DEAM: Accumulated Momentum with Discriminative Weight for Stochastic Optimization

Jiyang Bai
Department of Computer Science
Florida State University
jiyang@ifmlab.org

Jiawei Zhang
Department of Computer Science
Florida State University
jiawei@ifmlab.org

Abstract
Optimization algorithms with momentum, e.g., Nesterov Accelerated Gradient [10] and ADAM [4], have been widely used for building deep learning models because of their faster convergence rates compared to stochastic gradient descent (SGD). Momentum [11] is a method that helps accelerate SGD in the relevant directions in variable updating, which can minify the oscillations of variables update route. Optimization algorithms with momentum usually allocate a fixed hyperparameter (e.g., $\beta_1$ [4]) as the weight of the momentum term. However, using a fixed weight is not applicable to some situations, and such a hyper-parameter can be extremely hard to tune in applications. In this paper, we will introduce a new optimization algorithm, namely DEAM (Discriminative wEight on Accumulated Momentum). Instead of assigning the momentum term with a fixed weight, DEAM proposes to compute the momentum weight in the learning process automatically. DEAM also involves a "backtrack" term, which can help accelerate the algorithm convergence by restricting redundant updates. Extensive experiments have been done on several real-world datasets. The experimental results demonstrate that DEAM can achieve a faster convergence rate than the existing optimization algorithms in training both the classic machine learning models and the recent deep learning models.

1 Introduction
Deep learning methods can achieve outstanding performance for many tasks, especially for image classification [6, 7]. Optimization algorithms play a crucial role in training the deep learning models to learn the (locally) optimal model variables. Here, we can take the image classification task as an example. Given the training set, the objective functions of the deep learning models in classifying the image instances can be represented as

$$\min_{w \in \mathcal{W}} f(X, y; w)$$

Here, $f(\cdot, \cdot; w)$ denotes the loss function with variables vector $w \in \mathcal{W}$, where $\mathcal{W}$ is the solution space of variables; $X$ and $y$ denote the features and labels of the training data, respectively.

The most popular algorithms to address the above objective function include stochastic gradient descent (SGD) [15], SGD with momentum [11], AdaGrad [1], RMSProp [18], AdaDelta [19] and ADAM [4]. SGD is a stochastic optimization algorithm that uses only one training example to calculate gradient in each training iteration (epoch), and updates variables using the gradient directly. AdaGrad introduces adaptive learning rates that can update different variables in suitable scales. According to [18], AdaGrad can achieve much better performance than SGD when the gradients are sparse or small. However, the regularization term of AdaGrad is continuously increasing, which may cause much slower convergence. ADAM involves both adaptive learning [15] and momentum [11]. The momentum [11] can help ADAM accelerate the convergence in the relevant directions and
Stochastic Gradient Descent: Stochastic gradient descent (SGD) [15] performs variable updating for each training example \( X[i, :] \) and label \( y[i] \) (Here, \( X \) represents the training set matrix, every row of which is a training sample vector; \( y \) is the vector of all training samples' labels): 
\[
    w_t = w_{t-1} - \eta \cdot \nabla f_t(X[i,:], y[i]; w_t)
\]  
(2)
where \( \eta \) is the learning rate and \( \nabla \) is the derivative of the loss function. The advantages of SGD include fast converging speed compared with gradient descent and preventing redundancy [15]. However, the unified learning rate for all variables can lead to some problems. To solve these problems, some variant algorithms have been proposed, such as AdaGrad [1], RMSProp [18] and ADAM [4]. Besides, [2][12][13] use the variance reduction methods to accelerate the training process of SGD.

