DEAM: Adaptive Momentum with Discriminative Weight for Stochastic Optimization

Jiyang Bai*, Yuxiang Ren† and Jiawei Zhang†
*Department of Computer Science, Florida State University, FL, USA
†IFM Lab, Department of Computer Science, Florida State University, FL, USA
Email: bai@cs.fsu.edu yuxiang@ifmlab.org, jiawei@ifmlab.org

***Abstract*—Optimization algorithms with momentum, e.g., (ADAM) helps accelerate SGD in parameter updating, which can minify the oscillations of parameters update route. However, the fixed momentum weight (e.g., β_1 in ADAM) will propagate errors in momentum computing. Besides, such a hyperparameter can be extremely hard to tune in applications. In this paper, we introduce a novel optimization algorithm, namely Discriminative wEight on Adaptive Momentum (DEAM). DEAM proposes to compute the momentum weight automatically based on the discriminative angle. The momentum term weight will be assigned with an appropriate value which configures the influence of momentum in the current step. In addition, DEAM also contains a 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 various models. A full version of this paper can be accessed in [1].**

I. Introduction

Deep learning methods can achieve outstanding performance in multiple fields including computer vision [5], natural language processing [3], and graph analysis [11]. Optimization algorithms are critical for deep learning methods: not only the model performance, but also training efficiency are greatly affected. In order to cope with the high computational complexity of training deep learning methods, stochastic gradient descent (SGD) [12] is utilized to update parameters based on the gradient of each training sample instead. The idea of momentum [9], inspired by Newton's first law of motion, is used to handle the oscillations of SGD. SGD with momentum [15] achieves the 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 [4] is the first optimization algorithm with adaptive learning rates, which makes use of the learning rate decay. ADAM [6] involves both adaptive learning [12] and momentum [9] 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.

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 updates, the fixed momentum weight fails to take the different influence of the current gradient into consideration, which will render errors in momentum computing. For example, when there exist parts of opposite eigen components [9] 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, instead of being placed with a fixed influence. We will illustrate this problem through cases in Section III-A1 where ADAM with a fixed weight β_1 cannot handle some simple but intuitive convex optimization problems. Based on this situation, we need to control the influence of momentum by an adaptive weight. What's more, designing hyperparameter-free optimization algorithms has been a very important 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 DEAM (**D**iscrim-inative w**E**ight on **A**daptive **M**omentum) to deal with the aforementioned problems. DEAM computes an adaptive momentum weight $\beta_{1,t}$ based on the "discriminative angle" θ between the historical momentum and the newly calculated gradient in each training iteration automatically. Besides, DEAM introduces a novel backtrack term, i.e., 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. We also provide extensive experiments to verify that the adaptive momentum term weight $\beta_{1,t}$ and the operation of backtrack term d_t can be crucial for the performance of the learning algorithms.

II. Related Works

Adaptive Learning Rates: To overcome the problems brought by the unified learning rate, some variant algorithms applying adaptive learning rate [2] have been proposed, such as AdaGrad [4], RMSProp [16], ADAM [6] and recent AdaBound [8]. AdaGrad adopts different learning rates to different variables, and its variable updating equation can be represented as

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

IEEE/ACM ASONAM 2020, December 7-10, 2020
978-1-7281-1056-1/20/$31.00 © 2020 IEEE

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
$\mathbf{m}_0 \leftarrow \mathbf{0}$; /* Initialize first-order momentum */
$\mathbf{v}_0 \leftarrow \mathbf{0}$, $\hat{\mathbf{v}}_0 \leftarrow \mathbf{0}$; /* Initialize second-order momentum */
for $t = 1, 2, \ldots, T$ **do**
  $\mathbf{g}_t = \nabla f_t(\mathbf{w}_t)$;
  $\theta = \left\langle \frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, \mathbf{g}_t \right\rangle$; /* The operator $\langle \cdot, \cdot \rangle$ represents the angle between two vectors. */
  if $\theta \in [0, \frac{\pi}{2})$ **then**
    $\beta_{1,t} = \sin\theta / K + \epsilon$;
  else
    $\beta_{1,t} = 1/K$ /* Here, $K = \frac{10(2+\pi)}{2\pi}$. */;
  end
  $\mathbf{m}_t = (1 - \beta_{1,t}) \cdot \mathbf{m}_{t-1} + \beta_{1,t} \cdot \mathbf{g}_t$;
  $\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t \odot \mathbf{g}_t$; /* $\odot$ is element-wise multiplication.*/
  $\hat{\mathbf{v}}_t = \max\{\hat{\mathbf{v}}_{t-1}, \mathbf{v}_t\}$;
  $d_t = \min\{0.5 \cos\theta, 0\}$;
  $\boldsymbol{\Delta}_t = d_t \cdot \boldsymbol{\Delta}_{t-1} - \eta_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}$;
  $\mathbf{w}_t = \mathbf{w}_{t-1} + \boldsymbol{\Delta}_t$;
end
return $\mathbf{w}_T$

where η is the learning rate and ∇ is the derivative of the loss function. We have to mention that the $\sum$, $\odot$ and $\sqrt{\ }$ in the above equation are element-wise operations. One drawback of AdaGrad is that with the increasing of iteration number t, the adaptive term may inflate continuously, which leads to a very slow convergence rate in the later stage of the training process. RMSProp [16] can solve this problem by using the moving average of historical gradients.

