




OPEN

Improving the efficiency of RMSProp optimizer by utilizing Nesterov in deep learning

Reham Elshamy , Osama Abu-Elnasr, Mohamed Elhoseny & Samir Elmougy

There are several methods that have been discovered to improve the performance of Deep Learning (DL). Many of these methods reached the best performance of their models by tuning several parameters such as Transfer Learning, Data augmentation, Dropout, and Batch Normalization, while other selects the best optimizer and the best architecture for their model. This paper is mainly concerned with the optimization algorithms in DL. It proposes a modified version of Root Mean Squared Propagation (RMSProp) algorithm, called NRMSProp, to improve the speed of convergence, and to find the minimum of the loss function quicker than the original RMSProp optimizer. Moreover, NRMSProp takes the original algorithm, RMSProp, a step further by using the advantages of Nesterov Accelerated Gradient (NAG). It also takes in consideration the direction of the gradient at the next step, with respect to the history of the previous gradients, and adapts the value of the learning rate. As a result, this modification helps NRMSProp to convergence quicker than the original RMSProp, without any increase in the complexity of the RMSProp. In this work, many experiments had been conducted to evaluate the performance of NRMSProp with performing several tests with deep Convolution Neural Networks (CNNs) using different datasets on RMSProp, Adam, and NRMSProp optimizers. The experimental results showed that NRMSProp has achieved effective performance, and accuracy up to 0.97 in most cases, in comparison to RMSProp and Adam optimizers, without any increase in the complexity of the algorithm and with fine amount of memory and time.

When deciding to build a structure in Deep Learning (DL)¹ to solve a specific problem, many questions arise for creating a powerful structure with great accuracy, and optimal training time. The most popular questions are: which hyper parameters, such as learning rate², batch size³, momentum⁴, and weight decay⁵, should be used? How to tune them to achieve the desired result? Which optimizer is the optimal one to answer the research problem? Actually, there are no specific guidelines for setting up an optimal DL structure with these parameters suitable for solving all problems or all data sets. Recently, there have been many researches interested in investigating hyper parameters and finding out the suitable values of these parameters for solving different problems. However, this paper focuses on the work mechanism of optimizers, and how to enhance their performance with regard to solving specific problems.

The aim of any optimizer is to minimize the loss function⁶, which is the difference between the actual output from the structure, and the desired output. There are many types of optimizers that can be used to achieve the minimum value for the loss function. Many papers have proposed new enhancement techniques on the traditional optimizers such as Stochastic Gradient Descend (SGD)⁷, AdaGrad⁸, AdaDelta⁹, Nadam¹⁰, Adam¹¹, and RmsProp¹². They include modification of many aspects such as the momentum, and learning rate. This paper proposed an algorithm, NRMSProp, to improve the performance of RmsProp optimizer, by adding a further step that involves calculating the Nesterov for a further point, with respect to the average of the past squared gradients for the current point. The performance of NRMSProp model is compared with the performance of the RMSProp and Adam optimizers under the same conditions.

The remainder of this paper is divided into five sections. The second section contains the “Literature review”. Section “Proposed Model NRMSProp” discusses the proposed optimizer in detail. Section “Experiments” presents the experiments on different datasets. Finally, section “Conclusion” concludes the work.

Faculty of Computers and Information, Mansoura University, Mansoura, Egypt. ✉email: reham_elshamy@oi.edu.eg

Literature review

Background. In this subsection, the focus is on the idea of how the optimizers work in Deep Neural Networks (DNNs). It presents some previous enhancement techniques of different traditional optimizers that deal with DL problems. Then, some modified versions of these optimizers are presented, with discussing how they work on different problems under different conditions. Training deep models is effectively remaining one of the most required tasks for researchers and practitioners in both real-world DL research, and application work. So far, the vast majority of the deep model training is based on the back propagation algorithm, which propagates the errors from the output layer backward, and uses gradient descent-based optimization algorithms to update the parameters layer by layer. Therefore, in order to achieve an effective model, the suitable optimizer to handle the problem should be chosen. There are different types of optimizers, such as Gradient Descent based Learning (GD), Momentum based Learning, Adaptive Gradient based Learning, and Momentum Adaptive Gradient based Learning Algorithms.

Gradient descent based learning (GD) algorithms. GD¹³ is utilized to reduce some functions by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. It is used to update the parameters' model in DL. In addition; there are multiple types of GD, such as the stochastic gradient batch (vanilla)¹⁴, gradient descent, or mini-batch gradient descent. The main difference between Batch Gradient Descent (BGD)¹⁵ and Stochastic Gradient Descent (SGD)¹⁶ is the cost of one example for each procedure in SGD is only computed. In contrast, in BGD, the cost for all training examples in the dataset has to be computed. This extremely quickens the neural networks. Basically, this is what stimulates utilizing SGD. SGD is utilized to update parameters in a DL, as shown in Eq. (1). Besides, this equation is employed to update parameters in a backwards pass, with the help of back propagation¹⁷, to compute the gradient. Each parameter, θ , is taken and updated by getting the original parameter and subtracting the learning rate times the ratio of change.

$$\theta = \theta - \eta \cdot \nabla_F(\theta) \quad (1)$$

It is noteworthy that in order to solve these drawbacks of SGD, an enhancement should be done. Mini Batch Gradient Descent¹⁸ would be adopted because it has the best of the two approaches. It also executes an update for each batch of n training examples in every batch. The following sub-section is devoted for discussing the other types of optimization algorithms.

Momentum based learning algorithms. Instead of depending only on the gradient of the current step to steer the search, momentum, the gradient of the previous steps is considered¹⁹. The gradient descent equations are changed as given in Eq. (2).

$$\mu(\tau) = \gamma\mu(\tau - 1) + \eta\nabla_F(\theta) \quad (2)$$

In the end, the parameters are updated through $\theta_{\tau+1} = \theta_{\tau} - \mu_{\tau}$. Thus, this allows us to fine-tune the updates to the slopes of our error function, which speeds up SGD, as shown in Figure 1 which illustrates how Momentum speeds up the SGD in training process²⁰. Moreover, it helps adapt the updates to every individual parameter to execute larger or smaller updates relying on their impact. In the sub-section below, some adaptive algorithms are introduced.

The Nesterov Momentum update²¹ is a significantly modified adaptation of the momentum update that has recently gained popularity. It has higher efficiency convergence promises for convex functions. It also performs slightly better in practice than the ordinary momentum. When the current parameter vector is at x , it is interfered from the momentum update above that the momentum term alone is about to nudge the parameter vector by μu^*v . Therefore, to compute the gradient, the future approximate position, $x + \mu u^*v$, should be considered as a looked-ahead²² at a further point to stop at. As a result, instead of computing the gradient at the previous position, x , compute it at $x + \mu u^*v$ as clarified in Fig. 2.

