Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 10, No. 4, 2022, 467-477.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Original paper

Whitened gradient descent, a new updating method for optimizers in deep neural networks

Hossein Gholamalinejad[1,2*] and Hossein Khosravi [2]

1. *Department of Computer, Faculty of Engineering, Bozorgmehr University of Qaenat, Qaen, Iran.*
2. *Faculty of Electrical Engineering Shahrood University of Technology.*

Article Info

Article History:
Received 08 November 2021
Revised 06 January 2022
Accepted 05 April 2022

DOI:10.22044/jadm.2022.11325.2291

Keywords:
Deep learning, Optimizer, Whitened gradient descent, Momentum.

**Corresponding author: Gholamalinejad69@gmail.com (H. Gholamalinejad).*

Abstract

Optimizers are the vital components of deep neural networks that perform weight updates. This paper introduces a new updating method for optimizers based on a gradient descent called the whitened gradient descent (WGD). This method is easy to implement, and can be used in every optimizer based on the gradient descent algorithm. It does not increase the training time of the network significantly. This method smooths the training curve, and improves the classification metrics. In order to evaluate the proposed algorithm, we perform 48 different tests on two datasets, Cifar100 and Animals-10, using three network structures including densenet121, resnet18, and resnet50. The experiments show that using the WGD method in the gradient descent-based optimizers improves the classification results significantly. For example, integrating WGD in the RAdam optimizer increases the accuracy of DenseNet from 87.69% to 90.02% on the Animals-10 dataset.

1. Introduction

Today, artificial neural networks (ANNs) have become very widespread and diversified. It attempts to simulate the human brain's learning process on a computer and make the computers more intelligent. ANNs can be divided into two categories: shallow networks and deep networks. In shallow ANNs, the inputs are the feature vectors. These features are extracted using some handcrafted algorithms such as PCA, FFT, Wavelet, and HoG. This process is somewhat different from the human brain's learning process since the brain receives pure data without any changes, and completes the learning or experimentation process. Deep neural networks (DNNs) are another type of ANNs that typically work on images without a prior feature extraction phase. These networks benefit from the deep learning process. Although the basic idea of deep learning was introduced many years ago, it was not possible to implement these algorithms due to the weakness of the existing hardware systems at that time. The advancement of hardware technology made the implementation of these algorithms feasible. Recently, many deep learning algorithms have been proposed to solve the traditional artificial intelligence problems.

The architecture of DNNs is hierarchical. A DNN uses this architecture for feature extraction. Today, these networks are used in many applications of artificial intelligence including semantic parsing [1-3], transfer learning [4-8], natural language processing [9-11], computer vision [12-15], and many more. The main reasons for the rapid development of these networks can be summarized in three things: significant increase in the hardware processing capabilities, especially GPUs, reduced cost of computer hardware, and significant improvements in machine learning algorithms [16].

DNNs have been widely used in the recent years, and several models have been proposed for different applications. These models can be divided into five categories [17, 18]: Convolution Neural Networks (CNN), Restricted Boltzmann Machines (RBM), Autoencoders, Sparse Coders, and Recurrent Neural Networks. The back-

propagation (BP) [19] algorithm is used in the feed-forward neural networks. Forward structure means that the artificial neurons are placed in successive feeder layers, and send their outputs forward. The term back-propagation implies that the errors are fed backward in the network to correct the weights. The BP method is a supervised method in the sense that the input samples are labeled, and the expected output is known in advance. Therefore, the network output is compared with these ideal outputs, and the network error is calculated. In this algorithm, it is first assumed that the network weights are randomly selected. In each step, the network output is calculated, and following the difference between the output and the target (desired output), the weights are updated to minimize this error using a loss function. In order to compute the gradient of the loss function, some training data is used. According to the data, the gradient descent methods can be divided into three categories: batch gradient descent (BGD) method, mini-batch gradient descent (MBGD) method, and stochastic gradient descent (SGD) method [20]. In the BGD method, the gradient is calculated for the entire training data. Therefore, in the case of a large amount of data, this method is practically ineffective because it requires a lot of memory. In SGD, only one training sample is used in each step to calculate the gradient. This algorithm requires less storage but it is hard to establish the theoretical convergence conditions. In MBGD, the benefits of both methods are used; a subset of training data is used to update the weights and calculate the gradients.

In the neural network, we need an algorithm that changes the weights to minimize the loss score. The algorithm that performs such a task is called the optimizer. It is performed iteratively to achieve an optimal solution for the weights.

In this work, we propose a new updating method for gradient descent-based optimizers called Whitened Gradient Descent (WGD). It smooths and accelerates the training process of DNN and also improves the model generalization performance. In this method, we use the mean gradient matrix in each layer of the convolution filter bank; this value is calculated and subtracted from all gradient kernel gradients in that layer. In this way, similar information in the convolution layer is deleted. The experimens show that the proposed method improves the network behavior in terms of classification criteria without significantly changing the network processing time.

The rest of this paper is organized as what follows. The works related to our idea and famous optimizing methods are discussed in Section 2. The proposed method and its structure are described in the next section. In Section 4, the experimental results are discussed. The experiments are conducted in two stages: first, each network is trained using the original optimizer, and then the process is repeated using the modified WGD optimizers. Finally, in Section 5, we draw the paper to conclusions.

2. Related Works

There are a lot of optimizer algorithms used in DNNs [21-26], among which the gradient descent based algorithms are the most popular. In this section, we introduce the famous optimizers based on gradient descent. In order to mathematically present the problem, assume that *f(x)* is the desired objective function, $x \in R^n$ and n is the number of training data. The corresponding gradient is $\Delta f\ (x)$. The step size for iteration k is t_k.

2.1. Batch gradient descent

In this method, a new x is calculated after computing the gradient of the whole training data (Equation 1):

$$x_{k+1} = x_k - t_k \Delta f\ (x_k)^{(1:n)} \tag{1}$$

The batch gradient descent ensures that the back-propagation algorithm converges to a minimum for the convex and non-convex problems. In deep learning applications, calculating the gradient for all training data costs a lot of time and memory. For this reason, the weights are slowly updated in this method. Therefore, deep neural networks rarely use this method to calculate the gradients and update the weights.

