Journal of Digital Imaging (2019) 32:105–115
https://doi.org/10.1007/s10278-018-0107-6

AdaptAhead Optimization Algorithm for Learning Deep CNN Applied to MRI Segmentation

Farnaz Hoseini[1] · Asadollah Shahbahrami[2] · Peyman Bayat[1]

Published online: 23 July 2018

Abstract

Deep learning is one of the subsets of machine learning that is widely used in artificial intelligence (AI) field such as natural language processing and machine vision. The deep convolution neural network (DCNN) extracts high-level concepts from low-level features and it is appropriate for large volumes of data. In fact, in deep learning, the high-level concepts are defined by low-level features. Previously, in optimization algorithms, the accuracy achieved for network training was less and high-cost function. In this regard, in this study, AdaptAhead optimization algorithm was developed for learning DCNN with robust architecture in relation to the high volume data. The proposed optimization algorithm was validated in multi-modality MR images of BRATS 2015 and BRATS 2016 data sets. Comparison of the proposed optimization algorithm with other commonly used methods represents the improvement of the performance of the proposed optimization algorithm on the relatively large dataset. Using the Dice similarity metric, we report accuracy results on the BRATS 2015 and BRATS 2016 brain tumor segmentation challenge dataset. Results showed that our proposed algorithm is significantly more accurate than other methods as a result of its deep and hierarchical extraction.

Keywords Deep learning · Convolutional neural networks · MRI segmentation · Deep convolutional neural networks · Optimization algorithm

Introduction

Today, machine learning is used as an influential tool for solving artificial intelligence (AI) problems. However, this success depends on involving an optimal representation of raw data, and various learning algorithms do not provide satisfactory performance when faced with poorly represented data. Extracting a feature representation is often done manually and requires special field knowledge [1]. In addition, this process is complicated, tiring, time-consuming, and it is often not generalizable to other areas. Recently, to solve these problems, much attention has been paid to deep learning algorithms that perform feature extraction automatically and optimally. These algorithms learn the proper features that are hidden in raw data. For example, models of deep belief networks (DBN), sparse auto-encoders, convolution neural networks (CNN), and deep boltzmann machines (DBM) were successfully used in machine vision, audio processing, natural language processing, and information retrieval and to automatically extract the features [2]. The CNN is one of the more successful techniques in deep learning, which is inspired by the results of scientific studies of the visual field of the animal brain [3]. These networks typically consist of three different types of layers, which are arranged in succession and repeated frequently [4]. These three layers are the convolution layer, the nonlinear function layer, and the pooling layer. There are several different types of cores on the input in the convolution layer, and the important thing is that these cores are applied dispersedly and the parameters are shared; therefore, the computational burden is significantly reduced compared to normal neural networks. Using a nonlinear layer leads to nonlinear decisions, and consequently, the ability of the learner machine increases. In the pooling layer, in a certain range of features, the characteristics are summarized in different ways.

✉ Asadollah Shahbahrami
shahbahrami@guilan.ac.ir

Farnaz Hoseini
farnazhoseini@iaurasht.ac.ir

Peyman Bayat
bayat@iaurasht.ac.ir

[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

Hence, the tolerance of machine to noise and low variation is increased in addition to reducing the volume of calculations [5].

Deep learning algorithms need to be optimized from a different perspective. For example, generally, model-based inference requires solving an optimization problem. In deep learning, the most important and most comprehensive problem solved through optimization is network training. Learning a network may assign hundreds of computers for some months. In the process of training the CNN to avoid the over-fitting, many educational data are required. For this purpose, using the datasets of a related domain, the initial training is performed and then, using the data of the desired domain, a precise fine-tuning is conducted. Essentially, in the learning process, due to a large number of parameters and training data, parallel processing techniques are used to deal with high computational and spatial complexities [6]. In recent studies in order to train these networks, various algorithms were used to solve the optimization problem. Unfortunately, there is no general agreement regarding the most appropriate optimization algorithm. In a significant study [7], a comparison was made between large numbers of optimization algorithms versus various tasks. The result of this study shows that algorithms that have an effective learning rate are generally more robust than others. The most commonly used optimization algorithms are as follows: stochastic gradient descent (SGD) [8], Momentum [9], Nesterov [10], AdaGrad [11], RMSProp [12], and Adam [13]. Selecting these methods depends more on the familiarity with the application of the algorithms and how to arrange their meta-parameters. Improving optimization algorithms is not always the best way to improve the optimization process. In deep models, the model is designed so that the optimization can be made more easily. In other words, with selecting an appropriate model family, simple optimization conditions can be provided using an algorithm from the 80th decade, such as a SGD and a combination of momentum.