Adaptive gradient based learning algorithms. This subsection investigates a group of learning algorithms, such as Adagrad, Adadelata, and RMSProp, that use adaptive learning rates to update variables. Adagrad²³ is a technique that permits the learning rate to adapt depending on the parameters. Thus, it provides big updates for infrequent parameters, and small updates for frequent parameters. Also, it is well known for being appropriate for handling sparse data. This technique utilizes a different learning rate for each parameter at a time step based on the past gradients that were calculated for that parameter. Adadelata²⁴ is considered to be an expansion of AdaGrad that lean towards to eliminate the decay caused by the learning rate. This strategy limits the window of collected past gradients to few settled estimate weights rather than pressing the entire squared first-order derivative by using a decay average. RMSProp can be thought of as an extension of AdaGrad which uses a moving

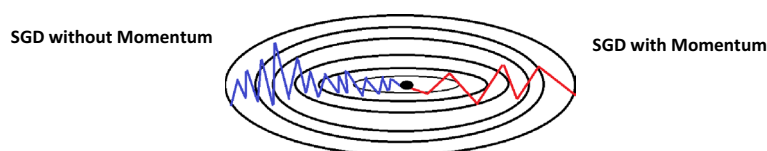


Figure 1. SGD without momentum via with momentum.

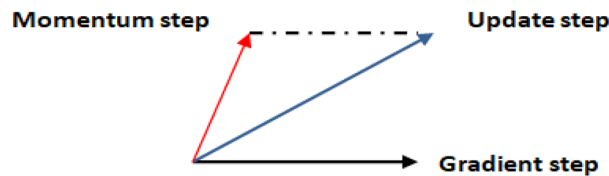


Figure 2. The effect of Nesterov on the Gradient step.

average (MA) of the partial derivatives instead of the sum in the calculation of the η for each instance as clarified in Eqs. (3) and (4) where S is the squared decay average.

$$S_t = \beta_1 S_{t-1} + (1 - \beta_1) g^2 \quad (3)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{S_t} + \epsilon} g_t \quad (4)$$

It is available to compute learning rates for every parameter. Moreover, individual changes for every parameter could be calculated and packed independently.

Momentum and adaptive gradient based learning algorithms. This subsection discusses some learning algorithms that incorporate the benefits of the previous two algorithms such as Adaptive Moment Estimation (Adam)²⁵ which can be defined as another approach which calculates the adaptive learning rates for every parameter. Besides, it keeps the average of all past squared gradients, such as AdaDelta. Adam is one of the effective optimization approaches in adaptive algorithms. It is an SGD algorithm that relies on the concept of momentum to quickly and effectively arrive at the loss function's global minimum. This aids in efficiently modifying the learning rate for each parameter, hastening the time to convergence to the minimum. Adam calculates distinct adaptive learning rates based on the first and second moment values for various parameters as clarified in Eqs. (5) and (6).

$$\hat{\mu}_t = \frac{\mu_t}{1 - \beta_1^t} \quad (5)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (6)$$

where μ is the mean and v is the variance of the first-order derivative in the same order. Equation (7) gives the final step of updating θ .

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{\mu}_t \quad (7)$$

Adam performs well in practice, and provides results that can be favorably compared to other optimization algorithms since it achieves the optima fast, and its performance is quiet quick and so effective. The technique of adaptive algorithms has the ability to also correct and fix each issue that can be confronted in other optimization algorithms that may cause fluctuation in the loss function adaptive techniques. There is an issue concerning the learning rate to be "just right", which is so tricky. If it is selected too small, there will be no progress. On the other hand, if it is too large, the solution will fluctuate, and be in the worst condition; it may even diverge. So, if it is specified and selected automatically, or even this step is avoided, the second order techniques, which search not only for the value and gradient of the objective but also for its curvature, can be beneficial in this case.

Related work. Dozat²⁶ improved Adam and explained and pointed out how to regenerate of Nesterov Accelerated Gradient (NAG) to be more direct and precise concerning performance. In this work, he had not executed the step of modifying the parameters with just the momentum procedure to compute the gradient in order to get back to the main parameter state. After that, the momentum procedure continues again during the real authentic update. Furthermore, the time of the momentum procedure can be applied only one time during the update of the previous time phase. Tato and Nkambou²⁷ introduced additional hyperparameters to Adam optimizer that preserves the direction of the gradient through ingrained optimization execution. On the other hand, Keskar and Socher²⁸ created a modified version of Adam (AAdam) to accelerate and quicken its performance. They aim to obtain a better minimum for the loss function, in comparison with the main algorithm, by extracting some thoughts from the momentum relying on the optimizer and exponential decay methodology. The clarified also that the procedure magnitude is produced by Adam to adjust the parameters. It is the way that the new procedure takes into account both of the direction of the gradient and the modification applied to the previous procedures. The authors also used MNIST data set for evaluation. The results showed that AAdam achieves the best results particularly on the validation set, even in the cases that require more memory. Also, they showed that it surpasses and outturns Adam and NAdam in decreasing training, and validation loss, and achieve better accuracy than the other methodologies.

Reddi et al.²⁹ introduced a combined hybrid methodology that begins with a flexible technique and then switches to SGD when convenient. They also presented SWATS which is considered as a simple methodology that converts from Adam to SGD, when a triggering case is satisfied. The case which is introduced is related to the projection of the Adam procedures on the gradient subspace. The cost of the observation procedure of this case is very low, and does not expand the number of hyperparameters in the optimizer. Furthermore, the authors created both switchover point and learning rate for SGD after the switch is assigned as a part of the algorithm. Due to these requirements, no convenient effort is added. Additionally, the authors demonstrated the adequacy of this methodology on ImageNet data sets. The results clarified that the proposed methodology compared with to SGD, while retaining the beneficial qualities of Adam such as the insensitivity, and quick initial advance of the hyperparameter. Hoseini et al.³⁰ proposed an algorithm to enable the training model in DCNN to switch between RMSprop and Nesterov optimizers to grantee that the AdaptAhead optimizer does not make any change of the structure of RMSprop and Nesterov algorithms. AdaptAhead built three switches; the first is responsible for a set the value of hyper-parameter norms, which corresponds to norm-1, Euclidean norm, and max-norm. The second switch determines whether gradients whether in the normal or in the Nesterov method. The third switch determines when the learning rate works whether by applying the calculated norm in an adaptive manner, or in the normal manner based on Nesterov method. Xue et al.³¹ suggested an approach to enhance the training of feed-forward NNs, which integrates the advantages of Differential Evolution and Adam. This approach explores the search space using a population-based method and adaptively modifies the learning rate to hasten convergence. Their results show that this proposed approach exhibited impressive outcomes in terms of accuracy and convergence speed across.