Adaptive Learning Rates: Methods applying adaptive learning rates includes AdaGrad [1], AdaDelta [19] and RMSProp [18]. AdaGrad adopts different learning rates to different variables, and its variable updating equation can be represented as follows:
\[
    w_t = w_{t-1} - \frac{g_t}{\sqrt{\sum_{i=1}^{t} g_i} + \epsilon}, \text{ where } g_t = \eta \cdot \nabla f_t(w_t).
\]  
(3)

We have to mention that the \( \sum \), \( \odot \) and \( \sqrt{ \) in the above equation are element-wise operations. By using the summation of all the previous gradients, AdaGrad can achieve adaptive learning rates for different variables. One drawback of AdaGrad is that with the increase of iteration number \( t \), the adaptive term \( \sum_{i=1}^{t} g_i \odot g_i \) will inflate continuously, which will lead to a very slow convergence rate. RMSProp can solve this problem by using the moving average of historical gradients. The update rule of RMSProp is shown as follows:
\[
    w_t = w_{t-1} - \eta \cdot g_t / \sqrt{v_t}, \text{ where } v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t \odot g_t.
\]  
(4)

In the above equation, term \( \beta_2 \) is a hyper-parameter in the interval \([0, 1]\). In this way, the adaptive term \( v_t \) will not increase continuously.

Momentum: Momentum [11][17] 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:
Algorithm 1 DEAM Algorithm

**Input:** learning rate \( \{ \eta_t \}_{t=1}^T \), \( \beta_2 = 0.999 \)

**Output:** trained variables

1. \( m_0 \leftarrow 0 \)
2. \( v_0 \leftarrow 0 \)
3. **for** \( t = 1, 2, \ldots, T \) **do**
   4. \( g_t = \nabla f_t(w_t) \),
   5. \( \theta = \left( \frac{m_{t-1} - v_{t-1}}{\sqrt{v_t}} \right) \) /* The operator \( \langle \cdot, \cdot \rangle \) represents the angle between two vectors. */
   6. **if** \( \theta \in [0, \frac{\pi}{2}] \) **then**
      7. \( \beta_{1,t} = \sin \theta / K + \epsilon \)
   8. **else**
      9. \( \beta_{1,t} = 1 / K \) /* \( K = \frac{10(2+\pi)}{2\pi} \) */
   10. **end if**
   11. \( m_t = (1 - \beta_{1,t}) \cdot m_{t-1} + \beta_{1,t} \cdot g_t \)
   12. \( v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t \odot g_t \)
   13. \( d_t = \text{min} \{ 0.5 \cos \theta, 0 \} \)
   14. \( \Delta_t = d_t \cdot m_{t-1} - \eta_t \cdot \frac{m_t}{\sqrt{v_t}} \)
   15. \( w_t = w_{t-1} + \Delta_t \)
   16. **end for**
17. **return** \( w_T \)

\[ w_t = w_{t-1} - m_t, \text{ where } m_t = \gamma \cdot m_{t-1} + \eta \cdot \nabla f_t(w_t). \] (5)

In the equation, \( \gamma \) is the weight of the momentum and \( \eta \) 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) [10], which can be presented as

\[ w_t = w_{t-1} - m_t, \text{ where } m_t = \gamma m_{t-1} + \eta \cdot \nabla f_t(w_{t-1} - \gamma m_{t-1}). \] (6)

**ADAM:** ADAM [4, 20] 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:

\[
\begin{align*}
g_t &= \nabla f_t(w) \\
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t; \quad \hat{m}_t = m_t / (1 - \beta_1^t) \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t \odot g_t; \quad \hat{v}_t = v_t / (1 - \beta_2^t) \\
w_t &= w_{t-1} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)
\end{align*}
\] (7)

ADAM records the first-order momentum \( \hat{m}_t \) and the second-order momentum \( \hat{v}_t \) of the gradients using the moving average (controlled by the parameters \( \beta_1 \) and \( \beta_2 \), respectively), and further computes the bias-corrected version of them (\( \hat{m}_t \) and \( \hat{v}_t \)). Based on ADAM, [3] proposes to switch from ADAM to SGD during the training process. In this way, it can combine the advantages of both SGD and ADAM.

**AMSGGrad:** AMSGGrad [14] is a modified version of ADAM. AMSGGrad changes the definition of second-order momentum by \( \hat{v}_t = \max \{ \hat{v}_{t-1}, v_t \} \), and other settings are almost the same as ADAM. What’s more, AMSGGrad applies a varied learning rate \( \eta_t \) comparing to ADAM, but the definition of \( \eta_t \) is not specified.