In this study, a new optimization algorithm based on deep learning is presented in order to optimize learning CNNs. While creating a CNN and preparing the expected input and output data, it is necessary to calculate the optimal values of the numerous parameters in the network. For this purpose, a special cost function is proposed and it is attempted to close the output of the network to the expected output, using adaptive gradient based optimization methods with various switches. In this method, the gradient direction of the cost function moves in the space of the parameters and the optimal value is obtained. In optimization, one of the important methods for obtaining optimal parameters is the use of gradient-based methods. In this method, the gradient direction of the cost function moves in the space of the parameters and the optimal value is obtained. The following sections of the present study are presented as follows: section "Basic Optimization Algorithms" introduces the basic optimization algorithms. Section "Previous Deep Learning Approaches to Brain Tumor Segmentation" introduces previous deep learning approaches to brain tumor segmentation. Section "Experimental Results" presents experimental results on the field of employing the proposed optimization algorithm in brain images segmentation. Our discussion is shown in section "Discussion". Finally, sections "Conclusions" and "Future Works" provide the conclusion of the research and future works.

Basic Optimization Algorithms

In this subsection, the basic algorithms for optimization used in deep learning are studied in the context of the considered topic [14]. In order to synchronize and create a common language, we have defined different parameters and variables in Table 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 [8]. 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

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
$\vartriangle\theta$	Calculation of learning rate
m	Number of sample in dataset
O	Maximum value
$x^{(1)}, \ldots, x^{(m)}$	The samples of data set
$y^{(1)}, \ldots, y^{(m)}$	Labels of samples in dataset

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 convex problem, the excess bound after k repetition is in the range [O ($\frac{1}{\sqrt{k}}$), O ($\frac{1}{k}$)]. 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 overfitting [15]. 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 [16]. The pseudo code of SGD algorithm is depicted in algorithm 1.

Algorithm 1 Pseudo code of Stochastic Gradient Descent algorithm

```
Enter initial values ($\epsilon_k$, θ)
while
    Sampling $\{x^{(1)}, \dots, x^{(m)}\}$ from Dataset by using $\{y^{(1)}, \dots, y^{(m)}\}$ Labels
    $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$
    $\theta \leftarrow \theta - \epsilon_k \hat{g}$
End while
```

Momentum Algorithm

Although the SGD is a comprehensive optimization strategy, it works slowly in many cases. The momentum method [9] is designed to accelerate the learning process, especially in facing with small but consistent gradients or high curvature surfaces, and noise gradients. Totally, the algorithm finds better scrolling speeds when facing with gentle slope directions. In this method, the running average is done with a damping, and the motion continues in this direction. 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 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 Eq. (1).

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

Therefore, the meta-parameter effect of the momentum method is better to be considered as the $\frac{1}{1-\alpha}$ factor. In other words, $\alpha = 0.9$ 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
$\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$
$\upsilon \leftarrow \alpha\,\upsilon - \epsilon\,\hat{g}$
$\theta \leftarrow \theta + \upsilon$
End while
```

Nesterov Algorithm

The accelerated gradient method of Nesterov [17] is a new method inspired by the momentum algorithm [10]. The update rule for this new method is presented in Eq. (2).

$$\begin{aligned}&\upsilon \leftarrow \alpha\,\upsilon - \epsilon\,\nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)}; \theta + \alpha\upsilon\right), y^{(i)}\right)\right]\\ &\theta \leftarrow \theta + \upsilon\end{aligned} \qquad (2)$$

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. 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 $\{x^{(1)}, \ldots, x^{(m)}\}$ from Dataset by using $\{y^{(1)}, \ldots, y^{(m)}\}$ Labels
$\tilde{\theta} \leftarrow \theta + \alpha\,\upsilon$
$\hat{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}, \tilde{\theta}), y^{(i)})$
$\upsilon \leftarrow \alpha\,\upsilon - \epsilon\,\hat{g}$
$\theta \leftarrow \theta + \upsilon$
End while
```