Wang et al.³² presented architecture for communication-efficient compressed federated adaptive gradient optimization, FedCAMS, which tackles the adaptively problem in federated optimization techniques while substantially reducing communication overhead. He suggested a universal adaptive federated optimization framework, FedAMS, as a base for FedCAMS. FedAMS, that includes different iterations of Adam characteristic max stabilization techniques. They offer an enhanced theoretical examination of the convergence of adaptive federated optimization, based on which they demonstrate that their suggested FedCAMS accomplishes the same convergence rate as its uncompressed counterpart FedAMS with a number of magnitude less communication cost in the no convex stochastic optimization context.

Proposed model NRMSProp

The main contribution of this work is to propose a model to enhance the performance of Adaptive Gradient based Learning algorithms in different ways such as the time and the accuracy. The proposed NRMSprop model gains its power from the advantages of Nesterov approach and the way in which RMSprop is calculating the gradients. The steps of NRMSprop are shown in Algorithm 1.

Algorithm 1. NRMSProp			
η : the learning rate β_1, β_2 : smoothing parameters $f(\theta)$: the objective function with parameter θ			
While: $f(\theta)$ does not converged do			
Step 1:	$\tau = \tau + 1$		
Step 2:	compute the gradient at step τ	$g_t = \nabla f(\theta_{t-1})$	(8)
Step 3:	calculate Nesterov momentum vector	$\mu_t = \beta_2^t \mu_{t-1} + (1 - \beta_2^t) g_t$	(9)
Step 4:	calculate the square of exponential moving average with term $(g_t - \mu_t)$	$s_t = \beta_1^t s_{t-1} + (1 - \beta_1^t) (g_t - \mu_t)^2$	(10)
Step 5:	correction of Bias	$\hat{\mu}_t = \frac{\mu_t}{1 - \beta_2^t}$	(11)
Step 6:	Apply the update of θ	$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \hat{\mu}_t$	(12)
End While			

As clarified in Algorithm 1, Step 1 allocates the initial time, τ , and Step 2 calculates the initial gradient g_t , which is the main parameter that optimizers of DL used to update the overall weights and the other parameters of the optimization process as the end of the process in Step 6. In the first iteration, g_t , is computed depending on initial random point as a start point. As the goal of optimization process is reducing the difference between this start point and the actual target minima NRMSprop computes the Nesterov vector μ in Step 3 and adding β_2^t to its computations that can effectively affect on how to apply look-ahead technique by calculating the gradients not at the current point, but with respect to the approximate future point. In Step 4, NRMSprop keeps the value of all past gradients as a history parameter of each movement and calculates the exponential moving average (s). Keeping this history helps NRMSprop to be more stable and prevents overshoots when the target minima is very close. As in adaptive learning technique, the step size is not a static value for all iterations, NRMSprop takes care of the step size and adapts it for the optimal value depending on the distance between the current position and the minima, so there is a need to penalty the gradient with μ in term $g_t - \mu_t$ to reduce the step size if the target point is near to the current position. To decide the next step and choose the right direction, NRMSprop calculates the correction of Bias $\hat{\mu}_t$ in step 5. Finally in step 6 we update the weights θ by the form of Eq. (12) with adding Nesterov by replacing $(\hat{\mu}_{t-1})$ of the earlier step by $(\hat{\mu}_t)$ of the current momentum vector to get θ_{t+1} as the last step of NRMSprop. To examine this proposed method we did experiments on Fashion-MNIST, CIFAR-10 and Tiny- ImageNet datasets using three Adam, RMSProp and the proposed optimizer NRMSPROP

under same environment to compare the result of NRMSProp with the others optimizers, the next subsection show the experiments.

Experiments

Datasets description. Fashion-MNIST³³ consists of 60,000 28×28 grayscale images of 10 fashion categories, along with a test set of 10,000 images. Figure 3 shows these 10 categories³⁴. CIFAR-10 data³⁵ includes 6000 images per class in 10 classes totaling 60,000 32×32 colour images; sample of this dataset are shown in Fig. 4. These images are splitting into 10,000 test photos and 50,000 training images are available. Tiny ImageNet dataset³⁶ is a version of Image Net dataset. It is a container of 200 categories in which there are 100,000 images in these categories and 10,000 images for each validation and test processes. Sample of this dataset are shown in Fig. 5.



Figure 3. Class names and example images in Fashion-MNIST dataset³⁴.

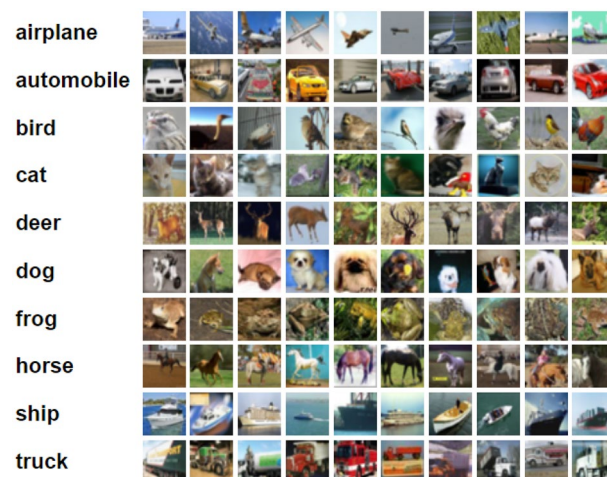


Figure 4. Class names and example images in CIFAR-10 dataset³⁵.



Figure 5. Sample of images in Tiny ImageNet dataset³⁷.

Accuracy (ACC), Precision (PREC), Recall (REC), and F1-score (F1) criteria, presented in Eqs. (13), (14), and (15), are used to evaluate NRMSPop, and to compare it with Adam and the original RmsProp optimizers³⁸.

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (13)$$

$$\text{PREC} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (14)$$

$$\text{REC} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (15)$$

$$\text{F1} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \quad (16)$$

The model structure. To examine NRMSPop, we used two models: a simple Convolution Neural Network (CNN)³⁹ model and ResNet model. Figure 6 illustrates the structure of CNN layers^{40,41}.

Table 1 illustrates the structure of NRMSPop model and its layers with 50 epochs for training, where the values of the hyper parameters are: η is the learning rate (the default is 10⁻³), β_1 and β_2 are the smoothing parameters ($\beta_1 = 0.9$, and $\beta_2 = 0.999$), and ϵ is a small number and tis usually set as 10⁻⁷.