Momentum: Momentum [9], [14] is a method that helps accelerate SGD in the relevant direction and prevent oscillations on the descent route. The momentum accelerates updates for dimensions whose gradients are in the same direction as historical gradients, and decelerates updates for dimensions whose gradients are the opposite. ADAM [6] is proposed based on momentum and adaptive learning rates for different variables. It updating rules can be presented as:

$$\begin{cases} \mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t; \ \ \hat{\mathbf{m}}_t = \mathbf{m}_t / (1 - \beta_1^t) \\ \mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t \odot \mathbf{g}_t; \ \ \hat{\mathbf{v}}_t = \mathbf{v}_t / (1 - \beta_2^t) \\ \mathbf{w}_t = \mathbf{w}_{t-1} - \eta \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) \end{cases} \tag{2}$$

ADAM records the first-order momentum and the second-order momentum of the gradients using the moving average, and further computes the bias-corrected version of them. AMSGrad [10] is a modified version of ADAM, which redefines second-order momentum by a maximum function.

III. Proposed Algorithm

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 mini-batches 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 t_{th} 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 previous first-order momentum $\mathbf{m}_{t-1}$ and current gradient $\mathbf{g}_t$ when calculation 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 $\boldsymbol{\Delta}_{t-1}$ (defined in Algorithm 1) and the current update volume $\mathbf{m}_t/\sqrt{\hat{\mathbf{v}}_t}$ when computing $\boldsymbol{\Delta}_t$. In the proposed DEAM, $\beta_{1,t}$ and d_t terms can collaborate with each other and achieve faster convergence.

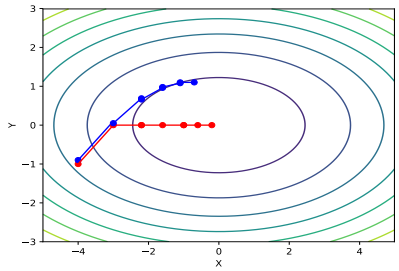


Figure 1. The update routes of ADAM with $\beta_1 = 0.9$ (the blue line) and $\beta_1 = 0.0$ (the red line).

A. Adaptive Momentum Weight $\beta_{1,t}$

1) Motivation: In the ADAM [6] paper, (the first-order) momentum's weight (i.e., β_1) is a pre-specified 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

$$f(x, y) = x^2 + 4y^2, \tag{3}$$

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 β_1 will lead to 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.

2) Mechanism: The momentum weight β_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 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 t_{th} iteration is calculated by

$$\theta = \left\langle -\frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, -\mathbf{g}_t \right\rangle = \left\langle \frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, \mathbf{g}_t \right\rangle \tag{4}$$

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 $\theta =< \frac{\mathbf{m}_{t-1}}{\sqrt{\hat{\mathbf{v}}_{t-1}}}, \mathbf{g}_t >$. 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:

$$\beta_{1,t} = \begin{cases} \sin\theta/K + \epsilon & \theta \in [0, \frac{\pi}{2}) \\ 1/K & \theta \in [\frac{\pi}{2}, \pi] \end{cases} \tag{5}$$

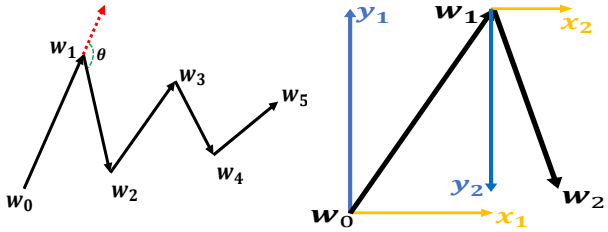
where $K = 10(2+\pi)/2\pi$ and ϵ is a very small value (e.g., $\epsilon = 0.001$). In the equation above, the threshold of the piecewise function is $\theta = \pi/2$, because $\sin\theta$ comes to the maximum at this point and goes down when $\theta > \frac{\pi}{2}$. If $\frac{\pi}{2} \leq \theta \leq \pi$, 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:

$$K = 10\left(\int_0^{\frac{\pi}{2}} \sin\theta d\theta + \int_{\frac{\pi}{2}}^{\pi} 1 d\theta\right) = \frac{10(2+\pi)}{2\pi} \tag{6}$$

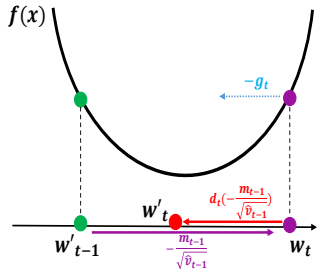
In the equation above, assume θ is randomly distributed on $[0, \pi]$, in this calculation we can get

$$\mathbb{E}[\beta_{1,t}] = \frac{1}{\pi}\int_0^{\pi} \beta_{1,t}(\theta) d\theta = 0.1 \tag{7}$$

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 [6]. 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.



