Received: 6 June 2021 | Accepted: 27 January 2022

DOI: 10.1002/int.22854

RESEARCH ARTICLE

WILEY

Adaptive momentum with discriminative weight for neural network stochastic optimization

Jiyang Bai[1] | **Yuxiang Ren[2]** | **Jiawei Zhang[3]**

[1]Department of Computer Science, Florida State University, Tallahassee, Florida, USA

[2]Department of Computer Science, IFM Lab, Florida State University, Tallahassee, Florida, USA

[3]Department of Computer Science, IFM Lab, University of California, Davis, Davis, California, USA

Correspondence
Yuxiang Ren, Department of Computer Science, IFM Lab, Florida State University, Tallahassee, FL 32306, USA.
Email: yuxiang@ifmlab.org

Funding information
National Science Foundation, Grant/Award Number: IIS-1763365

Abstract

Optimization algorithms with momentum have been widely used for building deep learning models because of the fast convergence rate. Momentum helps accelerate Stochastic gradient descent in relevant directions in parameter updating, minifying the oscillations of the parameters update route. The gradient of each step in optimization algorithms with momentum is calculated by a part of the training samples, so there exists stochasticity, which may bring errors to parameter updates. In this case, momentum placing the influence of the last step to the current step with a fixed weight is obviously inaccurate, which propagates the error and hinders the correction of the current step. Besides, such a hyperparameter can be extremely hard to tune in applications as well. In this paper, we introduce a novel optimization algorithm, namely, *D*iscriminative w*E*ight on *A*daptive *M*omentum (DEAM). Instead of assigning the momentum term weight with a fixed hyperparameter, DEAM proposes to compute the momentum weight automatically based on the discriminative angle. The momentum term weight will be assigned with an appropriate value that configures momentum in the current step. In this way, DEAM involves fewer hyperparameters. DEAM also contains a

Should be considered joint first author.

novel backtrack term, which restricts redundant updates when the correction of the last step is needed. The backtrack term can effectively adapt the learning rate and achieve the anticipatory update as well. Extensive experiments demonstrate that DEAM can achieve a faster convergence rate than the existing optimization algorithms in training the deep learning models of both convex and nonconvex situations.

KEYWORDS

deep learning, momentum, neural network training, optimization algorithm, stochastic optimization

1 | INTRODUCTION

Deep learning methods can achieve outstanding performance in multiple fields, including computer vision,[1–4] natural language processing,[5,6] speech and audio processing,[7] and graph analysis.[8] Training deep learning models involves an optimization process to find the parameters that minimize the loss function. Simultaneously, the number of parameters commonly used in deep learning methods can be huge.

Therefore, optimization algorithms are critical for deep learning methods: not only the model performance but also training efficiency are greatly affected. To cope with the high computational complexity of training deep learning methods, stochastic gradient descent (SGD)[9] is utilized to update parameters based on the gradient of each training sample instead. The idea of momentum,[10] inspired by Newton's first law of motion, is used to handle the oscillations of SGD. SGD with momentum[11] achieves a faster convergence rate and better optimization results compared with the original SGD. In gradient descent-based optimization, training efficiency is also greatly affected by the learning rate. AdaGrad[12] is the first optimization algorithm with adaptive learning rates, which uses the learning rate decay. AdaDelta[13] subsequently improves AdaGrad to avoid the extremely small learning rates. Adaptive momentum (ADAM)[14] involves both adaptive learning[9] and momentum[10] and utilizes the exponential decay rate β_1 (momentum weight) to accelerate the convergence in the relevant directions and dampen oscillations. However, the decay rate β_1 of the first-order momentum $\mathbf{m}_t$ in ADAM is a fixed number, and the selection of the hyperparameter β_1 may affect the performance of ADAM greatly. Commonly, $\beta_1 = 0.9$ is the most widely used parameter as introduced in Reference [14], but there is still no theoretical evidence proving its advantages. We summarize the contributions of the methods mentioned in Table 1.

During the optimization process, it is common that there exist errors in some update steps. These errors can be caused by the inappropriate momentum calculation and then lead to slower convergence or oscillations. For each parameter update, the fixed momentum weight fails to take the different influences of the current gradient into consideration, rendering errors in momentum computing. For example, when there exist parts of opposite eigencomponents[10] between the continuous two parameter updates (we regard this situation as an error), the current gradient should be assigned a larger weight to correct the momentum in the last update

TABLE 1 Contributions of related works

Methods	Contributions
SGD with momentum[10,11]	Propose the momentum mechanism to handle the oscillations of SGD
AdaGrad[12]	Propose the adaptive learning rate decay
AdaDelta[13]	Avoid the extremely small learning rates in AdaGrad
ADAM[14]	Combine both adaptive learning rate and momentum mechanisms

Abbreviations: AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; SGD, stochastic gradient descent.

instead of being placed with a fixed influence. We will illustrate this problem through cases in Section 3.1.1 where ADAM with a fixed weight β_1 cannot handle some simple but intuitive convex optimization problems. On the basis of this situation, we need to control the influence of momentum by an adaptive weight. Moreover, designing hyperparameter-free optimization algorithms has been a critical research problem in recent years. Reducing the number of hyperparameters will not only stabilize the performance of the optimization algorithm but also release the workload of hyperparameters tuning.

In this paper, we introduce a novel optimization algorithm, namely, *D*iscriminative w*E*ight on *A*daptive *M*omentum (DEAM) to deal with the aforementioned problems. DEAM proposes an adaptive momentum weight $\beta_{1,t}$, which will be updated in each training iteration automatically. Besides, DEAM employs a novel backtrack term d_t, which will restrict redundant updates when DEAM decides that the correction of the previous step is needed. We also provide the theoretical analysis about the adaptive momentum weight along with extensive experiments. On the basis of them, we verify that the adaptive momentum term weight $\beta_{1,t}$ and the operation of backtrack term d_t can be crucial for the learning algorithms' performance.

Here, we summarize the detailed learning mechanism of DEAM as follows:

- DEAM computes adaptive momentum weight $\beta_{1,t}$ based on the "discriminative angle" θ between the historical momentum and the newly calculated gradient.
- DEAM introduces a novel backtrack term, that is, d_t, which is proposed to correct the redundant update of the previous training epoch when necessary. The calculation of d_t is also based on the discriminative angle θ.
- DEAM involves fewer hyperparameters than the ADAM during the training process, which can decrease the workload of hyperparameter tuning.

Detailed information about the learning mechanism and the concepts mentioned above will be described in the following sections. This paper will be organized as follows. In Section 2, we cover related works about widely used optimization algorithms. In Section 3, we analyze more detail of our proposed algorithm, whose theoretical convergence rate will also be studied. Extensive experiments are exhibited in Section 4. Finally, we give a conclusion of this paper in Section 6.

2 | PROBLEM DEFINITION AND RELATED WORKS

Function optimization: Given a differentiable function f and its domain $\mathcal{X}$, the function optimization is to find the optima point $x^* \in \mathcal{X}$ such that $\forall\ x \in \mathcal{X}, f(x^*) \leq f(x)$. For the neural network function optimization, the optimization algorithms aim at finding the optima point of neural networks: that is, the weights of network producing the smallest loss function value. Commonly, the optimization algorithms are designed based on the gradient descent algorithm. We summarize the notations used in this paper in Table 2.

SGD: SGD[9,15] performs variable updating for each training example $\mathbf{X}[i,:]$ and label $\mathbf{y}[i]$. (Here, $\mathbf{X}$ represents the training set matrix, every row of which is a training sample vector; $\mathbf{y}$ is the vector of all training samples' labels).

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \cdot \nabla f_t(\mathbf{X}[i,:], \mathbf{y}[i]; \mathbf{w}_t), \tag{1}$$

where η is the learning rate and ∇ is the derivative of the loss function. The advantages of SGD include fast converging speed compared with gradient descent and preventing redundancy.[9] Reddi et al.[16] use the variance reduction methods to accelerate the training process of SGD. The works of stochastic average gradient (SAG)[17] and stochastic dual coordinate ascent (SDCA)[18] can achieve a variance reduction effect for SGD that leads to a linear convergence rate. On the basis of them, stochastic variance reduced gradient (SVRG)[19] does not require the storage of gradients; SAGA[20] is with better theoretical convergence rates and supports non-strongly convex problems.

