Neural Computing and Applications (2023) 35:17095–17112
https://doi.org/10.1007/s00521-023-08568-z

ORIGINAL ARTICLE

# A modified Adam algorithm for deep neural network optimization

Mohamed Reyad[1] · Amany M. Sarhan[1] · M. Arafa[1]

Received: 8 January 2022 / Accepted: 5 April 2023 / Published online: 25 April 2023

**Abstract**
Deep Neural Networks (DNNs) are widely regarded as the most effective learning tool for dealing with large datasets, and they have been successfully used in thousands of applications in a variety of fields. Based on these large datasets, they are trained to learn the relationships between various variables. The adaptive moment estimation (Adam) algorithm, a highly efficient adaptive optimization algorithm, is widely used as a learning algorithm in various fields for training DNN models. However, it needs to improve its generalization performance, especially when training with large-scale datasets. Therefore, in this paper, we propose HN Adam, a modified version of the Adam Algorithm, to improve its accuracy and convergence speed. The HN_Adam algorithm is modified by automatically adjusting the step size of the parameter updates over the training epochs. This automatic adjustment is based on the norm value of the parameter update formula according to the gradient values obtained during the training epochs. Furthermore, a hybrid mechanism was created by combining the standard Adam algorithm and the AMSGrad algorithm. As a result of these changes, the HN_Adam algorithm, like the stochastic gradient descent (SGD) algorithm, has good generalization performance and achieves fast convergence like other adaptive algorithms. To test the proposed HN_Adam algorithm performance, it is evaluated to train a deep convolutional neural network (CNN) model that classifies images using two different standard datasets: MNIST and CIFAR-10. The algorithm results are compared to the basic Adam algorithm and the SGD algorithm, in addition to other five recent SGD adaptive algorithms. In most comparisons, the HN Adam algorithm outperforms the compared algorithms in terms of accuracy and convergence speed. AdaBelief is the most competitive of the compared algorithms. In terms of testing accuracy and convergence speed (represented by the consumed training time), the HN-Adam algorithm outperforms the AdaBelief algorithm by an improvement of 1.0% and 0.29% for the MNIST dataset, and 0.93% and 1.68% for the CIFAR-10 dataset, respectively.

**Keywords** Optimizer · Adaptive Moment Estimation · Adam · AMSGrad · RMSprop · Nesterov Accelerated Adam · Deep Neural Networks

## 1 Introduction

Deep neural networks (DNNs) are widely regarded as the most popular and powerful machine learning method. They have been successfully applied in a variety of fields such as computer vision, natural language processing (NLP), bioinformatics, speech recognition, and medical computer technology, among others. DNNs are artificial neural networks (ANNs) that have multiple hidden layers. In DNNs, the number of hidden layers can reach 100 or more [1].

In recent years, the significant improvements in computer speeds, such as GPU accelerators and cloud computing, have greatly increased the prevalence of DNNs. They enable us to train deep neural networks much faster than before [2, 3]. Furthermore, the existence of a large

✉ Mohamed Reyad
m.r.elhelesy@xed.aucegypt.edu;
PG_38823@feng.tanta.edu.eg

Amany M. Sarhan
amany_sarhan@f-eng.tanta.edu.eg

M. Arafa
m.arafa@f-eng.tanta.edu.eg

1 Computers and Control Department, Faculty of Engineering, Tanta University, Tanta, Egypt

volume of data necessitates the development of deep neural networks to keep up with the growing volume of data. Furthermore, significant progress in DNN training methods has aided in the development of deep learning models that are now used in a wide range of applications. In the vast majority of applications, these models have delivered elegant results. All of these factors propelled DNNs to the forefront of the machine learning field, inspiring many researchers to work on improving their training methods [1, 4].

Deep neural networks can be optimized in a variety of ways, including optimizing the network model's structural design and determining the optimal parameters such as weights and biases of a predefined network structure, preprocessing of the datasets, and choosing the best optimization technique during the learning process. There are currently no established criteria for developing an ideal deep neural architecture [5]. Any optimizer's goal is to minimize an objective function, also known as a loss function, which is the difference between the expected and computed values. The minimization procedure determines the best set of parameters for designing DNNs for classification, prediction, and clustering tasks.

Many researchers [6, 7] present several optimization algorithms for deep neural networks in the literature. The gradient descent approach, which is a first-order differential method used to obtain an array of weights that satisfy the error criteria, is used by the majority of these algorithms. The most widely used optimization algorithms for deep neural networks in the literature are gradient descent techniques such as back-propagation and adaptive moment estimation (Adam) algorithms [8–10].

Training procedures remain relatively simple in comparison to the increasing complexity of deep neural network topologies. The majority of practical optimization approaches for DNNs employ the stochastic gradient descent (SGD) technique. However, as a hyper-parameter, the SGD learning rate is frequently difficult to tune and must be adjusted throughout the training process. To address this issue, several adaptive SGD variants have been developed, including adaptive gradient (AdaGrad), adaptive delta (Adadelta), root mean square propagation (RMSProp), and Adam. Based on the gradient statistics, these SGD variants aim to automatically adapt the learning rate of parameter updates. This explains why the SGD method is still used in training the most recent DNN models, particularly the feed-forward type [11].

The main goal of this paper is to create an optimization method that has good generalization performance like the SGD method while also achieving fast convergence like the adaptive methods. To address the shortcomings of current optimization algorithms, this paper proposes a modified Adam algorithm that does not require any additional parameters. The modified algorithm's main contribution is an increase in both convergence speed and accuracy. There is a mathematical proof that shows the differences between the modified and basic algorithms. Extensive experiments are carried out to demonstrate the proposed algorithm's superiority to state-of-the-art optimization algorithms, which are trained on two different datasets. The results show that the proposed algorithm outperforms the other algorithms in terms of convergence speed and accuracy.

The sections of the paper are organized as follows. The second section summarizes recent reviews of adaptive optimization techniques for deep neural network optimization. Section 3 describes the deep neural network architectures. The convolution neural network (CNN) is reviewed in Sect. 4. The concept and the mathematical proof of the modified Adam algorithm are discussed in detail in Sect. 5. The experiments and results discussion are given in Sect. 6. Section 7 concludes the paper and gives the future work.

## 2 Literature review of deep neural networks optimization

DNNs have been a hot topic in the machine learning community in recent years. The optimization methods used to train DNNs can be divided into two types: first-order optimization methods and second-order optimization methods [12]. The first-order derivative values of the objective function are used to direct the search process towards the steepest decreasing direction in first-order methods. It should be noted that the gradient denotes the first-order derivative of a multivariate objective function [12]. The gradient descent (GD) optimization algorithm is a popular first-order optimization algorithm that uses the objective function's negative gradient to find its minimum.

Since tuning the SGD algorithm's learning rate as a hyper-parameter is difficult, it is adjusted throughout the training process [14]. The adaptive variants of SGD algorithms attempt to automatically adapt the learning rate for parameter updates based on gradient statistics. Although these adaptive variants simplify learning rate settings and increase convergence speed, in some applications, their overall performance is significantly worse than the basic SGD algorithm. As a result, the SGD (possibly with momentum) algorithm is still used in training cutting-edge deep neural models such as feed-forward DNNs [15, 16]. Furthermore, recent studies have shown that the ability of DNN models to fit noisy data is dependent on the optimization methods used [17, 18].

The RMSProp algorithm [19] and the AdaGrad [20] are two optimization methods that are strongly attributable to

Adam. These connections will be demonstrated later. Other stochastic optimization methods discussed include vSGD [21], AdaDelta [22], and the natural newton method [23]. All of these optimizers use the first derivative (gradient) of the loss function to estimate the curvature of the loss surface and determine the optimal learning rate step sizes. As in the natural gradient descent (NGD) method, some variants of Adam use a preconditioner (like AdaGrad) that adjusts to the geometry of the data based on the approximation for the diagonal of the Fisher information matrix [24]. The adaptation mechanism in Adam's preconditioner is more conventional than vanilla NGD [25]. Other variants of Adam have also been proposed such as NosAdam [26], Sadam [27], and Adax [28].

Throughout this paper, we attempt to improve the learning rate update step during the training process for first-order optimization methods. Other approaches, such as Lookahead [29], update the weights slowly and quickly separately. It is regarded as a wrapper that can be combined with other optimizers.

When compared to the SGD algorithm, adaptive gradient methods like Adam typically converge quickly in the early training phases, but they still have poor generalization performance [31, 32]. Recent advances have attempted to combine the advantages of adaptive methods and the SGD method, such as switching from Adam to SGD with a hard schedule, as in SWATS [33], or with a smooth transition as in AdaBound [34]. Other Adam modifications are also proposed. The AMSGrad algorithm [35] solves Adam's convergence analysis problem. Dokkyun et al. [36] solve the problem of trapping into a local minimum for non-convex cost functions. The Adam algorithm's parameter update formula has been modified to include the cost function. The evolved gradient direction optimizer is a novel gradient-based algorithm introduced by the authors of [37]. (EVGO). It solves the vanishing gradient problem by updating the weights of the DNNs using the first-order gradient and a proposed hyperplane. The authors of [38] create YOGI, an adaptive optimization approach. It takes into account the training dataset's mini-batch size. The MSVAG algorithm [39] segregates Adam assign update and magnitude scaling, the RAdam algorithm [40] corrects learning rate variance, the Fromage algorithm [41] controls function space distance, and the AdamW algorithm [42] decouples weight decay from gradient descent.

Although these modifications outperform Adam in terms of accuracy, they perform worse in terms of generalization on large-scale datasets like ImageNet [43]. Furthermore, many optimizers are empirically unstable when training generative adversarial networks (GAN) compared to Adam [44].

Aside from the first-order methods, there are second-order methods that use the objective function's second-order derivative values (also known as the Hessian matrix) to minimize it. They provide us with additional information about the objective function's curvature surface, which aids in estimating a better step size for the learning rate. Newton's method, quasi-newton method, gauss–newton method [45, 46], and conjugate- gradient [47] are some common examples of second-order optimization methods. To train deep auto-encoders without using pre-training, Hessian-free optimization (HFO) [48] is used. The sum-of-functions optimizer (SFO) [49] is a quasi-newton method that employs mini-batches, which are small subsets of the dataset. Its performance is determined by the number of mini-batches generated from the dataset. This method is frequently impractical on memory-constrained systems such as GPUs.

Second-order optimization methods are not widely used because they require more computations to obtain second-order derivatives [49]. Table 1 summarizes the survey results for some selected optimization methods for improving the performance of deep network networks that have recently appeared in the literature.