2.2. Stochastic gradient descent

Another method to optimize the gradient is SGD. Since the selection of training samples is random, it is called stochastic. In this method, only one training data is used in each step to update the weights and calculate the gradient (Equation 2):

$$x_{k+1} = x_k - t_k \Delta f\ (x_k)^{(i)} \tag{2}$$

in which i is the number of training data. Since gradient computation and weight update are conducted sample by sample, the computed gradient is not necessarily equal to the actual gradient of the error curve for all training data. As a result, the likelihood of convergence decreases.

2.3. Mini-batch gradient descent

The Mini-Batch gradient descent has the advantages of both of the previous methods, i.e. BGD and SGD. In this algorithm, the training data is split into several mini-batches, and then one of them is used to calculate the gradient at each step. (Equation 3):

$$x_{k+1} = x_k - t_k \Delta f\ (x_k)^{(i:i+m)} \tag{3}$$

where m represents the mini-batch size. This method does not guarantee convergence, and if the learning rate is not adjusted correctly, the possibility of divergence is high. Therefore, other techniques are used for convergence.

2.4. Gradient descent with momentum

Qian has considered momentum to reduce variance in SGD [28]. Momentum accelerates convergence in the desired direction, and prevents the tendency to irrelevant directions. In this method, instead of using only the gradient of the current step to guide the search, the momentum from previous steps is also employed to determine the direction of the gradient [27]. (Equation 4):

$$\begin{aligned} v_k &= m v_{k-1} + t_k \Delta f\ (x_k) \\ x_{k+1} &= x_k - v_k \end{aligned} \tag{4}$$

Where v_k is the rate of weight change, and $m \in [0,1]$ is the momentum factor and often has a value close to 1.0 such as 0.8, 0.9, or 0.99. A momentum of 0.0 is the same as gradient descent without momentum.

2.5. Nesterov accelerated gradient

Nesterov accelerated gradient (NAG) [29] is another momentum-based algorithm. This method calculates the gradient relative to the next approximate value of the parameters, not their current value. The difference between NAG and classical momentum is that NAG puts more weight on the recent gradients. In other words, NAG forgets old gradients more quickly. Weight update in this method is shown in Equation 5:

$$\begin{aligned} v_k &= m v_{k-1} + t_k \Delta f\ (x_k - m v_{k-1}) \\ x_{k+1} &= x_k - v_k \end{aligned} \tag{5}$$

In this method, we give the momentum coefficient *m* a value of about 0.9. In the gradient method, the current gradient is first calculated, and then this gradient makes a significant leap in the direction of the new accumulated gradient. The Nesterov method first makes a large leap in the direction of the previously accumulated gradient, measures the gradient, and then makes a minor correction. These changes created by predicting future values prevent the changes from accelerating too much.

2.6. Adagrad

One of the disadvantages of all the optimizers described so far is that the learning rate is constant across all parameters and cycles. Adagrad [30] changes the learning rate for each parameter in each time interval, and is a second-order optimization algorithm that works with the derivative of the error function (Equation 6):

$$\begin{aligned} G_k &= G_{k-1} + \Delta f\ (x_k)^{(2)} \\ x_{k+1} &= x_k - \frac{t}{\sqrt{G_k + \varepsilon}} \Delta f\ (x_k) \end{aligned} \tag{6}$$

Here, *G* is a diagonal matrix in which each *i*-th object of diameter is the sum of the squares of the gradients, and ε is a smoothing coefficient that prevents zero in the denominator. One of the main advantages of this method is that it eliminates the requirement to determine the learning rate. In most applications, the value 0.01 is selected, and does not change. The main disadvantage of the Adagrad is the square aggregation matrix of the gradients at the denominator. Since $\Delta f\ (x_k)^2$ is a positive value, the sum always increases during the network's training. This reduces the rate of learning (Equation 6), and eventually, weight update will have no effect. After that, the algorithm cannot learn anything else.

2.7. Adadelta

The Adadelta method [31] is a more robust version of the Adagrad. It tries to overcome the main drawbacks of Adagrad: 1) the continual decay of learning rates throughout training, and 2) the need for a manually selected global learning rate. Adadelta adapts the learning rates based on a moving window of gradient updates instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done.

This method has the benefits of both Adagrad and momentum. Followings are the operation details of Adadelta (Equation 7):

$$\begin{aligned} &E[\Delta f\ (x)^2]_k = \rho E[\Delta f\ (x)^2]_{k-1} + \\ &(1-\rho)\Delta f\ (x)^2 \\ &\hat{x}_k = -\frac{\sqrt{E[\hat{x}]_{k-1} + \varepsilon}}{\sqrt{E[\Delta f\ (x)^2]_k + \varepsilon}} \Delta f\ (x_k) \\ &E[\hat{x}^2]_k = \rho E[\hat{x}^2]_{k-1} + (1-\rho)\hat{x}_k^2 \\ &x_{k+1} = x_k + \hat{x}_k \end{aligned} \tag{7}$$

where ρ is a decay constant (e.g, 0.95) and ε is a small value (e.g. 1e-6) for numerical stability.

2.8. RMSprop

The RMSprop [5] method is an Adagrad-based method. The RMS and Adadelta deduction methods were developed independently and almost simultaneously following the need for a solution to the problem of drastically reducing the learning rate in the Adagrad method. This method also divides the learning rate by the average attenuated square of the gradients (Equation 8):

$$\begin{aligned} &G_k = G_{k-1} + \Delta f E[\Delta f\ (x)^2]_k = \\ &\rho E[\Delta f\ (x)^2]_{k-1} + (1-\rho)\Delta f\ (x_k) \\ &x_{k+1} = x_k - \frac{t}{\sqrt{E[\Delta f\ (x)^2]_k + \varepsilon}} \Delta f\ (x_k) \end{aligned} \tag{8}$$

where ρ is a decay constant, and is commonly considered as 0.9.