Adaptive learning rates: To overcome the problems brought by the unified learning rate, some variant algorithms applying adaptive learning rate[21] have been proposed, such as AdaGrad,[12] AdaDelta,[13] RMSProp,[22] ADAM,[14] recent ESGD,[23] and AdaBound.[24] AdaGrad adopts different learning rates to different variables, and its variable updating equation can be represented as

TABLE 2 Abbreviated notations used in the paper

Notation	Description
η_t	The learning rate at the tth training epoch
$\mathbf{m}_t$	The first-order momentum at the tth training epoch
γ	The weight of the first-order momentum in the SGD with momentum optimizer
$\mathbf{v}_t$	The second-order momentum in the RSMProp, ADAM, and DEAM optimizers at the tth training epoch
β_1	The weight of the previous first-order momentum when computing the updated first-order momentum in the ADAM optimizer
β_2	The weight of the previous second-order momentum when computing the updated first-order momentum in the ADAM and DEAM optimizers
$\beta_{1,t}$	The adaptive weight of the current gradient when computing the updated first-order momentum in the DEAM optimizer at the tth training epoch
Δ_t	The model weights update term in the DEAM optimizer at the tth training epoch
d_t	The backtrack term when computing the current weight update term Δ_t in the DEAM optimizer at the tth training epoch
α_d	The backtrack term coefficient
θ	The discriminative angle in the DEAM optimizer

Abbreviations: ADAM, adaptive momentum; DEAM, Discriminative wEight on Adaptive Momentum; SGD, stochastic gradient descent.

$$\mathbf{g}_t = \eta \cdot \nabla f_t(\mathbf{w}_t),$$
$$\mathbf{w}_t = \mathbf{w}_{t-1} - \frac{\mathbf{g}_t}{\sqrt{\sum_{i=1}^{t} \mathbf{g}_i \odot \mathbf{g}_i}}, \tag{2}$$

where η is the learning rate and ∇ is the derivative of the loss function. We have to mention that the $\sum$, $\odot$, and $\surd$ in the above equation are elementwise operations. One drawback of AdaGrad is that with the increase of iteration number t, the adaptive term $\sum_{i=1}^{t}\mathbf{g}_i \odot \mathbf{g}_i$ will inflate continuously, which will lead to a very slow convergence rate. RMSProp[22] can solve this problem by using the moving average of historical gradients. The update rule of RMSProp is shown as follows:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \cdot \mathbf{g}_t / \sqrt{\mathbf{v_t}},$$
$$\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1-\beta_2) \cdot \mathbf{g}_t \odot \mathbf{g}_t. \tag{3}$$

In the above equation, term β_2 is a hyperparameter in the interval $[0, 1]$. In this way, the adaptive term $\mathbf{v}_t$ will not increase continuously.

Momentum: Momentum[10,11,25–27] is a method that helps accelerate SGD in the relevant direction and discourage oscillations on the descent route. SGD with momentum updates variables with the following equations:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \mathbf{m}_t,$$
$$\mathbf{m}_t = \gamma \cdot \mathbf{m}_{t-1} + \eta \cdot \nabla f_t(\mathbf{w}_t). \tag{4}$$

In the equation, γ is the weight of the momentum, and η is the learning rate. The momentum accelerates updates for dimensions whose gradients are in the same direction as historical gradients, and reduces updates for dimensions whose gradients are the opposite. Momentum is also applied in Nesterov accelerated gradient (NAG),[28] which can be presented as

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \mathbf{m}_t,$$
$$\mathbf{m}_t = \gamma \mathbf{m}_{t-1} + \eta \cdot \nabla f_t(\mathbf{w}_{t-1} - \gamma \mathbf{m}_{t-1}). \tag{5}$$

ADAM[14,29] is proposed based on SGD and momentum concept, and it also computes individual adaptive learning rates for different variables. The variable updating rules in ADAM can be represented by the following equations:

$$\begin{cases} \mathbf{g}_t = \nabla f_t(\mathbf{w}), \\ \mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1-\beta_1) \cdot \mathbf{g}_t; \quad \hat{\mathbf{m}}_t = \mathbf{m}_t / \left(1 - \beta_1^t\right), \\ \mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1-\beta_2) \cdot \mathbf{g}_t \odot \mathbf{g}_t; \quad \hat{\mathbf{v}}_t = \mathbf{v}_t / \left(1 - \beta_2^t\right), \\ \mathbf{w}_t = \mathbf{w}_{t-1} - \eta \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon). \end{cases} \tag{6}$$

ADAM records the first-order momentum $\mathbf{m}_t$ and the second-order momentum $\mathbf{v}_t$ of the gradients using the moving average (controlled by the parameters β_1 and β_1, respectively), and further computes the bias-corrected version of them ($\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$). On the basis of ADAM, Keskar and Socher[30] propose to switch from ADAM to SGD during the training process. In this way, it can combine the advantages of both SGD and ADAM.

AMSGrad[31] is a modified version of ADAM. AMSGrad changes the definition of the second-order momentum by $\hat{\mathbf{v}}_t = \max\{\hat{\mathbf{v}}_{t-1}, \mathbf{v}_t\}$, and other settings are almost the same as ADAM. This formula is to make sure that the second moment will always increase along with t, which ensures the decreasing of step size. What is more, AMSGrad applies a varied learning rate η_t comparing to ADAM, but the definition of η_t is not specified.

DEAM is an optimization algorithm that involves the adaptive momentum weights and backtrack mechanisms, and is first proposed in Reference [32]. In this paper, we further explore the effectiveness of DEAM by giving a detailed theoretical analysis on the convergence and comprehensive experiments on more types of models (e.g., Graph Neural Networks and RNNs). Both the theoretical convergence analysis and the experimental results further demonstrate the validity of DEAM.

Algorithm convergence: Most of the machine learning and deep learning tasks are under nonconvex conditions. However, most convergence analysis of the mentioned optimization algorithms is based on convex situations. Chen et al.[33] give a convergence rate of order $O(\log T/\sqrt{T})$ for nonconvex stochastic optimization with respect to the ADAM-type methods.

3 | PROPOSED ALGORITHM

Algorithm 1: DEAM Algorithm

Input: loss function $f(\mathbf{w})$ with parameters $\mathbf{w}$; learning rate $\{\eta_t\}_{t=1}^{T}$; $\beta_2 = 0.999$

Output: trained parameters

```
m_0 ← 0;  /* Initialize first-order momentum. */
v_0 ← 0, v̂_0 ← 0;  /* Initialize second-order momentum. */
for t = 1, 2, ..., T do
    g_t = ∇f_t(w_t);
    θ = ⟨ m_{t-1}/√(v̂_{t-1}), g_t ⟩;  /* The operator ⟨·,·⟩ represents the angle between two vectors. */
    if θ ∈ [0, π/2) then
        β_{1,t} = sin θ/K + ε;
    else
        β_{1,t} = 1/K   /* Here, K = 5(2+π)/π. */;
    end
    m_t = (1 − β_{1,t}) · m_{t-1} + β_{1,t} · g_t;
    v_t = β_2 · v_{t-1} + (1 − β_2) · g_t ⊙ g_t;  /* ⊙ is element-wise multiplication.*/
    v̂_t = max{v̂_{t-1}, v_t};
    d_t = min{α_d cos θ, 0}   /* Here, α_d = 0.5 in default. */;
    Δ_t = d_t · Δ_{t-1} − η_t · m_t/√(v̂_t);
    w_t = w_{t-1} + Δ_t;
end
return w_T
```

Our proposed algorithm DEAM is presented in Algorithm 1. In the algorithm, $f_1, f_2, \ldots, f_T$ is a sequence of loss functions computed with the training minibatches in different iterations (or epochs). DEAM introduces two new terms in the learning process: (1) the adaptive momentum weight $\beta_{1,t}$, and (2) the "backtrack term" d_t. In the tth training iteration, both $\beta_{1,t}$ and d_t are calculated based on the "discriminative angle" θ, which is the angle between previous $\mathbf{m}_{t-1}/\sqrt{\hat{\mathbf{v}}_{t-1}}$ and current gradient $\mathbf{g}_t$ (since essentially both $\mathbf{m}_{t-1}/\sqrt{\hat{\mathbf{v}}_{t-1}}$ and $\mathbf{g}_t$ are vectors, there exists an angle between them). Here, $\mathbf{m}$ is the first-order momentum that records the exponential moving average of historical gradients; $\mathbf{v}$ is the exponential moving average of the squared gradients, which is called the second-order momentum. In the following parts of this paper, we will denote $\mathbf{m}_{t-1}/\sqrt{\hat{\mathbf{v}}_{t-1}}$ as the "update volume" in the $(t-1)$th iteration. Formally, $\beta_{1,t}$ determines the weights of the previous first-order momentum $\mathbf{m}_{t-1}$ and current gradient $\mathbf{g}_t$ when calculating the present $\mathbf{m}_t$. Meanwhile, the backtrack term d_t represents the returning step of the previous update on parameters. We can notice that in each iteration, after the θ has been calculated, the $\beta_{1,t}$ and d_t are directly obtained according to the θ. In this way, we can calculate appropriate $\beta_{1,t}$ as the discriminative angle changes. The d_t term balances between the historical update term Δ_{t-1} (defined in Algorithm 1) and the current update volume $\mathbf{m}_t/\sqrt{\hat{\mathbf{v}}_t}$ when computing Δ_t. In the proposed DEAM, $\beta_{1,t}$ and d_t terms can collaborate with each other and achieve faster convergence.

3.1 | Adaptive momentum weight $\beta_{1,t}$

3.1.1 | Motivation

In the ADAM[14] paper, (the first-order) momentum's weight (i.e., β_1) is a prespecified fixed value, and commonly $\beta_1 = 0.9$. It has been used in many applications and the performance can usually meet the expectations. However, this setting is not applicable in some situations. For example, for the case

$$f(x, y) = x^2 + 4y^2, \tag{7}$$

where x and y are two variables, it is obvious that f is a convex function. If $f(x, y)$ is the objective function to optimize, we try to use ADAM to find its global optima.

Let us assume ADAM starts the variable search from $(-4, -1)$ (i.e., the initial variable vector is $\mathbf{w}_0 = (-4, -1)^\top$) and the initial learning rate is $\eta_1 = 1$. Different choices of β_1 will lead to a very different performance of ADAM. For instance, in Figure 1, we illustrate the update routes of ADAM with $\beta_1 = 0.9$ and $\beta_1 = 0.0$ as the blue and red lines, respectively. In Figure 1, the ellipse lines are the contour lines of $f(x, y)$, and points on the same line share the same function value. We can observe that after the first updating, both of the two approaches will update variables to $(-3, 0)$ point (i.e., the updated variable vector will be $\mathbf{w}_1 = (-3, 0)^\top$). In the second step, since the current gradient $\mathbf{g}_2 = (-6, 0)^\top$, the ADAM with $\beta_1 = 0.0$ will update variables in the $(1, 0)$ direction. Meanwhile, for the ADAM with $\beta_1 = 0.9$, its $\mathbf{m}_2$ is computed by integrating $\mathbf{m}_1$ and $\mathbf{g}_2$ together (whose weights are β_1 and $1 - \beta_1$, respectively). Therefore the updating direction of it will be more inclined to the previous direction instead. Compared with ADAM with $\beta_1 = 0.0$, the ADAM with $\beta_1 = 0.9$ takes much more iterations until converging.