Recently, other new versions of Adam have been arisen. A new version of Adam based on combining adaptive coefficients and composite gradients using randomized block coordinate descent is proposed in [50]. It enhances the performance of the Adam algorithm to a certain extent in terms of accuracy and convergence speed. The effect of the second-order momentum and the use of different learning rates was not considered on the performance of the original algorithm. In [51], an Adam-style algorithm, denoted by Amos, is introduced. It uses adaptive learning-rate decay and weight decay to improve the performance of the original algorithm. It utilizes model-specific information to establish the initial learning rate and decaying schedules. In [52], a faster version of Adam algorithm named Adan is suggested to accelerate the training process of deep neural networks effectively. It develops a new Nesterov momentum estimation method to estimate the first- and second-order moments of the gradient in adaptive gradient algorithms like Adam. This method increases the convergence speed of the Adam algorithm.

## 3 Deep neural networks (DNNs) architectures

Deep neural network (DNN) is a type of artificial neural network (ANN) with multiple hidden layers between the input and output layers [1]. DNN structures vary, but they all share the same basic building blocks, such as neurons, synapses, weights, biases, and activation functions [1]. They can be trained to perform functions similar to human

Table 1 Performances results for some selected optimization algorithms

| DNN Models | Datasets | Algorithm | Performance |
|---|---|---|---|
| CNN (2015) [8] | MNIST | Adam, AdaMax | Loss = 0.26 |
| WRN-22, WRN-28 (2018) [13] | CIFAR-10- CIFAR-100 | ND-Adam | Loss = (3.70–19.30) (3.70–18.42) |
| Deep4Net ResNet (2019) [42] | CIFAR-10 | (AdamW)- (SGDW) | Accuracy = (73.68%) (72.04%) |
| ResNet18, PreActResNet18 (2019) [35] | CIFAR-10 | AMSGrad and AdamX | – – |
| CNN1, CNN2 (2019) [61] | MNIST | HuperAdam | Accuracy = 98.63% 99.78% After 1000 steps |
| (ResNet20, ResNet32) (2020) [62] | CIFAR-10 | SGD | Accuracy = (92.08–93.14%) |
| | | Adam | (90.33%–91.63%) |
| | | AdamW | (91.97%–92.72%) |
| | | AdaHessian | (92.13%–93.08%) |
| VGG11, ResNet18, DenseNet121 (2020) [60] | CIFAR-10 | EAdam | Accuracy = (91.45%–94.99%- 95.61%) |
| | CIFAR-100 | Adam | (88.95%–92.88%93.55%) |
| | | RAdam | (89.54%–94.26%- 94.97%) |
| | | Adabelief | (91.66%–94.85%–95.69%) all accuracy for 150 epochs |
| AlexNet- ResNet20 (2020) [37] | MNIST | EVGO | For MNIST(Val = 98.06%- Test = 98.12%) |
| | CIFAR-10 | | For CIFAR(Loss = 0.05240.4616 |
| | | | Accuracy = 80.92%–87.52%) |
| BPNN (2023) [50] | MNIST | ACGB-Adam | For MNIST (Loss(MSE) = 0.253- Accuracy = 95.9%) |
| | CIFAR-10 | | For CIFAR(Loss(MSE) = 2.287- Accuracy = 94.1%) |
| ResNet50 (2023) [51] | ImageNet | Amos | (Loss = 0.261) |
| ResNet, ConvNext (2022) [52] | ImageNet | ADAN | (Top-1 accuracy on ViT Small = 80.9, on ViT Base = 82.3, on Swin Tiny = 81.6, on Swin small = 83.7, Swin Base = 83.8) |

brains using supervised or unsupervised learning algorithms [53].

## 3.1 Activation functions

Because of matrix operations in artificial neural networks, the network and its components are linear. The established linear structure is transformed into a nonlinear one using the activation functions. Choosing appropriate activation functions makes it simple to increase the network's computation speed. The common activation functions that are used in the deep neural networks are sigmoid, tangent hyperbolic (Tanh), rectified linear unit (ReLU) and leaky rectified linear unit (Leaky ReLU) [53, 54].

## 3.2 Training of deep neural networks

Deep neural network training (or learning) is the process of determining the weight of neuron connections to achieve the required relationships between inputs and outputs with a certain precision. There are two types of learning methods used to train neural networks [53]: supervised learning and unsupervised learning. In most machine learning practical applications, where the network model has a training dataset of inputs and outputs, supervised learning is used. This type of learning is used to provide an approximation of a mapping function to represent the relationship between inputs and outputs.

Classification and regression are two common problems addressed by supervised learning. Unsupervised learning is used when the network model only has input data and no corresponding outputs, such as in clustering problems. The goal of this type of learning is to learn more about the data by modeling the underlying structure or distribution [13, 54].

The basic learning algorithm used to train DNNs for supervised learning is back-propagation, which has two operating phases, forward propagation and backward propagation [53, 60]. It is based on the gradient descent algorithm (GD), which calculates gradients across the entire dataset. This results in a large number of iterations and increases the risk of becoming trapped in local optimums with early convergence. Due to these issues, the mini-batch gradient descent method was proposed. The training dataset is divided into fixed-size batches for use during the training process in this method. The total error is computed, and the weights for each sub batch are updated. When the mini-batch value is set to "1," the stochastic gradient descent algorithm is used. In this case, the error is calculated for one sample at a time, and the weights are also updated, resulting in faster convergence through direct data vectoring. The error value is then propagated back through the networks, and the weights are updated using GD in the opposite direction of the curvature. The network parameters ($\theta$) to be optimized are updated according to the following formula [10].

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t) \tag{1}$$

where $\eta$ is the learning rate, $\nabla_\theta J(\theta_t)$ is the gradient of the loss function $J(\theta_t)$ with respect to $\theta_t$.

The updated weight values are affected by the value of the learning rate. It converges for convex surface areas and non-convex surfaces on a global minimum. The batch-gradient descent is also known as the vanilla gradient descent. It works with extremely large training datasets where it performs intensive calculations that take up a lot of memory space, making it difficult to use. Furthermore, it provides numerous redundant updates that we do not require. As a result, several methods based on stochastic gradient descent have been developed for use in practical applications. Because the network only processes one training sample at a time, stochastic gradient descent is easy to fit in memory and fast in computations. This suits the large datasets as it updates the parameters more frequently and converges faster. Some of the improved algorithms that are based on the stochastic gradient descent are illustrated in the following section.

### 1) Stochastic-Gradient Descent (SGD)

The stochastic gradient descent (SGD) algorithm calculates the lost function for a single training sample at a time rather than considering all training data samples. Memory deficiency problems can be avoided in this manner. SGD was created to address the shortcomings of the batch-gradient descent algorithm. The problem with using SGD is determining the proper learning rate value to avoid oscillations and reach the global optimal. For parameter updating, it employs the following equation [6, 10].

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t; x^{(i)}; y^{(i)}) \tag{2}$$

where $\eta$ is the learning rate, $\nabla_\theta J(\theta_t)$ is the gradient of the loss function $J(\theta_t)$ with respect to $\theta_t$. Also $x^{(i)}$ and $y^{(i)}$ represent the training data in the form of inputs-outputs pairs. If the loss function curve has saddle points where one dimension slopes up and the other dimension slopes down, the SGD algorithm does not perform well [6].

### 2) Gradient descent with momentum

The learning steps in gradient descent methods is desired to move faster towards the best result. When the learning steps are very large, the global optimal cannot always be reached. These large steps can have a direct impact on the time required to achieve global optimal. To address these issues, the momentum gradient descent method has been proposed. It limits the speed of the next learning step by using the average speed of previous learning steps. In this method, the dynamic average of the past gradients $(m_{t-1})$ is exponentially decreased and it is kept. Its direction is determined by taking these dynamic averages into account. In this way, learning steps move faster towards the best result with less deviation [6, 10]. We can express the updating rules as [10]:

$$m_t = \gamma m_{t-1} + \eta \cdot \nabla_\theta J(\theta_t) \tag{3}$$

$$\theta_{t+1} = \theta_t - \mathrm{m}_t \tag{4}$$

where $\gamma$ is the momentum parameter, it is usually set to 0.9 or a similar value.

### 3) Nesterov Accelerated Gradient (NAG)

Nesterov accelerated gradient (NAG) is a method to give our momentum term this kind of prediction. The NAG algorithm determines the first step in the direction of the average gradient for the current position before measuring the new position. The momentum term $\gamma m_{t-1}$ will be used to move the parameters $\theta_t$. Computing the term $\theta_t - \gamma m_{t-1}$ will give an approximation to the next position of the parameters and this is considered a rough idea to know where our parameters are going to be [6, 10]. The parameters are updated based on the following two equations [10].

$$m_t = \gamma m_{t-1} + \eta \cdot \nabla_\theta J(\theta_t - \gamma m_{t-1}) \tag{5}$$

$$\theta_{t+1} = \theta_t - \mathrm{m}_t \tag{6}$$

### 4) Adaptive Gradient Algorithm (AdaGrad)

The adaptive gradient (AdaGrad) algorithm divides the learning rate component by the square root of $v_t$, which is the sum of the current and past squared gradients up to time instant t. The gradient component, like in SGD, remains unchanged. AdaGrad makes different updates for each parameter by using different learning rates for each step. The most significant advantage of using AdaGrad is that the learning rate is not manually adjusted, as in other adaptive learning systems. The update equations of the AdaGrad can be expressed as [10]:

$$g_t = \nabla_{\theta_t} J(\theta_t) \tag{7}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \in}} g_t \tag{8}$$

where $\in$ is a smoothing term that avoids division by zero (usually set as $10^{-8}$), and $v_t$ is the exponential moving average of the gradient.

E) Root Mean Square Propagation (RMSProp)

RMSProp is an optimization method that is closely related to Adam [19]. A version with momentum has sometimes been used. There are a few key differences between RMSProp with momentum and Adam. RMSProp with momentum updates the parameters by using a momentum on the rescaled gradient, whereas Adam updates the parameters directly by using the moving average of the gradient's first and second moments $v_t$. The update rules of the RMSProp algorithm are [10, 19]:

$$g_t = \nabla_{\theta_t} J(\theta_t) \tag{9}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{10}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \in}} g_t \tag{11}$$

where $g_t$ is first derivative for loss function (the gradient) and $\beta_2$ is the exponential decay rate.

6) Adaptive Delta (Adadelta):

The parameters in the Adadelta optimization method have their own learning speeds that gradually decrease until it is no longer possible to continue the learning process. To address this issue, the RMSProp method was developed [6, 10, 22]. The following equations [10] are used to update the parameters.

$$RMS[\Delta_{\theta_t}] = \sqrt{E\left[\Delta_{\theta_t}^2\right] + \in} \tag{12}$$

$$\Delta_{\theta t} = -\frac{\mathrm{RMS}[\Delta_{\theta_{t-1}}]}{\mathrm{RMS}[g_t]} g_t \tag{13}$$

$$\theta_{t+1} = \theta_t + \Delta_{\theta_t} \tag{14}$$

where $RMS[\Delta_{\theta_t}]$ and $E\left[{\Delta_{\theta_t}}^2\right]$ are the root mean squared error and the expected moving average of the parameter updates $\Delta_{\theta_t}$ at time step $t$, respectively.

