Journal of Advances in Computer Research
Quarterly pISSN: 2345-606x eISSN: 2345-6078
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 9, No. 4, November 2018), Pages: 59-71
www.jacr.iausari.ac.ir

A Hybrid Optimization Algorithm for Learning Deep Models

Farnaz Hoseini[1], Asadollah Shahbahrami[✉2], Peyman Bayat[1]
1) Department of Computer Engineering, Rasht Branch, Islamic Azad University, Rasht, Iran
2) Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran
farnazhoseini@iaurasht.ac.ir; shahbahrami@guilan.ac.ir; bayat@iaurasht.ac.ir

Received: 2018/05/06; Accepted: 2018/06/02

Abstract

The learning algorithms require optimization in multiple aspects. Generally, model-based inferences need to solve an optimized problem. In deep learning, the most important problem that can be solved by optimization is neural network training, but training a neural network can involve thousands of computers for months. In the present study, basic optimization algorithms in deep learning were evaluated. First, a performance criterion was defined based on a training data set, which makes an objective function along with an adjustment phrase. In the optimization process, a performance criterion provides the least value for objective function. Finally, in the present study, in order to evaluate the performance of different optimization algorithms, recent algorithms for training neural networks were compared for the segmentation of brain images. The results showed that the proposed hybrid optimization algorithm performed better than the other tested methods because of its hierarchical and deeper extraction.

Keywords: *Deep Learning, Optimization Algorithms, Stochastic Gradient Descent, Momentum, Nestrove, Adam*

1. Introduction

In some aspects, the used algorithms in deep models differ from traditional optimization algorithms. In most of the scenarios related to deep learning, the performance criterion (P) is defined based on the test set and it often can be managed. Therefore, the P criterion is optimized indirectly. In other words, another cost function J (θ) is defined, and improvement of P criterion is expected by its optimization. It should be noted that in pure optimization, the performance criterion and objective function are the same. On the other hand, the other optimization algorithms in deep models are highly dependent on the specific structure of the objective function. Generally, the cost function is formed on a training data set by Equation (1) [1].

$$J(\theta) = \mathbb{E}_{(x,\,y)\sim \hat{p}_{data}} L\left(f\left(x;\theta\right), y\right) \quad (1)$$

In Equation (1), L is the cost of a data sample, θ is performance criterion and f (x; θ) is the output corresponding to the input x. Moreover, $\hat{p}_{data}$ shows the empirical distribution (distribution based on training data), which is replaced with unspecified p_{data}data generator distribution. In supervised learning, y presents the output of objective. It should be noted that in some algorithms studied in the present study, the

topics are presented based on the supervision mode and objective function without adjustment. It is worth pointing out that by adding θ or x parameters to L, and omitting the y parameter, evaluation of the adjustment and unsupervised mode is possible, respectively. In Equation (1), the objective function is defined based on a training data set. The aim of machine learning algorithm is to reduce the hope of generality error by Equation (2) [2].

$$J^{*}(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L\left(f(x;\theta), y\right) \quad (2)$$

The calculated value obtained by Equation 2 is called risk. Here, the emphasis is on the hope calculation based on the actual data generator distribution p_{data}. Assuming that the actual distribution of $p_{data}(x,y)$is known, risk minimization is an optimization action performed by an optimization algorithm, otherwise, by having a training data set, we encounter a deep learning problem [2]. The simplest way to convert a machine-learning problem to an optimization one, the hope calculation is performed based on training data (m is a bias parameter). In other words, the actual distribution of $p_{data}(x,y)$ is replaced with empirical distribution of$\hat{p}_{data}(x,y)$, which is based on a training data set. Therefore, the empirical risk is minimized by Equation (3).

$$\mathbb{E}_{(x,y)\sim \hat{p}_{data}(x,y)}\left[L\left(f(x;\theta), y\right)\right] = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right) \quad (3)$$

The learning process conducted based on minimization of mean training error is called empirical risk minimization. In this condition, the deep learning is similar to normal optimization. Here, instead of direct risk optimization, the empirical risk is optimized, and the same behavioral risks are expected [3]. It should be noted that the empirical risk minimization is prone to overfitting. The models with high capacity easily can memorize the data set. On the other hand, in most cases, minimization of empirical risk is impossible. In other words, most of modern algorithms are performed based on gradient descent, while most of cost functions such as 0-1 loss do not provide useful derivatives (their derivatives are either zero or infinity). It means that in the deep learning field we can use empirical risk minimization, and the quantity with a higher difference with actual value should be optimized. In the present study, the basic algorithms in optimization process were evaluated and used in the training process of neural network through the empirical risk minimization approach. The following sections of the present study are presented as follows: Section 2 introduces the basic algorithms in optimization process. Section 3 presents experimental results on the field of employing the algorithm in brain images segmentation for optimization process in training network. Finally, Section 4 provides the conclusion of the research.