From the analysis above, we can observe that a careful tuning and updating of β_1 in the learning process can be crucial for the performance of ADAM. However, by this context so far, there still exist no effective approaches for guiding the parameter tuning yet. To deal with this problem, DEAM introduces the concept of discriminative angle θ for computing β_1 automatically as follows.

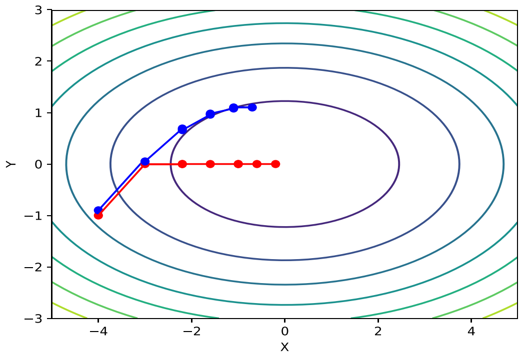

FIGURE 1 The update routes of ADAM with $\beta_1 = 0.9$ (the blue line) and $\beta_1 = 0.0$ (the red line). ADAM, adaptive momentum [Color figure can be viewed at wileyonlinelibrary.com]

3.1.2 | Mechanism

The momentum weight β_1 will be updated in each iteration in DEAM, and we can denote its value computed in the tth iteration as $\beta_{1,t}$ formally. Essentially, in the tth iteration of the training process, both the previous update volume and $\mathbf{g}_t$ are vectors (or directions), and these directions directly decide the updating process. Thus we try to extract their relation with the help of angle, and subsequently determine the weight $\beta_{1,t}$ (or $1 - \beta_{1,t}$) by the angle.

In Algorithm 1, the discriminative angle θ in the tth iteration is calculated by

$$\theta = \left\langle -\frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, -\mathbf{g}_t \right\rangle = \left\langle \frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, \mathbf{g}_t \right\rangle. \tag{8}$$

Here, the operator $\langle \cdot, \cdot \rangle$ denotes the angle between two vectors (the angle is calculated according to the cosine similarity). This expression is easy to understand, since the $-\mathbf{m}_{t-1}/\sqrt{\hat{\mathbf{v}}_{t-1}}$ can represent the updating direction of $(t-1)$th iteration in AMSGrad, meanwhile $-\mathbf{g}_t$ is the reverse of the present gradient. So we can simplify it as $\theta = \left\langle \frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, \mathbf{g}_t \right\rangle$. If θ is close to zero (denoted by $\theta \to 0°$), the $\mathbf{m}_{t-1}/\sqrt{\hat{\mathbf{v}}_{t-1}}$ (previous update volume) and $\mathbf{g}_t$ are almost in the same direction, and the weights for them will not be very important. Meanwhile, if θ approaches 180° (denoted by $\theta \to 180°$), the previous update volume and $\mathbf{g}_t$ will be in totally reverse directions. This means in the current step, the previous momentum term is already in a wrong direction. Therefore, to rectify this error of the last momentum, DEAM proposes to assign the current gradient's weight (i.e., $\beta_{1,t}$ in our paper) with a larger value instead. As the $\beta_{1,t}$ varies when θ changes from 0° to 180°, we intend to define $\beta_{1,t}$ with the following equation:

$$\beta_{1,t} = \begin{cases} \sin\theta/K + \epsilon, & \theta \in \left[0, \frac{\pi}{2}\right), \\ 1/K, & \theta \in \left[\frac{\pi}{2}, \pi\right], \end{cases} \tag{9}$$

where $K = 10(2+\pi)/2\pi$ and ϵ is a very small value (e.g., $\epsilon = 0.001$). In the equation above, the threshold of the piecewise function is $\theta = \pi/2$, because $\sin\theta$ comes to the maximum at this point and goes down when $\theta > \frac{\pi}{2}$. If $\frac{\pi}{2} \le \theta \le \pi$, which is exactly the situation $\theta \to 180°$ we discussed above, we intend to keep $\beta_{1,t}$ in a relatively large value. The reason we rescale $\sin\theta$ by $1/K$ is that directly applying $\beta_{1,t} = \sin\theta$ will overweight $\mathbf{g}_t$, which may cause fluctuations on the update routes. The value of K is determined by

$$K = \frac{10}{\pi}\left(\int_0^{\frac{\pi}{2}} \sin\theta \, d\theta + \int_{\frac{\pi}{2}}^{\pi} 1 \, d\theta\right) = \frac{5(2+\pi)}{\pi}. \tag{10}$$

In the equation above, assume θ is randomly distributed on $[0, \pi]$. Here, we specify $K = \frac{5(2+\pi)}{\pi}$ in this calculation so that we can get

$$\mathbb{E}[\beta_{1,t}] = \frac{1}{\pi}\int_0^{\pi} \beta_{1,t}(\theta) \, d\theta = 0.1. \tag{11}$$

In other words, the expectation of $\beta_{1,t}$ (i.e., $\mathbb{E}(\beta_{1,t})$) will be identical to the β_1 used in ADAM paper.[14] After obtaining $\beta_{1,t}$, it will be applied to calculating $\mathbf{m}_t$ as shown in Algorithm 1. In this way, we have achieved momentum with adaptive weights, and this weight is automatically computed during the training process, fewer hyperparameters will be involved.

3.2 | Backtrack mechanism d_t

To further speed up the convergence rate, we employ a novel backtrack mechanism for DEAM. As a mechanism computed based on the discriminative angle θ, the backtrack term allows DEAM to eliminate redundant update in each iteration. Besides, according to our following analysis, the backtrack term d_t virtually collaborates with the $\beta_{1,t}$ term to further accelerate the convergence of the training process.

3.2.1 | Motivation

When optimizer (e.g., ADAM) updates variables of the loss function (e.g., $f(x, y)$), some update routes will look like the black arrow lines shown in Figure 2A, especially when the discriminative angle θ is larger than 90°. We call this phenomenon the "zig-zag" route. In Figure 2A, it shows the update routes of a two-dimensional function. Each black arrow line in the figure represents the variables' update in each epoch; the red dashed line is the direction of the update routes; the θ is the discriminative angle. If $\theta \ge 90°$, the "zig-zag" phenomenon will appear severely, which may lead to slower convergence speed. The main reason is when $\theta \ge 90°$, if we map two neighboring update directions onto the coordinate axes, there will be at least one axis of the directions being opposite. This situation is shown in Figure 2B. For the example of a function with two-dimensional variables, the update volume $\mathbf{m}_1/\sqrt{\hat{\mathbf{v}}_1}$ can be decomposed into $(x_1, y_1)^\top$ in Figure 2B, and the same with $\mathbf{m}_2/\sqrt{\hat{\mathbf{v}}_2}$. We can notice that y_1 and y_2 are in the opposite directions, so the first and second steps practically have inverse updates subject to the y-axis. We attribute this situation to the overupdate (or redundant update) of the first step. Therefore the backtrack term d_t is proposed to restrict this situation.

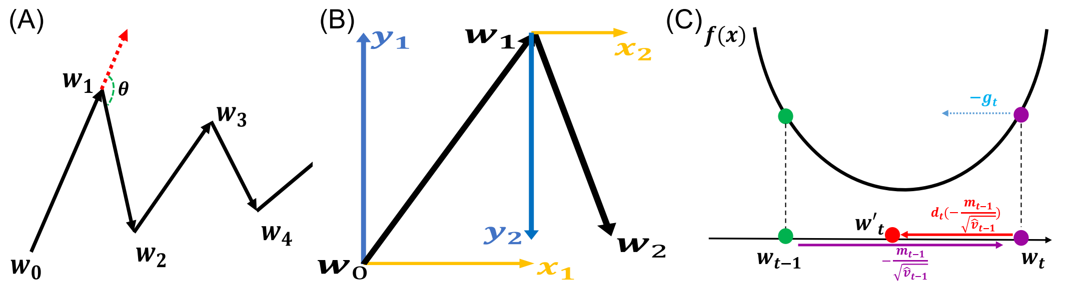

FIGURE 2 Examples about d_t. (A) The update routes of a two-dimensional function. The θ in the figure denotes the angle between the update direction from w_0 to w_1 and the update direction from w_1 to w_2; (B) an example of the opposite axis directions in the update routes of (A); (C) the mechanism of the backtrack term d_t [Color figure can be viewed at wileyonlinelibrary.com]