7) Adaptive Moment Estimation (Adam)

Adam is a highly efficient adaptive optimization algorithm that is frequently used as a replacement for the traditional stochastic gradient reduction method. It dynamically updates the learning rate for each parameter and is thought to be computationally efficient with low memory requirements [8]. Adam employs a parameter update method akin to gradient descent with RMSProp and momentum. It uses the exponential moving average of the squared gradient ($\mathrm{v_t}$), as in RMSProp, in addition to the exponential moving average of the gradient ($\mathrm{m_t}$). So, it combines between the benefits of the RMS-Prop and the momentum [6, 10]. For parameter updates, the following equations are used [6, 10]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{16}$$

$$m_t^\wedge = \left(\frac{m_t}{1 - \beta_1^t}\right) v_t^\wedge = \left(\frac{v_t}{1 - \beta_2^t}\right) \tag{17}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t^\wedge + \in}} m_t^\wedge \tag{18}$$

where $\beta_1$ is the exponential decayrate. The default values for $\beta_1$ and $\beta_2$ are 0.9 and 0.999, respectively. $m_t^\wedge$and $v_t^\wedge$ are correction biases for $m_t$ and $v_t$ respectively.

8) Nesterov-accelerated adaptive momentum estimation (Nadam)

Nadam consists of a combination of the three algorithms Nadam, Adam and NAG. In order to include the NAG algorithm in the Adam algorithm, the momentum expression is modified using the following update rule [6, 10].

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t^\wedge + \in}} \left(\beta_1 m_t^\wedge + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t}\right) \tag{19}$$

## 4 Convolutional neural networks architectures

The Convolutional Neural Network (CNN) is a type of DNN that is made up of multi-layer perceptrons. It is the most commonly used model for DNNs. The key advantage of CNN is that it can automatically identify relevant features without the need for human intervention [1]. It is frequently used in classification problems involving data with a topological input, such as time-series data in one dimension and image data in two dimensions [3, 5]. A CNN model is used as a classifier in the current study, with two different benchmark datasets. Figure 2 depicts the general architecture of a CNN model [55]. It is composed of a number of layers known as multi-building blocks [53]. Each layer in the CNN architecture, including its function, will be explained in detail below.

(1) Convolutional layer

A CNN is a neural network that has at least one convolutional layer. The most important component of the CNN model is the convolutional layer. It is made up of a set of convolutional filters known as kernels. The input image is convolved with these filters to map an output feature, which is expressed as N-dimensional metrics. The Kernel is a grid of discrete values representing the kernel weights. The convolutional operation is carried out in the following order. The CNN input format is first described. The vector format is the traditional neural network's input, whereas the multi-channeled image is the CNN's input. For instance, the single-channeled is the format of the gray-scale image, while the RGB image format is three-channeled. In the CNN model, a convolutional layer often incorporates with the ReLU activation function to be as one layer and then it followed by a pooling layer [56].

(B) Pooling layer

The pooling layer's primary purpose is to subsample the feature maps. These maps are created by using convolutional operations. The pooling layer is available in several variations, but its general purpose is to replace the output of the convolutional layer with a summary statistic of the neighboring outputs. There are several types of pooling methods that can be used in different pooling layers. Tree pooling, gated pooling, average pooling, min pooling, max pooling, global average pooling (GAP), and global max pooling are examples of these methods. The most common and widely used pooling methods are the max, min, and GAP pooling.

(C) Fully connected layer

The pooling layer's primary function is to subsample the feature. This layer is typically found at the end of the CNN architecture. Each neuron in this layer is connected to all neurons in the previous layer, which is known as the fully connected (FC) approach. It serves as the CNN classifier. It adheres to the fundamental layers of the conventional multiple-layer perceptron.

Because the CNN is a feed-forward ANN, the input to the FC layer comes from the previous pooling or convolutional layer. This input takes the form of a vector, which is generated after flattening the feature maps. As shown in Fig. 1, the FC layer output represents the final CNN output.

There are many reasons to use a CNN instead of a standard multi-layer perceptron network for classifying images [55]. The main reason is the weight sharing feature, which reduces the number of trainable network parameters and enhances the generalization performance and prevents overfitting of the network model. Moreover, concurrently learning of the feature extraction layers and the classification layer causes the model output to be both highly organized and highly reliant on the extracted features.

## 5 The proposed modification of Adam algorithm

There are various reasons to use a CNN rather than a standard multi-layer perceptron network for image classification [55]. The main reason is the weight sharing feature, which reduces the number of trainable network parameters while improving generalization performance and preventing network overfitting. Furthermore, learning the feature extraction and classification layers concurrently results in a model output that is both highly organized and heavily reliant on the extracted features.

In this section, we will present our proposed modified algorithm, which is based on the standard Adam optimizer. Adam is one of the best optimization algorithms for training DNNs, and it is gaining popularity [53]. As a result of some issues that arose when it was used in some applications, such as the generalization performance problem and the convergence problem, several trials were conducted to improve its performance, as in the case of the SGD optimizer with momentum. Algorithm 1 describes the pseudo-code for the basic Adam algorithm.

Throughout this paper, we attempt to tackle the convergence problem associated with the standard Adam

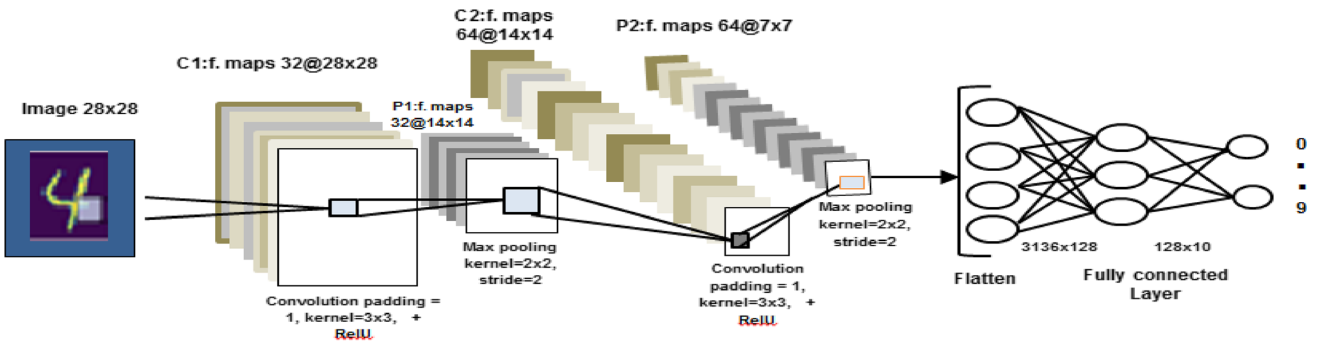


Image 28x28
C1:f. maps 32@28x28
P1:f. maps 32@14x14
C2:f. maps 64@14x14
P2:f. maps 64@7x7
Convolution padding = 1, kernel=3x3, + RelU
Max pooling kernel=2x2, stride=2
Convolution padding = 1, kernel=3x3, + RelU
Max pooling kernel=2x2, stride=2
Flatten
3136x128
128x10
Fully connected Layer
0
9

Fig. 1 General architecture of a CNN model

algorithm in order to achieve a high convergence speed. The proposed modified algorithm, denoted by HN Adam, is based on the adaptive norm technique and the hybrid technique between the original Adam algorithm and the AMSGrad algorithm, with the letters “H” and “N” referring to the hybrid mechanism and the adaptive norm, respectively. To improve the generalization performance of the basic Adam algorithm, we use a hybrid mechanism with some modifications between the Adam algorithm and the AMSGrad algorithm.

The main challenge that our proposed algorithm attempts to overcome is having good generalization performance like the SGD while also achieving quick convergence like the adaptive methods. The basic idea behind our modification is to automatically adjust the learning rate step size based on the adaptive norm for each current and past gradient, where the norm function for any two points is considered the Euclidean distance between them. The adaptive norm means that the norm value is changed dynamically based on the gradient values obtained in each epoch. Furthermore, a hybrid mechanism between the original Adam algorithm and the AMSGrad algorithm has also been made to enhance the generalization performance and achieve a high speed of convergence for the most architecture of DNNs. We validate the proposed algorithm in extensive experiments of image classification through two different standard datasets.

At first, the modified algorithm, HN_Adam, trains the network model using the Adam algorithm but with the adaptive (or dynamic) norm function to increase the step size of the learning rate and avoid dropping in a local minimum.

**Algorithm 1:** The Basic Adam Optimizer

```
Require:
  1: Initialized parameter $\theta_0$ , step size η , batch size N_B
  2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0, v_0 = 0$
  3: For all $t = 1, \ldots\ldots, T$ do
  4:     Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
  5:     $g_t \leftarrow \sum_{k=1}^{N} \nabla \{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
  6:     $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
  7:     $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
  8:     $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$ , $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
  9:     $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
  10:    end for
  11:    return final parameter $\theta_T$
```

Once it closes to the global minimum, the hybrid technique is invoked where it switches to use the AMSGrad algorithm also with the adaptive norm function. It can thus achieve an accurate optimization at an acceptable switch point based on the value of the threshold ($\Lambda_{t0}$), the absolute value of gradients ($|g_t|$) and the exponential moving averages of the past gradient $m_{t-1}$.

The pseudo-code of the modified algorithm, HN_Adam, is described in algorithm 2. The modifications that are made compared to the original Adam algorithm are in bold. As the HN_Adam algorithm uses a dynamic norm value, the absolute value of the gradient must be taken before the power is calculated. This is done to ensure that only positive values will be added if it uses the possibly odd values for the norm.

The threshold value of the norm ($\Lambda_{t0}$) is randomly chosen in the range from 2 to 4 and then the norm value $\Lambda(t)$ is adaptively computed depending on the value of the absolute gradient $|\boldsymbol{g_t}|$ and the exponential moving average of the past gradient $m_{t-1}$, as described in the following equation.