Residual Network (ResNet)⁴² is a group of DNN models that has attained outstanding results on a variety of tasks associated with computer vision, including segmentation based on semantics, recognizing objects, and classification of images. The introduction of residual links, which enables the network to develop residual mappings that may be quickly optimized using gradient-based approaches, is the primary improvement of the ResNet models. The difference between the input and output of a group of convolutional layers, which is then added back to the input, is used for computing these residual mappings. Instead of trying to learn the complete mapping from scratching the network can learn to concentrate on the disparities between the input and the desired output in this manner. There are many depths of ResNet models, ranging from the original ResNet-18 to the far deeper ResNet-152. These models have already been trained using massive datasets like ImageNet. Here, ResNet

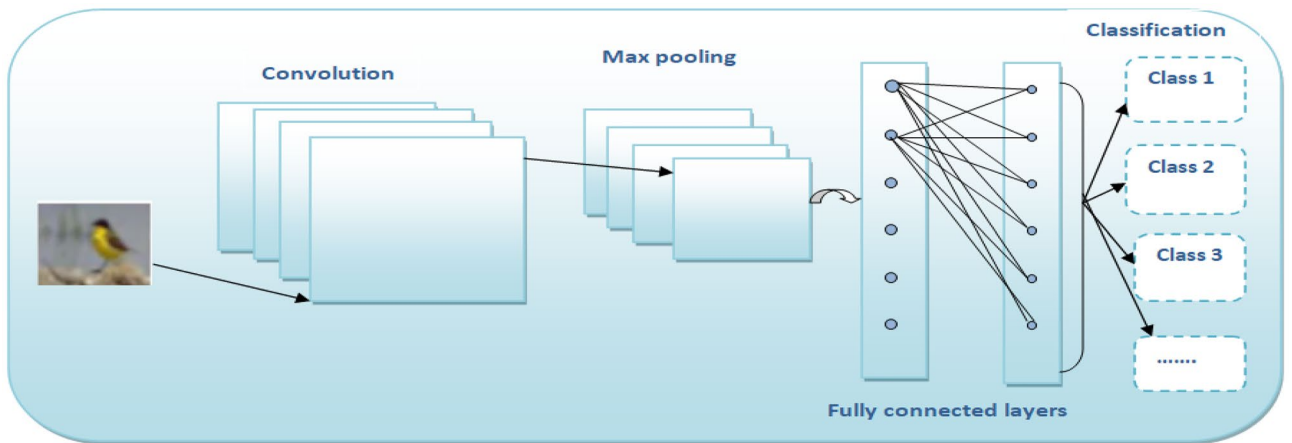


Figure 6. CNN layers.

CNN Model		
Layer (type)	Output shape	Param
conv2d14 (conv2d)	(None, 26, 26, 32)	320
(maxpooling)	(None, 13, 13, 32)	0
flatten18 (flatten)	(None, 5408)	0
dense37 (dense)	(None, 128)	692,352
dense38 (dense)	(None, 10)	1290
Total params: 693,962		
Trainable params: 693,962		
Non-trainable params:		

Table 1. The structure of the proposed CNN Model.

V2 model is used to perform the experiments on NRMSporop. Figure 7 illustrates the structure of ResNet_V2 layers. The results of the two models on the three datasets are illustrated in the next section.

Results of the experiments. Table 2 illustrates the overall performance of Adam on Fashion-MNIST dataset in which it achieves precision 98% which belongs to Class Trouser and Sandal, where 97% recall is the highest rate belongs to Class Sneaker and Bag. On the other hand, achieving 97% F1-score is the highest value belongs to Class Trouser. We present the classes of Fashion-MNIST dataset as (Class 1 refer to T-shirt/top, Class 2 refer to Trouser, Class 3 refer to Pullover, Class 4 refer to Dress, Class 5 refer to Coat, Class 6 refer to Sandal, Class 7 refer to Shirt, Class 8 refer to Sneaker, Class 9 refer to Bag and Class 10 refer to Ankle boot).

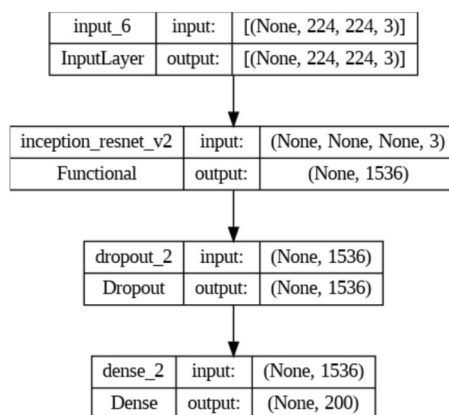


Figure 7. ResNet_V2 layers.

Fashion-MNIST dataset				
Classes	Precision	Recall	F1-score	Support
Class 1	0.83	0.78	0.80	1000
Class 2	0.98	0.96	0.97	1000
Class 3	0.81	0.80	0.80	1000
Class 4	0.84	0.87	0.86	1000
Class 5	0.71	0.88	0.78	1000
Class 6	0.98	0.93	0.95	1000
Class 7	0.71	0.56	0.63	1000
Class 8	0.91	0.97	0.94	1000
Class 9	0.94	0.97	0.96	1000
Class 10	0.95	0.94	0.95	1000
Accuracy			0.87	10,000
Macro avg	0.87	0.87	0.86	10,000
Weighted avg	0.87	0.87	0.86	10,000

Table 2. The Precision, Recall and F1-score of Adam.

Table 3 illustrates the overall performance of RMSProp on Fashion-MNIST dataset in which it achieves precision 99% which belongs to Class Sandal, where 97% recall is the highest rate belongs to Class Trouser. On the other hand, achieving 98% F1-score is the highest value belongs to Class Trouser.

Table 4 illustrates the overall performance of NRMSProp on Fashion-MNIST dataset in which it achieves precision 99% which belongs to Class Sandal and Trouser, where 98% recall is the highest rate belongs to Class Trouser, Sandal and Bag. On the other hand, achieving 98% F1-score is the highest value belongs to Class Trouser and Sneaker.

Next experiment we present classes of CIFAR-10 dataset as (Class 1 refer to Airplane, Class 2 refer to automobile, Class 3 refer to Bird, Class 4 refer to Cat, Class 5 refer to deer, Class 6 refer to Dog, Class 7 refer to Frog, Class 8 refer to horse, Class 9 refer to Ship and Class 10 refer to truck). Table 5 illustrates the overall performance of Adam on CIFAR-10 dataset in which it achieves precision 83% which belongs to Class ship, where 81% recall is the highest rate belongs to Class truck. On the other hand F1-score is 78% is the highest value belongs to Class automobile.

Table 6 illustrates the overall performance of RMSProp on CIFAR-10 dataset in which it achieves precision 91% which belongs to Class automobile, where 89% recall is the highest rate belongs to Class ship. On the other hand F1-score is 81% is the highest value belongs to Class automobile.

Table 7 illustrates the overall performance of NRMSProp on CIFAR-10 dataset in which it achieves precision 85% which belongs to Class automobile, where 87% recall is the highest rate belongs to Class ship. On the other hand F1-score is 81% is the highest value belongs to Class automobile.

From Figs. 2, 3, 4, 5, 6, 7, the overall performance of NRMSProp is showed to be higher than Adam and RMSprop in reaching high value in most classes of Fashion-MNIST and CIFAR-10 datasets.