3.2.2 | Mechanism

Since the redundant update situation is caused by over updating of the previous iteration, simply we intend to deal with it through a backward step. Meanwhile, during the updating process of variables, not every step will suffer from the redundant update: if $\theta \to 0°$, the updating process becomes smooth, not like the situation shown in Figure 2A. Besides, from the analysis above we conclude that if $\theta \geq 90°$, there will be at least one dimension involves the redundant update. Thus, in the tth iteration we quantify d_t as the following equation:

$$d_t = \min\{\alpha_d \cos\theta, 0\} \tag{12}$$

and we rewrite the updating term with backtrack in DEAM as

$$\Delta_t = d_t \cdot \Delta_{t-1} - \eta_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}, \tag{13}$$

where θ is the discriminative angle, α_d equals to 0.5 in our default setting, and Δ_t is the updating term in Algorithm 1. By designing d_t in this way, when $\theta \to 0°$, $d_t = 0$ and there is no backward step, the updating term $\Delta_t = -\eta_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}$ is similar to AMSGrad; when $\theta \to 180°$, d_t equals to $0.5\cos\theta$ and comes to the maximum value when $\theta = 180°$. In Equation (12), $\cos\theta$ is rescaled by α_d, the reason of our default setting $\alpha_d = 0.5$ is that: in Figure 2C, $\mathbf{w}_{t-1}$ and $\mathbf{w}_t$ are the variables updated by DEAM without d_t term in the $(t-1)$th and tth iterations, respectively. If the backtrack mechanism is implemented, in the $(t+1)$th iteration, since $\theta = 180°$, first $d_t = \alpha_d \cos\theta \to -0.5$ makes the backtrack to the $\mathbf{w}_t^{'}$ point (the middle point of $\mathbf{w}_{t-1}$ and $\mathbf{w}_t$). Thus, this backtrack step allows the variable to further approach the optima. For more complicated situations, since it is too hard to find the optimal α_d value for every specific learning task, we use the following expectation to set the default value of α_d. In the tth iteration, considering when $\theta \to 180°$, $d_t < 0$ and $w_t^{'}$ should locate between the w_{t-1} and w_t. As the optimal relative location of $w_t^{'}$ is unknown, we assume that $w_t^{'}$ is randomly distributed between w_{t-1} and w_t. Thus, the statistical expectation of the location of $w_t^{'}$ is the central point between w_{t-1} and w_t. In other words, d_t should be -0.5 when $\theta \to 180°$, and $\alpha_d = 0.5$ is the optimal choice under our backtrack mechanism.

By implementing the backtrack term d_t, DEAM can combine it with the adaptive momentum weight $\beta_{1,t}$ to achieve the collaborating of them. For the situation of large discriminative angle ($\theta \geq 90°$), both $\beta_{1,t}$ and d_t in the current step can make corrections to the last update. Since when $\theta \geq 90°$, the last update is in conflict direction compared with the current gradient, and $\beta_{1,t}$ will increase to allocate a large weight for the present gradient, which subsequently corrects the previous step. Meanwhile, the d_t will also conduct a backward step of to further rectify the last update.

3.3 | Theoretical analysis

In this part, we give the detailed analysis on the convergence of our DEAM algorithm. According to References [14,31,33,34], given an arbitrary sequence of convex objective functions $f_1(\mathbf{w}), f_2(\mathbf{w}), ..., f_T(\mathbf{w})$, we intend to evaluate our algorithm using the regret function, which is denoted as

$$R(T) = \sum_{t=1}^{T} [f_t(\mathbf{w}_t) - f_t(\mathbf{w}^*)], \tag{14}$$

where $\mathbf{w}^*$ is the globally optimal point. In the following Theorem 1, we will show that the above regret function is bounded. Before proving Theorem 1, there are some properties and lemmas as the prerequisites.

Proposition 1. *If a function $f: \mathbb{R}^d \to \mathbb{R}$ is convex, then $\forall x, y \in \mathbb{R}^d$, $\forall \phi \in [0, 1]$, we have*

$$f(\phi x + (1-\phi)y) \leq \phi f(x) + (1-\phi)f(y).$$

Proposition 2. *If a function $f: \mathbb{R}^d \to \mathbb{R}$ is convex, then $\forall x, y \in \mathbb{R}^d$ we have*

$$f(y) \geq f(x) + \nabla f(x)^{\top}(y - x).$$

Lemma 1. *Assume that the function f_t has bounded gradients, $\|\nabla f_t(\mathbf{w})\|_\infty \leq G_\infty$. Let $\mathbf{m}_{t,i}$ represents the* ith *element of $\mathbf{m}_t$ in DEAM, then the $\mathbf{m}_{t,i}$ is bounded by*

$$\mathbf{m}_{t,i} \leq \frac{(1-\epsilon_0)G_\infty}{K(1-\lambda)},$$

where ϵ_0 and λ are defined in Theorem 1.

Proof. Let $g_t = \nabla f_t(\mathbf{w})$. According to the definition of $\mathbf{m}_{t,i}$ in our algorithm,

$$\begin{aligned}\mathbf{m}_{t,i} &= \sum_{j=1}^{t} \beta_{1,j} \prod_{l=1}^{t-j} (1-\beta_{1,t-l+1}) g_{j,i} \\ &\le \frac{G_\infty}{K} \sum_{j=1}^{t} \prod_{l=1}^{t-j} (1-\epsilon) \le \frac{G_\infty}{K} \sum_{j=1}^{t} (1-\epsilon_0)\lambda^{t-j} \\ &\le \frac{(1-\epsilon_0)G_\infty}{K(1-\lambda)},\end{aligned}$$

where K and ϵ are the terms in Algorithm 1. □

For the following proof, $\mathbf{g}_t := \nabla f_t(\mathbf{w}_t)$ and $\mathbf{g}_{t,i}$ will represent the ith element of $\mathbf{g}_t \in \mathbb{R}^d$, and $\mathbf{g}_{1:t,i} = [\mathbf{g}_{1,i}, \mathbf{g}_{2,i}, \ldots, \mathbf{g}_{t,i}]$.

Theorem 1. *Assume $\{f_t\}_{t=1}^{T}$ have bounded gradients $\|\nabla f_t(\mathbf{w})\|_\infty \le G_\infty$ for all $\mathbf{w} \in \mathbb{R}^d$, all variables are bounded by $\|\mathbf{w}_p - \mathbf{w}_q\|_2 \le D$ and $\|\mathbf{w}_p - \mathbf{w}_q\|_\infty \le D_\infty$, $\forall\ p, q \in \{1, 2, \ldots, T\}$, $\eta_t = \eta/\sqrt{t}$, $\gamma_1 = (1-\epsilon_0)/\sqrt{\beta_2}$ and satisfies $\gamma_1 < 1$, $\epsilon = 1-(1-\epsilon_0)\lambda^{t-1}, \lambda \in (0,1)$. Our proposed algorithm can achieve the following bound on regret:*

$$\begin{aligned}R(T) \le \frac{D^2}{\epsilon_0 \eta} \sum_{i=1}^{d} \sqrt{T\hat{\mathbf{v}}_{T,i}} + \frac{(1-\epsilon_0)^2 G_\infty D_\infty d}{K(1-\lambda)^2 \epsilon_0} \\ + \frac{\eta\sqrt{1+\log T}}{2\epsilon_0^2(1-\gamma_1)\sqrt{1-\beta_2}} \sum_{i=1}^{d} \|\mathbf{g}_{1:T,i}\|_2.\end{aligned}$$

Proof. According to Proposition 2, for $\forall\ t \in \{1, 2, \ldots, T\}$, we have

$$\begin{aligned}f_t(\mathbf{w}_t) - f_t(\mathbf{w}^*) &\le \nabla f_t(\mathbf{w}_t)^{\mathrm{T}}(\mathbf{w}_t - \mathbf{w}^*) \\ &= \sum_{i=1}^{d} \mathbf{g}_{t,i}\left(\mathbf{w}_{t,i} - \mathbf{w}_i^*\right)\end{aligned}$$

From the definition of Δ_t in the updating rule of DEAM, we know it is equal to multiplying the learning rate η_t in some iterations by a number in $[0.5, 1]$, which means $\mathbf{w}_{t+1} = \mathbf{w}_t - \hat{\eta}_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}; \hat{\eta}_t = \mu_t \cdot \eta_t$, where $\mu_t \in [0.5, 1]$. Thus if we first focus on the ith element of $\mathbf{w}_t$, we can get

$$\begin{aligned}\left(\mathbf{w}_{t+1,i} - \mathbf{w}_i^*\right)^2 = \left(\mathbf{w}_{t,i} - \mathbf{w}_i^* - \hat{\eta}_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}\right)^2 = \left(\mathbf{w}_{t,i} - \mathbf{w}_i^*\right)^2 \\ -2\hat{\eta}_t\left(\frac{(1-\beta_{1,t})}{\sqrt{\hat{\mathbf{v}}_{t,i}}}\mathbf{m}_{t-1,i} + \frac{\beta_{1,t}}{\sqrt{\hat{\mathbf{v}}_{t,i}}}\mathbf{g}_{t,i}\right) \cdot \left(\mathbf{w}_{t,i} - \mathbf{w}_i^*\right) + \left(\hat{\eta}_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}\right)^2.\end{aligned}$$

Then,

$$2\hat{\eta}_t \cdot \frac{\beta_{1,t}}{\sqrt{\hat{\mathbf{v}}_{t,i}}}\mathbf{g}_{t,i}\left(\mathbf{w}_{t,i}-\mathbf{w}_i^*\right)=\left(\mathbf{w}_{t,i}-\mathbf{w}_i^*\right)^2-\left(\mathbf{w}_{t+1,i}-\mathbf{w}_i^*\right)^2 -2\hat{\eta}_t \cdot \frac{(1-\beta_{1,t})}{\sqrt{\hat{\mathbf{v}}_{t,i}}}\mathbf{m}_{t-1,i}\cdot\left(\mathbf{w}_{t,i}-\mathbf{w}_i^*\right)+\hat{\eta}_t^2\cdot\frac{\mathbf{m}_{t,i}^2}{\hat{\mathbf{v}}_{t,i}}.$$