### 3 Proposed Algorithm

Our proposed algorithm DEAM is presented in Algorithm [1]. In the algorithm, \( \{ f_t \}_{t=1}^T \) is a set of objective functions computed with the training mini-batches in different iterations (or epochs). DEAM introduces two new terms in the learning process: (1) the "discriminative angle" \( \theta \), and (2) the "backtrack term" \( d_t \). Formally, \( \theta \) denotes the angle between previous \( m_{t-1} / \sqrt{v_{t-1}} \) and current gradient \( g_t \), and it subsequently achieves the discriminative weight \( \beta_{1,t} \). Meanwhile, the backtrack
Figure 1: The updating routes of ADAM with $\beta_1 = 0.9$ (the blue line) and $\beta_1 = 0.0$ (the red line).

term $d_t$ represents the small returning step of the previous update on variables. We can notice that in each iteration, after the $\theta$ has been calculated, the $\beta_{1,t}$ is directly obtained according to the $\theta$ (the $\beta_1$ in our paper is not the exact one in ADAM [4], since previously $\beta_1$ is the weight of historical first-order momentum, but in our paper $\beta_1$ is the weight of current gradient). In this way, we can achieve different $\beta_{1,t}$ as the discriminative angle changes. The $d_t$ term balances between the historical update $\Delta_{t-1}$ and the current $\mathbf{m}_t/\sqrt{\nu_t}$ when updating $\Delta_t$. For the rest of Algorithm [1] the notations of $\mathbf{m}_t$ and $\nu_t$ are identical to those used in ADAM.

3.1 Discriminative Angle $\theta$

Before talking about the mechanism of discriminative angle, we will explain why we propose to compute it in the learning algorithm.

3.1.1 Motivation

In the ADAM [4] paper, the historical first-order momentum’s weight (i.e., $\beta_1$) and the current gradient's weight (i.e., $(1-\beta_1)$) are pre-specified fixed values, 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

$$f(x, y) = x^2 + 4y^2$$

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’s 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 $\beta_1$ will lead to very different performance of ADAM. For instance, in Figure 1 we illustrate the updating 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 $\mathbf{m}_t$ is computed by integrating $\mathbf{m}_1$ and $\mathbf{g}_2$ together (whose weights are $\beta_1$ 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 $\beta_1 = 0.9$ will take much more iterations until reaching the convergence.

From the analysis above, we can observe that, a careful tuning and updating of $\beta_1$ 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 $\theta$ for computing $\beta_1$ automatically as follows.

3.1.2 Mechanism

The momentum weight $\beta_1$ will be updated in each iteration in DEAM, and we can denote its value computed in the $t_{th}$ iteration as $\beta_{1,t}$ formally. Essentially, in the $t_{th}$ iteration of the training process, both $\mathbf{m}_{t-1}/\sqrt{\nu_{t-1}}$ 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 $\theta$ in the $t_{th}$ iteration is calculated by

$$\theta = \left\langle \frac{m_{t-1}}{\sqrt{v_{t-1}}}, -g_t \right\rangle = \left\langle \frac{m_{t-1}}{\sqrt{v_{t-1}}}, g_t \right\rangle$$

(8)

Here, operator $\langle \cdot, \cdot \rangle$ denotes the angle between two vectors. This expression is easy to understand, since the $-m_{t-1}/\sqrt{v_{t-1}}$ can represent the updating direction of last step in ADAM, meanwhile $-g_t$ is the reverse of the present gradient. So we can simplify it as $\theta = \left\langle \frac{m_{t-1}}{\sqrt{v_{t-1}}}, g_t \right\rangle$. If $\theta$ is close to zero (denoted by $\theta \to 0^\circ$), the $m_{t-1}/\sqrt{v_{t-1}}$ and $g_t$ are almost in the same direction, and the weights for them will not be very important. Meanwhile, if $\theta$ approaches 180° (denoted by $\theta \to 180^\circ$), $m_{t-1}/\sqrt{v_{t-1}}$ and $g_t$ will be in totally reverse directions. This means in the current step, the previous momentum term is already in a wrong direction. Therefore, the current gradient’s weight (i.e., $\beta_{1,t}$ in our paper) should be assigned with a larger value instead. As the $\beta_{1,t}$ varies when $\theta$ changes from 0° to 180°, we intend to define $\beta_{1,t}$ with the following equation:

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

(9)

where $K = 10(2 + \pi)/2\pi$ and $\epsilon$ 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 $\frac{\pi}{2} \leq \theta \leq \pi$, which is exactly the situation $\theta \to 180^\circ$ 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 $g_t$, which may cause fluctuations on the updating routes. The value of $K$ is determined by:

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

(10)

In the equation above, assume $\theta$ is randomly distributed on $[0, \pi]$, in this calculation we can get

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

(11)

In other words, the expectation of $\beta_{1,t}$ (i.e., $\mathbb{E}(\beta_{1,t})$) will be identical to the $\beta_3$ used in ADAM [4] (the $\beta_{1,t}$ in our paper equals to the $1 - \beta_{1,t}$ in ADAM [4] paper). After obtaining $\theta$, it will be applied to calculate $m_t$ as shown in Algorithm 1. In this way, we have achieved momentum with discriminative weights, and this weight is automatically computed during the training process.

3.2 Backtrack term $d_t$

To further speed up the convergence rate of DEAM, we will introduce a novel backtrack mechanism in this paper. Backtrack allows DEAM to eliminate redundant update in each iteration, which has never been used in the existing algorithms, like AdaGrad [1], RMSProp [18] and ADAM [4].

3.2.1 Motivation

When optimizer (e.g., ADAM) updates variables of the objective function (e.g., $f(x, y)$), some update routes will look like the black arrow lines shown in Figure 2(a) especially when the discriminative angle $\theta$ is larger than 90°. We call this phenomenon the "zig-zag" route. In Figure 2(a), it shows the updating routes of a 2-dimension function. Each black arrow line in the figure represents the variables’ update in each epoch; the red dashed line is the direction of the updating routes; the $\theta$ is the discriminative angle. If $\theta \geq 90^\circ$, the "zig-zag" phenomenon will appear severely, which may lead to slower convergence speed. The main reason is when $\theta \geq 90^\circ$, 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 2(b) For the example of function with 2-dimension variables, the update direction $m_{1}/\sqrt{v_{1}}$ can be decomposed into $(x_1, y_1)^\top$ in Figure 2(b) and the same with $m_{2}/\sqrt{v_{2}}$. 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 over-update (or redundant update) of the first step. Therefore the backtrack term $d_t$ is proposed to restrict this situation.
Axis decomposition

Example: When \( \theta = 180^\circ \)

Figure 2: Some examples about \( d_t \)

3.2.2 Mechanism

Since the redundant update situation is caused by over updating of the previous epoch, 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 \( \theta \to 0^\circ \), the updating process becomes smooth, not like the situation shown in Figure 2(a). Besides, from the analysis above we conclude that if \( \theta \geq 90^\circ \), there will be at least one dimension involves the redundant update. Thus, in the \( t \)-th iteration we quantify \( d_t \) as the following equation:

\[
d_t = \min \{0.5 \cos \theta, 0\}
\]

and we rewrite the updating term with backtrack in DEAM as

\[
\Delta_t = d_t \cdot \Delta_{t-1} - \eta_t \cdot \frac{m_{t-1}}{\sqrt{t}}
\]

where \( \theta \) is the discriminative angle and \( \Delta_t \) is the updating term in Algorithm 1. By designing \( d_t \) in this way, when \( \theta \to 0^\circ \), \( d_t = 0 \) and there is no backward step, the updating term \( \Delta_t = -\eta_t \cdot \frac{m_{t-1}}{\sqrt{t}} \) is similar to ADAM: when \( \theta \to 180^\circ \), \( d_t = 0.5 \cos \theta \) and comes to the maximum value when \( \theta = 180^\circ \). The reason that \( \cos \theta \) is rescaled by 0.5 is that: in Figure 2(c) \( w_{t-1} \) and \( w_t \) are the variables updated by DEAM without \( d_t \) term in the \( (t-1) \)-th and \( t \)-th iterations respectively. If the backtrack mechanism is implemented, in the \( (t-1) \)-th iteration, since \( \theta = 180^\circ \), firstly \( d_t = 0.5 \cos \theta \to -0.5 \) makes the backtrack to the \( w^* \) point (the middle point of \( w_{t-1} \) and \( w_t \)). Thus, this backtrack step allows the variable to further approach the optima.