The second criteria for the evaluation stage is constructing confusion matrix⁴³, in which it gives an accurate assessment of the model's accuracy in terms of true positives, true negatives, false positives, and false negatives. This aids in comprehending the model's performance and locating potential improvement areas. Also, it offers a

Fashion-MNIST dataset				
Class	Precision	Recall	F1-score	Support
Class 1	0.74	0.88	0.80	1000
Class 2	0.98	0.97	0.98	1000
Class 3	0.84	0.69	0.76	1000
Class 4	0.86	0.88	0.87	1000
Class 5	0.80	0.72	0.76	1000
Class 6	0.99	0.93	0.95	1000
Class 7	0.58	0.62	0.60	1000
Class 8	0.91	0.96	0.93	1000
Class 9	0.98	0.95	0.96	1000
Class 10	0.94	0.96	0.95	1000
Accuracy			0.86	10,000
Macro avg	0.86	0.86	0.86	10,000
Weighted avg	0.86	0.86	0.86	10,000

Table 3. The Precision, Recall and F1-score of RMSProp.

Fashion-MNIST dataset				
Class	Precision	Recall	F1-score	Support
Class 1	0.81	0.88	0.85	1000
Class 2	0.99	0.98	0.98	1000
Class 3	0.78	0.86	0.82	1000
Class 4	0.86	0.91	0.89	1000
Class 5	0.83	0.80	0.82	1000
Class 6	0.99	0.98	0.98	1000
Class 7	0.80	0.63	0.71	1000
Class 8	0.95	0.98	0.98	1000
Class 9	0.96	0.98	0.97	1000
Class 10	0.98	0.95	0.97	1000
Accuracy			0.90	10,000
Macro avg	0.90	0.90	0.98	10,000
Weighted avg	0.90	0.90	0.98	10,000

Table 4. The Precision, Recall and F1-score of NRMSProp.

CIFAR-10 dataset				
Class	Precision	Recall	F1-score	Support
Class 1	0.68	0.74	0.71	1000
Class 2	0.81	0.75	0.78	1000
Class 3	0.65	0.46	0.54	1000
Class 4	0.45	0.56	0.50	1000
Class 5	0.57	0.66	0.61	1000
Class 6	0.59	0.56	0.58	1000
Class 7	0.77	0.72	0.75	1000
Class 8	0.76	0.70	0.73	1000
Class 9	0.83	0.71	0.77	1000
Class 10	0.67	0.81	0.73	1000
Accuracy			0.67	10,000
Macro avg	0.68	0.67	0.67	10,000
Weighted avg	0.68	0.67	0.67	10,000

Table 5. The Precision, Recall and F1-score of Adam.

CIFAR-10 dataset				
Class	Precision	Recall	F1-score	Support
Class 1	0.69	0.69	0.69	1000
Class 2	0.91	0.73	0.81	1000
Class 3	0.73	0.49	0.58	1000
Class 4	0.52	0.43	0.62	1000
Class 5	0.73	0.53	0.62	1000
Class 6	0.63	0.60	0.62	1000
Class 7	0.81	0.72	0.76	1000
Class 8	0.57	0.85	0.69	1000
Class 9	0.63	0.89	0.74	1000
Truck	0.69	0.85	0.76	1000
Accuracy			0.68	10,000
Macro avg	0.69	0.68	0.67	10,000
Weighted avg	0.69	0.67	0.67	10,000

Table 6. The Precision, Recall and F1-score of RMSProp.

CIFAR-10 dataset				
Class	Precision	Recall	F1-score	Support
Class 1	0.72	0.74	0.73	1000
Class 2	0.85	0.77	0.81	1000
Class 3	0.57	0.63	0.60	1000
Class 4	0.48	0.58	0.53	1000
Class 5	0.74	0.51	0.61	1000
Class 6	0.71	0.49	0.58	1000
Class 7	0.73	0.79	0.76	1000
Class 8	0.71	0.77	0.74	1000
Class 9	0.69	0.87	0.77	1000
Class 10	0.79	0.73	0.76	1000
Accuracy			0.69	10,000
Macro avg	0.70	0.69	0.69	10,000
Weighted avg	0.70	0.69	0.69	10,000

Table 7. The Precision, Recall and F1-score of NRMSprop.

more in-depth and a picture of the model’s performance. It can assist in pinpointing certain instances where the model is functioning successfully or incorrectly. The model’s general effectiveness can be improved by detecting problem areas with the use of the confusion matrix, which can also be used to guide parameter adjustment and refining. Three confusion matrices are constructed for each dataset as given below.

Figures 8, 9, 10 show the confusion matrices for Fashion-MNIST datasets. Figure 8 illustrates that Adam is getting confused mostly between similar classes in comparison between actual and predicted values. Some of the higher values of negative classification are between these classes: (T-shirt/top and Shirt) (Pullover and Shirt) (Coat and Shirt) (Coat and Pullover) with values (117, 68, 165, 105). Figure 9 illustrates that RMSProp is getting confused in many classes in comparison between actual and predicted values. Some of the higher values of negative classification are between these classes: (T-shirt/top and Shirt) (Pullover and Coat) (Dress and Coat) (Coat and Pullover) (Shirt and coat) (Shirt and Pullover) with values (224, 66, 52, 92, 154, 168). Figure 10 illustrates NRMSprop is getting confused just in very similar classes in comparison between actual and predicted values. Some of the higher values of negative classification are between these classes: (T-shirt/top and Shirt) (Pullover



Figure 8. Confusion matrix of Adam for Fashion-MNIST dataset.



Figure 9. Confusion matrix of Rmsprop for Fashion-MNIST dataset.



Figure 10. Confusion matrix of NRMSprop for Fashion-MNIST dataset.

and Coat) (Pullover and Shirt) with values (147, 123,92). Depending on these experiments on Fashion-MNIST dataset, NRMSprop is shown to be the less confusion degree than Adam and Rmsprop.

Figures 11, 12, 13 show the confusion matrices for CIFAR-10 datasets. Figure 11 illustrates that Adam is getting confused in many classes in comparison between actual and predicted values. By focus on some of higher values of negative classification are between these classes we found many classes have high confusion value like (airplanes, cats, deer, dogs, and trucks). Figure 12 illustrates that that RMSProp is getting confused in many classes in comparison between actual and predicted values. By analyzing the behavior or RMSProp, the most of values are shown to be very close to each other and RMSProp is hardly distinguished images between