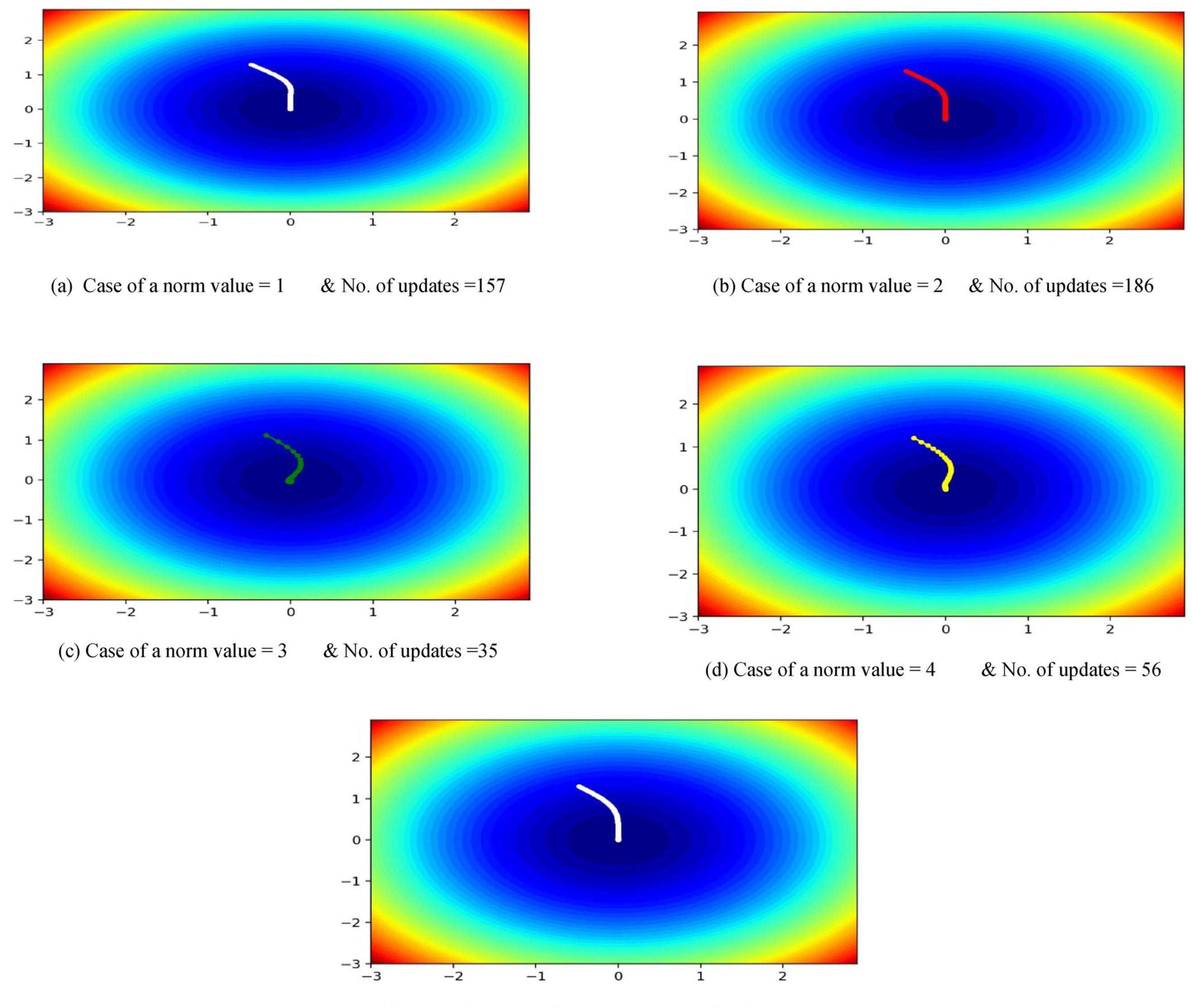
$$\Lambda(t) \leftarrow \Lambda_{t0} - \frac{m_{t-1}}{m_{max}} \tag{20}$$

where $m_{max}$ is the maximum value between $|\boldsymbol{g_t}|$ and $m_{t-1}$.

**Algorithm 2:** Hybrid and Adaptive Norming of Adam with AMSGrad (HN_Adam): The Modified Algorithm

```
Require:
  1: Initialized parameter $\theta_0$ , step size η , batch size N_B,
  2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$ , $v_0 = 0$ , amsgrad = False, $v_{hat(0)} = 0$
  3: For all $t = 1, \ldots\ldots, T$ do
  4:     Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
  5:     $g_t \leftarrow \sum_{k=1}^{N} \nabla \{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
  6:     $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
  7:     $\boldsymbol{m_{max} \leftarrow Max(m_{t-1}, |g_t|)}$
  8:     $\boldsymbol{\Lambda(t) \leftarrow \Lambda_{t0} - \frac{m_{t-1}}{m_{max}}}$
  9:     $\boldsymbol{v_t \leftarrow \beta_2 . v_{t-1} + (1-\beta_2).(|g_t|)^{\Lambda(t)}}$
  10:    If $\boldsymbol{\Lambda(t) < 2}$: // Switching Between Adam and AMSgrad
  11:        amsgrad = True
  12:        $v_{hat(t)} \leftarrow Max(v_{hat(t-1)}, | v_t)| )$
  13:        $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{m_t}{((v_{hat(t)}^{1/\Lambda(t)}) + \varepsilon)}$
  14:    else :
  15:        amsgrad = False
  16:        $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{m_t}{((v_t^{1/\Lambda(t)}) + \varepsilon)}$
  17:    end for
  18:    return final parameter $\theta_T$
```

It is observed that in the case of higher power values of the norm, longer steps are required to converge and reach to the global minimum of the cost function. In the case of lower power values of the norm, short steps are required to converge. This indicates that using a higher norm value leads to more exploration and less exploitation, and vice versa. It is known that any optimization problem needs different rates of exploration and exploitation during the search process. Thus, instead of using a fixed norm value (equal to 2 as in the original Adam algorithm), it is dynamically changed in the proposed HN_Adam algorithm to make a balance between the exploration and exploitation. The dynamic value of the norm function can achieve


(a) Case of a norm value = 1 & No. of updates =157

(b) Case of a norm value = 2 & No. of updates =186

(c) Case of a norm value = 3 & No. of updates =35

(d) Case of a norm value = 4 & No. of updates = 56

(e) Case of a norm value = 5 & No. of updates = 241

**Fig. 2** Example of plotting the Adam search on a contour plot with different norm values for the loss function $f(x) = x^{\wedge 2} + y^{\wedge 2}$

better results in terms of accuracy and convergence speed. HN_Adam is designed to adjust the adaptation of the norm value for the standard Adam algorithm by changing its power value at every update. This is done based on the information of the previous gradient updates.

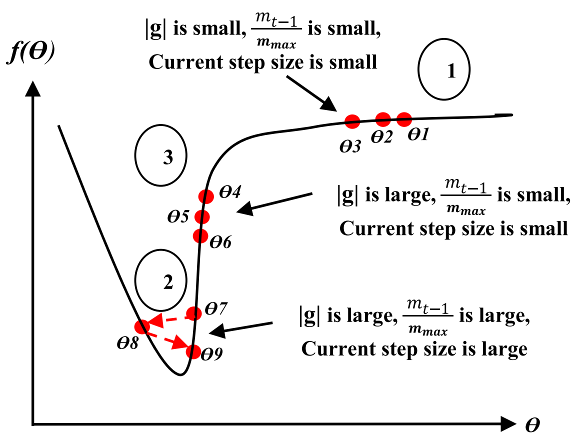
It is highly recommended to keep the norm value in the range between 1 and 4 since smaller values lead generally to bad results and higher values lead hardly to improvements with more expensive computations being exerted, see Fig. 2.

In Eq. (20), the ratio $\frac{m_{t-1}}{m_{max}}$ is less than or equal to 1, this implies that the norm value will be in the range from 1 to 4. So, the sequence is switched to the AMSGrad algorithm under the condition that $\Lambda(t) < 2$. This means that HN_Adam uses the modified Adam algorithm with more exploration ability of search as long as the norm value is within the range from 2 to 4. Otherwise, it uses the AMSGrad algorithm with more exploitation ability.

Figure 2 illustrates the effect of increasing the norm value from 1 to 5 for the standard Adam algorithm using the loss function $f(x) = x^{\wedge 2} + y^{\wedge 2}$. It shows that increasing the norm values leads to a decrease in the number of updates, the number of epochs and the learning period.

## 5.1 Comparison to Adam

In this part, the comparison between the standard Adam algorithm and the modified algorithm, HN_Adam, will be highlighted as well as the differences between them will be explained. Also, the enhancement of the HN_Adam algorithm in terms of accuracy and convergence speed will be



|g| is small, $\frac{m_{t-1}}{m_{max}}$ is small, Current step size is small
f(Θ)
1
3
θ3 θ2 θ1
θ4
θ5
θ6
|g| is large, $\frac{m_{t-1}}{m_{max}}$ is small, Current step size is small
2
θ7
θ8
θ9
|g| is large, $\frac{m_{t-1}}{m_{max}}$ is large, Current step size is large
θ

**Fig. 3** Curvature of the loss function for an ideal optimizer [57]

discussed. As shown in Algorithm (1) and Algorithm (2), the update direction of the original Adam algorithm is $\frac{m_t^\wedge}{\sqrt{v_t^\wedge}}$, where $m_t^\wedge$ is bias corrected for the exponential moving average (EMA) of the gradient ($g_t$) and $v_t^\wedge$ is bias corrected for the exponential moving average (EMA) of the squared gradient ($g_t^2$). The update direction of the HN_Adam algorithm is $\frac{m_t}{v_t^{1/\Lambda}}$, where $v_t^{1/\Lambda}$ is the EMA of $g_t^\Lambda$ and $\Lambda$ is the adaptive norm value.

We can observe that HN_Adam takes a small step size of the learning rate when the absolute gradient $|\boldsymbol{g_t}|$ is close to $m_{max}$, like Adam, and a large step size when the gradient significantly deviates from $m_{max}$.

Now, we will demonstrate that HN_Adam can use the curvature information of the loss functions to choose a proper step size of the learning rate in order to enhance the training process. For explanation, Let us consider the loss function shown in Fig. 3 [57]. We use three regions on the graph to explain the behavior of the HN_Adam algorithm that concerns with the amount of parameter updates while searching the loss function to find the global minimum. These regions are used the same as in [57]. The learning rate can be expressed in terms of the step size which is responsible for the amount of change in parameter updates. So, we will clarify that the HN_Adam algorithm can choose an appropriate value of the step size and matches the ideal behavior to make suitable amount of changes for parameters updating.

Figure 3 shows how an ideal optimizer considers the curvature information to determine the proper step size for the three tested regions. We use it as a reference in evaluating the HN_Adam algorithm. Furthermore, we make a comparison between the HN_Adam algorithm and two other algorithms SGD and Adam. The step size formulas for SGD, Adam, and HN_Adam can be written as:

$$\Delta\theta_t^{SGD} = -\eta.m_t \tag{21}$$

$$\Delta\theta_t^{\mathrm{Adam}} - \eta.\frac{m_t^\Delta}{\sqrt{v_t^\Delta}} \tag{22}$$

$$\Delta\theta_t^{HN\_Adam} = -\eta.\frac{m_t}{v_t^{1/\Lambda}} \tag{23}$$

where $|\Delta\theta_t|$ is the step size for the parameter update at the instant $t$.