So we can obtain

$$\mathbf{g}_{t,i}\left(\mathbf{w}_{t,i}-\mathbf{w}_i^*\right)=\frac{\sqrt{\hat{\mathbf{v}}_{t,i}}}{2\hat{\eta}_t\beta_{1,t}}\left[\left(\mathbf{w}_{t,i}-\mathbf{w}_i^*\right)^2-\left(\mathbf{w}_{t+1,i}-\mathbf{w}_i^*\right)^2\right] \tag{15}$$

$$+\frac{(1-\beta_{1,t})}{\beta_{1,t}}\mathbf{m}_{t-1,i}\left(\mathbf{w}_i^*-\mathbf{w}_{t,i}\right) \tag{16}$$

$$+\frac{\hat{\eta}_t}{2\beta_{1,t}}\cdot\frac{\mathbf{m}_{t,i}^2}{\sqrt{\hat{\mathbf{v}}_{t,i}}}. \tag{17}$$

For the right-hand part of (15) in the above formula, if we sum it from $t=1$ to $t=T$,

$$\begin{aligned}&\sum_{t=1}^{T}\frac{\sqrt{\hat{\mathbf{v}}_{t,i}}}{2\hat{\eta}_t\beta_{1,t}}\left[\left(\mathbf{w}_{t,i}-\mathbf{w}_i^*\right)^2-\left(\mathbf{w}_{t+1,i}-\mathbf{w}_i^*\right)^2\right]\\ &\le\frac{1}{\epsilon_0}\left\{\left(\mathbf{w}_{1,i}-\mathbf{w}_i^*\right)^2\cdot\frac{\sqrt{\hat{\mathbf{v}}_{1,i}}}{\eta_1}+\cdots\right.\\ &\left.+\left(\mathbf{w}_{T,i}-\mathbf{w}_i^*\right)^2\left(\frac{\sqrt{\hat{\mathbf{v}}_{T,i}}}{\eta_T}-\frac{\sqrt{\hat{\mathbf{v}}_{T-1,i}}}{\eta_{T-1}}\right)\right\}\\ &\le\frac{D^2}{\epsilon_0\eta}\sqrt{T\hat{\mathbf{v}}_{T,i}}.\end{aligned}$$

The first inequality is satisfied because of the $\hat{\mathbf{v}}_t=\max\{\hat{\mathbf{v}}_{t-1},\mathbf{v}_t\}$ in Algorithm 1. For Equation (16) in the formula, if we sum it from $t=1$ to $t=T$,

$$\begin{aligned}&\sum_{t=1}^{T}\frac{(1-\beta_{1,t})}{\beta_{1,t}}\mathbf{m}_{t-1,i}\left(\mathbf{w}_i^*-\mathbf{w}_{t,i}\right)\le\frac{(1-\epsilon_0)G_\infty D_\infty}{K(1-\lambda)\epsilon_0}\sum_{t=1}^{T}(1-\beta_{1,t})\\ &\le\frac{(1-\epsilon_0)G_\infty D_\infty}{K(1-\lambda)\epsilon_0}\sum_{t=1}^{T}(1-\epsilon)=\frac{(1-\epsilon_0)G_\infty D_\infty}{K(1-\lambda)\epsilon_0}\sum_{t=1}^{T}(1-\epsilon_0)\lambda^{t-1}\\ &\le\frac{(1-\epsilon_0)^2G_\infty D_\infty}{K(1-\lambda)^2\epsilon_0}.\end{aligned}$$

The first inequality is according to Lemma 1. Finally, we will infer Equation (17) in previous formula. According to the Lemma 2 of Reference [31], we have

$$\sum_{t=1}^{T} \frac{\hat{\eta}_t}{2\beta_{1,t}} \cdot \frac{\mathbf{m}_{t,i}^2}{\sqrt{\hat{\mathbf{v}}_{t,i}}} \le \frac{1}{2\epsilon_0} \sum_{t=1}^{T} \eta_t \frac{\mathbf{m}_{t,i}^2}{\sqrt{\mathbf{v}_{t,i}}} \le \frac{\eta}{2\epsilon_0} \sum_{t=1}^{T} \frac{1}{\sqrt{t}} \cdot \frac{\mathbf{m}_{t,i}^2}{\sqrt{\mathbf{v}_{t,i}}}$$

$$= \frac{\eta}{2\epsilon_0} \sum_{t=1}^{T} \frac{\left(\sum_{j=1}^{t} \beta_{1,j} \prod_{l=1}^{t-j} (1-\beta_{1,t-l+1}) \mathbf{g}_{j,i}\right)^2}{\sqrt{t\left((1-\beta_2)\sum_{j=1}^{t} \beta_2^{t-j} \mathbf{g}_{j,i}^2\right)}}$$

$$\le \frac{\eta}{2\epsilon_0} \sum_{t=1}^{T} \frac{\left(\sum_{j=1}^{t} (1-\epsilon_0)^{t-j}\right)\left(\sum_{j=1}^{t} (1-\epsilon_0)^{t-j} \mathbf{g}_{j,i}^2\right)}{\sqrt{t\left((1-\beta_2)\sum_{j=1}^{t} \beta_2^{t-j} \mathbf{g}_{j,i}^2\right)}}$$

$$\le \frac{\eta}{2\epsilon_0^2\sqrt{1-\beta_2}} \sum_{t=1}^{T} \frac{\sum_{j=1}^{t} (1-\epsilon_0)^{t-j} \mathbf{g}_{j,i}^2}{\sqrt{t\left(\sum_{j=1}^{t} \beta_2^{t-j} \mathbf{g}_{j,i}^2\right)}}$$

$$\le \frac{\eta}{2\epsilon_0^2\sqrt{1-\beta_2}} \sum_{t=1}^{T} \frac{1}{\sqrt{t}} \sum_{j=1}^{t} \frac{(1-\epsilon_0)^{t-j} \mathbf{g}_{j,i}^2}{\sqrt{\beta_2^{t-j} \mathbf{g}_{j,i}^2}}$$

$$\le \frac{\eta}{2\epsilon_0^2\sqrt{1-\beta_2}} \sum_{t=1}^{T} |\mathbf{g}_{t,i}| \sum_{j=t}^{T} \frac{\gamma_1^{j-t}}{\sqrt{t}} \le \frac{\eta\sqrt{1+\log T}}{2\epsilon_0^2(1-\gamma_1)\sqrt{1-\beta_2}} \|\mathbf{g}_{1:T,i}\|_2.$$

In the above inequalities, some inferences are based on Cauchy–Schwarz Inequality. Therefore, the final bound of $R(T)$ can be expressed as

$$R(T) \le \frac{D^2}{\epsilon_0 \eta} \sum_{i=1}^{d} \sqrt{T\hat{\mathbf{v}}_{T,i}} + \frac{(1-\epsilon_0)^2 G_\infty D_\infty d}{K(1-\lambda)^2 \epsilon_0} + \frac{\eta\sqrt{1+\log T}}{2\epsilon_0^2(1-\gamma_1)\sqrt{1-\beta_2}} \sum_{i=1}^{d} \|\mathbf{g}_{1:T,i}\|_2. \qquad \square$$

For the bound term, as $T \to +\infty$, $\frac{R(T)}{T} \to 0$ and we can infer that $\lim_{T\to\infty}[f_t(\mathbf{w}_t) - f_t(\mathbf{w}^*)] = 0$, which means the proposed algorithm can finally converge.

4 | EXPERIMENTS

We have applied the DEAM algorithm on multiple popular machine learning and deep learning structures, including logistic regression (LR), deep neural networks (DNNs), convolutional neural networks (CNNs), graph convolutional networks (GCNs), and recurrent neural networks (RNNs). These structures cover both convex and nonconvex situations. Through experiments, we demonstrate that DEAM has universal advantages for different types of machine learning structures. Below we introduce the experimental results in detail for each learning structure.

4.1 | Comparison algorithms

To show the advantages of the algorithm, we compare it with various popular optimization algorithms.

- *DEAM*: DEAM is the proposed algorithm in this paper.
- *DEAM without* d_t: We remove the backtrack mechanism from DEAM to verify the effectiveness of d_t by ablation study.
- *ADAM*[14]: ADAM is an algorithm for the first-order gradient-based optimization based on adaptive estimates of lower-order moments. But the momentum is controlled by hypermeters (i.e., β_1 and β_2).
- *RMSProp*[22]: RMSprop belongs to the realm of adaptive learning rate algorithms.
- *AdaGrad*[12]: AdaGrad adapts the learning rate to the parameters, which strategy is setting low learning rates for parameters associated with frequently occurring features but high learning rates for parameters associated with infrequent features.
- *SGD*[9]: SGD performs a parameter update for each training example, which leads to more frequent parameter updates but more fluctuated objective functions.

To ensure fairness, we use the same parameter initialization when testing each optimization method and fine-tune the hyperparameters (e.g., learning rate and decay weight) of each optimization method and report the best results. The experimental device is a Dell PowerEdge T630 Tower Server, with 80 cores 64-bit Intel Xeon CPU E5-2698 v4@2.2 GHz. The total memory is 256 GB, with an extra (SSD) swap of 256 GB. The operating system is Ubuntu 16.04.3, and all codes are implemented in Python.