2. The Basic Algorithms in the Optimization Process

In this section, the basic algorithms of optimization are discussed. In order to synchronization and create a common language, we have defined different parameters and variables in Table 1.

Table 1. The meaning of parameters and variables which are used in discussion of optimization algorithms

Variable name	Variable function
ϵ	Learning rate
θ	Performance criterion
$\tilde{\theta}$	Updating performance criterion (θ)
α	Momentum parameter
υ	Speed of movement in parameters space
g	Calculation of gradient
$\hat{g}$	Estimation of gradient vector
$g \odot g$	Calculation of partial derivatives squares
r	Integrating partial derivatives vector
$\hat{r}$	Bias correlation of first order (r)
s	Integrating vector of square partial derivatives
$\hat{s}$	Bias correlation of second order (s)
w	Weighted of neurons
t	Time slot
ρ	The damping rate in the weighted sum
δ	The numerical stability constant
$\Delta\ \theta$	Calculation of learning rate
m	Number of sample in dataset
$x^{(1)}, \ldots, x^{(m)}$	The samples of data set
$y^{(1)}, \ldots, y^{(m)}$	Labels of samples in dataset

2.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) and its variants are the most common optimization algorithm in deep learning. In an SGD, convergence is much faster than the Standard (or Batch) Gradient Descent. Since the update of weights (W) in this algorithm is much higher than the standard one. With an average of the gradient on the small groups which are sampled as Independent Identical Distraction (i.i.d). A non-biased estimation can be obtained from the gradient of the cost function [4]. An important parameter of the SGD is the learning rate (ε). In practice, learning rates over

the time and repetitions are reduced. The reason is that the SGD estimator creates a noise, which does not even disappear even by reaching the local minimum.

Therefore, to achieve convergence, this noise must be tended to zero. However, in the normal gradient, due to the absence of a random element, the real gradient becomes zero by reaching the optimal point and convergence is achieved with a constant learning rate. The most important feature of the SGD optimization algorithm is the independence of the calculation time on the number of educational samples. In this way, convergence is possible even with a large set of data. Under these conditions, it is possible that the model error is placed within a defined range of final test errors before the whole processing data is processed. To study the convergence rate of an optimization algorithm, the measurement of the actual overrun error is common. Excess Bound shows the distance between the current value and the minimum value of the cost function. When the SGD algorithm is applied to a strong and convex problem, the excess bound after K (constant parameter) repetition is at O ($\frac{1}{\sqrt{k}}$) and O ($\frac{1}{k}$), respectively. Theoretically, the descending gradient algorithm has better convergence rates compared to its stochastic version. In learning a machine, pursuing algorithms with a convergence rate faster than O ($\frac{1}{k}$) is inadequate, and faster convergence will correspond to the higher over-fitting [5]. With a large dataset, the SGD algorithm is capable of generating rapid progress at the beginning of the work by calculating the gradient for only a few samples, so that its slow convergence will somehow be compensated. Most of the algorithms described in the following will actually deliver better results; however, they change the constant coefficients in the asymptotic analysis, and will not affect the degree of convergence. Practically, by gradually increasing the categories, it is possible to compromise between the benefits of descending gradient algorithms and a SGD [6]. The pseudo code of SGD algorithm is depicted in algorithm 1.

Algorithm 1. Pseudo code of Stochastic Gradient Descent Algorithm

1 *Enter initial values* (ϵ_k, θ)
2 ***while***
3 $Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ \ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
4 $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathrm{L}\left(f(x^{(i)}, \theta), y^{(i)}\right)$
5 $\theta \leftarrow \theta - \epsilon_k\, \hat{g}$
6 ***End while***

2.2 Momentum Algorithm

Although the SGD is a comprehensive optimization strategy, it works slowly in many cases. The momentum method [7] is designed to accelerate the learning process, especially in facing with small but consistent gradients or high curvature surfaces, and noise gradients. In this method, the running averaging the preceding step is drawn with a damping, and the motion continues in this direction. In this method, the cost function is interpreted as the height of a ground with many ups and downs, which the vector of parameters are attempted to be directed to the lowest height.