2.9. Adam

Adam [32] is another method that calculates the learning rate according to the data. In addition to storing the exponentially squared averages of the previous gradients, such as the Adadelta method, it also preserves exponentially averaging the gradients like the NAG method [32].

Adam's updating rule consists of the following steps (Equation 9):

$$\begin{aligned} &m_k = \beta_1 m_{k-1} + (1-\beta_1)\Delta f\ (x_k) \\ &v_k = \beta_2 m_{k-1} + (1-\beta_2)\Delta f\ (x_k)^2 \\ &\hat{m}_k = \frac{m_t}{1-\beta_1^k} \\ &\hat{v}_k = \frac{m_t}{1-\beta_2^k} \\ &x_{k+1} = x_k - \frac{\eta k}{\sqrt{\hat{v}_k} + \varepsilon} \hat{m}_k \end{aligned} \tag{9}$$

in which m_k is the exponential moving averages of the gradient, and v_k is the moving averages of the squared gradient. The hyper-parameters β_1 and β_2 control the exponential decay rates of these moving averages. These moving averages are initialized as (vectors of) 0's, leading to moment estimates that are biased towards zero. $\hat{m}_k$ and $\hat{v}_k$ are the bias-corrected estimates. The developers have proposed default values of 0.9 for β_1, 0.999 for β_2 and 10e-8 for ε. They have shown empirically that the Adam method works well in practice, and is superior to other adaptive learning methods.

2.10. Nadam

Nadam (Nesterov-accelerated Adaptive Moment Estimation) [29] combines the Adam and Nesterov accelerated gradient (NAG) [33]. The weight update of this method is shown in Equation 10:

$$\begin{aligned} &g_k = \nabla_{xk} J(x_k) \\ &m_k = \gamma m_{k-1} + \eta g_k \\ &\hat{m}_k = \frac{m_t}{1-\beta_1^k} \\ &\hat{v}_k = \frac{m_t}{1-\beta_2^k} \\ &x_{k+1} = x_k - \frac{\eta}{\sqrt{\hat{v}_k} + \varepsilon} (\beta_1 \hat{m}_k \\ &+ \frac{(1-\beta_1) g_k}{1-1-\beta_1^k}) \end{aligned} \tag{10}$$

where β_1 can be 0.9, β_2 can be 0.999, and ε can be 1e-8.

2.11. Radam

Rectified Adam or Radam [34] is a variant of the Adam stochastic optimizer that introduces a term to rectify the variance of the adaptive learning rate. It seeks to tackle the convergence problem suffered by Adam. The authors in [34] argue that the root cause of this behavior is that the adaptive learning rate has an undesirably large variance in the early stage of model training due to the limited amount of training samples being used. Thus to reduce such variance, it is better to use lower learning rates in the first few epochs of training-that justifies the warmup heuristic. The Radam optimizer algorithm is shown in Algorithm 1.

3. Proposed Method

In DNNs, the convolution layer weights are the factors that change the information between layers. These weights are stored in memory as a matrix. They often contain similar information. The targeted removal of this information can help to produce better feature maps. In this article, we propose a new optimizer algorithm that removes this similar information. We use the mean matrix for this purpose. In each convolution layer, the mean matrix to produce better feature maps. In this article, we propose a new optimizer algorithm that removes this similar information. We use the mean matrix for this purpose.

Algorithm 1. RAdam

Input: $\{\alpha_k\}_{k=1}^{K}$: step size, $\{\beta_1, \beta_2\}$: Degradation rate for calculating the moving average and moving 2nd moment, x_0: initial parameter, $f_k(x)$: stochastic objective function.

Output: x_k: resulting values

$m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)
$\rho_\infty \leftarrow \frac{2}{1-\beta_2} - 1$ (Compute the maximum length of the approximated SMA)
While $k = \{1, \dots, K\}$ **do**
  $g_k \leftarrow \nabla_x f_k(x_{k-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)
  $v_k \leftarrow \frac{1}{\beta_2 v_{k-1}} + (1-\beta_2) g_k^2$ (Update exponential moving 2nd moment)
  $m_k \leftarrow \beta_1 m_{k-1} + (1-\beta_1) g_k$ (Update exponential moving 1st moment)
  $\widehat{m_k} \leftarrow m_k / (1-\beta_1^k)$ (Compute bias-corrected moving average)
  $\rho_k \leftarrow \rho_\infty - 2t\beta_2^k / (1-\beta_2^k)$ (Compute the length of the approximated SMA)
  If *the variable is tractable, i.e.,* $\rho_k > 4$ **then**
    $l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$ (Compute adaptive learning rate)
    $r_k \leftarrow \sqrt{\frac{(\rho_k - 4)(\rho_k - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_k}}$ (Compute the variance rectification term)
    $x_k \leftarrow x_{k-1} - \alpha_k r_k \widehat{m_k} l_k$ (Update parameters with adaptive momentum)
  Else
    $x_k \leftarrow x_{k-1} - \alpha_k \widehat{m_k}$ (Update parameters with un-adapted momentum)
return x_k

In each convolution layer, the mean matrix of the weights is subtracted from all weight kernels. The formulation of the proposed algorithm is shown in Equation 11:

$$h_{k,x} = \nabla_x J(k,x) - \mu_x$$
$$\mu_x = \frac{1}{P}\sum_{i=1}^{P} \nabla_x J(i,x) \quad (11)$$

where J is the objective function, P is the weight kernel number in a specific layer, and μ_x is the mean of weight kernels in a specific layer. Due to subtracting the mean from each gradient vector, we named our method whitened gradient descent (WGD). This algorithm can be integrated with any gradient descent-based optimizer by replacing $\nabla_x J(x)$ with $h_{k,x}$. As an example, embedding the WGD to the RAdam optimizer is shown in Algorithm 2.

4. Experiments

In this section, we study the effect of embedding the proposed updating algorithm in four famous optimizers in the classification tasks. Three different image datasets are used to evaluate the impact of integrating WGD with different optimizers.