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 [11]. 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. In this way, the learning rate for parameters with a large partial derivative history is rapidly reduced and the minimal updates are experiences for parameters of the model.

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* $(\epsilon, \theta)$
$\delta = 10^{-7}$
$r = 0$
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
**while**
    $g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i} L\big(f(x^{(i)}, \theta), y^{(i)}\big)$
    $r \leftarrow r + g \odot g$
    $\triangle\,\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$
    $\theta \leftarrow \theta + \triangle\,\theta$
**End while**
```

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 [12]. 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 good results in training deep models, which is nowadays used as one of the most commonly used methods [18]. Algorithm 5 presents the pseudo code of RMSProp algorithm.

Algorithm 5 Pseudo code of RMPSProp algorithm

```
*Enter initial values* $(\epsilon, \theta, \rho)$
$\delta = 10^{-6}$
$r = 0$
$Sampling\ \{x^{(1)}, \dots, x^{(m)}\}\ from\ Dataset\ by\ using\ \{y^{(1)}, \dots, y^{(m)}\}\ Labels$
**while**
    $g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i} L\big(f(x^{(i)}, \theta), y^{(i)}\big)$
    $r \leftarrow \rho r + (1 - \rho) g \odot g$
    $\triangle\,\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$
    $\theta \leftarrow \theta + \triangle\,\theta$
**End while**
```

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. The Adam algorithm is known as an algorithm that hyper parameter selection is less important. There are certain differences between our proposed algorithm and Dozat [19] previous work. We will use multiple switches to test the combination of different tricks and the hyper-parameter p is added to test other moments in normalization of gradient, instead of just second moment. In this paper, the algorithm is implemented so that the each update is done in much less time that Adam's method at the cost of a little losing of mathematical precision. 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* $(\theta)$
*Sampling* $\{x^{(1)}, \ldots, x^{(m)}\}$ *from Dataset by using* $\{y^{(1)}, \ldots, y^{(m)}\}$ *Labels*
**while**
    $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$
    $s \leftarrow \rho_1 s + (1 - \rho_1) g$
    $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
    $\hat{s} \leftarrow \frac{s}{1 - \rho_1}$
    $\hat{r} \leftarrow \frac{r}{1 - \rho_2}$
    $\Delta\theta = -\frac{\hat{s}}{\delta + \sqrt{\hat{r}}} \epsilon$
    $\theta \leftarrow \theta + \Delta\theta$
**End while**
```

Previous Deep Learning Approaches to Brain Tumor Segmentation

In this subsection, the articles of recent years are discussed using the basic optimization algorithms regarding the segmentation of medical images. Zikic et al. [20] proposed an architecture for the segmentation of the magnetic resonance images (MRI) of the brain consist of two nested windows with dimensions d × d and d′' × d′, where d < d′, and labeled with the help of the external window of the inner pixels of the internal window. In this work, the size of the windows is set to the extent that optimal results are obtained. In this architecture, instead of determining the label of each pixel, for each region, the probability distribution is provided, segmentation is obtained by combining them. In other architecture, using the first stage CNN, Long et al. [21] obtained the preliminary estimation of fragmentation. Then, it is given to the second stage CNN as the input, and the segmentation of the first stage is improved. In an architectural model for segmentation of the MR image of the brain, Dvořák et al. [22] used the idea of local structural estimation and correlation of close-up pixel labels. This model was created using k-means clustering of visual words. Pereira et al. [23] proposed a multi-path architectural model for fragmenting the brain MR image produces windows of different dimensions that have a different degree of the locality, which connect before the output layer. In this model, leaky-ReLUs and 3×3 convolutional kernels are used to prevent over-fitting, and data normalization is carried out as a continuous pre-processing phase. On the other hand, the architecture of the model is fixed and no attempt was made to optimize the architecture. In another architectural model for segmentation of the MR image of the brain, Havaei et al. [24, 25] used two ways to learn local features and total features. This model uses two-stage learning to deal with the imbalance problem in the data set. While these precautions had an extensive improvement over the previous traditional methods since further improvements in the performance accuracy and optimization of the training process are important. We already used the DCNN model to improve segmentation of MR images. The model was evaluated on the public BRATS 2016 dataset with patch-based approach for segmentation, resulting in dice similarity metric 0.90 for complete, 0.85 for core, and 0.84 for enhancing regions [26].