The effect of implementing the momentum method is presented in Fig. 1. In this figure, the black arrows indicate direction of gradient and the red path is associated with

the path followed by the momentum algorithm. It is observed that the gradient path tends to show zigzag movements in the valley while the use of a momentum causes the movement direction to tend to the length of the valley.

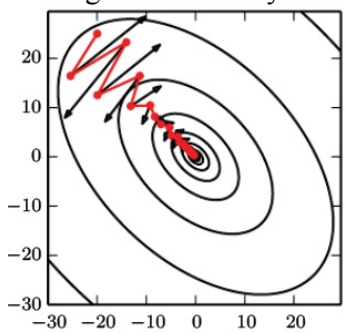

Fig. 1. Modifying the improper condition of implementing the momentum algorithm [19]

In this method, the cost function is interpreted as the height in a ground full of rise and falls and it is attempted to lead a ball, which is the vector of parameters to the lowest height. Considering the relationship between potential energy with the height of U = H (U is potential energy and H is height), the cost function is proportional to potential energy JαU (J is cost function). By random initialization of the parameters, the ball of interest is located at a random point with the initial velocity of zero. Since the force (F) applied to the ball is equal to the potential energy gradient (F = -∇U), the force that is felt by the ball is proportional to the gradient of the cost function (Fα-∇J). Since the relation F = A (A is acceleration) exists between force and acceleration, the acceleration of the ball is proportional to the negative gradient of the cost function (Aα-∇J). In this way, the velocity changes in the time unit is proportional to the negative gradient.

Therefore, in this method, the gradient affects the velocity of the ball directly and the velocity changes the location of the ball indirectly. However, in the conventional random gradient descent method the negative gradient was directly related to spatial variation.

$$\left(\upsilon^{t+1} - \upsilon^{t} = -\nabla J\right) \tag{4}$$

The υ variable in Equation (4) represents the velocity and direction of movement in the space of the momentum method parameters. The running average of the gradients is embedded in this variable. By equalizing the mass unit, the gradient vector can be considered as equivalent to the particle momentum. The α∈ [0, 1] hyper-parameter determines the severity of the previous gradients' damping the updating rule of which is in accordance with Equation (5).

$$\upsilon \leftarrow \alpha\upsilon - \epsilon\nabla_{\theta}\frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)},\theta\right), y^{(i)}\right)), \qquad \theta \leftarrow \theta + \upsilon \tag{5}$$

In this equation, υ aggregates the gradients. As α value is greater than ε, the effect of the previous gradients on the choice of movement direction will be higher. Therefore, the coefficient α plays the role of friction and reduces the energy of the system.

The momentum name is taken from a physical similarity, where the gradient is a force moving a particle in the parameter space in accordance with Newton's law of motion. In the momentum physics, it is equal to mass multiplied by the velocity. In the normal

gradient, the magnitude of each step is equal to the soft multiplication gradient by the learning rate. While the magnitude of the steps here depends on the size and alignment of the previous gradient sequence, the biggest steps are created when the successive gradients are in the same direction. If we assume that all successive gradients are equal and show it with g, then the velocity in the opposite direction of the gradient is incremental and reaches the limit of Equation (6).

$$\frac{\epsilon g}{1-\alpha} \tag{6}$$

Therefore, the meta-parameter effect of the momentum method is better to be considered as the $\frac{1}{1-\alpha}$ factor. In other words, α = 0.9 means that the maximum speed in this method will be 10 times that in the normal gradient method. In practice, 0.5, 0.9 and 0.99 are common values for α meta-parameter. Like the learning rate, α can be variably changed with time. In most cases, work starts with a small amount of α and its value gradually increases. Another point is that gradual decrease of ε value is more important than applying gradual changes in α. In the descending gradient algorithm, we take only one step toward the highest descent, while using momentum, the speed of the particle movement is also controlled. Algorithm 2 presents the pseudo code of momentum algorithm.