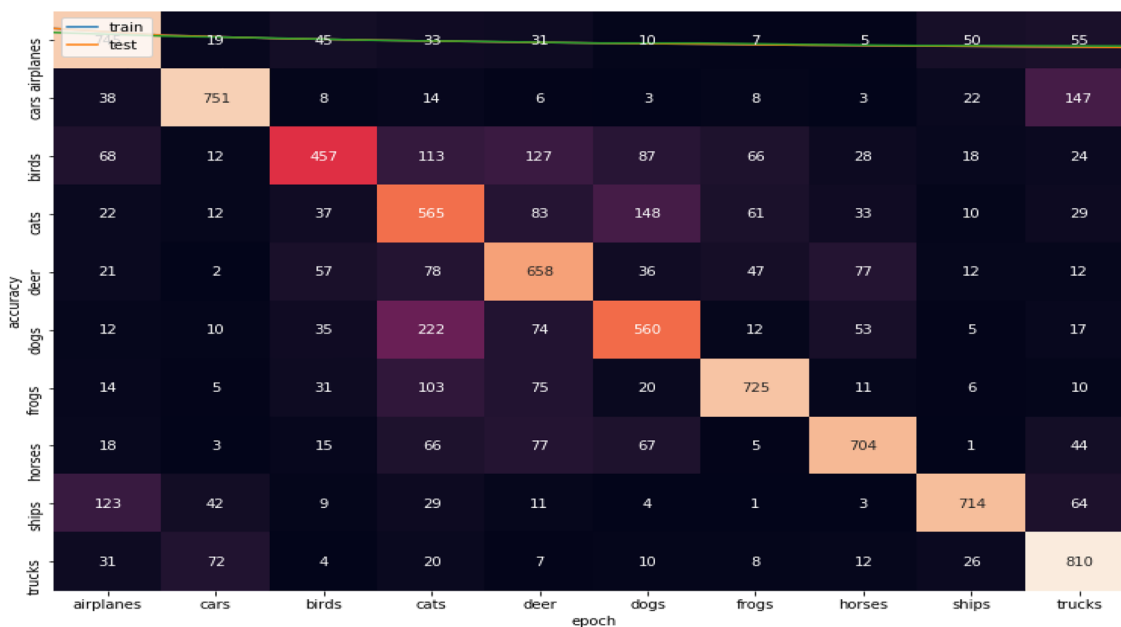


Figure 11. Confusion matrix of Adam for CIFAR-10.

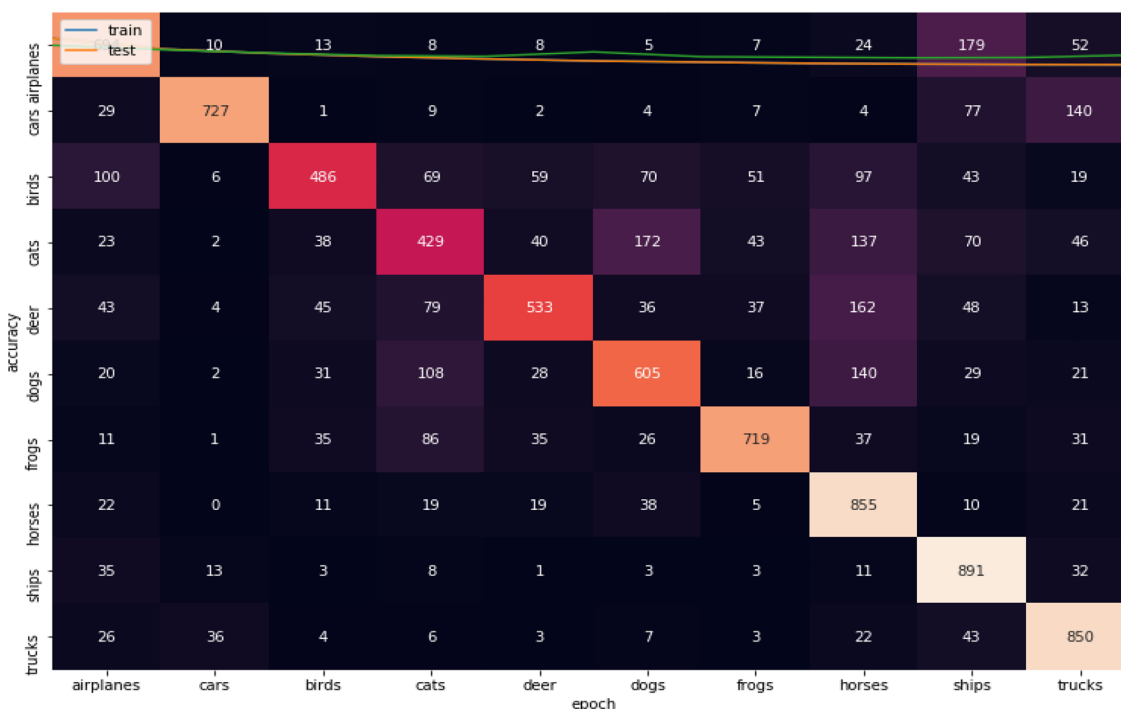


Figure 12. Confusion matrix of Rmsprop for CIFAR-10.

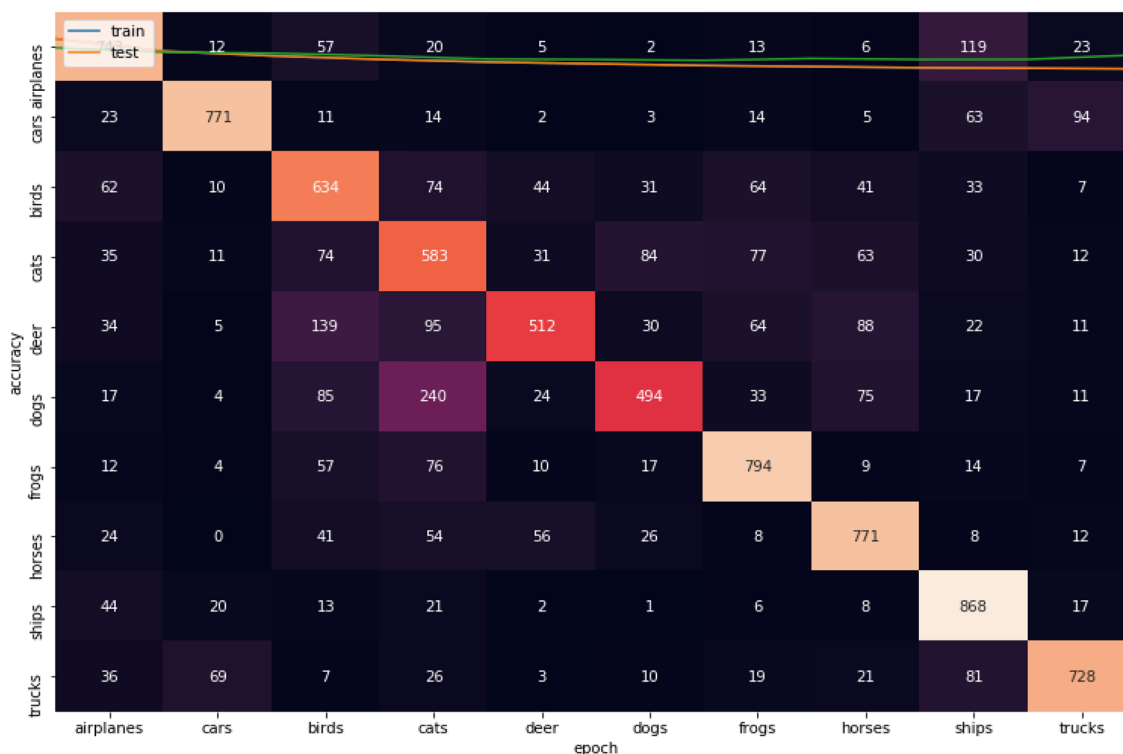


Figure 13. Confusion matrix of NRMSProp or CIFAR-10.