3.3 Convergence Analysis

In this part, we will give the detailed analysis on the convergence of our DEAM algorithm. According to [4][14][21], given an arbitrary sequence of convex objective functions \( f_1(w), f_2(w), \ldots, f_T(w) \), we intend to evaluate our algorithm using the regret function, which is denoted as:

\[
R(T) = \sum_{t=1}^{T} [f_t(w_t) - f_t(w^*)]
\]

where \( w^* \) is the globally optimal point. In the following Theorem 3.1, we will show that the above regret function is bounded. For the following proof, \( g_t := \nabla f_t(w_t) \) and \( g_{t,i} \) will represent the \( i \)-th element of \( g_t \in \mathbb{R}^d \), and \( g_{1:t,i} = [g_{1,i}, g_{2,i}, \ldots, g_{t,i}] \).

**Theorem 3.1.** Assume \( \{f_t\}_{t=1}^{T} \) have bounded gradients \( \|\nabla f_t(w)\|_{\infty} \leq G \) for all \( w \in \mathbb{R}^d \), all variables are bounded by \( \|w_p - w_q\|_2 \leq D \) and \( \|w_p - w_q\|_{\infty} \leq D \), \( \forall p,q \in \{1,2,\ldots,T\} \), \( \eta_t = \eta/\sqrt{t} \), \( \gamma_1 = (1 - \epsilon_0)/\sqrt{32} \) and satisfies \( \gamma_1 < 1 \), \( \epsilon = 1 - (1 - \epsilon_0)\lambda^{-1} \), \( \lambda \in (0,1) \). Our proposed algorithm can achieve the following bound on regret:

\[
R(t) \leq \frac{D^2}{\epsilon_0 \eta} \sum_{i=1}^{d} \sqrt{T V_{T,i}} + \frac{(1 - \epsilon_0)^2 G D}{K(1 - \lambda)^2 \epsilon_0} + \frac{\eta \sqrt{T + \log T}}{2\epsilon_0 \gamma_1 \sqrt{1 - \beta_2}} \sum_{i=1}^{d} \|g_{1:T,i}\|_2
\]

For the bound term, as \( T \to +\infty \), \( R(T) \to 0 \) and we can infer that \( \lim_{T \to \infty} [f_t(w_t) - f_t(w^*)] = 0 \), which means the proposed algorithm can finally converge. The detailed proof of Theorem 3.1 is given in the supplementary document submitted together with this paper.
4 Experiments

We have applied the DEAM algorithm on multiple popular machine learning and deep learning structures, based on different datasets. Meanwhile, to show the advantages of the algorithm, we compare it with various popular learning algorithms, including ADAM [4], RMSProp [18] and AdaGrad [1], based on the training loss and testing loss. For all the experiments, the loss function (objective function) we have selected is the cross-entropy loss function, and the size of the minibatch is 128. Besides, the learning rate is 0.0001.

4.1 Experiment Settings and Results

Logistic Regression: We firstly evaluate our algorithm on the multi-class logistic regression model, since it is widely used and owns a convex objective function. We conduct logistic regression on the MNIST dataset [7]. MNIST dataset includes 60,000 training samples and 10,000 testing samples, where each sample is a $28 \times 28$ image of hand-written numbers from 0 to 9. The loss of objective functions on both training set and testing set are shown in Figure 3.

Deep Neural Network: We use the deep neural network (DNN) with two fully connected layers of 1,000 hidden units and the ReLu [9] activation function. The dataset we use is MNIST [7]. The result is exhibited in Figure 4.