Algorithm 2. Pseudo code of Momentum Algorithm

Enter initial values ($\epsilon, \alpha, \theta, \upsilon$)
while
Sampling $\{x^{(1)}, \ldots, x^{(m)}\}$ *from Dataset by using* $\{y^{(1)}, \ldots, y^{(m)}\}$ *Labels*
$\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathrm{L}\left(f(x^{(i)}, \theta), y^{(i)}\right)$
$\upsilon \leftarrow \alpha \upsilon - \epsilon \hat{g}$
$\theta \leftarrow \theta + \upsilon$
End while

2.3 Nesterov Algorithm

The accelerated gradient method of Nesterov [8] is a new method inspired by the momentum algorithm [9]. The update rule for this new method is presented in Equation (7).

$$\upsilon \leftarrow \alpha \upsilon - \epsilon \nabla_\theta \left[\frac{1}{m} \sum_{i=1}^{m} \mathrm{L}\left(f\left(x^{(i)}; \theta + \alpha\upsilon \right), y^{(i)} \right) \right] \tag{7}$$

$$\theta \leftarrow \theta + \upsilon$$

Where, the parameters α and ε have a role similar to the momentum method. The difference between the Nesterov and momentum methods is where the gradient is calculated. As shown in Fig. 2, the gradient is calculated after applying the current velocity contribution. In other words, the Nesterov method can be described as an attempt to modify the gradient calculation location. In this way, there is a look ahead and it is attempted to calculate at the next hop point.

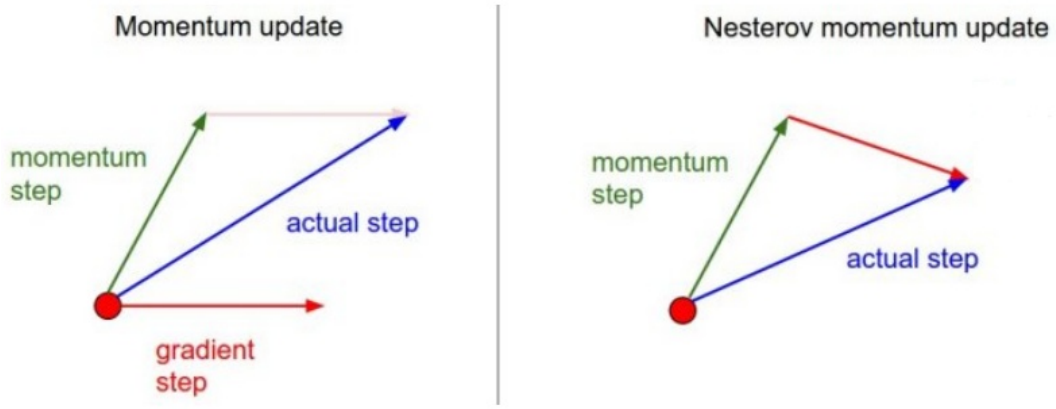

Fig. 2. Calculation of the gradient of the Nesterov algorithm [8]

The difference between the Nesterov and the momentum methods is where the gradient is calculated. The gradient is calculated after applying the current speed. In other words, the Nesterov method can be described as an attempt to correct the gradient calculation location. Accordingly, there is a look-ahead view. It is worth noting that in performing complex analysis, it was found that the Nesterov algorithm in ordinary descending gradient mode yielded the convergence rate in terms of the number of repetitions from O (1/ (k) to $O(1/k^2)$; however, in a SGD, there is no improvement in convergence rates. Algorithm 3 presents the pseudo code of Nestrove algorithm.

Algorithm 3. Pseudo code of Nestrove Algorithm

Enter initial values ($\epsilon, \alpha, \theta, \upsilon$)
while
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ \ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
$\tilde{\theta} \leftarrow \theta + \alpha\, \upsilon$
$\hat{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i \mathrm{L}\left(f(x^{(i)}, \tilde{\theta}), y^{(i)}\right)$
$\upsilon \leftarrow \alpha\, \upsilon - \epsilon\, \hat{g}$
$\theta \leftarrow \theta + \upsilon$
End while

2.4 AdaGrad Algorithm

In this algorithm, each parameter has its own learning rate, and its scale is proportionally changed to the total squared history of the previous partial derivatives [10]. Therefore, the learning rate for parameters with a large partial derivative history is rapidly reduced, and the minimal reductions are experienced for parameters with a small minor derivative history. Totally, the algorithm finds better scrolling speeds when facing gentle slope directions. This algorithm possesses theoretical properties in the domain of convex cost functions. In practice, however, using this method in deep networks and dividing the learning rate on the aggregation of the entire partial derivative history, in some cases, the learning rate is reduced reaching the optimal point. Algorithm 4 presents the pseudo code of AdaGrad algorithm.