The Proposed Optimization Algorithm

This proposed optimization algorithm for learning DCNN is based on a combination of Nesterov and RMSProp techniques that we name it AdaptAhead. Initially, the algorithm constants are set before entering the main loop. In this algorithm, ϵ is added for numerical stability and avoids the denominator in line 12 to be zero. λ_m is the first-norm decaying rate, which is set to 0.9 to create a relatively small sliding window for calculating the average norm. λ_v is a second-norm decaying rate which is set to 0.999 to create a relatively long sliding window for calculating the average norm. Then, the initial values are given to the parameters or the main variables of the model. The learning rate in the proposed algorithm is initially set to 0.01 and then changed using annealing. In other words, in repeating the algorithm, when the validation error rate flattens, the rate of learning is multiplied by 0.1. p is the norm number of the algorithm and is set at line 4. The values for this hyper-parameter are considered as 1, 2, and∞, which corresponds to norm-1, Euclidean norm, and max-norm are set by switch 1(sw1). In empirical observations, for each of these values, the behavior of the algorithm is evaluated. In line 5, the network weights are set to Gaussian random values with a mean of zero and a 2/N variance. In this relation, N is the number of corresponding neuron inputs. The algorithm continues as long as the error rate in the validation data is decreasing. In line 9, switch 2 (sw2) determines calculating gradients whether in the normal or in the Nesterov method. In line 10, the first moment is calculated which actually calculates the gradient in the sliding window mode. In the 11th line, the p-norm or p-th moment is calculated by the sliding window method. In line12, switch 3 (sw3) determines that the learning rate works whether by applying the calculated norm in line 11 in an adaptive manner, or in the normal manner based on Nesterov method. In this way, by adjusting the three hybrid switches, a total of eight different methods are examined. It is noticeable that in Algorithm 7, it is observed that by disabling adaptive learning rate (switch 3), the norm type (switch 1) will be affectless.

Algorithm 7 The Proposed Optimization algorithm

```
1   Input Dataset (N examples)
2   $\epsilon \leftarrow 10^{-8}, \lambda_v \leftarrow 0.999, \lambda_m \leftarrow 0.9$
3   $\alpha_0 = 0.01$
4   $p = or_{sw1}\{1, 2, \infty\}$
5   $\theta_0 \leftarrow N(0, \sqrt{2/N})$
6   $t, m_0, v_0 \leftarrow 0$
7   while (Reduce Error in Validation Data)
8       $t = t + 1$
9       $g_t \leftarrow \nabla_\theta L(\theta_{t-1} - \alpha_t \lambda_m m_{t-1} or_{sw2}\{0, 1\})$
10      $m_t \leftarrow \lambda_m m_{t-1} + (1 - \lambda_m) g_t$
11      $v_t \leftarrow \lambda_v v_{t-1} + (1 - \lambda_v)|g_t|^p$
12      $\Delta_t \leftarrow \alpha_t m_t or_{sw3}\left\{\frac{1}{\sqrt[p]{v_t} + \epsilon}, 1\right\}$
13      $\theta_t = \theta_{t-1} - \Delta_t$
14  End while
```

Table 2 Architecture of the proposed DCNN model

Layer	Type	Maps	Map size	Kernel size	Stride	Padding	Activation
out	Fully connected	–	5	–	–	–	Softmax
F7	Fully connected	–	100	–	–	–	ReLU
F6	Fully connected	–	5670	–	–	–	ReLU
C5	Convolution	70	9 × 9	3 × 3	1	1	ReLU
C4	Convolution	60	9 × 9	3 × 3	2	1	ReLU
C3	Convolution	50	17 × 17	3 × 3	1	1	ReLU
C2	Convolution	70	17 × 17	3 × 3	2	1	ReLU
C1	Convolution	50	33 × 33	3 × 3	1	1	ReLU
In	Input	4	33 × 33	–	–	–	–

Experimental Results

In this section, evaluation of the proposed optimization algorithm is presented.