The first, second, and third regions are denoted as 1, 2, and 3, respectively, in Fig. 4. Now, for these three regions, we will compare the step sizes of HN Adam, SGD, and Adam to the ideal optimizer's step size. The gradient is close to 0 in the first region because the loss function is flat. To increase its learning rate, the ideal optimizer should take large steps. The SGD algorithm, unlike the ideal optimizer, will take small steps because it is proportional to the EMA of the gradient $m_t$. While both the Adam algorithm and the HN_Adam will make large step sizes like the ideal optimizer because $v_t^\wedge$ is a small value and the norm value $\Lambda$ is a large value.

In the second region, both $|\boldsymbol{g_t}|$ and $m_t$ are large since the loss function in this region oscillates in a steep and narrow valley. To reach the global optimum, the ideal optimizer should decrease its learning rate and make small steps. The SGD algorithm, unlike the ideal optimizer, will take large steps because its learning rate is proportional to $m_t$. Finally, the ideal optimizer should increase its learning rate and use a large step size in the third region where the loss function has a large $v_t^\wedge$ value and the norm value $\Lambda$ is a small value. Finally, in the third region where the loss function has a large $|g_t|$ with a small curvature, the ideal optimizer should increase its learning rate and apply a large step size. Unlike the ideal optimizer, the Adam algorithm will make a small step size because the denominator $\sqrt{v_t^\wedge}$ in its update formula is large. Despite that $|g_t|$ and $v_t$ are large, the norm value $\Lambda(t)$ is also large this could happen because the ratio of the exponentially moving average of past gradients and the current absolute gradient is small and the HN_Adam will use a large step size as in the ideal optimizer. The SGD algorithm will also take a large step size.

We summarize these three cases in Table 2, where S and L refer to small and large values, respectively. $|\Delta\theta_t|^{ideal}$ is the step size for the parameter update of the ideal optimizer. The HN_Adam algorithm matches the behavior of the ideal optimizer over the three tested regions.

## 5.2 Mathematical illustration of the learning rate step size for HN_Adam

The HN Adam algorithm's updated term differs slightly from the standard Adam algorithm. It is based on a

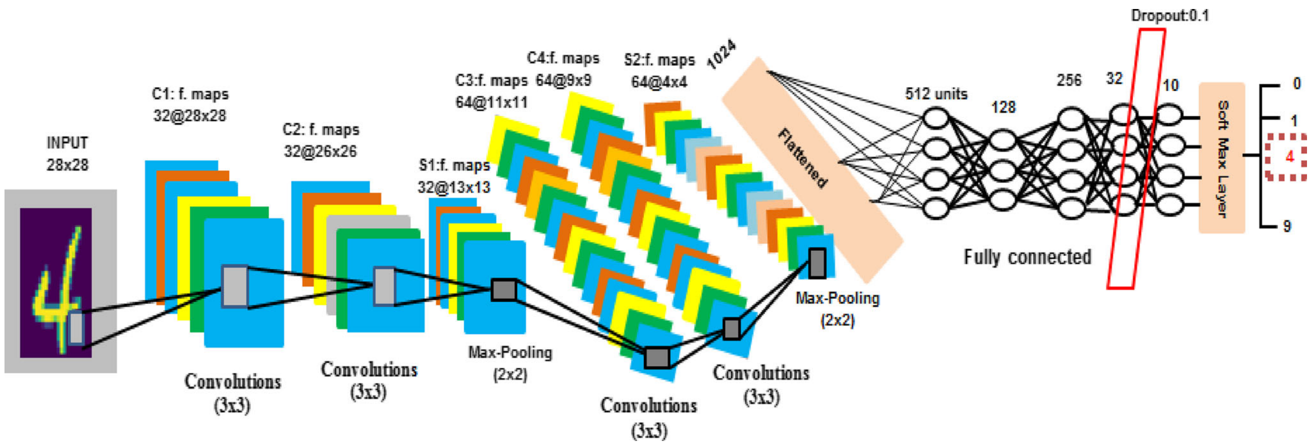


**Fig. 4** Architecture of the deep CNN model using the MNIST dataset

**Table 2** Comparison of optimizers' behavior for determining the step size of parameter updates over three different regions of a loss function based on the curvature information

| Case No | $\lvert g_t \rvert, v_t$ | $\frac{m_{t-1}}{m_{max}}$ | $\lvert\Delta\theta_t\rvert^{ideal}$ |
|---|---|---|---|
| Case 1 | S | S | L |
| | $\lvert\Delta\theta_t\rvert^{SGD}$ | $\lvert\Delta\theta_t\rvert^{Adam}$ | $\lvert\Delta\theta_t\rvert^{HN\_Adam}$ |
| | S | L | L |
| Case 2 | L | L | S |
| | $\lvert\Delta\theta_t\rvert^{SGD}$ | $\lvert\Delta\theta_t\rvert^{Adam}$ | $\lvert\Delta\theta_t\rvert^{HN\_Adam}$ |
| | L | S | S |
| Case 3 | L | S | L |
| | $\lvert\Delta\theta_t\rvert^{SGD}$ | $\lvert\Delta\theta_t\rvert^{Adam}$ | $\lvert\Delta\theta_t\rvert^{HN\_Adam}$ |
| | L | S | L |

dynamic norm value that changes with the gradient values during epochs. In Adam, it is left constant value of 2 during the learning process for ease of use [33, 40]. Changing the norm value influences the size of the learning step.

Now we will investigate how the Adam algorithm's fixed norm differs from the HN_Adam algorithm's dynamic norm. For both Adam and, the general equation for the step size of the parameter updates at the moment $t$, for both Adam and HN_Adam, can be reformulated as illustrated in the following steps.

The formulas of the exponential moving average terms,$m_t$ and $v_t$, are [8, 35]:

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t \tag{24}$$

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . g_t^2 \tag{25}$$

If we assume that $m_{t=0} = 0$ and$v_{t=0} = 0$, we can rewrite these formulas of the moving averages as:

$$m_t = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} . g_i \tag{26}$$

$$v_t = (1 - \beta_2) \sum_{i=0}^{t} \beta_2^{t-i} . g_i^2 \tag{27}$$

Using Eqs. 26 and 27, the correction bias terms for $m_t$ and $v_t$ will be,

$$m_t^{\wedge} = \frac{m_t}{1 - \beta_1^t} = \sum_{i=0}^{t} \beta_1^{t-i} . g_i \tag{28}$$

$$v_t^{\wedge} = \frac{v_t}{1 - \beta_2^t} = \sum_{i=0}^{t} \beta_2^{t-i} . g_i^2 \tag{29}$$

From Eqs. 22, 23, 28 and 29, we can rewrite the general equation of the step size for the parameter updates for both Adam and HN_Adam as:

$$\Delta\theta_t = -\eta . \frac{\sum_{i=0}^{t} g_i(\theta_i) . \beta_1^{t-i}}{\sqrt[\Lambda]{\sum_{i=0}^{t} \lvert g_i(\theta_i) \rvert^{\Lambda} . \beta_2^{t-i}} + \varepsilon} \tag{30}$$

where the norm value is fixed, $\Lambda = 2$, for the parameter updates of the Adam algorithm.

To illustrate the difference of using Eq. 30 by the Adam algorithm from the HN_Adam algorithm, the equality relation between them for the step size of the parameter updates will be examined as follows.

$$-\eta . \frac{\sum_{i=0}^{t} g_i(\theta_i) . \beta_{1,x}^{t-i}}{\sqrt[\Lambda]{\sum_{i=0}^{t} \lvert g_i(\theta_i) \rvert^{\Lambda} . \beta_{2,x}^{t-i}} + \varepsilon} \overset{?}{=} - \eta . \frac{\sum_{i=0}^{t} g_i(\theta_i) . \beta_{1,y}^{t-i}}{\sqrt[2]{\sum_{i=0}^{t} g_i(\theta_i)^2 . \beta_{2,y}^{t-i}} + \varepsilon} \tag{31}$$

where the left side of the examined equality represents the step size of the HN_Adam algorithm and the right side is the step size of the Adam algorithm. To distinguish between the parameters of the two algorithms, we use the subscript $x$ for HN_Adam's hyper-parameters and the subscript $y$ for Adam's hyper-parameters. The same symbols are used for the learning rate ($\eta$) and the smoothing term ($\varepsilon$). Taking all of the above into account, Eq. 31 can be simplified to,

$$\frac{\sum_{i=0}^{t} g_i(\theta_i).\beta_{1,x}^{t-i}}{\sqrt[\Lambda]{\sum_{i=0}^{t} |g_i(\theta_i)|^{\Lambda}.\beta_{2,x}^{t-i}} + \varepsilon} \overset{?}{=} \frac{\sum_{i=0}^{t} g_i(\theta_i).\beta_{1,y}^{t-i}}{\sqrt[2]{\sum_{i=0}^{t} g_i(\theta_i)^2.\beta_{2,y}^{t-i}} + \varepsilon} \tag{32}$$

If we assume that $\beta_{1,x}$=$\beta_{1,y}$, the condition that makes the above examined equality to be true is:

$$\sqrt[\Lambda]{\sum_{i=0}^{t} |g_i(\theta_i)|^{\Lambda}.\beta_{2,x}^{t-i}} = \sqrt[2]{\sum_{i=0}^{t} g_i(\theta_i)^2.\beta_{2,y}^{t-i}} \tag{33}$$

Just by looking, $\Lambda = 2$ makes the two sides of Eq. 33 to be the same if $\beta_{2,x}$=$\beta_{2,y}$. So we will choose a different norm value $\Lambda \neq 2$ with $t > 0$. For easy, let try to use $\Lambda = 1$ and $t = 1$, Eq. 31 becomes,

$$\sum_{i=0}^{1} |g_i(\theta_i)|.\beta_{2,x}^{t-i} = \sqrt[2]{\sum_{i=0}^{1} g_i(\theta_i)^2.\beta_{2,y}^{t-i}} \tag{34}$$

After expanding the summation, Eq. (34) becomes

$$|g_0(\theta_0)|.\beta_{2,x} + |g_1(\theta_1)| = \sqrt[2]{g_0(\theta_0)^2\beta_{2,y} + g_1(\theta_1)^2} \tag{35}$$

By squaring the both sides,

$$|g_0(\theta_0)|^2.\beta_{2,x}^2 + 2.|g_0(\theta_0)|.\beta_{2,x}.|g_1(\theta_1)| + |g_1(\theta_1)|^2 = g_0(\theta_0)^2\beta_{2,y} + g_1(\theta_1)^2 \tag{36}$$

By omitting $|g_1(\theta_1)|^2$ from both sides and dividing both sides by $g_0(\theta_0)^2$, we obtain

$$\beta_{2,x}^2 + \frac{2.\beta_{2,x}.|g_1(\theta_1)|}{|g_0(\theta_0)|} = \beta_{2,y} \tag{37}$$

This means that, in order for the Adam algorithm to behave like the HN Adam algorithm, its hyper-parameter $\beta_2$ needs to be modified to be dependent on current and past gradients, rather than just the HN Adam algorithm's hyper-parameter $\beta_2$. This ensures that the hyper-parameter of the modified algorithm, HN_Adam, is dependent on the obtained gradients in each epoch and makes use of the loss function's curvature information.