Algorithm 4. Pseudo code of AdaGrad Algorithm

Enter initial values (ϵ, θ)
$\delta = 10^{-7}$
$r = 0$
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
while
$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathrm{L}\left(f(x^{(i)}, \theta), y^{(i)}\right)$
$r \leftarrow r + g \odot g$
$\Delta\ \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathrm{r}}} \odot g$
$\theta \leftarrow \theta + \Delta\ \theta$
End while

2.5 RMSProp Algorithm

In this algorithm, the gradient aggregation in the AdaGrad algorithm is performed as moving averaging with exponential weighting. Thus, this algorithm, when used in non-convex cost functions, can forget the history of long-range gradients and, in the middle, easily move downside [11]. In the RMSProp algorithm, compared to AdaGrad, a new meta-parameter is added determining the average length of motion averaging. In practice, the RMSProp algorithm provides optimal results in training deep models, which is nowadays used as one of the most commonly used methods [12]. Algorithm 5 presents the pseudo code of RMSProp algorithm.

Algorithm 5. Pseudo code of RMPSProp Algorithm

Enter initial values (ϵ, θ, ρ)
$\delta = 10^{-6}$
$r = 0$
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
while
$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathrm{L}\left(f(x^{(i)}, \theta), y^{(i)}\right)$
$r \leftarrow \rho r + (1 - \rho) g \odot g$
$\Delta\ \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathrm{r}}} \odot g$
$\theta \leftarrow \theta + \Delta\ \theta$
End while

2.6 Adam Algorithm

The Adam algorithm [13] is another algorithm with an adaptive learning rate. The word “Adam” is derived from the term "adaptive moments". This algorithm can be considered as the combination of two momentum and RMSProp algorithms. This algorithm directly uses the gradient first-order moments with exponential weights. The

most straightforward way to add the momentum to RMSProp is to apply the momentum to the scaled gradient. In this algorithm, bias correction is applied to first and second-order estimates. In RMSProp, a second-order moment estimate is used without a correction factor imposing a bias at the early stages of the learning process. The Adam algorithm is known as an algorithm working persistently against the selection of meta-parameters. Algorithm 6 presents the pseudo code of Adam algorithm.

Algorithm 6. Pseudo code of Adam Algorithm

$\epsilon = 0.001$
$\rho_1 = 0.9, \rho_2 = 0.999$
$\delta = 10^{-8}$
$r = 0, s = 0$
Enter initial value (θ)
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ \ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
while
$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i \mathrm{L}\left(f(x^{(i)}, \theta), y^{(i)}\right)$
$s \leftarrow \rho_1 s + (1 - \rho_1) g$
$r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
$\hat{s} \leftarrow \frac{s}{1 - \rho_1}$
$\hat{r} \leftarrow \frac{r}{1 - \rho_2}$
$\Delta\theta = -\frac{\hat{s}}{\delta + \sqrt{\hat{r}}} \epsilon$
$\theta \leftarrow \theta + \Delta\theta$
End while

2.7 Advantages and disadvantages of basic optimization algorithms

In previous subsections, the basic algorithms in the optimization process for deep learning were briefly explained. In this subsection, the advantages and disadvantages of these algorithms are presented in Table 2.

Table 2. A comparison between different optimization algorithms

Algorithm name	Advantages	Disadvantages
SGD	Independence of training data	Low convergence speed
Momentum	Increasing convergence speed	Inaccurate steps on steep slopes
Nesterov	Precise steps on steep slopes	Accuracy reduction in calculating local gradients
AdaGrad	Creating unique learning rates for each parameter	Creating small steps with constant length
RMSProp	Making steps with variable size	The lack of a precise bias in the gradient
Adam	Bias correction of first order	Large learning rates and step downs

3. The Proposed Hybrid Algorithm

By considering the advantages and disadvantages of the previous algorithms, we aim to introduce a hybrid algorithm which is improved the previous algorithms disadvantages. The main goal of proposed hybrid algorithm is to obtain the best result by emphasizing the mentioned benefits. The different optimization algorithms including SGD, Adam, AdaGrad, and PMSProp were combined in our previous work [14].