Convolutional Neural Network: The CNN models in our experiments are based on the LeNet-5 [7], and it is implemented on multiple datasets: ORL [16], MNIST [7] and CIFAR-10 [5]. ORL dataset consists of face images of 40 people, each person has ten images and each image is in the size of $112 \times 92$; the CIFAR-10 dataset consists of 60,000 $32 \times 32$ images in 10 classes, with 6,000 images per class. The dataset has no augmentation. For different datasets, the structures of CNN models are modified: for the ORL dataset, 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 dataset, the CNN structure follows the LeNet-5 structure in [7]; for the CIFAR-10 dataset, the CNN model consists of three convolutional layers with 64, 128, 256 kernels respectively, and a fully connected layer having 1024 neurons. All the experiments apply ReLu [9] activation function, and 0.5 dropout rate on fully connected layers. The results are shown in Figure 5.

From the experimental results, we can observe that our proposed algorithm can converge faster than other widely used optimization algorithms in most of the cases.

4.2 Effectiveness Analysis of the Backtrack Mechanism

To show the effectiveness of the backtrack term $d_t$, we also carry out the experiments of DEAM algorithm without the $d_t$ term, and exhibit the results in Figures 3, 4, 5. The results indicate that after applying $d_t$ term, the converging speed will become slightly faster, especially for the results
in Figure 5(a) and 5(b). In other words, for some cases, the backtrack term $d_t$ can truly further accelerate the training process.

4.3 Time-consuming Analysis

We have recorded the running time of ADAM and DEAM in every experiment, and list them in the Table 1. The results indicate that the average running time of DEAM is slightly higher than ADAM for about 10%, which is in an acceptable range. (The device we used is the Dell PowerEdge T630 Tower Server, with 64-bit Intel Xeon CPU E5-2698 v4@2.2GHz (80 cores). The total memory is 256 GB, with an extra (SSD) swap of 250 GB.)

| Experiment               | ADAM | DEAM |
|--------------------------|------|------|
| Logistic Regression on MNIST | 165  | 185  |
| DNN on MNIST              | 299  | 370  |
| CNN on ORL                | 155  | 170  |
| CNN on MNIST              | 1761 | 1969 |
| CNN on CIFAR              | 2138 | 2451 |

5 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, and the time consuming is as good as the ADAM. Not only the proposed algorithm can outperform other popular optimization algorithms, but less hyperparameter will be introduced, which makes the DEAM much more applicable.
References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[2] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NeurIPS*, 2013.

[3] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. In *arXiv preprint arXiv:1712.07628*, 2017.

[4] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[5] Alex Krizhevsky. Learning multiple layers of features from tiny images. In *CRC Press*, 2009.

[6] Alex Krizhevsky and Ilya Sutskever Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[7] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-base learning applied to document recognition. In *IEEE*, 1998.

[8] H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In *COLT*, 2010.

[9] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[10] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). In *Doklady ANUSSR*, 1983.

[11] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : The Official Journal of the International Neural Network Society*, 1999.

[12] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In *ICML*, 2016.

[13] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alexander J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *NeurIPS*, 2015.

[14] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *ICLR*, 2018.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms. In *arXiv:1609.04747v2*, 2017.

[16] Ferdinando Samaria and Andy Harter. Parameterisation of a stochastic model for human face identification. In *IEEE Workshop on Applications of Computer Vision*, 1994.

[17] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[18] Tijmen Tieleman and Geoffrey E. Hinton. Leture 6.5 rmsprop,coursera: Neural networks for machine learning. In *Techcnical report*, 2012.

[19] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. In *arXiv:1212.5701v1*, 2012.

[20] Zijun Zhang, Lin Ma, Zongpeng Li, and Chuan Wu. Normalized direction-preserving adam. In *arXiv: 1709.04546v2*, 2018.

[21] Martin Zinkevich. On convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.
6 Appendix

6.1 Convergence Proof

Definition 6.1. 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
\[
f(\phi x + (1 - \phi) y) \leq \phi f(x) + (1 - \phi) f(y)
\]

Lemma 6.1. If a function \( f : \mathbb{R}^d \to \mathbb{R} \) then \( \forall x, y \in \mathbb{R}^d \) we have
\[
f(y) \geq f(x) + \nabla f(x)^\top(y - x)
\]

We will use the above definition and lemma to prove the Theorem 3.1.