## 6 Experiments and results

The modified algorithm, HN_Adam, is tested by using it to train a deep convolutional neural network using two different datasets CIFAR-10 [30] and MNIST [13]. Each of these datasets contains ten classes. The experiments are carried out using the Python programming language as well as two open-source libraries called Tensorflow and Keras. All experiments and results are obtained using the same hardware device, a digital computer equipped with a CPU core i5-5300U (2.30 GHz) and 8.00 GB of RAM.

The HN Adam algorithm is compared to the basic Adam algorithm and the SGD algorithm, as well as five other SGD adaptive algorithms: AdaBeilf [30], Adam, RMSprop, AMSGrad, and Adagrad. We use the default parameter settings where $\beta_1$ = 0.9, $\beta_2$ = 0.99, $\varepsilon = 10^{-8}$, and $\eta$ = 0.001. For all compared algorithms, the training, validation, and testing datasets are batched with a size of 128. The experimental findings are divided into two sections, one for each dataset.a) The first experiment: training a deep CNN model using the MNIST dataset

The MNIST dataset [13] contains 60,000 handwritten digit images. It is divided into three sets: the first set of 40,000 images is the training, the second one of 10,000 images is the validation set and the third set of 10,000 images is the testing set. The digits have been centered in a fixed-size (28 × 28 pixel) image with values ranging from 0 to 255. All images are converted to float32 data type with size-normalized values in the range from 0 to 1.

2) Network architecture

The convolutional neural network is built in the first experiment, as shown in Fig. 4. It begins with two 3 × 3 convolutional layers of 32 kernels each, followed by a max-pooling layer with a 2 × 2 window. Following that, a ReLU activation function is used. Following that, two more convolutional layers with 64 kernels of size 3 × 3 are added, followed by a max-pooling layer with a 2 × 2 window. A ReLU activation function is also used. Following that, the max-pooling layer's 2-dimensional output vector is converted to a 1-dimensional vector with a size of 1024 × 1 using a flatten module from the tensorflow package. The converted vector is then passed through four hidden layers. These hidden layers have 512, 128, 256, and 32 nodes, respectively. The ReLU activation function is applied after each hidden layer. Then the dropout layer is included with a default probability value of 0.1. Finally, a hidden layer of 10 nodes is used and the Softmax activation function is applied to produce the output from the output layer.

3) Experimental setup

The MNIST dataset is used to train a deep CNN model with a total of 697,034 parameters. The model is trained using the optimization algorithms HN_Adam, AdaBelief, Adam, AMSGrad, SGD, RMSProp, and AdaGrad individually. These algorithms are used to train the CNN model as learning algorithms. The performance for each

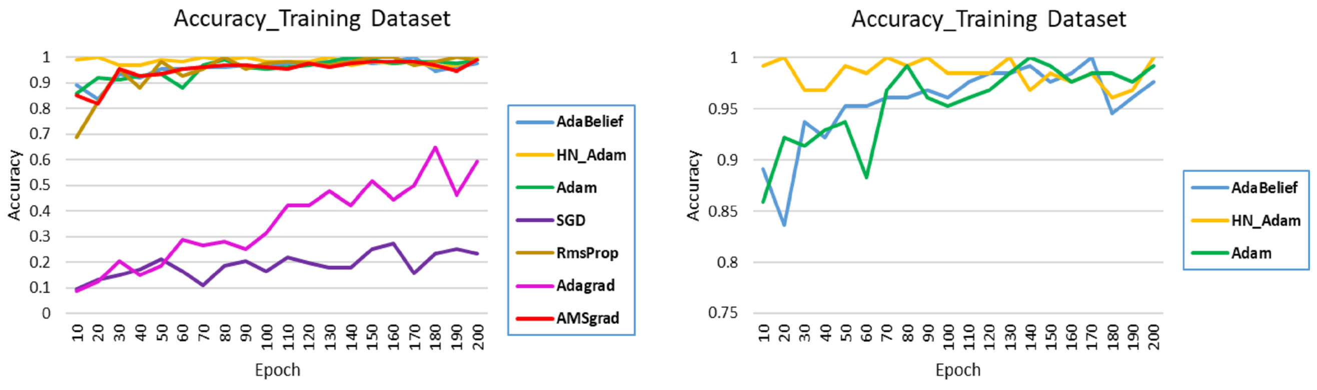


**Fig. 5** Training accuracy of the CNN model for the compared algorithms, case of using MNIST dataset

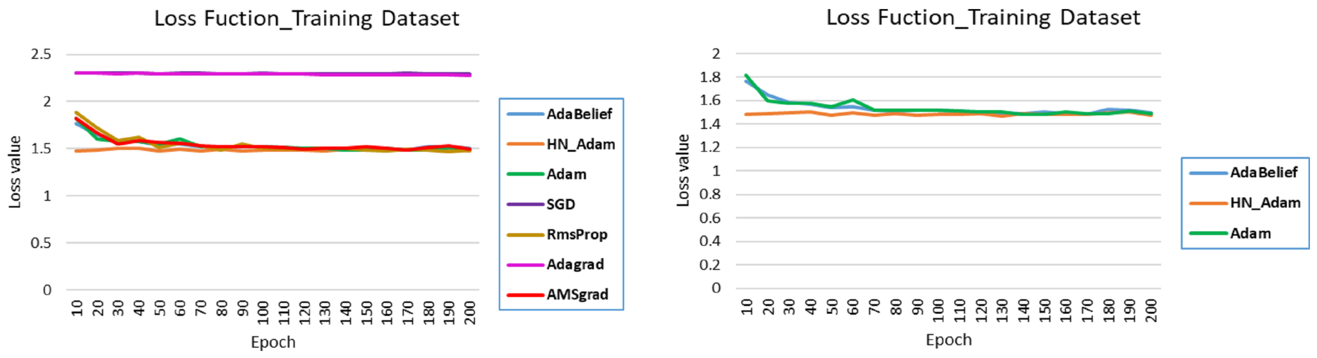


**Fig. 6** Loss function minimization during the training process for HN_Adam, Adam and AdaBelief, case of using MNIST dataset

**Table 3** Accuracy results, case of using MNIST dataset

| Algorithm | Min_Loss_Training Dataset | Test_Accuracy |
|---|---|---|
| HN_Adam | **1.471718** | **98.59%** |
| AdaBelief [30] | 1.48115 | 97.6% |
| Adam [8] | 1.483735 | 97.04% |
| AMSGrad [35] | 1.484545 | 97.09% |
| SGD [33] | 2.296593 | 96.84% |
| RMSprop [19] | 1.476105 | 97.09% |
| Adagrad [20] | 2.279421 | 96.97% |

Bold indicates the best achieved value for each response characteristic

**Table 4** The consumed training time for the compared algorithms, case of using MNIST dataset

| Algorithm | Training Time |
|---|---|
| HN_Adam | 1048 s |
| AdaBelief [29] | 1051 s |
| Adam [8] | 1067 s |
| AMSGrad [38] | 1051 s |
| SGD [36] | 1320 s |
| RMSprop [18] | 1075 s |
| Adagrad [19] | 1050 s |

compared algorithm is measured in terms of the minimum training loss function and the testing accuracy.

4) Results and discussions

The response curves of the compared algorithms during training process are indicated in Figs. 5 and 6. Figure 5 shows the accuracy curves for the compared algorithms during the training process for the CNN model. We focus on the basic Adam algorithm and the AdaBelief algorithms as they are the most competitive ones of the compared algorithms. Figure 6 shows the loss function minimization curves of the compared algorithms through the training process for the CNN model.

To demonstrate the differences between these response curves, the response characteristics in terms of the minimum loss function during training process, the accuracy of the testing on test dataset are calculated and listed in Table 3. For simplicity, the minimum training loss function and testing accuracy are determined after 200 epochs for 5 independent runs with randomly shuffled training data. The best achieved value for each response characteristic is highlighted in bold.

As shown in Figs. 5 and 6, HN_Adam could achieve fast convergence like the adaptive methods with better accuracy. The results illustrated in Table 3 confirm this, as it outperforms the other compared algorithms and achieves values of 1.471718, and 98.59%, for the minimum training loss function, and the testing accuracy, respectively. Table 4 also includes the training time in seconds consumed by the compared algorithms during the training process, demonstrating the increase in convergence speed. The learning algorithms use these values of the training time to train the CNN model and achieve the reported accuracy results in Table 3, where 10 epochs are considered for simplicity.

With a minimum training time of 1048 s, the HN_Adam algorithm clearly outperforms the other optimizers and achieves a high speed of convergence.b) The second experiment: training a deep CNN model using the CIFAR-10 dataset

Like the first experiment, the second one is conducted on another convolutional neural network, with slight differences in architecture from the previous model and using a different type of input data. The CIFAR-10 dataset [30] is used to train the CNN model. It consists of 60,000 color images fragmented into 10 classes, with 6000 images in each. The dataset is divided into three sets: the training set of 40,000 images, the validation set of 10,000 images and the testing set of 10,000 images. The images have been centered in a fixed-size image (32 × 32 pixels) with values ranging from 0 to 255. All image sizes are n1ormalized on a scale of 0 to 1.

(1) Network architecture

In this experiment, the CNN model is constructed as shown in Fig. 7. It starts with two convolutional layers of 32 kernels of size 3 × 3, followed by a max-pooling layer with a 2 × 2 window. Following that, a ReLU activation function is used. After that, two more convolutional layers with 64 kernels of size 3 × 3 are added, followed by a max-pooling layer with a 2 × 2 window. A ReLU activation function is also used. The max-pooling layer's 2-dimensional output vector is then converted to a 1-dimensional vector with a size of 1600 × 1 using a flatten module from the TensorFlow package. The converted vector is then passed through four hidden layers. These hidden layers have 512, 128, 32, and 10 nodes, respectively. Following each of these hidden layers, the ReLU activation function is used. Finally, the output layer is generated using a hidden layer of ten nodes and the Softmax activation function.

(B) Experimental setup

The CIFAR-10 dataset is used to train a deep CNN model with a total of 955,512 parameters. The model is trained using the optimization algorithms HN_Adam, AdaBelief, Adam, AMSGrad, SGD, RMSProp, and AdaGrad individually. These algorithms are used as learning algorithms to train the CNN model. The performance for each compared algorithm is measured in terms of the minimum training loss function and the testing accuracy.

(C) Results and discussions

The response curves of the compared algorithms during training process are indicated in Figs. 8 and 9. Figure 8 shows the accuracy curves for the compared algorithms during the training process for the CNN model. We focus on the basic Adam algorithm and the AdaBelief algorithms as they are the most competitive ones of the compared algorithms. Figure 9 shows the loss function minimization curves of the compared algorithms through the training process for the CNN model.