4.2 | Experiments in LR

We first evaluate our algorithm on the multiclass LR model since it is widely used and owns a convex objective function.[14] We conduct LR on the ORL data set.[35] During the training process, the minibatch size is 128, and the learning rate is 0.0001. ORL data set consists of face images of 40 people, each person has 10 images, and each image is in the size of 112 × 92. The loss of objective functions on both the training set and testing set is shown in Figure 3. From the figure, it can be found that DEAM has obtained the fastest convergence rate with apparent advantages and converges to the global minimum. From the running time listed in Table 3, the

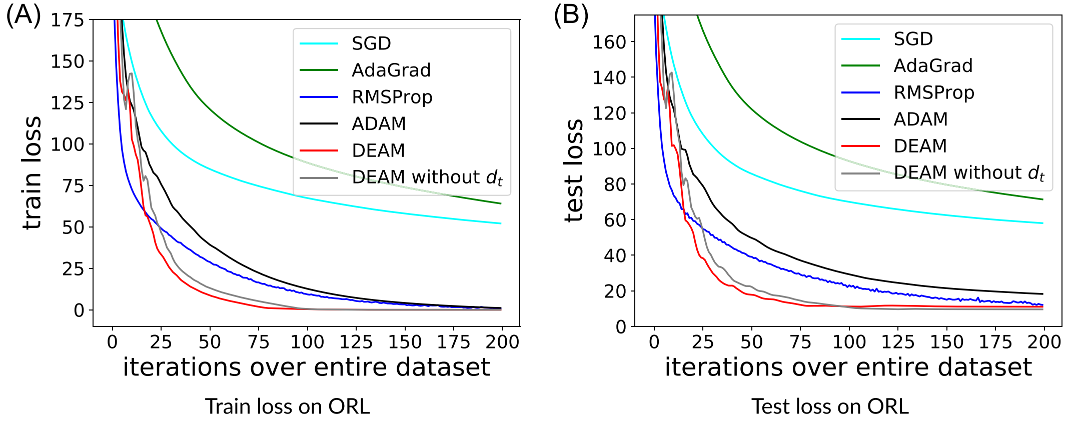

FIGURE 3 The optimization process of Logistic Regression. AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; DEAM, Discriminative wEight on Adaptive Momentum; SGD, stochastic gradient descent [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 3 Running time of DEAM and comparison methods (the unit of values is second)

Comparison methods	Running time on all models								
	LR on ORL	**DNN on MNIST**	**CNN on ORL**	**CNN on MNIST**	**CNN on CIFAR-10**	**GCN on Cora**	**GCN on Citeseer**	**RNN on Reddit**	**LSTM on MNIST**
ADAM[14]	102	664	47,418	21,775	67,584	20	24	>10,000	212.58
RMSProp[22]	48	307	36,722	11,997	84,305	15	22	>10,000	>5000
AdaGrad[12]	>200	667	>100,000	>50,000	>100,000	>50	>100	7172.81	286.72
SGD[9]	>200	346	>100,000	16,985	67,564	>50	>100	>10,000	223.83
DEAM	**38**	**302**	**35,064**	**11,679**	**57,761**	**6.22**	**7.58**	**5356.99**	**208.79**

Abbreviations: AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; CNN, convolutional neural network; DEAM, Discriminative wEight on Adaptive Momentum; DNN, deep neural network; GCN, graph convolutional network; LR, logistic regression; LSTM, Long Short-Term Memory; RNN, recurrent neural network; SGD, stochastic gradient descent.

optimization time required for DEAM is also the shortest, which shows that the additional overhead brought by the adaptive momentum and backtrack mechanism in DEAM is worthwhile for the overall running time improvement.

4.3 | Experiments in DNNs

We use a DNN with two fully connected layers of 64 hidden units and the Rectified Linear Unit (ReLU)[36] activation function. The data set we use is MNIST.[4] The MNIST data set includes 60,000 training samples and 10,000 testing samples, where each sample is a 28 × 28 image of hand-written numbers from 0 to 9. The minibatch size is set as 128, and the learning rate is 0.0001. Results are exhibited in Figure 4, which shows DEAM achieves the best convergence performance. Besides, DEAM requires the least running time (50% running time compared with ADAM) to finish the optimization process in Table 3.

4.4 | Experiments in CNNs

The CNN model in our experiments is based on the LeNet-5.[4] We test it on multiple data sets: ORL, MNIST, and CIFAR-10.[37] The CIFAR-10 data set consists of 60,000 32 × 32 images of 10 classes, with 6000 images per class. For different data sets, the structures of CNN models are modified: for the ORL data set, the CNN model has two convolutional layers with 16 and 36 feature maps of 5 kernels and 2 max-pooling layers, and a fully connected layer with 1024 neurons; for the MNIST data set, the CNN structure follows the LeNet-5 structure in Reference [4]; for CIFAR-10 data set, the CNN model consists of three convolutional layers with 64, 128, and 256, respectively, and a fully connected layer having 1024 neurons. All experiments apply ReLU[36] activation function, and the minibatch size is set as 128 together with the learning rate of 0.0001. From the results shown in Figure 5, DEAM can converge to optimum faster in a smoother process. All three data sets demonstrate the same advantage. The running time of

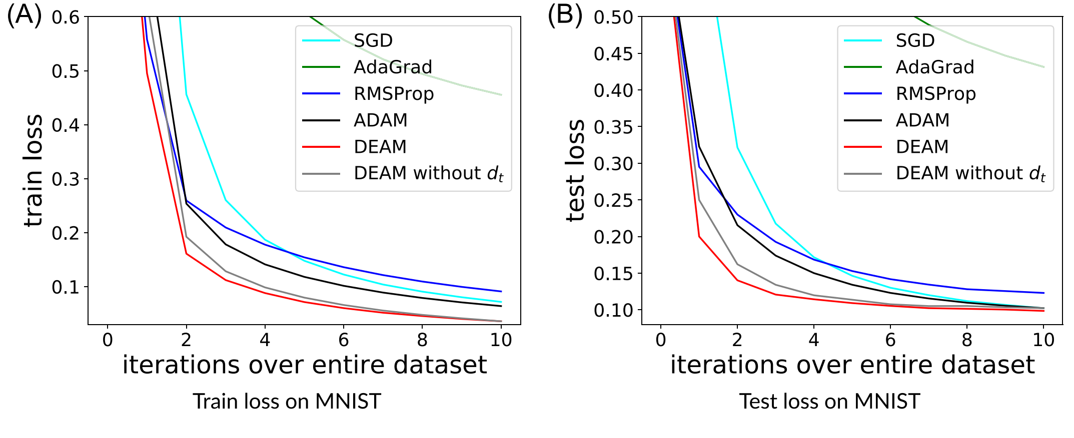

FIGURE 4 The optimization process of the DNN structure. AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; DEAM, Discriminative wEight on Adaptive Momentum; DNN, deep neural network; SGD, stochastic gradient descent [Color figure can be viewed at wileyonlinelibrary.com]

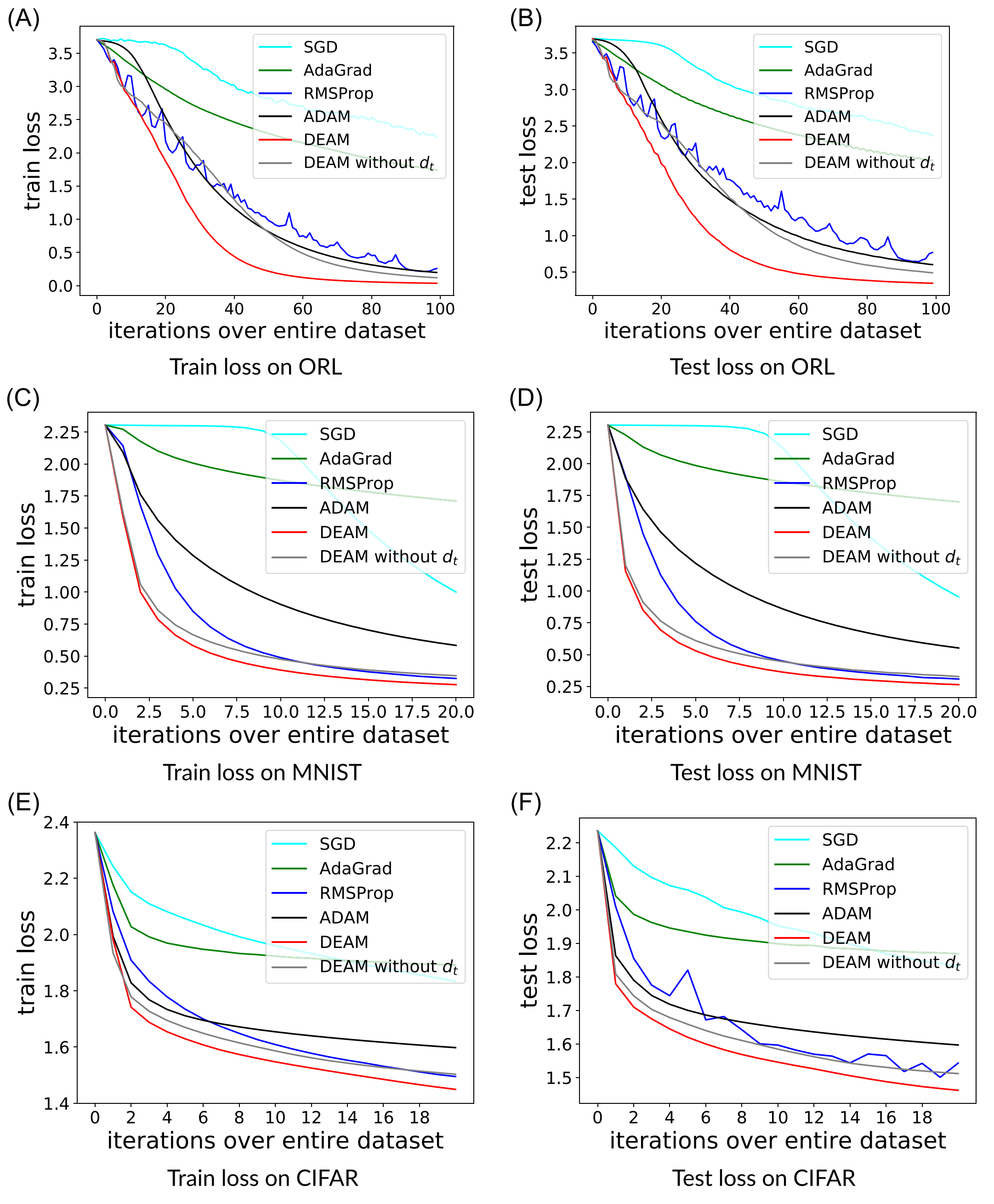

FIGURE 5 The optimization process of CNN structure. AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; CNN, convolutional neural network; DEAM, Discriminative wEight on Adaptive Momentum; SGD, stochastic gradient descent [Color figure can be viewed at wileyonlinelibrary.com]

DEAM on three data sets is less than other optimization methods as well. Through the comparison between DEAM and DEAM without d_t term, we observe that the performance improvement brought by the backtrack mechanism is evident, which verifies that the backtrack mechanism proposed in DEAM is critical for optimizing the CNN model.

