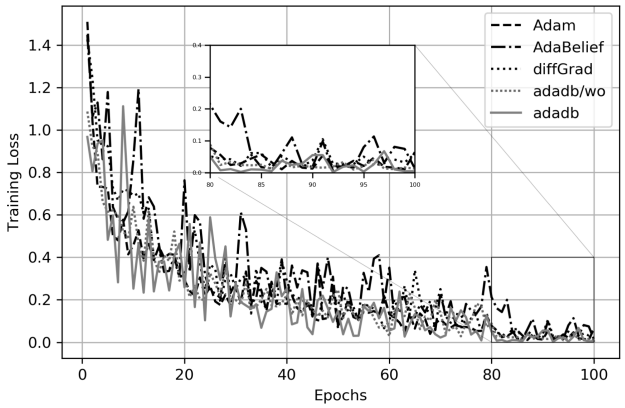

FIGURE 7. Training loss using the different optimizer techniques.

settles at the best available minima. Figure 7 and Figure 8 presents this phenomenon. Figure 7 shows the training loss per epoch whereas Figure 8 shows the testing accuracy per epoch against all the techniques on CIFAR10 dataset. The adadb shows the highest oscillation in training loss between epoch 1 and epoch 20, however, this oscillation gradually decreases as the number of epochs increases. The high oscillation helps in finding the best minima quickly and adadb shows the best performance in testing accuracy results.

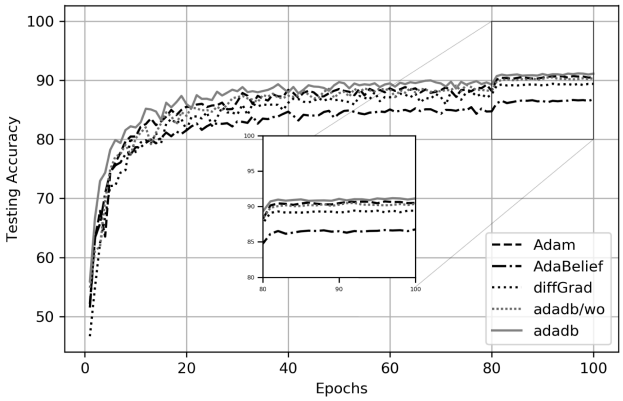

FIGURE 8. Testing accuracy using the different optimizer techniques.

IX. CONCLUSION

This paper presents a new optimization technique named as adaptive diff-batch or adadb for optimizing the gradient descent algorithm. The proposed technique removes the problem of slow convergence in the diffGrad algorithm by replacing the friction constant with three past differences of gradients rather than a single one. Moreover, the proposed technique presents a condition to decide when to use the friction constant. The technique performs faster than the diffGrad and finds a better solution than the Adam, and AdaBelief optimization techniques. The proposed technique is combined with the adaptive batch size to further increase the convergence rate. The proposed technique has outperformed both the Adam, AdaBelief, and diffGrad optimizers on synthetic complex non-convex functions and CIFAR10 & CIFAR100 datasets.

REFERENCES

[1] C. Lemaréchal, "Cauchy and the gradient method," *Doc Math Extra*, vol. 251, p. 254, Dec. 2012.
[2] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 2nd ed. New York, NY, USA: Academic, 2003.
[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[4] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. B. Chaudhuri, "DiffGrad: An optimization method for convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4500–4511, Nov. 2020.
[5] J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan, "Adabelief optimizer: Adapting stepsizes by the belief in observed gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 18795–18806.
[6] A. Devarakonda, M. Naumov, and M. Garland, "AdaBatch: Adaptive batch sizes for training deep neural networks," 2017, *arXiv:1712.02029*. [Online]. Available: http://arxiv.org/abs/1712.02029
[7] W. Khan, S. Ali, M. U. S. Khan, M. Jawad, M. Ali, and R. Nawaz, "AdaDiffGrad: An adaptive batch size implementation technique for diffgrad optimization method," in *Proc. 14th Int. Conf. Innov. Inf. Technol. (IIT)*, Nov. 2020, pp. 209–214.
[8] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.
[9] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Feb. 2011.
[10] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: http://arxiv.org/abs/1212.5701
[11] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, Video Lectures*, vol. 264, no. 1, pp. 2146–2153, 2012.
[12] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 2019, *arXiv:1904.09237*. [Online]. Available: http://arxiv.org/abs/1904.09237
[13] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, 2003, pp. 928–936.

MUHAMMAD U. S. KHAN (Member, IEEE) received the Ph.D. degree from North Dakota State University, Fargo, ND, USA. He is currently an Assistant Professor with COMSATS University Islamabad, Abbottabad Campus, Abbottabad, Pakistan. His research interests include cloud computing, data mining, big data, the IoT, network security, and artificial intelligence.

MUHAMMAD JAWAD received the Ph.D. degree from North Dakota State University, Fargo, ND, USA. He is currently an Assistant Professor with COMSATS University Islamabad, Lahore Campus, Lahore, Pakistan. His research interests include power systems, smart grids, renewable energy, power management, and optimization.

SAMEE U. KHAN (Senior Member, IEEE) received the Ph.D. degree from The University of Texas, in 2007. He is currently the Head of Department and a James W. Bagley Chair Professor in electrical and computer engineering with the Mississippi State University (MSU). Before arriving at MSU, he was a Cluster Lead of computer systems research with National Science Foundation, from 2016 to 2020. He was the Walter B. Booth Professor with North Dakota State University. His work has appeared in over 400 publications. His research interests include optimization, robustness, and security of computer systems. He is an Associate Editor of IEEE TRANSACTIONS ON CLOUD COMPUTING, *Journal of Parallel and Distributed Computing*, and *ACM Computing Surveys*.