To illustrate the differences between these response curves, the response characteristics in terms of the

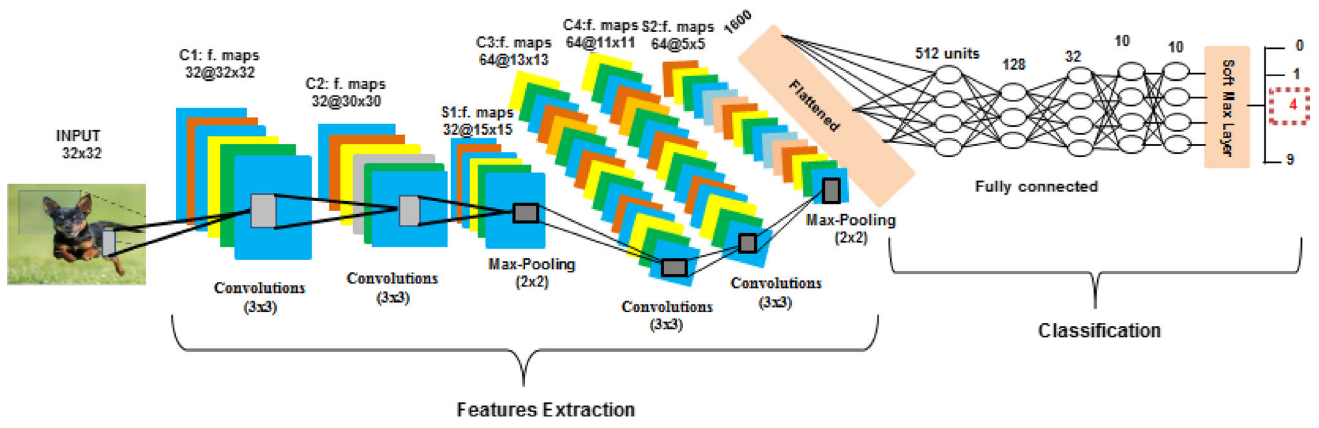
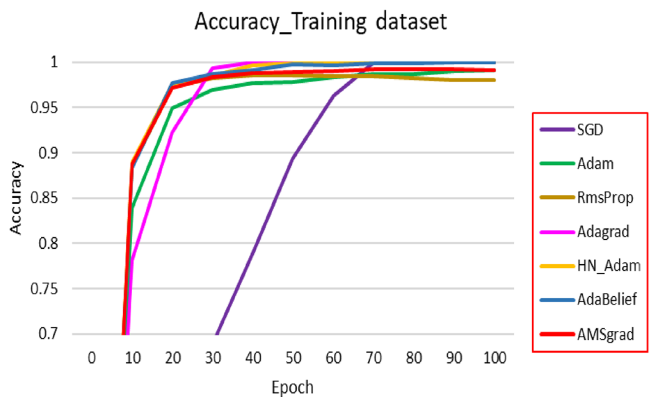


**Fig. 7** Architecture of the deep CNN model using the CIFAR-10 dataset



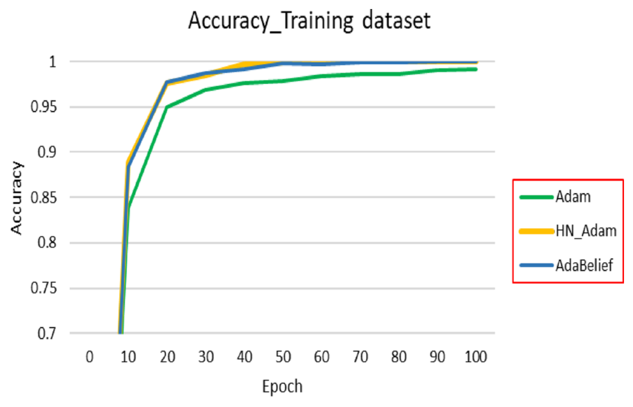
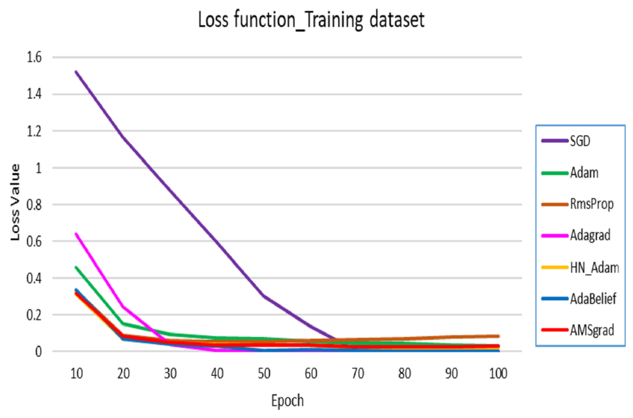


**Fig. 8** Training accuracy of the CNN model for the compared algorithms, case of using CIFAR-10 dataset



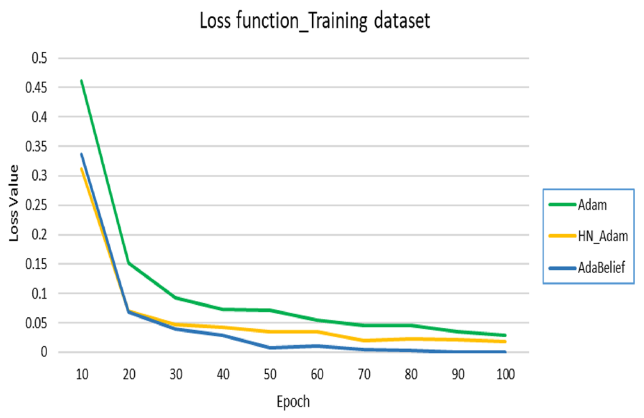


**Fig. 9** Loss function minimization during the training process for HN_Adam, Adam and AdaBelief, case of using CIFAR-10 dataset

**Table 5** Accuracy results, case of using CIFAR-10 dataset

| Algorithm | Min_Loss_Training Dataset | Test_Accuracy |
|---|---|---|
| HN_Adam | 0.0188 | **97.51%** |
| AdaBelief [30] | **0.0101** | 96.60% |
| Adam [8] | 0.0292 | 96.0431% |
| AMSGrad [35] | 0.0281 | 96.096% |
| SGD [33] | 0.0318 | 96.042% |
| RMSprop [19] | 0.0577 | 96.091% |
| Adagrad [20] | 0.387 | 96.97% |

Bold indicates the best achieved value for each response characteristic

**Table 6** The consumed training time for the compared algorithms, case of using CIFAR-10 dataset

| Algorithm | Training Time |
|---|---|
| HN_Adam | **2737 s** |
| AdaBelief [30] | 2784 s |
| Adam [8] | 2767 s |
| AMSGrad [35] | 2822 s |
| SGD [33] | 2788 s |
| RMSprop [19] | 2851 s |
| Adagrad [20] | 2780 s |

Bold indicates the best achieved value for each response characteristic

minimum loss function during training process, the accuracy of the testing on test dataset are calculated and listed in Table 5. For simplicity, the minimum training loss function and testing accuracy are determined after 100 epochs for 5 independent runs with randomly shuffled training data. The best achieved value for each response characteristic is highlighted in bold.

According to the test accuracy, the proposed HN_Adam algorithm outperforms the other compared algorithms with a value of 97.51%. For the minimum training loss function, AdaBelief was in the first rank with a value of 0.0101. HN_Adam was in the second rank with a value of 0.0188. Adagrad gives the worst values among the compared algorithms.

Table 6 shows the training time in seconds consumed by the compared algorithms during the training process. The learning algorithms use these training time values to train the CNN model and achieve the previous accuracy results shown in Table 5, where 20 epochs are considered for simplicity. The HN_Adam outperforms the other

**Table 7** Top-1 accuracy results using ImageNet dataset

| Algorithm | Top -1 Accuracy |
|---|---|
| HN_Adam | **73.20%** |
| AdaBelief [30] | 70.08% |
| Adam [8] | 63.79% |
| SGD [33] | 70.23% |
| Yogi[38] | 68.23% |
| RAdam [40] | 67.62% |
| MSVAG [39] | 65.99% |

Bold indicates the best achieved value for each response characteristic

optimizers with a minimum training time value of 2737 s and thus it achieves a high speed of convergence.

***Remark 1.*** Without loss of generality, the modified algorithm, HN_Adam, is applied to the deep CNN models of the sequential architecture. It can also be applied to more complex and diverse deep CNN architectures such as LeNet-5 [58], ResNet [59] and AlexNet [60]. The authors of [58], for example, use the EVGO algorithm to train three different CNN models based on these architectures. The first model employs the LeNet-5 architecture, which has a total of 81,194 parameters. The second model employs the AlexNet architecture, which has a total of 1,250,666 parameters (1,249,866 trainable and 800 non-trainable). The final model employs the ResNet architecture, which has a total of 271,690 parameters. The first model, like ours, is trained on the MNIST dataset [13], while the other two models are trained on the CIFAR-10 dataset [30].

Their results in terms of maximum training accuracy, minimum training cost, maximum validation accuracy, and minimum validation cost are (99.90%, 9.69E-06, 97.98%, and 0.066) for the first model, (98.11%, 0.0534, 80.42%, and 0.066) for the second model, and (91.06%, 0.6192, 87.25%, and 0.4666) for the third model, as shown in [37]. To ensure that the HN_Adam algorithm can be used efficiently with a variety of CNN model architectures, we used it to train the same CNN model architectures as in [37]. Based on the results, HN_Adam outperforms the EVGO algorithm for all three architectures tested. The maximum training accuracy, minimum training cost, maximum validation accuracy, and minimum validation cost for the LeNet-5 architecture are (100%, 5.81E-06, 99.23%, and 0.0388), respectively, for the AlexNet architecture (99.29%, 0.0230, 97.89%, and 0.0827), and for the ResNet architecture (98.00%, 0.2689, 95.49%, and 0.3382). This demonstrates that the HN Adam algorithm can deal with various CNN architectures while achieving high performance results.

***Remark 2.*** It should be noted that while advanced computational devices can train deep CNN models quickly, they cannot solve the convergence problem for more complex deep neural network models with different architectures that can be handled by the proposed algorithm. Furthermore, the proposed algorithm can be easily applied to computational devices with limited hardware resources.