4.5 | Experiments in RNNs

To evaluate the performance of DEAM in RNN structures, we employ two kinds of structures (i.e., basic RNN and Long Short-Term Memory [LSTM]) to implement experiments.

We use the basic RNN model structure first. The hidden size is 100, and the vocabulary size is set as 8000. During the training process, the training batch size is 128. The experiment is run on Reddit data set,* which collects real comments from the Reddit website. We sample 2000 sentences from the data set as the training set and 200 sentences as the test set. The basic RNN is trained to work on text generation tasks. From Figure 6A,B, we can observe that DEAM can converge to a lower position with a higher rate compared with other optimization algorithms.

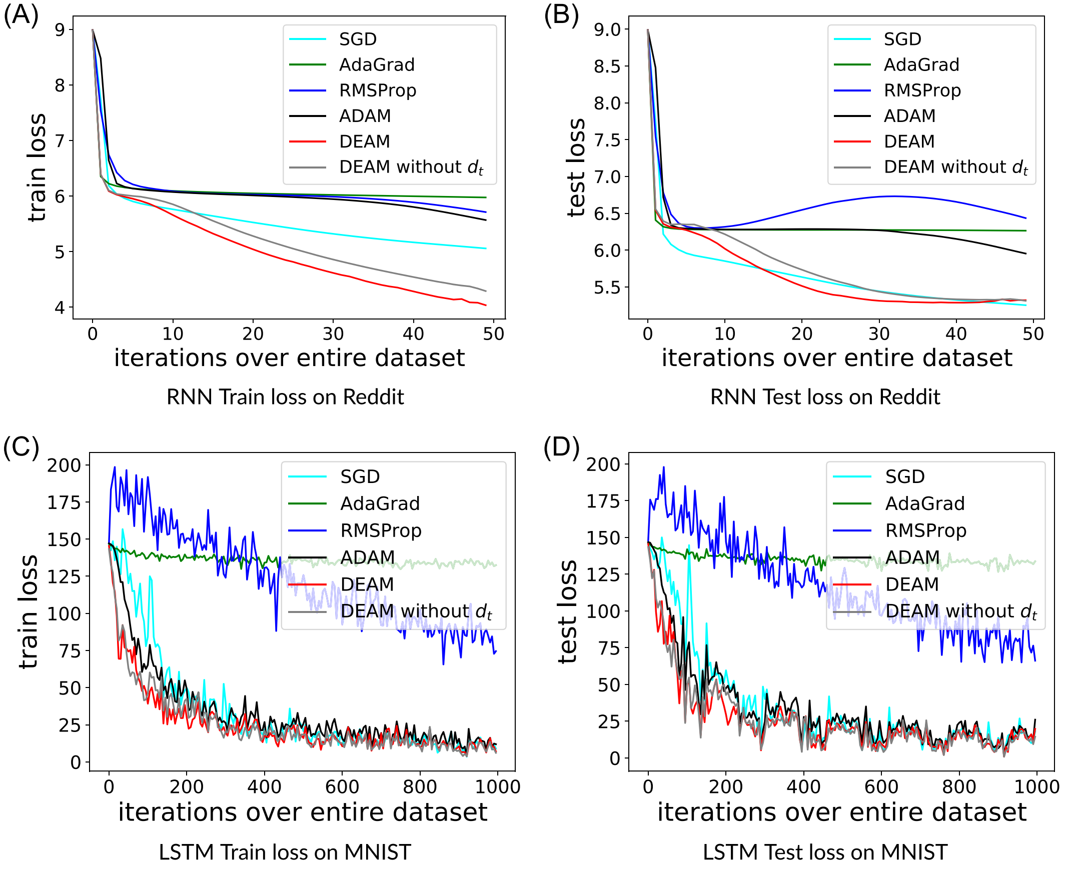

FIGURE 6 The optimization process of the recurrent neural network structures. AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; DEAM, Discriminative wEight on Adaptive Momentum; LSTM, Long Short-Term Memory; SGD, stochastic gradient descent [Color figure can be viewed at wileyonlinelibrary.com]

The backtrack mechanism (i.e., d_t) significantly enhanced model performance in this experiment.

The other RNN structure we run in the experiment is the LSTM model. Here, we implement the LSTM model on the MNIST data set to classify the image. As the MNIST data set images have a size 28×28, each row of the images is considered a word, and each image can be transferred to a sentence with 28 words (rows). In our experiment, the hidden size of LSTM is 128, and the learning rate is 0.001. From Figure 6C,D, DEAM, ADAM, and SGD can achieve comparable performance which beats the results from RMSProp and AdaGrad.

4.6 | Experiments in GCNs

Graph Neural Networks[38–42] are deep models to serve tasks involving graph-structured data. In this paper, we evaluate the proposed algorithms on the GCN structure in Reference [39] on Cora and Citeseer data sets.[43] The Cora data set contains 2708 nodes from 7 classes and 1433 features per node. The Citeseer data set has 3327 nodes, 3703 features per node, and nodes belong to 6 classes. GCN works on the node classification task, and the loss function (objective function) we have selected is the cross-entropy loss function. All other experimental details (e.g., training/test ratio) follow the settings in Reference [39]. The learning rate is 0.01 for GCN optimization. The results are shown in Figure 7. We can observe that DEAM converges faster than other widely used optimization algorithms in all the cases. Within the same number of epochs, DEAM can converge to the lowest loss on both the training set and test set.

4.7 | Analysis of the backtrack mechanism

To show the effectiveness of the backtrack term d_t, we carry out the ablation study of DEAM without d_t term, and exhibit the results from Figures 4 to 7. The results indicate that after applying d_t term, the converging speed becomes faster for most of the neural network structures. To thoroughly prove the effectiveness of our proposed d_t term, we compare it with other definitions of backtrack terms e.g., $d_t = 0.5\cos\theta$, and present the results in Figure 8. In Figure 8, $d_t = sigmoid\_based$ represents that $d_t = -1/(1 + e^{-(\theta - \frac{1}{2}\pi)}) + \frac{1}{2}$, and $d_t = tanh\_based$ means that $d_t = -2/(1 + e^{-2(x - \frac{1}{2}\pi)}) + 1$. Due to the limited space, here we only exhibit the results of the logistic regression on the ORL data set. The experimental results of other machine learning structures on different data sets are consistent. The results in Figure 8 demonstrate that the designed d_t in DEAM is effective and can achieve the best convergence performance.

4.8 | Time-consuming analysis

We have recorded the running time of DEAM and other comparison algorithms in every experiment and list them in Table 3. The running time shown in Table 3 contains “>,” which means the model still does not converge at the specific time. From the results, we can observe that in all of our experiments, DEAM finally converges within the smallest amount of time. From the results from Figures 4 to 7 and Table 3, we can conclude that DEAM can converge not