Datasets and Evaluation Criteria

The data used in the study were extracted from the BRATS2015[1] and BRATS2016[2] Challenge, including four MRI modalities named spin-lattice relaxation (T1), spin-spin relaxation (T2), spin-lattice relaxation contrasted (T1c), and attenuation inversion recovery (FLAIR). The dataset includes 230 brain images. In order to prevent over-fitting in the learning of multi-million parameters of this network, the data augmentation technique was used in the dataset. In general DCNNs, there are usually three input channels, while the input images in this model are four-channel. In the existing training DCNNs, there are usually three input channels requiring the adjustment of meta-parameters such as learning rate, small size of batches, and weight decay rate. To train the proposed model, we extracted around 880,000 patches from BraTS 2015 and BraTS 2016 images. Then, we separated 20% of the data for testing and 10% for validation sets, when training batch size of 128 is used for each iteration. By the way, the BraTS competition organizers hide these data to prevent competitor over fitting, so we have extracted the dataset based on BraTS Original images. However, the input images in this model are four-channel. Finally, the DCNN classifies each voxel according to a dangerous level in five different classes. For each class, there are two binary maps, one obtained by the model (P) and the other by the consensus of experts (T) available in the dataset. Therefore, the dice similarity coefficient metric is calculated according to Eq. (3) using the model output. The criterion of correctness for segmentation of MR images usually is presented as dice similarity. But to calculate the accuracy, Eq. (4) can be used. In the dice similarity metric, P represents the model predictions, T represents the ground truth labels, and (A) is all the binary map.

$$\mathrm{Dice}(\mathrm{P},\mathrm{T}) = \frac{|\mathrm{P}\wedge\mathrm{T}|}{(|\mathrm{P}|+|\mathrm{T}|)/2} \tag{3}$$

$$\mathrm{Accuracy}\ \ (\mathrm{P},\mathrm{T}) = \frac{|\mathrm{P}\wedge\mathrm{T}|+|(\mathrm{A}-\mathrm{P})\wedge(\mathrm{A}-\mathrm{T})|}{|\mathrm{A}|} \tag{4}$$

Platform and Results

The experimental process was carried out using a NVIDIA GeForce series GeForce GTX 1080 Ti. To evaluate the algorithm and the proposed model, using the BRATS dataset, the segmentation of the four-dimensional MR images was performed and the dice similarity metric was used to compare the performance. The DCNN model used in this experiment is according to Table 2. Using this architecture instead of U-Net is for performance purposes. The simplicity and the representational bottleneck in the proposed network in comparison with U-Net cause the computational and memory costs decrease significantly. As it is observed, in this model, the dimensional reduction was not used in the pooling and the convolution with step 2 was used instead. As it is observed, in this model, the dimensional reduction was not done using pooling and the convolution with stride 2 was used instead. Hence, in addition to reducing the number of dimensions, the risk of discarding critical information is partly avoided. Moreover, in all layers, 3 × 3 convolutions were used, which leads to a reduction in the number of model parameters. Additionally, a batch normalization technique was used in the convolution layer between the output of convolution and the nonlinear function of the ReLU. This helps to improve the optimization process and improves the speed and accuracy of the operation. In two fully connected layers of F6 and F7, dropout and a maintenance ratio of 50% were used to enhance the generalizability of the model. Output neurons in the last convolutional layer are connected to a fully connected layer with 150 neurons. A second fully connected layer with five