Generally, for momentum algorithm, four various type can be considered. In addition, to calculate the momentum, one can use normal gradients or scale gradients, and on the other hand, it is possible to use the previous gradient or Nesterov accelerated gradient (NAG). In the AdaGrad method, twelve different types can be considered. It is possible to use sum or average and different norms. Finally, one can use first or second-order information. Moving average calculation can be performed in four different ways. In the first method, the related coefficient is constant and time-independent. In the second type, the initialization-bias is used. In the third type, it is possible to use actual average instead of moving average. Finally, in the fourth type, it is possible to invent a scheduling for changes and increase the related coefficient by passing time. In addition, it is possible to reduce the overall learning rate based on a scheduling by passing time. Algorithm 7 depicts the pseudo code of proposed hybrid Algorithm.

Algorithm 7. Pseudo code of proposed hybrid Algorithm

Input Dataset (θ_0, α_i)
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ \ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
$\mu_i \sim [0, 1)$
$\nu_i \sim [0, 1)$
$p \sim [1, \infty)$
$\epsilon \approx 10^{-8}$
while
$t = t + 1$
$h_t \leftarrow \{\nabla_\theta L(\theta_{t-1})\ \text{or}\ \mathcal{R}_{u\sim\mathcal{N}(0,1)}(\nabla_\theta L(\theta_{t-1}))\}$
$v_t \leftarrow \nu_t v_{t-1} + \{(1-\nu_t)\ \text{or}\ 1\} h_t^p$
$v_t \leftarrow \lambda_v v_{t-1} + (1-\lambda_v)|g_t|^p$
$\bar{v}_t \leftarrow \sqrt[p]{v_t} + \epsilon$
$g_t \leftarrow \left\{\frac{\alpha_t}{\bar{v}_t}\ \text{or}\ 1\right\} \nabla_\theta L(\theta_{t-1})$
$m_t \leftarrow \mu_t m_{t-1} + \{(1-\mu_t)\ \text{or}\ 1\} g_t$
$\bar{m}_t \leftarrow \{(\mu_{t+1} m_t + \{(1-\mu_t)\ \text{or}\ 1\} g_t)\ \text{or}\ m_t\}$
$\Delta_t \leftarrow \left\{\frac{\alpha_t}{\bar{v}_t}\ \text{or}\ 1\right\} \bar{m}_t$
$\theta_t = \theta_{t-1} - \Delta_t$
End while

A four-layer convolutional neural network architecture had been used to perform a hybrid optimization algorithm. It should be noticed that networks with three and eight layers were tested. However, it was observed that increasing the number of layers from four layers to eight does not increase the accuracy of the model; therefore, the four-layer model is chosen due to less computational overhead. In all layers of the proposed architecture, a convolution and the Rectified Liner Unit (ReLU) are used. In the first layer of the proposed architecture, the pooling is used to keep useful information. In the

second and third layers, the normalization operation is performed to improve the performance of the optimization algorithm. In the fourth layer, the dropout technique is used to reduce weights, and finally, to create the output classes, a softmax layer is added to the convolutional neural network.

4. Experimental Results

In the present study, the employed data is related to BRATS2012, BRATS2013, BRATS2014, BRATS2015 and BRATS2016 competitions consisting four MRI modality such as T1[1], T2[2], T1c[3], and FLAIR[4]. To evaluate performance of the proposed hybrid algorithm, it is necessary to calculate the abundant optimal parameters in the network by forming convolutional network and preparing expected output and output data. Therefore, the cost function is created aiming at approximating the network output to the expected one with multiple repeats using gradient-based optimization methods. The loss function of SGD, Momentum, Adam and the proposed hybrid algorithm were shown in Table 3. The segmentation accuracy that was calculated by using dice coefficient metric for brain images were presented in Table 4. There are two binary maps for each class. One of them was obtained by the model (P) and another one (T) by a group of researchers. Therefore, the Dice criterion is calculated by the Equation (8) through the output of the model. In the other word, this criterion means the ratio of common area to determined average area by model and experts.

$$Dice\left(P,T\right)=\frac{\left|P\wedge T\right|}{\left(\left|P\right|+\left|T\right|\right)/2} \tag{8}$$

Table 3. The loss function for SGD, Momentum, Adam and the proposed hybrid algorithm

Algorithm	Test Loss
SGD	0.0189
Momentum	0.0242
Adam	0.0271
The proposed hybrid algorithm	0.0171

Table 4. The results of comparing the measured accuracy of the proposed hybrid algorithm with some related works for brain images segmentation.