(a) The "zig-zag" route (b) Axis decomposition



(c) Example: When $\theta = 180°$

Figure 2. Some examples about d_t

B. Backtrack Term d_t

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 2(a), especially when the discriminative angle θ is larger than $90°$. We call this phenomenon the "zig-zag" route. In Figure 2(a), it shows the update 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 update routes; the θ is the discriminative angle. If $\theta \geq 90°$, the "zig-zag" phenomenon will appear severely, which may lead to slower convergence speed. The main reason is when $\theta \geq 90°$, 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 2(b). For the example of a function with 2-dimension variables, the update volume $\mathbf{m}_1/\sqrt{\hat{\mathbf{v}}_1}$ can be decomposed into $(x_1, y_1)^\top$ in Figure 2(b), 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 over-update (or redundant update) of the first step. Therefore the backtrack term d_t is proposed to restrict this situation.

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 2(a). 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 t_{th} iteration we quantify d_t as the following equation:

$$d_t = \min\{0.5\cos\theta, 0\} \tag{8}$$

and we rewrite the updating term with backtrack in DEAM as

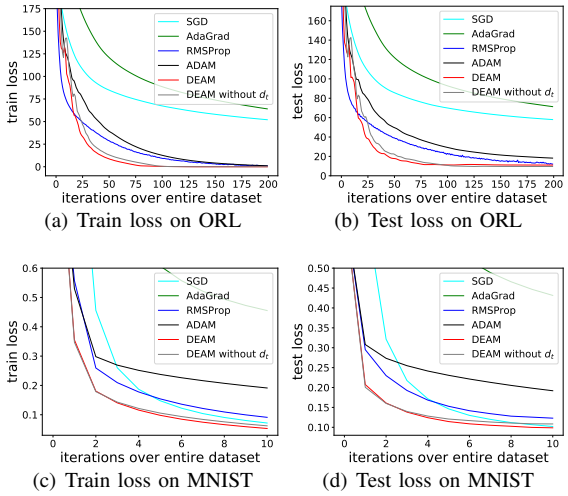
$$\mathbf{\Delta}_t = d_t \cdot \mathbf{\Delta}_{t-1} - \eta_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}} \tag{9}$$

where θ is the discriminative angle and $\mathbf{\Delta}_t$ 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 $\mathbf{\Delta}_t = -\eta_t \cdot \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t}}$ is similar to AMSGrad; when $\theta \to 180°$, $d_t = 0.5\cos\theta$ and comes to the maximum value when $\theta = 180°$. The reason that $\cos\theta$ is rescaled by 0.5 is that: in Figure 2(c), $\mathbf{w}_{t-1}$ and $\mathbf{w}_t$ 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 $\theta = 180°$, firstly $d_t = 0.5\cos\theta \to -0.5$ 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.

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.

IV. Experiments

We have applied the DEAM algorithm on multiple popular machine learning and deep learning structures, including logistic regression, deep neural networks (DNN), convolutional neural networks (CNN). These structures cover both convex and non-convex situations. To show the advantages of the algorithm, we compare it with various popular optimization algorithms, including ADAM [6], RMSProp [16], AdaGrad [4] and SGD. For all the experiments, the loss function we have selected is the cross-entropy loss. During the training process, the minibatch size for logistic regression, DNN and CNN structures is 128 and the learning rate is 0.0001.