[1] https://www.smir.ch/BRATS/Start2015
[2] https://www.smir.ch/BRATS/Start2016

Table 3 The results of the accuracy obtained from the proposed optimization algorithm for selecting different switches

Exp. No.	Switches States			Accurancy $= \frac{\lvert P\wedge T\rvert + \lvert (A-P)\wedge(A-T)\rvert}{\lvert A\rvert}$
	SW1(Norm)	SW2(Nestrov)	SW3(Adaptive)	
(1)	–	Yes	No	85.1
(2)	–	No	No	83.5
(3)	1	Yes	Yes	87.3
(4)	1	No	Yes	86.1
(5)	2	Yes	Yes	91.1
(6)	2	No	Yes	88.2
(7)	∞	Yes	Yes	86.4
(8)	∞	No	Yes	85.5

neurons and a softmax layer is finally added to the model in order to create five output classes and define the target function. This layer divides each voxel (volume pixel) into five output classes. It should be noted that while this process stretches out the convergence process of the model, however implicitly, creates the regularizing effect of aggregation of multiple models. The end of training session is detected using early stopping. In other words using a validation set that is separated from training set, the accuracy of the model is tested during training. If the validation accuracy has no improvements for two epochs, the training session is terminated.

In the Table 3 corresponding to the selection of each switch in the proposed optimization algorithm of the present paper, the results obtained from the training of the DCNN model are presented.

We have compared our algorithm with the algorithms used in similar problems that have worked on the same dataset. The three switches have eight combinations that some of them is the same as known algorithms. As is mentioned in Table 3, using 2-norm in calculating the moment and applying the Nesterov technique in calculating the gradient resulted in the best outcomes, indeed accuracy of 91.1 percentage. On the other hand, it is observed that the failure to use the rate of learning and the technique of Nesterov in calculating the gradient leads to failure to reach the optimal point. Another noteworthy point is that the use other norms in calculating the moment led to a slight decrease in the final accuracy.

Figures 1 and 2 show the presence of noise fluctuations in the loss function graph and the accuracy chart of the training set due to the small randomized behaviors of the classes. The accuracy chart of the test set model in Fig. 2 shows that the optimization process was convergent.

Table 4 depicts the accuracy and loss function on the BRATS dataset of using the proposed optimization algorithm in comparison to some related works. Comparisons in this table are mostly based on the architecture of referenced networks. The optimization methods that are listed in this table are given according to the associated network for completeness.