Reference	Segmentation Accuracy					Algorithm
	BRATS 2012	BRATS 2013	BRATS 2014	BRATS 2015	BRATS 2016	
[15,16]	0.81	0.84	-	-	-	SGD+Momentum
[17]	-	0.84	-	0.78	-	SGD+Nesterov
[18]	-	-	0.83	-	-	SGD
#	0.86	0.85	0.81	0.89	0.85	The proposed hybrid Algorithm

5. Conclusion

The results of various theoretical analyses showed existence of performance limitation in each designed optimization algorithm for deep models. The results had no effect on

[1] Spin-Lattice Relaxation
[2] Spin-Spin Relaxation
[3] Spin-Lattice Relaxation Contrasted
[4] Attenuation Inversion Recovery

the empirical use of the network. Furthermore, some theoretical results indicated insolvability of some issues or impossibility of finding a solution for a dimensioned network. However, empirically, it is possible to solve the problem by making the network bigger, leading to an increase in the number of responses in more space of parameters. Moreover, in deep models, the aim is not to find the exact point of minimum objective function, but to find a point to reduce generic error. However, in the theoretical analysis of optimization algorithm capability, achieving this aim is difficult. The results showed that the algorithms with adaptive learning rate are stronger than others. The most common optimization algorithms include SGD, RMS Momentum, Nesrove, AdaGrad, PMSProp, and Adam [19]. Choosing among these methods depends on familiarity with the application of algorithms and the way of their meta-parameters regulation.

References

[1] Judd, J. S. (1988). Neural network design and the complexity of learning (No. CALTECH-CS-TR-88-20). California Inst of Tech Pasadena Dept of Computer Science.

[2] Blum, A., & Rivest, R. L. (1989). Training a 3-node neural network is NP-complete. In Advances in neural information processing systems (pp. 494-501).

[3] Schaul, T., Antonoglou, I., & Silver, D. (2014). Unit tests for stochastic optimization. International Confrence on Learning Representation.

[4] Zhang, T. (2004, July). Solving large scale linear prediction problems using stochastic gradient descent algorithms. Proceedings of the twenty-first ACM international conference on Machine learning (p. 116).

[5] Bousquet, O., & Bottou, L. (2008). The tradeoffs of large scale learning. In Advances in neural information processing systems (pp. 161-168).

[6] Saad, D. (1998). Online algorithms and stochastic approximations. Online Learning, 5.

[7] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-114).

[8] Nesterov, Y. (2013). Introductory lectures on convex optimization: A basic course (Vol. 87). Springer Science & Business Media.

[9] Nesterov, Y. (1983, February). A method of solving a convex programming problem with convergence rate O (1/k2). In Soviet Mathematics Doklady (Vol. 27, No. 2, pp. 372-376).

[10] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5), 1-17.

[11] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

[12] Boureau, Y. L., Chopra, S., & LeCun, Y. (2007, March). A unified energy-based framework for unsupervised learning. In Artificial Intelligence and Statistics (pp. 371-379).

[13] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[14] Hoseini, F., Shahbahrami, A., & Bayat, P. (2018). An Efficient Implementation of Deep Convolutional Neural Networks for MRI Segmentation. Journal of digital imaging, 1-10.

[15] Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., ... & Larochelle, H. (2017). Brain tumor segmentation with deep neural networks. Medical image analysis, 35, 18-31.

[16] Havaei, M., Guizard, N., Larochelle, H., & Jodoin, P. M. (2016). Deep learning trends for focal brain pathology segmentation in MRI. In Machine Learning for Health Informatics (pp. 125-148).

[17] Pereira, S., Pinto, A., Alves, V., & Silva, C. A. (2016). Brain tumor segmentation using convolutional neural networks in MRI images. IEEE transactions on medical imaging, 35(5), 1240-1251.

[18] Dvořák, P., & Menze, B. (2015, October). Local Structure Prediction with Convolutional Neural Networks for Multimodal Brain Tumor Segmentation. In International MICCAI Workshop on Medical Computer Vision (pp. 59-71).

[19] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.