(a) Train loss on ORL (b) Test loss on ORL

(c) Train loss on MNIST (d) Test loss on MNIST

Figure 3. Results of Logistic Regression and DNN structures

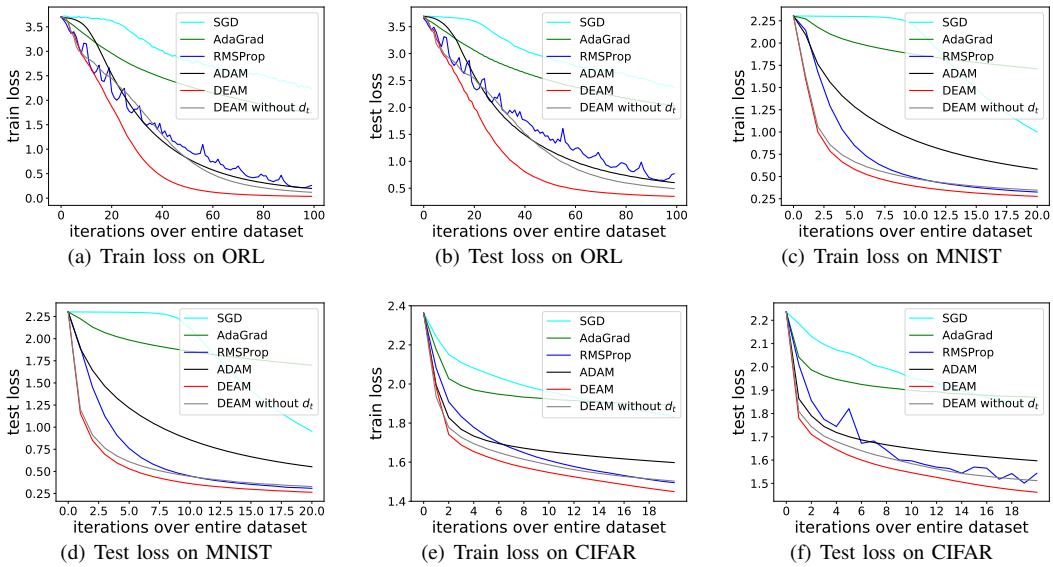
A. Experiment Settings and Results

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 ORL dataset [13]. ORL dataset consists of face images of 40 people, each person has ten images and each image is in the size of 112×92. The loss of objective functions on both training set and testing set are shown in Figure 3(a), 3(b).

Deep Neural Network: We use deep neural network (DNN) with two fully connected layers of 1,000 hidden units and the Relu activation function. The dataset we use is MNIST. The MNIST dataset 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. Result are exhibited in Figure 3(c), 3(d).

Convolutional Neural Network: The CNN models in our experiments are based on the LeNet-5 [7], and it is implemented on multiple datasets: ORL, MNIST and CIFAR-10. The CIFAR-10 dataset consists of 60,000 32×32 images in 10 classes, with 6,000 images per class. 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 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. The results are shown in Figure 4.

We can observe that DEAM converges faster than other widely used optimization algorithms in all the cases. Within the same number of epoches, DEAM can converge to the lowest loss on both the training set and test set.



(a) Train loss on ORL (b) Test loss on ORL (c) Train loss on MNIST

(d) Test loss on MNIST (e) Train loss on CIFAR (f) Test loss on CIFAR

Figure 4. Results of CNN structure

Table I
RUNNING TIME OF DEAM AND COMPARISON METHODS (THE UNIT OF VALUES IS SECOND)

Comparison Methods	Running time on all models				
	Logistic Regression on ORL	DNN on MNIST	CNN on ORL	CNN on MNIST	CNN on CIFAR-10
DEAM	**36**	**282**	**33382**	**11485**	**55928**
ADAM	102	664	47418	21775	67584
RMSProp	48	307	36722	11997	84305
AdaGrad	> 200	667	> 100000	> 50000	> 100000
SGD	> 200	346	> 100000	16985	67564

B. Time-consuming Analysis

We have recorded the running time of DEAM and other comparison algorithms in every experiment, and list them in the Table I. The running time shown in Table I 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 mount of time. From the results in Figures 3, 4, and Table I, we can conclude that DEAM can converge not only in fewer epochs, but using less running time. The device we used is the Dell PowerEdge T630 Tower Server, with 80 cores 64-bit Intel Xeon CPU E5-2698 v4@2.2GHz. The total memory is 256 GB, with an extra (SSD) swap of 256 GB.

V. 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. Extensive experiments have shown that the proposed algorithm can converge faster than existing methods on both convex and non-convex situations, and the time consuming is better than existing methods.

VI. ACKNOWLEDGEMENT

This work is partially supported by NSF through grant IIS-1763365 and by FSU.

REFERENCES

[1] Jiyang Bai, Yuxiang Ren, and Jiawei Zhang. Deam: Adaptive momentum with discriminative weight for stochastic optimization. *arXiv*, pages arXiv–1907, 2019.

[2] Laxmidhar Behera, Swagat Kumar, and Awhan Patnaik. On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE transactions on neural networks*, 2006.

[3] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. Question answering over freebase with multi-column convolutional neural networks. In *ACL*, 2015.

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

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

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

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

[8] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In *ICLR*, 2019.

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

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

[11] Yuxiang Ren, Bo Liu, Chao Huang, Peng Dai, Liefeng Bo, and Jiawei Zhang. Heterogeneous deep graph infomax. *arXiv preprint arXiv:1911.08538*, 2019.

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

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

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

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

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