4.1. Network training

We conducted our experiments on a PC with GeForce Turbo RTX-2080 GPU and Corei3-9100f CPU running at 4000MHz. The proposed algorithm and model are implemented in Python 3.7 using the Pytorch and OpenCV libraries. Due to the imbalance of the samples in different classes of the datasets, we used a weighted random sampler [35] to balance the training data. The training parameters of the proposed model are listed in Table 1.

Table 1. Training parameters of proposed model.

Batch size	**32**
Epochs	**50**
Momentum	**0.9**
Learning rate	**0.01**
Weight decay	**1e-3**
Epsilon	**1e-10**
Weighted random sampler	**Sampler**

We used three different DNNs including ResNet18, ResNet50, and Densenet121. The first structure that we used was ResNet [36]. It has a novel structure, having the new residual connections; it can be trained on every dataset without the problem of vanishing gradients. It supports very deep layers, and exists in different versions from ResNet18 to ResNet152, and even more. We trained two structures, ResNet18 and ResNet50. Another DNN that we used in this article is DenseNet [37]. DenseNet is a type of convolutional neural network that uses dense connections between layers through Dense Blocks, where all layers are connected (with matching feature-map sizes) directly with each other.

Algorithm 2. WGD RAdam

Input: $\{\alpha_k\}_{k=1}^{K}$: step size, $\{\beta_1, \beta_2\}$: Degradation rate for calculating the moving average and moving 2nd moment, x_0: initial parameter, $f_k(x)$: stochastic objective function.

Output: x_k: resulting values

$m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)
$\rho_\infty \leftarrow \frac{2}{1-\beta_2} - 1$ (Compute the maximum length of the approximated SMA)
While $k = \{1, \dots, K\}$ **do**
$g_k \leftarrow \nabla_x f_k(x_{k-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)
$g_k \leftarrow g_k - \mu$
$v_k \leftarrow \frac{1}{\beta_2 v_{k-1}} + (1-\beta_2) g_k^2$ (Update exponential moving 2nd moment)
$m_k \leftarrow \beta_1 m_{k-1} + (1-\beta_1) g_k$ (Update exponential moving 1st moment)
$\widehat{m_k} \leftarrow m_k / (1-\beta_1^k)$ (Compute bias-corrected moving average)
$\rho_k \leftarrow \rho_\infty - 2t\beta_2^k / (1-\beta_2^k)$ (Compute the length of the approximated SMA)
If *the variable is tractable, i.e.,* $\rho_k > 4$ **then**
$l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$ (Compute adaptive learning rate)
$r_k \leftarrow \sqrt{\frac{(\rho_k - 4)(\rho_k - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_k}}$ (Compute the variance rectification term)
$x_k \leftarrow x_{k-1} - \alpha_k r_k \widehat{m_k} l_k$ (Update parameters with adaptive momentum)
Else
$x_k \leftarrow x_{k-1} - \alpha_k \widehat{m_k}$ (Update parameters with un-adapted momentum)
return x_k

We used Densenet121 in our experiments. This architecture is shown in Figure 1.

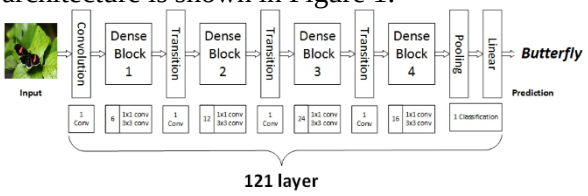

Figure 1. Densenet121 block diagram.

4.2. Datasets

In order to evaluate the proposed method, we used two popular datasets: Animals-10 with ten classes and CIFAR100 with one hundred classes. Animals-10 dataset [38] consists of 26,183 medium quality animal images grouped in 10 categories: dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel, and elephant. Some images from the Animals-10 dataset are shown in Figure 2.

Figure 2. Sample images from Animals-10 dataset.

All the images have been collected from Google images, and have been verified by a human. This dataset contains 23,563 images for training and 2,620 images for the test. The number of training and test data of this dataset are listed in Table 2.

Table 2. Size of each class in Animals-10 dataset.

Class	Training	Testing
Dog	4,376	**487**
Horse	2,361	**262**
Elephant	1,301	**145**
Butterfly	1,901	**211**
Chicken	2,789	**309**
Cat	1,502	**166**
Cow	1,680	**186**
Sheep	1,638	**186**
Spider	4,339	**482**
Squirrel	1,676	**186**
Total	23,563	**2,620**

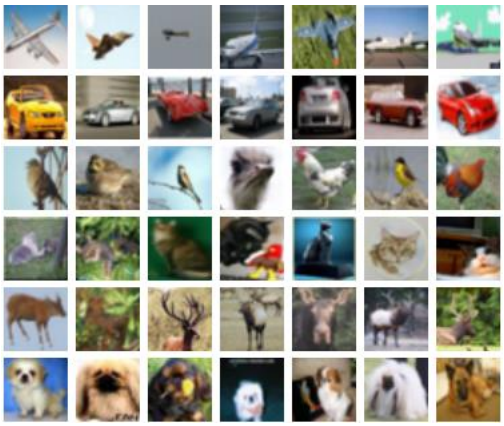

Figure 3. Sample images from CIFAR-100 dataset.

The CIFAR-100 dataset is a labeled subsets of the 80 million tiny images dataset. They were

collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. This dataset is just like the CIFAR-10, except that it has 100 classes containing 600 images in each. There are 500 training images and 100 testing images per class. Some sample images from the CIFAR-100 dataset are shown in Figure 3.

4.3. Metrics

There are various criteria for evaluating the performance of deep neural networks used in the classification problems. In this section, we examine these criteria. These are the mean recall (mRe), the mean precision (mPr), and the Cohen Kappa Score (Kappa), which is a measure to express the agreement between two annotators. In this case, the first annotator is a method under evaluation, and the second annotator is the ground truth. Equation 12 shows these metrics.

$$\begin{aligned} \text{Accuracy} &= \frac{\text{TP}}{\text{Total number of images}} \\ \text{mRe} &= \frac{\sum_{i=1}^{\text{num of classes}} \text{Re}_i}{\text{num of classes}} \\ \text{mPr} &= \frac{\sum_{i=1}^{\text{num of classes}} \text{Pr}_i}{\text{num of classes}} \\ \text{Kappa} &= \frac{\text{Acc-Pe}}{\text{1-Pe}} \end{aligned} \tag{12}$$

Where

$$\text{Re}_i = TP_i / (TP_i + FN_i)$$

$$\text{Pr}_i = TP_i / (TP_i + FP_i)$$

And *Pe* is the probability of agreement when both annotators assign random labels.