Dataset	Adam	RMSProp	NRMSProp
Fashion-MNIST	0.91	0.86	0.97
CIFAR-10	0.71	0.82	0.84
Tiny Imagenet	0.69	0.70	0.72

Table 8. The accuracy results of Adam, RMSProp, and NRMSProp.

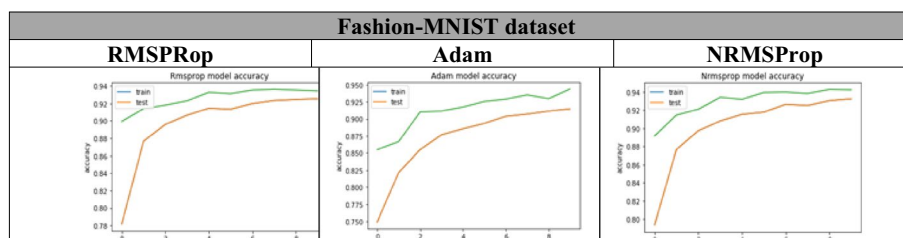


Figure 14. The accuracy of training and test curves of RMSProp, Adam, and NRMSProp on Fashion-MNIST dataset.

overall classes in dataset. Figure 13 illustrates that NRMSProp is getting confused in less number of classes in comparison between actual and predicted values compared with Adam and RMSProp. NRMSProp. By focus on some of higher values of negative classification are between these classes we found that classes with have high confusion values are (birds, cats, ships). Depending on these experiments on CIFAR-10 dataset, NRMSProp is shown to be the less confusion degree than Adam and RMSProp.

From Table 8, Figs. 14, 15, 16, it is clear that the proposed NRMSProp achieves good results in comparison with the other optimizers under all measurement criteria. The overall performance of the NRMSProp optimizer is more efficient after adding the Nestrove term to its steps. Therefore, the power of Nestrove can be utilized in to enhance the accuracy and to speed up the process of the optimizer in general. Moreover, NRMSProp has an effect feature that keeps the history of the current point to be more efficient in speeding up the process of taking the decision of which direction should be chosen. When adding Nestrove to NRMSProp’s steps, it gives

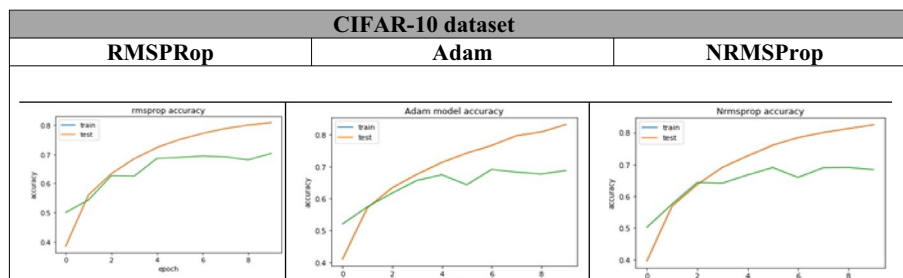


Figure 15. The accuracy of training and test curves of RMSPProp, Adam, and NRMSProp on CIFAR-10 dataset.

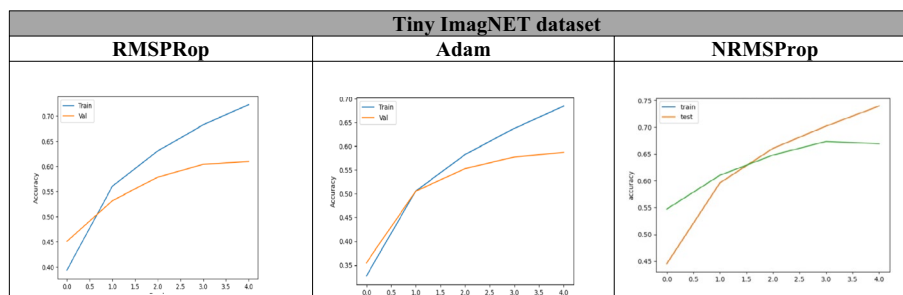


Figure 16. The accuracy of training and test curves of RMSPProp, Adam, and NRMSProp on Tiny Imagenet dataset.

NRMSProp the power to take a big step in the right direction by look ahead for projected positions not only the actual position and by adapting the learning step for each iteration depend on the minima position.

Conclusion

This paper proposed NRMSProp as a modified version of the Adaptive optimizer to increase its efficiency. It takes RMSPProp algorithm one step further since the efficiency of the Nesterov technique is combined with RMSPProp optimizer. Moreover, to examine the proposed algorithm NRMSProp, experiments are conducted on the Adam, RMSPProp, and NRMSProp optimizers on Fashion-MNIST, CIFAR-10 and Tiny-ImagNet datasets. Accuracy, Precision, Recall, F1-score, and confusion matrix are used to evaluate NRMSProp and compare its performance with Adam and RMSPProp optimizers on different datasets. The results showed that the NRMSProp achieved high accuracy under all measurements with fast convergence without noticeably increasing complexity.

Data availability

The datasets (Fashion-MNIST, cifar and ImagNet) used during the current study available in the following links respectively: Xiao et al.³³. Available: [arXiv:1708.07747](https://arxiv.org/abs/1708.07747), <https://www.cs.toronto.edu/~kriz/cifar.html>, <https://tiny-imagenet.herokuapp.com>.

Received: 5 March 2023; Accepted: 22 May 2023

Published online: 31 May 2023

References

- Shrestha, A. & Mahmood, A. Review of deep learning algorithms and architectures. *IEEE Access* 7(c), 53040–53065. <https://doi.org/10.1109/ACCESS.2019.2912200> (2019).
- Peace, I. C., Uzoma, A. O. & Ita, S. A. Effect of learning rate on artificial neural network in machine learning. *Int. J. Eng. Res. Technol.* 4(2), 359–363. <http://www.ijert.org> (2015) (online).
- Radiuk, P. M. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Inf. Technol. Manag. Sci.* 20(1), 20–24. <https://doi.org/10.1515/itms-2017-0003> (2018).
- Dahl, G. E., Sainath, T. N. & Hinton, G. E. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, pp. 8609–8613. <https://doi.org/10.1109/ICASSP.2013.6639346> (2013).
- Nakamura, K. & Hong, B. W. Adaptive weight decay for deep neural networks. *IEEE Access* 7, 118857–118865. <https://doi.org/10.1109/ACCESS.2019.2937139> (2019).
- Keren, G. Neural Network Supervision: Notes on Loss Functions, Labels and Confidence Estimation, Doctoral Thesis (Universität Passau, 2019).
- Alag, M. Stochastic Gradient Descent Variants and Applications (2022).
- Ward, R., Wu, X. & Bottou, L. Adagrad stepsizes: Sharp convergence over nonconvex landscapes. *36th Int. Conf. Mach. Learn. ICML 2019*, vol. 2019–June, pp. 11574–11583 (2019).
- Zeiler, M. D. ADADELTA: An Adaptive Learning Rate Method. [arXiv:1212.5701](https://arxiv.org/abs/1212.5701) (2012) (online).