Lemma 6.2. Assume that the function \( f_t \) has bounded gradients, \( \|\nabla f_t(w)\|_\infty \leq G_\infty \). Let \( m_{t,i} \) represents the \( i \)th element of \( m_t \) in Algorithm 1, then the \( m_{t,i} \) is bounded by
\[
m_{t,i} \leq (1 - \epsilon_0) G_\infty K (1 - \lambda)
\]

Proof. Let \( g_t = \nabla f_t(w) \). According to the definition of \( m_{t,i} \) in our algorithm,
\[
m_{t,i} = \sum_{j=1}^{t} \beta_{1,j} \prod_{l=1}^{t-j} (1 - \beta_{1,t-l+1}) g_{j,i}
\]
\[
\leq \frac{G_\infty}{K} \sum_{j=1}^{t} \prod_{l=1}^{t-j} (1 - \beta_{1,t-l+1}) \leq \frac{G_\infty}{K} \sum_{j=1}^{t} \prod_{l=1}^{t-j} (1 - \epsilon)
\]
\[
\leq \frac{G_\infty}{K} \sum_{j=1}^{t} (1 - \epsilon_0) \lambda^{t-j}
\]
\[
\leq (1 - \epsilon_0) G_\infty K (1 - \lambda)
\]

where \( K \) and \( \epsilon \) are the terms in Algorithm 1.

Lemma 6.3. Let \( \eta_t = \eta / \sqrt{t}, \epsilon = 1 - (1 - \epsilon_0) \lambda^{t-1}, \lambda \in (0, 1), \gamma_1 = (1 - \epsilon_0) / \sqrt{\beta_2} \) and satisfies \( \gamma_1 < 1 \). The bound of \( R(T) \) will be
\[
R(T) \leq \frac{D^2}{\epsilon_0 \eta} \sum_{i=1}^{d} \sqrt{TV_{T,i}} + \frac{(1 - \epsilon_0)^2 G_\infty D_\infty d}{K(1 - \lambda)^2 \epsilon_0} + \frac{\eta \sqrt{1 + \log T}}{2 \epsilon_0^2 (1 - \gamma_1) \sqrt{1 - \beta_2}} \sum_{i=1}^{d} \|g_{i:T,i}\|_2
\]

Proof. According to Lemma 1.1, for \( \forall t \in \{1, 2, \ldots, T\} \), we have
\[
f_t(w_t) - f_t(w^*) \leq \nabla f_t(w_t)^\top(w_t - w^*)
\]
\[
= g_t^\top(w_t - w^*)
\]
\[
= \sum_{i=1}^{d} g_{t,i}(w_{t,i} - w^*_i)
\]

From the update rule of the proposed algorithm, we can indicate that \( w_{t+1} = w_t + \Delta_t \). From the definition of \( \Delta_t \), we know that it is equal to multiplying the learning rate \( \eta_t \) in some iterations by a number in \([0.5, 1]\), which means
\[
\hat{\eta}_t = \mu_t \cdot \eta_t
\]
\[
w_{t+1} = w_t - \hat{\eta}_t \frac{m_t}{\sqrt{\eta_t}}
\]
where $\mu_t \in [0.5, 1]$. Thus we can refer the Theorem 10.5 of [5] in our proving.

If we firstly focus on the $i_{th}$ element of $w_t$, we can get

$$
(w_{t+1,i} - w^*_i)^2 = (w_{t,i} - w^*_i)^2 \cdot \frac{m_t}{\sqrt{v_t}}^2
$$

$$
= (w_{t,i} - w^*_i)^2 - 2\hat{h}_t \cdot \frac{m_t}{\sqrt{v_t}} (w_{t,i} - w^*_i) + (\hat{h}_t \cdot \frac{m_t}{\sqrt{v_t}})^2
$$

$$
= (w_{t,i} - w^*_i)^2 - 2\hat{h}_t (1 - \beta_{1,t}) \frac{m_{t-1,i}}{\sqrt{v_{t,i}}} \cdot (w_{t,i} - w^*_i)
$$

$$
+ (\hat{h}_t \cdot \frac{m_t}{\sqrt{v_t}})^2
$$

Then

$$
2\hat{h}_t \cdot \beta_{1,t} \frac{m_{t-1,i}}{\sqrt{v_{t,i}}} (w_{t,i} - w^*_i) = (w_{t,i} - w^*_i)^2 - (w_{t+1,i} - w^*_i)^2
$$