4.4. Classification experiments

We embedded our method into some popular gradient-based optimizers including SGDM, Adam, RAdam, and Nadam in order to evaluate their classification metrics performance in the two datasets using three network structures[1]. The training parameters are set as shown in Table 1. Tables 3, 4, 5 show the classification metrics on Animals-10, and Tables 6, 7, and 8 show the results on CIFAR-100.

4.4.1. Animals-10 experiments

Animals-10 is a complex dataset with high-quality images. It has several challenges including different backgrounds, different vision angles, and imbalanced samples, i.e. the number of samples in each category is different. As shown in Tables 3, 4, and 5, embedding WGD to the selected optimizers increased the accuracy and decreased the loss in all the 24 experiments. The best accuracy was achieved using the WGD Radam optimizer in DenseNet121, which was 90%. The best accuracy improvement was achieved in the Nadam optimizer with ResNet50, which increased the accuracy from 67.20% to 76.64%.

As described in [39], Cohen Kappa Score is a statistic metric used to measure the inter-rater reliability (and also the intra-rater reliability) for the qualitative classification items. Thus it could show the power of classifier architecture in classification. As shown in Tables 3, 4, and 5, the best improvement of these parameters was in the Nadam cases that were about 6% in DenseNet121 and ResNet18 and about 10% in ResNet50. According to Cohen Kappa Score, the best result was in WGD Radam for DenseNet121.

Table 3. Classification results for Animals-10 dataset using DenseNet121.

Optimizer	Accuracy	Loss	Mean recall	Mean precision	CKS
SGDM	87.00	0.5074	85.84	86.51	85.20
WGD SGDM	**87.65**	**0.4535**	**87.17**	**87.37**	**85.95**
Adam	77.41	0.6808	76.34	75.99	74.32
WGD Adam	**79.55**	**0.6604**	**77.61**	**78.19**	**76.69**
Nadam	73.66	0.7967	71.17	72.79	69.96
WGD Nadam	**78.36**	**0.6218**	**76.57**	**77.08**	**75.36**
Radam	87.69	0.7606	87.40	86.77	86.01
WGD Radam	**90.02**	**0.6222**	**90.09**	**89.56**	**88.66**

Table 4. Classification results for Animals-10 dataset using ResNet18.

Optimizer	Accuracy	Loss	Mean recall	Mean precision	CKS
SGDM	81.19	0.7271	79.19	80.64	78.54
WGD SGDM	**81.57**	**0.6540**	**79.93**	**81.34**	**79.00**
Adam	72.90	0.9012	70.31	71.58	69.09
WGD Adam	**75.31**	**0.8535**	**74.20**	**73.15**	**71.99**
Nadam	69.19	0.9652	**67.08**	66.87	65.01
WGD Nadam	**75.27**	**0.8185**	52.55	**74.51**	**71.78**
Radam	82.91	1.2590	81.12	82.34	80.51
WGD Radam	**84.71**	**1.1105**	**83.69**	**83.97**	**82.60**

The training curves on Animals-10 using the Radam optimizer are shown in Figure 4 and Figure 5. As shown, using WGD in Radam, the accuracy was increased, and the loss was decreased.

[1] Codes can be found here https://github.com/hoseingh69/WGD

Table 5. Classification results for Animals-10 dataset using ResNet50.

Optimizer	Accuracy	Loss	Mean recall	Mean precision	CKS
SGDM	77.03	1.9745	74.57	76.32	73.79
WGD SGDM	**81.38**	**0.8031**	**79.44**	**80.66**	**78.76**
Adam	73.97	0.7936	69.63	73.84	70.19
WGD Adam	**76.64**	**0.7692**	**74.56**	**76.25**	**73.39**
Nadam	67.20	0.9498	64.73	63.80	62.75
WGD Nadam	**76.07**	**0.7422**	**74.14**	**74.50**	**72.76**
Radam	83.41	1.0974	82.19	82.37	81.12
WGD Radam	**85.28**	**0.8933**	**84.63**	**84.36**	**84.36**

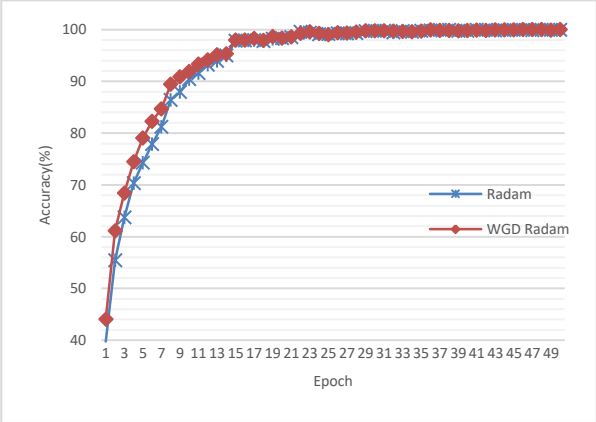

Figure 4. Accuracy of Densenet121 in train data on Animals-10.

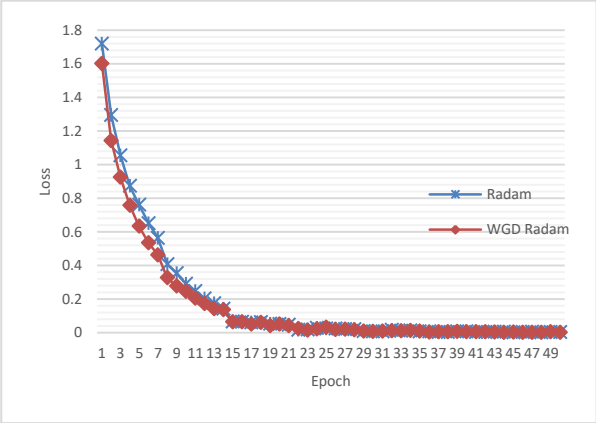

Figure 5. Loss of Densenet121 in train data on Animals-10.