10. Soydaner, D. A comparison of optimization algorithms for deep learning. *Int. J. Pattern Recognit. Artif. Intell.* **34**(13), 2052013. <https://doi.org/10.1142/S0218001420520138> (2020).
11. Schmidt, R. M., Schneider, F. & Hennig, P. Descending through a Crowded Valley—Benchmarking Deep Learning Optimizers. *arXiv:2007.01547* (2020) (**online**).
12. Reddy, S. V. G., Thammi Reddy, K. & Valli Kumari, V. Optimization of deep learning using various optimizers, loss functions and dropout. *Int. J. Innov. Technol. Explor. Eng.* **8**(2S), 272–279 (2018).
13. Ashok, A. Stochastic Gradient Descent for Deep Learning, pp. 1–10. <https://www.andrew.cmu.edu/user/anubhava/bettersgd.pdf>
14. Zhang, J. Gradient Descent based Optimization Algorithms for Deep Learning Models Training. *arXiv:1903.03614* (2019) (**online**).
15. Mandal, B. Stochastic Gradient Descent in Deep Learning. *Anal. Vidhya*. <https://medium.com/analytics-vidhya/stochastic-gradient-descent-in-deep-learning-b19c038ec025> (2021) (**online**).
16. Teng, Y. *et al.* Leader stochastic gradient descent for distributed training of deep learning models. *Adv. Neural Inf. Process. Syst.* **32**(NeurIPS), 1–11 (2019).
17. Busecma, M. Back propagation neural networks. *Subst. Use Misuse* **33**(2), 233–270. <https://doi.org/10.3109/10826089809115863> (1998).
18. Masters, D. & Luschi, C. Revisiting Small Batch Training for Deep Neural Networks, pp. 1–18. *arXiv:1804.07612* (2018) (**online**).
19. Craven, M. & Page, D. Deep Learning II: Momentum and Adaptive Step Size (2018).
20. Devarakonda. Revisiting Small Batch Training for Deep Neural Networks (2021).
21. Yang, Z., Bao, W., Yuan, D., Tran, N. H. & Zomaya, A. Y. Federated learning with Nesterov accelerated gradient. *IEEE Trans. Parallel Distrib. Syst.* **33**(12), 4863–4873. <https://doi.org/10.1109/TPDS.2022.3206480> (2022).
22. Lu, J. Gradient Descent, Stochastic Optimization, and Other Tales. *arXiv:2205.00832* (2022) (**online**).
23. Duchi, J. C., Bartlett, P. L. & Wainwright, M. J. Randomized smoothing for (parallel) stochastic optimization. *Proc. IEEE Conf. Decis. Control* **12**, 5442–5444. <https://doi.org/10.1109/CDC.2012.6426698> (2012).
24. Shazeer, N. & Stern, M. Adafactor: Adaptive learning rates with sublinear memory cost. *35th Int. Conf. Mach. Learn. ICML* **10**, 7322–7330 (2018).
25. Kingma, D. P. & Ba, J. L. Adam : A method for stochastic optimization. *3rd International Conference for Learning Representations, San Diego* (2015).
26. Dozat, T. Incorporating Nesterov momentum into Adam. *ICLR Workshop* **1**, 2013–2016 (2016).
27. Tato, A. & Nkambou, R. Improving ADAM Optimizer, Work. track-ICLR 2018, pp. 1–4. <http://yann.lecun.com/exdb/mnist/> (2018) (**online**).
28. Keskar, N. S. & Socher, R. Improving Generalization Performance by Switching from Adam to SGD. *arXiv:1712.07628* (2017) (**online**).
29. Reddi, S. J., Kale, S. & Kumar, S. On the convergence of Adam and beyond. *6th Int. Conf. Learn. Represent. ICLR 2018—Conf. Track Proc.*, pp. 1–23 (2018).
30. Hoseini, F., Shahbahrami, A. & Bayat, P. AdaptAhead optimization algorithm for learning deep CNN applied to MRI segmentation. *J. Digit. Imaging* **32**(1), 105–115. <https://doi.org/10.1007/s10278-018-0107-6> (2019).
31. Xue, Y., Tong, Y. & Neri, F. An ensemble of differential evolution and Adam for training feed-forward neural networks. *Inf. Sci.* **608**, 453–471 (2022).
32. Wang, Y., Lin, L. & Chen, J. Communication-efficient adaptive federated learning. *Proceedings of the 39th International Conference on Machine Learning, PMLR*, vol. 162, pp. 22802–22838 (2022).
33. Xiao, H., Rasul, K. & Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747* (2017) (**online**).
34. <https://github.com/zalandoresearch/fashion-mnist>
35. <https://www.cs.toronto.edu/~kriz/cifar.html>
36. Deng, J. *et al.* ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, pp. 248–255 (2009).
37. Chrabaszcz, P., Loshchilov, I. & Hutter, F. A Downsampled Variant of ImageNet as an Alternative to the CIFAR Dataset. *arXiv:1707.08819v3* (2017).
38. Powers, D. M. W. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *J. Mach. Learn. Technol.* **2**, 37–63 (2020).
39. Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A. & De, D. *Fundamental Concepts of Convolutional Neural Network* Vol. 172 (Springer, 2019). https://doi.org/10.1007/978-3-030-32644-9_36.
40. Kuo, C. C. J. Understanding convolutional neural networks with a mathematical model. *J. Vis. Commun. Image Represent.* **41**, 406–413. <https://doi.org/10.1016/j.jvcir.2016.11.003> (2016).
41. Bezdán, T. & Bačanin Džakula, N. Convolutional neural network layers and architectures. In *Sinteza 2019—International Scientific Conference on Information Technology and Data Related Research* 445–451. <https://doi.org/10.15308/Sinteza-2019-445-451> (Singidunum University, 2019).
42. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, June 27–30 (2016).
43. Ting, K. M. Confusion matrix. *Encycl. Mach. Learn. Data Min.*, pp. 260–260. https://doi.org/10.1007/978-1-4899-7687-1_50 (2017).

Author contributions

R.E. and O.A.-E. prepared the main algorithm of the manuscript, wrote and did all the experiments of this manuscript. M.E. reviewed the algorithm and all the results of all experiments. S.E. reviewed the algorithm and all the results of all experiments. All authors reviewed the manuscript.

Funding

Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to R.E.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2023