$$
- 2\hat{h}_t (1 - \beta_{1,t}) \frac{m_{t-1,i}}{\sqrt{v_{t,i}}} \cdot (w_{t,i} - w^*_i)
$$

$$
+ \hat{h}_t^2 \cdot \frac{m^2_{t-1,i}}{v_{t,i}}
$$

So we can obtain

$$
g_{t,i}(w_{t,i} - w^*_i) = \frac{\sqrt{v_{t,i}}}{2\hat{h}_t \beta_{1,t}} [(w_{t,i} - w^*_i)^2 - (w_{t+1,i} - w^*_i)^2]
$$

$$
+ \frac{(1 - \beta_{1,t})}{\beta_{1,t}} \frac{m_{t-1,i}}{\sqrt{v_{t,i}}} (w^*_i - w_{t,i})
$$

$$
+ \frac{\hat{h}_t}{2\beta_{1,t}} \frac{m^2_{t-1,i}}{\sqrt{v_{t,i}}}
$$

(15)

(16)

(17)

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

$$
\sum_{t=1}^{T} \frac{\sqrt{V_{t,i}}}{2\hat{h}_t \beta_{1,t}} [(w_{t,i} - w^*_i)^2 - (w_{t+1,i} - w^*_i)^2] \leq \frac{1}{\epsilon_0} \left\{ [(w_{1,i} - w^*_i)^2 - (w_{2,i} - w^*_i)^2] \cdot \frac{\sqrt{V_{1,i}}}{\eta_1}
$$

$$
+ [(w_{2,i} - w^*_i)^2 - (w_{3,i} - w^*_i)^2] \cdot \frac{\sqrt{V_{2,i}}}{\eta_2}
$$

$$
+ \ldots
$$

$$
+ [(w_{T-1,i} - w^*_i)^2 - (w_{T,i} - w^*_i)^2] \cdot \frac{\sqrt{V_{T-1,i}}}{\eta_{T-1}}
$$

$$
\leq \frac{1}{\epsilon_0} \left\{ [(w_{1,i} - w^*_i)^2] \cdot \frac{\sqrt{V_{1,i}}}{\eta_1}
$$

$$
+ (w_{2,i} - w^*_i)^2 \left( \frac{\sqrt{V_{2,i}}}{\eta_2} - \frac{\sqrt{V_{1,i}}}{\eta_1} \right)
$$

$$
+ \ldots
$$

$$
+ (w_{T,i} - w^*_i)^2 \left( \frac{\sqrt{V_{T,i}}}{\eta_T} - \frac{\sqrt{V_{T-1,i}}}{\eta_{T-1}} \right)
$$

$$
\leq \frac{D^2}{\epsilon_0 \eta} \sqrt{T \nu_{T,i}}
$$
Therefore, the final bound of \( R(T) \) can be expressed as

\[
R(T) \leq \frac{D^2}{\epsilon_0^2} \sum_{i=1}^{d} \sqrt{Tv_{T,i}} + \frac{(1 - \epsilon_0)^2 G_\infty D_\infty d}{K(1 - \lambda)^2 \epsilon_0} + \frac{\eta \sqrt{1 + \log T}}{2(1 - \gamma_1) \sqrt{1 - \beta_2}} \sum_{i=1}^{d} \|g_{1:T,i}\|_2.
\]

The third and ninth inequality is based on Cauchy-Schwarz Inequality.

Therefore, the final bound of \( R(T) \) can be expressed as

\[
R(T) \leq \frac{D^2}{\epsilon_0^2} \sum_{i=1}^{d} \sqrt{Tv_{T,i}} + \frac{(1 - \epsilon_0)^2 G_\infty D_\infty d}{K(1 - \lambda)^2 \epsilon_0} + \frac{\eta \sqrt{1 + \log T}}{2(1 - \gamma_1) \sqrt{1 - \beta_2}} \sum_{i=1}^{d} \|g_{1:T,i}\|_2.
\]