The accuracy results of all experiments on Animals-10 are shown in Figure 6. As presented, in all cases, the embedding WGD improves the accuracy.

4.4.2. CIFAR-100 experiment

CIFAR-100 is an image dataset with a low quality and small images in 100 categories. The input size for our test networks is 224 × 224. Thus the images will be blurred in the input of networks. We did 24 different tests in this stage using DenseNet121, ResNet18, and ResNet50. The results are shown in Tables 6, 7, and 8. As shown, although this dataset has low-quality images, we achieved an accuracy of 70.35% on the test data with WGD Radam. In this case, we had about 7% improvement in accuracy and about 8% in the Cohen Kappa Score, using our method.

Table 6. Classification results for CIFAR-100 dataset using DenseNet121.

Optimizer	Accuracy	Loss	Mean recall	Mean precision	CKS
SGDM	66.16	1.3812	66.16	66.31	65.81
WGD SGDM	**66.70**	**1.3411**	**66.70**	**67.18**	**66.36**
Adam	45.02	2.0750	45.01	44.58	44.46
WGD Adam	**50.86**	**1.8090**	**50.85**	**51.22**	**50.36**
Nadam	40.44	2.3100	40.44	40.76	39.83
WGD Nadam	**46.10**	**1.9762**	**46.09**	**47.31**	**45.55**
Radam	63.23	2.6760	63.23	63.18	62.85
WGD Radam	**70.35**	**2.0094**	**70.35**	**70.73**	**70.05**

Table 7. Classification results for CIFAR-100 dataset using ResNet18.

Optimizer	Accuracy	Loss	Mean recall	Mean precision	CKS
SGDM	58.38	1.6075	58.38	57.99	57.95
WGD SGDM	**60.57**	**1.5235**	**60.57**	**60.30**	**60.17**
Adam	44.13	2.1271	44.13	43.77	43.56
WGD Adam	**46.96**	**1.9737**	**46.96**	**46.96**	**46.42**
Nadam	43.89	2.1221	43.89	44.00	43.32
WGD Nadam	**45.27**	**2.0659**	**45.27**	**44.88**	**44.71**
Radam	62.25	3.0041	62.24	62.53	61.86
WGD Radam	**63.28**	**2.9368**	**63.28**	**63.55**	**62.90**

Table 8. Classification results for CIFAR-100 dataset using ResNet50.

Optimizer	Accuracy	Loss	Mean recall	Mean precision	CKS
SGDM	56.34	1.7131	56.34	56.47	56.89
WGD SGDM	**58.70**	**2.1040**	**58.70**	**58.83**	**58.28**
Adam	42.61	2.1868	42.61	42.35	42.03
WGD Adam	**47.00**	**1.9774**	**47.00**	**46.83**	**46.46**
Nadam	30.08	2.7722	30.08	29.14	29.37
WGD Nadam	**45.15**	**2.0368**	**45.14**	**45.46**	**44.59**
Radam	**64.93**	**2.4143**	**64.92**	**65.46**	**64.57**
WGD Radam	64.79	2.3893	64.78	64.99	64.43

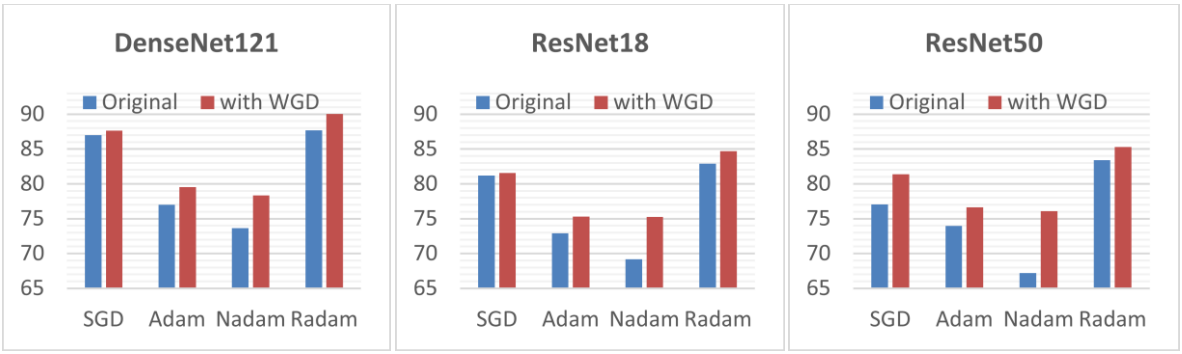

Figure 6. Accuracy (%) of all experiments on Animals-10.

We measured the accuracy and loss of test data after each epoch. The training curves on CIFAR-100 using Radam optimizer are shown in Figures 7 and 8. As shown, in this case, using WGD in Radam will increase the accuracy and decrease the loss.

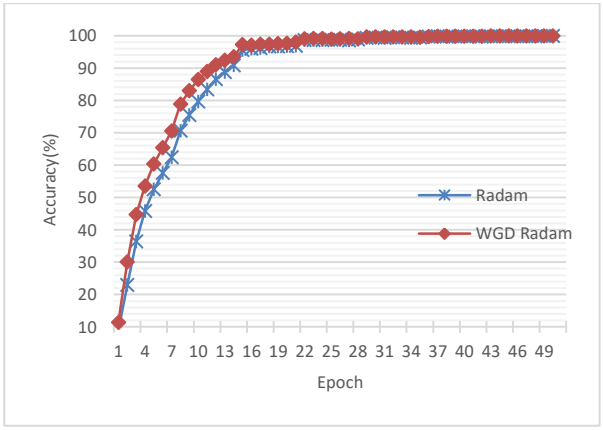

Figure 7. Accuracy of Densenet121 in train data on CIFAR-100.