***Remark 3.*** To ensure the good performance of the modified algorithm, HN_Adam, over large-scale datasets, we evaluate it using the ImageNet dataset that contains 3.2 million cleanly annotated images spread over 5247 categories [64]. This dataset is used to train a deep CNN model of the ResNet-18 architecture [65], which has a total of 11,196,042 parameters (11,186,442 trainable parameters and 9,600 non-trainable parameters). We use HN_Adam, AdaBelief [30], Adam [8], SGD [33], Yogi [38], RAdam [40] and MSVAG [39] as learning algorithms during the training process of the ResNet18 deep network model. The results are obtained in terms of the top-1 accuracy considering the testing dataset for 100 epochs. The top-1 accuracy represents conventional accuracy considering the class with the highest probability (the top one). The results of the top-1 accuracy for the learning algorithms are listed in Table 7. The results of the compared algorithms are taken the same as in [30, 66]. The results of our proposed HN_Adam algorithm are obtained considering the parameter settings for Mini-batch size, learning rate ($\eta$), $\beta_1$, $\beta_2$, and $\varepsilon$ to be the same as in [30].

As illustrated in Table 7, HN_Adam achieves the highest top-1 accuracy with a value of 73.2% and outperforms the other adaptive methods. This confirms that HN_Adam has a good generalization performance for different deep CNN models over different sizes of datasets.

## 7 Conclusion and future work

We proposed a simple and intuitive approach for modifying the basic Adam algorithm to address the generalization performance and convergence issues. The modified algorithm, denoted by HN_Adam, can improve the basic Adam algorithm's generalization performance and reduce training time without increasing its complexity. HN_Adam is used to train a deep CNN model over two different benchmark datasets. To evaluate the HN_Adam algorithm, it is compared to the following learning algorithms: AdaBelief, Adam, AMSGrad, SGD, RMSProp, and AdaGrad. The results are presented in terms of the minimum training cost, maximum training accuracy, minimum validation cost, maximum validation accuracy, maximum test accuracy, and training time consumed. Where the minimum training and validation costs are the least values of the loss function that are attained by the learning algorithms during the training and validation processes, respectively. Moreover, the accuracy curves during the training and validation processes are also given. The results demonstrate that HN_Adam outperforms the compared algorithms for the majority of the compared items.

For future work, the modified algorithm can be used to enhance the learning stability for other more complex deep learning models such as the generative adversarial networks (GANs), and the autoencoders networks.

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

**Data availability** All data generated or analyzed during this study are included in this published article. Derived data supporting the findings of this study are available from the corresponding author on request.

## Declarations

**Conflict of interest** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

1. Alzubaidi L, Zhang J, Humaidi AJ (2021) Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8:53
2. Michael G, Kaldewey T, Tam D (2017) Optimizing the efficiency of deep learning through accelerator virtualization. IBM J Res Dev 61:121–1211. https://doi.org/10.1147/JRD.2017.2716598
3. Maurizio C, Beatrice B, Alberto M, Muhammad S, Guido M (2020) An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. J Fut Inter 12:113
4. Pouyanfar S, Sadiq S, Yan Y (2018) A survey on deep learning: algorithms, techniques, and applications. ACM Comput Surv 51:5
5. Hassen L, Slim B, Ali L, Chih CH, Lamjed BS (2021) Deep convolutional neural network architecture design as a bi-level optimization problem. J Neuro Comput 439:44–62
6. Shiliang S, Zehui C, Han Z, Jing Z. (2019) A survey of optimization methods from a machine learning perspective, supported by NSFC Project 61370175 and Shanghai Sailing Program 17YF1404600
7. Qbal I, Sarker H (2021) Machine learning: algorithms, real-world applications and research directions, J SN Comput Sci
8. Kingma DP, Jimmy B (2015) Adam: a method for stochastic optimization, Presented at International Conference on Learning Representations (ICLR)
9. Liangchen L, Yuanhao X, Liu Y, Sun X (2019) Adaptive gradient methods with dynamic bound of learning rate, arXiv preprint arXiv:1902.09843
10. Ruder S, Park SM, Sim KB (2017) An overview of gradient descent optimization algorithms, arXiv:1609.04747v2 [cs.LG]
11. Sebastian B, Josef G, Martin W (2018) An improvement of the convergence proof of the Adam-optimizer, CoRR, abs/1804.10587
12. Agnes L, Sagayaraj F (2019) A survey of optimization techniques for deep learning networks, Int J Res Eng Appl Manag (IJREAM) 5:2
13. Zhang Z (2018) Improved Adam optimizer for deep neural networks, IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), pp. 1–2
14. Wilson AC, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning, in Advances in Neural Information Processing Systems
15. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V , Rabinovich A (2015) Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9
16. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–77
17. Zhang C, Bengio S, Hardt M, Recht B, Vinyals O (2017) Understanding deep learning requires rethinking generalization, in ICLR 2017
18. Arpit D, Jastrzębski S, Ballas N, Krueger D, Bengio E, Kanwal MS, Maharaj T, Fischer A, Courville A, Bengio .Y (2017) A closer look at memorization in deep networks, arXiv preprint arXiv:1706.05394
19. Tieleman T, Hinton G (2012) Lecture 6.5—RmsProp: divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning
20. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
21. Tom S, Sixin Z, Yann L (2012) No more pesky learning rates. arXiv preprint arXiv:1206.1106
22. Zeiler MD (2012)Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701
23. Nicolas RL, Andrew FW (2010) A fast natural newton method. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 623–630
24. Razvan P, Yoshua B (2013) Revisiting natural gradient for deep networks. arXiv preprint arXiv:1301.3584
25. Amari S (1998) Natural gradient works efficiently in learning. Neural Comput 10(2):251–276
26. Huang H, Wang C, B Dong (2018) Nostalgic adam: weighting more of the past gradients when designing the adaptive learning rate, arXiv preprint arXiv:1805.07557
27. Wang G, Lu S, Tu W, Zhang. (2019) LSadam: A variant of adam for strongly convex functions, arXiv preprint arXiv:1905.02957
28. Li W, Zhang Z, Wang X, Luo P (2020) Adax: Adaptive gradient descent with exponential long term memory, arXiv preprint arXiv:2004.09740
29. Zhang M, Lucas J, Ba J, Hinton GE (2019) Lookahead optimizer: k steps forward, 1 step back, in Advances in Neural Information Processing Systems, pp. 9593–9604

30. Zhuang J, Tang T, Ding Y, Tatikonda S, Dvornek, X. Papademetris N, Duncan JS (2020) AdaBelief Optimizer: Adapting stepsizes by the belief in observed gradients, 34th Conference on Neural Information Processing Systems (NeurIPS
31. Wilson AC, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning. In Advances in Neural Information Processing Systems, pp. 4148–4158
32. Lyu K, Li J (2019) Gradient descent maximizes the margin of homogeneous neural networks,“ arXiv preprint arXiv:1906.05890
33. Keskar NS, Socher R (2017) Improving generalization performance by switching from adam to sgd, arXiv preprint arXiv:1712.07628
34. Luo L, Xiong Y, Liu Y, Sun X (2019) Adaptive gradient methods with dynamic bound of learning rate, arXiv preprint arXiv:1902.09843
35. Reddi SJ, Kale S, Kumar S (2019) On the convergence of adam and beyond, arXiv preprint arXiv:1904.09237
36. Yi D, Ahn J, Ji S (2020) An effective optimization method for machine learning based on ADAM. Appl Sci 10:1073. https://doi.org/10.3390/app10031073
37. Karabayir I, Akbilgic O, Tas N (2020) A novel learning algorithm to optimize deep neural networks: evolved gradient direction optimizer (EVGO). IEEE Transactions on Neural Networks and Learning Systems
38. Manzil Z, Sashank R, Devendra S, Satyen K, and Sanjiv K (2018) Adaptive methods for nonconvex optimization. Adv Neural Inf Process Syst 9793–9803
39. Balles L, Hennig P (2017) Dissecting Adam: the sign, magnitude and variance of stochastic gradients, arXiv preprint arXiv:1705.07774
40. Liu L, Jiang H, He P, Chen W, Liu X, Gao J, and Han J (2019) On the variance of the adaptive learning rate and beyond, arXiv preprint arXiv:1908.03265
41. Bernstein J, Vahdat A, Yue Y, Liu M (2019) On the distance between two neural networks and the stability of learning, arXiv preprint arXiv:2002.03432
42. Loshchilov I, Hutter F (2017) Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101
43. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252
44. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. Adv Neural Inf Process Syst 2672–2680
45. Wedderburn RW (1974) Quasi-likelihood functions, generalized linear models, and the gauss—newton method. Biometrika 61(3):439–447
46. Nocedal J (1980) Updating quasi-newton matrices with limited storage. Math Comput 35(151):773–782
47. Pascanu .R, Bengio .Y (2013) Revisiting natural gradient for deep networks, arXiv preprint arXiv:1301.3584
48. Martens J (2010) Deep learning via hessian-free optimization. ICML 27:735–742
49. Jascha SD, Ben P, Surya G (2014) Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods, Proceedings of the 31 st International Conference on Machine Learning, Beijing, China
50. Miaomiao L, Dan Y, Zhigang L, Jingfeng G, Jing C (2023) An Improved adam optimization algorithm combining adaptive coefficients and composite gradients based on randomized block coordinate descent. Hindawi Computational Intelligence and Neuroscience Volume, Article ID 4765891(2023).
51. Ran. T, Ankur. P. P “Amos: An Adam-style Optimizer with adaptive weight decay towards model-oriented scale”, conference paper at ICLR (2023).
52. Xingyu X, Pan Z, Huan L, Zhouchen L, Shuicheng Y (2022) Adan: adaptive nesterov momentum algorithm for faster optimizing deep models. arXiv:2208.06677v3 [cs.LG]
53. Keijsers NLW (2010) Neural Networks, in Encyclopedia of Movement Disorders
54. Yang ZR, Yang Z (2014) Bioinformatics. In Comprehensive Biomedical Physics
55. Jiuxiang G, Zhenhua W, Jason K, Lianyang M, Amir S, Bing S, Ting L, Xingxing W, Wangb L, Gang W, Jianfei C , Tsuhan C (2017) Recent advances in convolutional neural networks. Adv Neural Inf Process Syst 4148–4158
56. Wang B, Sun Y, Xue B, Zhang M (2018) Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. arXiv preprint arXiv:1803.06492
57. Toussaint M (2012) Lecture notes, Some notes on gradient descent
58. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
59. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In Proceedings of IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp. 770–778
60. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks In Proc Adv Neural Inf Process Syst 1097–1105
61. Wang S, Sun J, Xu Z HyperAdam (2019) A learnable task-adaptive adam for network training, The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)
62. Yao Z, Gholami A, Shen S, Keutzer K, Mahoney MW (2020) Adahessian: An adaptive second order optimizer for machine learning, arXiv preprint arXiv:2006.00719
63. Yuan W, Gao K (2020) Eadam optimizer: How epsilon impact Adam, arXiv preprint arXiv:2011.02150
64. Jia Li, Kai Li, Li Fei-Fei (2009) Imagenet: A large-scale hierarchical image database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 248–255
65. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016) Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770– 778.
66. Jinghui C, Quanquan G (2018) Closing the generalization gap of adaptive gradient methods in training deep neural networks,” arXiv preprint arXiv:1806.06763