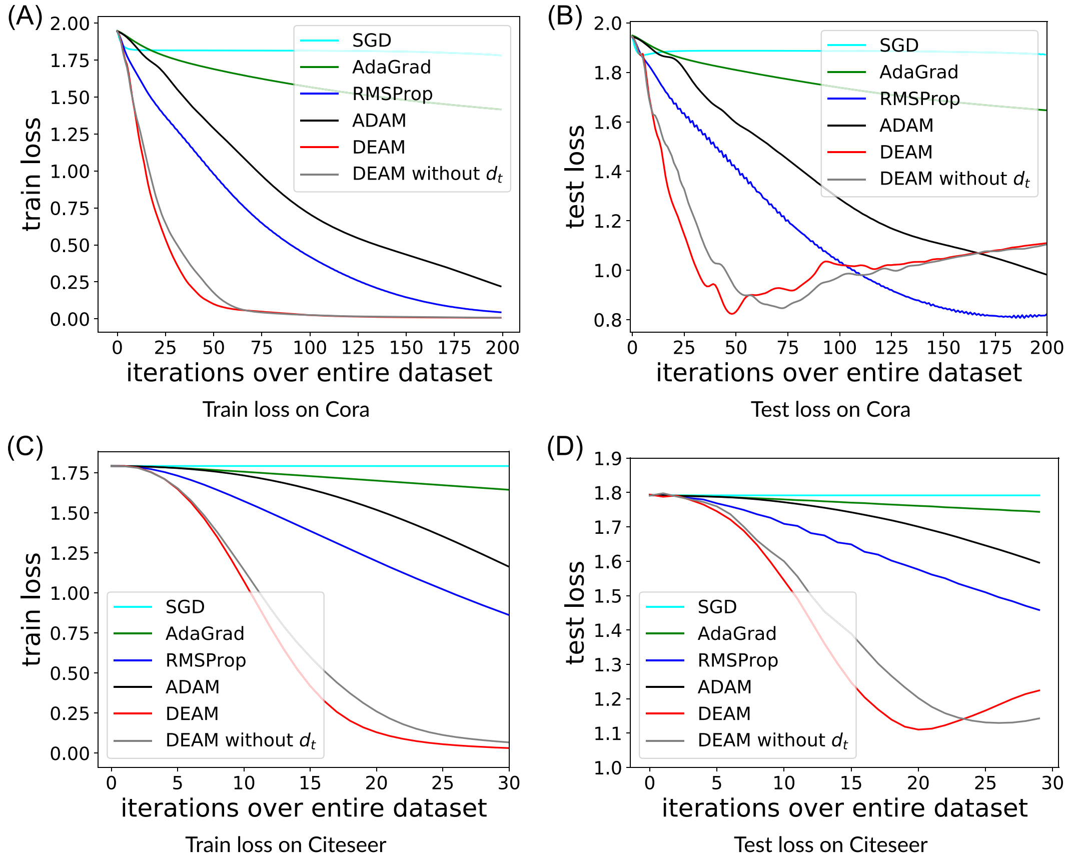

FIGURE 7 The optimization process of GCN structure. AdaGrad, Adaptive Gradient; ADAM, adaptive momentum; DEAM, Discriminative wEight on Adaptive Momentum; GCN, graph convolutional network; SGD, stochastic gradient descent [Color figure can be viewed at wileyonlinelibrary.com]

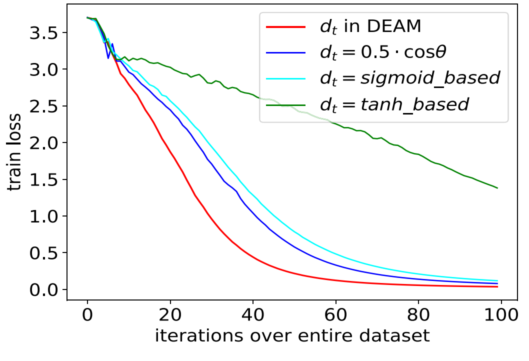

FIGURE 8 Comparison between various designed d_t terms. DEAM, Discriminative wEight on Adaptive Momentum [Color figure can be viewed at wileyonlinelibrary.com]

only in fewer epochs, but also cost less running time. The results in Table 3 also show that the computational cost of the proposed adaptive learning rate and the backtrack mechanism is worthwhile compared with the faster convergence speed they bring because the total running time required is shorter.

5 | FUTURE EXPLORATION

On the basis of the current version of DEAM algorithm, there are still some directions that can be further improved: (1) the second-order momentum weight β_2 in DEAM is a fixed hyperparameter under our design. Such design may not be the ideal solution for different neural networks optimization tasks, thus we expect to propose an adaptive β_2 in the future work; (2) according to Equation (12), our proposed backtrack term d_t is computed by $d_t = \min\{0.5\cos\theta, 0\}$. Here, the 0.5 is also a hyperparameter for the d_t computation. To further eliminate redundant update in the optimization process, this hyperparameter can be specified before the optimization process. For example, we can sample a subset of the training data to grid-search the optimal hyperparameter for the current model optimization task.

6 | CONCLUSION

In this paper, we have introduced a novel optimization algorithm, the DEAM, which implements the momentum with discriminative weights and the backtrack term. We have analyzed the advantages of the proposed algorithm and proved it by theoretical inference. Extensive experiments have shown that the proposed algorithm can converge faster than existing methods by almost 50% on both convex and nonconvex situations, and the time consuming is better than existing methods: the time consuming of DEAM is only half of the most widely used optimizers SGD and ADAM on average. Not only the proposed algorithm can outperform other popular optimization algorithms, but also fewer hyperparameters will be introduced, which makes the DEAM much more applicable.

FUNDING INFORMATION

National Science Foundation, Grant Number: IIS-1763365.

ORCID

Jiyang Bai https://orcid.org/0000-0002-0621-8815
Yuxiang Ren https://orcid.org/0000-0001-8829-3984
Jiawei Zhang https://orcid.org/0000-0003-3356-6361

ENDNOTE

*https://www.kaggle.com/nursen/redditcomments.

REFERENCES

1. 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*; 2016:770-778.
2. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *Adv Neural Inf Process Syst*. 2012;25:1097-1105.
3. Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*; 2015:1-9.
4. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proc IEEE*. 1998;86(11):2278-2324.
5. Dong L, Wei F, Zhou M, Xu K. Question answering over freebase with multi-column convolutional neural networks. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*; 2015:260-269.

6. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:14090473. 2014.
7. Neumann M, Vu NT. Attentive convolutional neural network based speech emotion recognition: a study on the impact of input features, signal length, and acted speech. In: *Proceedings of the Interspeech 2017*; 2017: 1263-1267.
8. Zhang J. Graph neural networks for small graph and giant network representation learning: an overview. arXiv preprint arXiv:190800187. 2019.
9. Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:160904747. 2016.
10. Qian N. On the momentum term in gradient descent learning algorithms. *Neural Networks.* 1999;12(1): 145-151.
11. Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. In: *International Conference on Machine Learning*. PMLR; 2013:1139-1147.
12. Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. *J Mach Learn Res*. 2011;12(7):2121-2159.
13. Zeiler MD. AdaDelta: an adaptive learning rate method. arXiv preprint arXiv:12125701. 2012.
14. Kingma DP, Ba J. ADAM: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014.
15. Bianchi P, Jakubowicz J. Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization. *IEEE Trans Autom Control*. 2012;58(2):391-405.
16. Reddi SJ, Hefny A, Sra S, Poczos B, Smola A. On variance reduction in stochastic gradient descent and its asynchronous variants. arXiv preprint arXiv:150606840. 2015.
17. Roux NL, Schmidt M, Bach F. A stochastic gradient method with an exponential convergence rate for finite training sets. arXiv preprint arXiv:12026258. 2012.
18. Shalev-Shwartz S, Zhang T. Stochastic dual coordinate ascent methods for regularized loss minimization. *J Mach Learn Res*. 2013;14(2):567-599.
19. Johnson R, Zhang T. Accelerating stochastic gradient descent using predictive variance reduction. *Adv Neural Inf Process Syst*. 2013;26:315-323.
20. Defazio A, Bach F, Lacoste-Julien S. SAGA: a fast incremental gradient method with support for non-strongly convex composite objectives. arXiv preprint arXiv:14070202. 2014.
21. Behera L, Kumar S, Patnaik A. On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE Trans Neural Networks*. 2006;17(5):1116-1125.
22. Tieleman T, Hinton GE. *Leture 6.5 RMSProp, COURSERA: Neural Networks for Machine Learning*. Technical Report; 2012.
23. Dauphin YN, De Vries H, Bengio Y. Equilibrated adaptive learning rates for non-convex optimization. arXiv preprint arXiv:150204390. 2015.
24. Luo L, Xiong Y, Liu Y, Sun X. Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint arXiv:190209843. 2019.
25. Li Q, Zhou Y, Liang Y, Varshney PK. Convergence analysis of proximal gradient with momentum for nonconvex optimization. In: *International Conference on Machine Learning*. PMLR; 2017:2111-2119.
26. Dozat T. *Incorporating Nesterov Momentum into ADAM*; 2016.
27. Mitliagkas I, Zhang C, Hadjis S, Ré C. Asynchrony begets momentum, with an application to deep learning. In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE; 2016:997-1004.
28. Nesterov Y. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). In: *Doklady ANUSSR*; 1983.
29. Zhang Z, Ma L, Li Z, Wu C. Normalized direction-preserving ADAM. arXiv preprint arXiv:170904546. 2017.
30. Keskar NS, Socher R. Improving generalization performance by switching from ADAM to SGD. arXiv preprint arXiv:171207628. 2017.
31. Reddi SJ, Kale S, Kumar S. On the convergence of ADAM and beyond. arXiv preprint arXiv:190409237. 2019.
32. Bai J, Ren Y, Zhang J. DEAM: adaptive momentum with discriminative weight for stochastic optimization. In: *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE; 2020:37-41.

33. Chen X, Liu S, Sun R, Hong M. On the convergence of a class of ADAM-type algorithms for non-convex optimization. arXiv preprint arXiv:180802941. 2018.
34. Zinkevich M. Online convex programming and generalized infinitesimal gradient ascent. In: *Proceedings of the 20th International Conference on Machine Learning*; 2003:928-936.
35. Samaria FS, Harter AC. Parameterisation of a stochastic model for human face identification. In: *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*. IEEE; 1994:138-142.
36. Nair V, Hinton GE. Rectified linear units improve restricted Boltzmann machines. In: *Proceedings of the 27th International Conference on Machine Learning*; 2010.
37. Krizhevsky A, Hinton G, et al. *Learning Multiple Layers of Features from Tiny Images*; 2009.
38. Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*; 2016:3844-3852.
39. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: *ICLR*; 2017.
40. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. arXiv preprint arXiv:171010903. 2017.
41. Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*; 2017:1025-1035.
42. Bai J, Ren Y, Zhang J. Ripple walk training: a subgraph-based training framework for large and deep graph neural network. In: *IJCNN*; 2021.
43. Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T. Collective classification in network data. *AI Mag*. 2008;29(3):93.

How to cite this article: Bai J, Ren Y, Zhang J. Adaptive momentum with discriminative weight for neural network stochastic optimization. *Int J Intell Syst*. 2022;37:6531-6554. doi:10.1002/int.22854