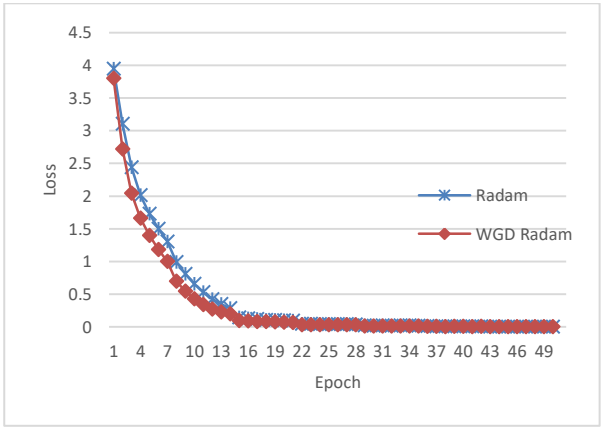

Figure 8. Loss of Densenet121 in train data on CIFAR-100.

The accuracy results of all experiments on CIFAR-100 are shown in Figure 9. As shown, in 11 cases, embedding WGD improves the accuracy.

Using WGD in the Radam decreased the accuracy by about 0.3%.

4.5. Decreasing network entropy

Another effect of the proposed method, i.e. integrating WGD in optimizers, is that it decreases the network entropy. The network entropy can be described using the Optimization Landspace Smoothing (OLS) factors. One of the famous OLS factors is the Lipschitz function [40]. Assume that N is the convolution kernel dimension, e is an N-dimensional unit vector, D is an N × N matrix, and h_t is the WGD. Assume that D is computed as in Equation 13:

$$D = 1 - ee^T \tag{13}$$

We re-write Equation 11 in matrix form:

$$h_t = D\nabla_\theta J \tag{14}$$

The Lipschitz function says that if $\|h_t\|_2^2 \le \|\nabla_\theta J\|_2^2$ then the network optimized using WGD is smoother than the optimizers without WGD. We have:

$$\begin{aligned} \|h_t\|_2^2 &= (h_t)^T h_t = (D\nabla_\theta J)^T (D\nabla_\theta J) \\ &= (\nabla_\theta J)^T D^T D \nabla_\theta J \end{aligned} \tag{15}$$

Using Equation 14, we have:

$$\begin{aligned} &(\nabla_\theta J)^T (1-ee^T)^T (1-ee^T)\nabla_\theta J \\ &= (\nabla_\theta J)^T (1-ee^T)\nabla_\theta J \end{aligned} \tag{16}$$

Finally, we achieve Equation 17:

$$\begin{aligned} \|h_t\|_2^2 &= (\nabla_\theta J)^T (1-ee^T)^T \nabla_\theta J \\ &= (\nabla_\theta J)^T \nabla_\theta J - (\nabla_\theta J)^T ee^T \nabla_\theta J \\ \|h_t\|_2^2 &= \|\nabla_\theta J\|_2^2 - \|e^T \nabla_\theta J\|_2^2 \\ \|h_t\|_2^2 &\le \|\nabla_\theta J\|_2^2 \end{aligned} \tag{17}$$

Equation (17) shows that using WGD in optimizers leads to a better Lipschitz factor.

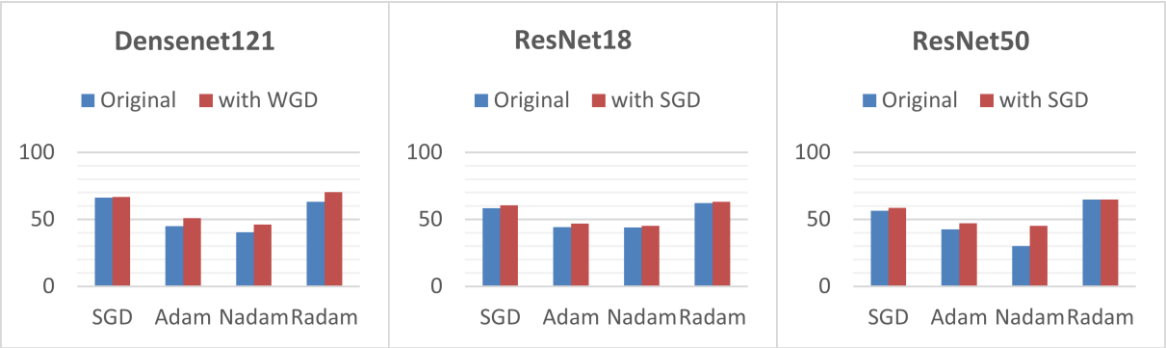

Figure 9. Accuracy (%) of all experiments on CIFAR-100.

Thus the gradients used in the training phase are better, and the proposed method improves the training procedure.

5. Conclusion

In this paper, we introduced WGD, a new updating method for gradient descent based optimizers. Briefly, we removed the mean from the gradient vectors to have zero mean. We evaluated this method by embedding it into four popular optimizers: SGDM, Adam, Nadam, and Radam. Using this method for the training phase of a proposed convolutional-based neural network on two image datasets improved all the classification metrics. In some cases, using the proposed updating method can improve the classification accuracy by up to 9%. Thus using WGD in gradient descent-based optimizers can increase the network training capability. Another benefit of our proposed updating method is improving the stability of the network in the training phase.

References

[1] A. Bordes, X. Glorot, J. Weston, and Y. Bengio, "Joint learning of words and meaning representations for open-text semantic parsing," in *Artificial Intelligence and Statistics*, 2012, pp. 127-135.

[2] W. Ma, W. Ma, S. Xu, and H. Zha, "Pyramid ALKNet for Semantic Parsing of Building Facade Image," *IEEE Geoscience and Remote Sensing Letters*, 2020.

[3] V. Lialin, R. Goel, A. Simanovsky, A. Rumshisky, and R. Shah, "Continual Learning for Neural Semantic Parsing," *arXiv preprint arXiv:2010.07865*, 2020

[4] D. C. Cireşan, U. Meier, and J. Schmidhuber, "Transfer learning for Latin and Chinese characters with deep neural networks," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 2012, pp. 1-6: IEEE.

[5] J. S. Ren and L. Xu, "On vectorization of deep convolutional neural networks for vision tasks," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[6] T. Kaur and T. K. Gandhi, "Deep convolutional neural networks with transfer learning for automated brain image classification," *Machine Vision and Applications*, vol. 31, pp. 1-16, 2020.

[7] I. D. Apostolopoulos and T. A. Mpesiana, "Covid-19: automatic detection from x-ray images utilizing transfer learning with convolutional neural networks," *Physical and Engineering Sciences in Medicine*, p. 1, 2020.

[8] X. Li, Y. Grandvalet, and F. Davoine, "A baseline regularization scheme for transfer learning with convolutional neural networks," *Pattern Recognition*, vol. 98, p. 107049, 2020.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111-3119.

[10] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006, vol. 2, pp. 2169-2178: IEEE. [11] K. Chowdhary, "Natural language processing," in *Fundamentals of Artificial Intelligence*: Springer, 2020, pp. 603-649.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[13] D .Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*, 2012, pp. 3642-3649: IEEE.

[14] O. Badmos, A. Kopp, T. Bernthaler, and G. Schneider" ,Image-based defect detection in lithium-ion battery electrode using convolutional neural networks," *Journal of Intelligent Manufacturing*, vol. 31, no. 4, pp. 885-897, 2020.

[15] X. Gou, L. Qing, Y. Wang, M. Xin, and X. Wang, "Re-training and parameter sharing with the Hash trick for compressing convolutional neural networks," *Applied Soft Computing*, p. 106783, 2020.

[16] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[17] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27-48, 2016.

[18] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533-536, 1986.

[20] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[21] C. Y. Miao, A. Yang, and M. J. Anderson, "Deep Learning Workload Performance Auto-Optimizer," EasyChair2516-2314, 2020.

[22] R. Marcus, P. Negi, H. Mao, N .Tatbul, M. Alizadeh, and T. Kraska, "Bao: Learning to Steer Query Optimizers," *arXiv preprint arXiv:2004.03814*, 2020.

[23] G.-H. Liu, T. Chen, and E. A. Theodorou, "A Differential Game Theoretic Neural Optimizer for Training Residual Networks," *arXiv preprint arXiv:2007.08880*, 2020.

[24] I. Kandel, M. Castelli, and A. Popovič, "Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images," *Journal of Imaging*, vol. 6, no. 9, p. 92, 2020

[25] S. Postalcıoğlu, "Performance Analysis of Different Optimizers for Deep Learning-Based Image Recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 34, no. 02, p. 2051003, 2020.

[26] S. Kim and T.-S. Choi, "Design of Multichannel FIR Filter using Gradient Descent Optimizer for Personal Audio Systems," in *Audio Engineering Society Convention 148*, 2020: Audio Engineering Society.

[27] R. Sutton, "Two problems with back propagation and other steepest descent learning procedures for networks," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, 1986, pp. 823-832.

[28] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145-151,1999.

[29] T. Dozat, "Incorporating nesterov momentum into adam.(2016),"

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[31] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] M. Kögel and R. Findeisen, "A fast gradient method for embedded linear predictive control," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1362-1367, 2011.

[34] L. Liu *et al.*, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[35] P. Efraimidis and P. Spirakis, "Weighted Random Sampling," in *Encyclopedia of Algorithms*, M.-Y. Kao, Ed. Boston, MA: Springer US, 2008, pp. 1024-1027.

[36] K. He, X. Zhang, and S. Ren, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.

[37] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700-4708.

[38] *Animals-10 image dataset*. Available: https://www.kaggle.com/alessiocorrado99/animals10.

[39] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica: Biochemia medica*, vol. 22, no. 3, pp. 276-282, 2012

[40] G. Beliakov, "Smoothing Lipschitz functions," *Optimisation Methods and Software*, vol. 22, no. 6, pp. 901-916, 2007.

نزول گرادیان سفیدشده، یک روش به روزرسانی جدید برای بهینه سازها در شبکه های عصبی عمیق

حسین غلامعلی نژاد[۱،۲*] و حسین خسروی[۲]

[۱] **گروه کامپیوتر، دانشکده فنی، دانشگاه بزرگمهر قائنات، قاین، ایران.**

[۲] **دانشکده برق و رباتیک، دانشگاه صنعتی شاهرود، شاهرود، ایران.**

ارسال ۲۰۲۱/۱۱/۰۸؛ بازنگری ۲۰۲۲/۰۱/۰۶؛ پذیرش ۲۰۲۲/۰۴/۰۵

چکیده:

بهینه سازها اجزای حیاتی شبکه های عصبی عمیق هستند که به روز رسانی وزن را انجام می‌دهند. این مقاله یک روش به روز رسانی جدید را برای بهینه سازهای مبتنی بر نزول گرادیان، به نام نزول گرادیان سفید شده (WGD) معرفی می‌کند. پیاده سازی این روش آسان است و در هر بهینه ساز مبتنی بر نزول گرادیان قابل استفاده است. این روش، زمان آموزش شبکه را به میزان قابل توجهی افزایش نمی‌دهد. روش پیشنهادی، منحنی یادگیری را صاف می‌کند و معیارهای طبقه بندی را بهبود می‌بخشد. به منظور ارزیابی الگوریتم پیشنهادی، ما ۴۸ تست مختلف را بر روی دو مجموعه داده Cifar100 و Animals-10 با استفاده از سه ساختار شبکه شامل densenet121، resnet18 و resnet50 انجام دادیم. آزمایش‌ها نشان می‌دهند که استفاده از روش WGD در بهینه‌سازهای مبتنی بر گرادیان، نتایج طبقه‌بندی را به طور قابل‌توجهی بهبود می‌بخشد. به عنوان مثال، ادغام WGD در بهینه ساز RAdam، دقت DenseNet را از ٪۸۷,۶۹ به ٪۹۰,۰۲ در مجموعه داده Animals-10 افزایش می‌دهد..

کلمات کلیدی: یادگیری عمیق، بهینه ساز، نزول گرادیان سفیدشده، مومنتوم.