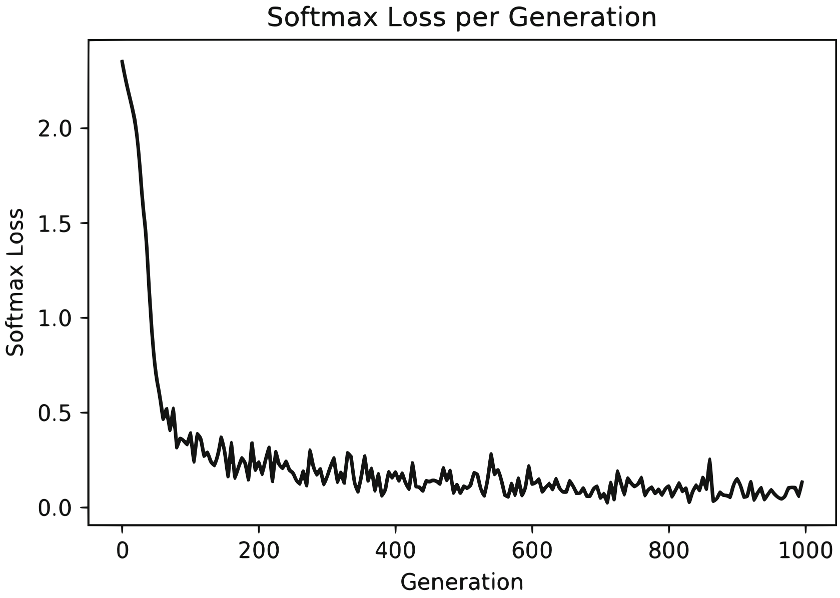

Fig. 1 The loss function changes in terms of the number of repetitions × 10

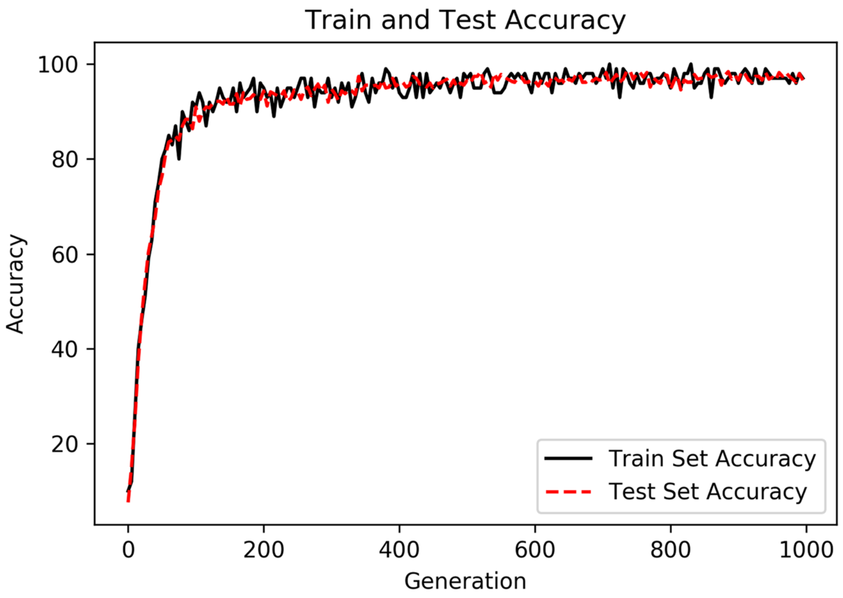

Fig. 2 The model accuracy in terms of the number of repetitions × 10

Experiences show that changing optimization method affects mostly the training time not the final accuracy of the network. As the table's last row reveals the proposed optimization algorithm accuracy is more than other previous techniques.

Discussion

In this paper, we proposed AdaptAhead optimization algorithm for deep CNNs that presented in Algorithm 7. As is mentioned in experimental results section, the proposed model is tested in the context of MRI image segmentation according to accuracy measure of Eq. 4. According to Table 3, using L_2-norm in calculating the moment and applying the Nesterov technique in calculating the gradient delivers the best outcomes, i.e., accuracy of 91.1 percentage. On the other hand, it is observed that the failure to use the adaptive learning rate and the technique of Nesterov in calculating the gradient leads to failure to reach the optimal convergence point. This phenomenon is due to Nesterov gradient and adaptive nature of learning rate. Another noteworthy point is that using other kinds of norm in calculating the moment led to a little decrease in the final accuracy. This is because of the nature of L_2-norm that considers all dimensions but with more emphasis on larger dimensions.

The accuracy vs. iterations in Fig. 2 visually demonstrate the good convergence behavior of the proposed optimization algorithm. During the process of learning, the coefficient α_t in Algorithm 7 is deceased linearly with iterations and consequently convergence is guaranteed.

As is presented in Table 4, the accuracy of the proposed model is compared with related works on BRATS dataset. It is clear that our proposed algorithm demonstrates better accuracy. This accounts for the ability of the algorithm to escape from local minima, better estimation of gradient due to look ahead nature of Nesterov method and learning rate adaptation according to the slope of loss function.

Generally speaking, we can say our proposed optimization algorithm is able to attain better convergence points in

Table 4 The results of comparing the measured dice similarity metric and loss function of the proposed optimization algorithm with some related works for MRI segmentation

Reference	Dataset http://www.braintumorsegmentation.org/	Dice similarity coefficient metric	Loss function	Methods
[24, 25]	BRATS 2012 BRATS 2013	0.81 0.84	average negative log-probability over random subset of patches	SGD + Momentum
[23]	BRATS 2013 BRATS 2015	0.84 0.78	Categorical Cross-entropy	SGD + Nesterov
[22]	BRATS 2014	0.83	Cross-entropy over patches	SGD
#	BRATS 2015 BRATS 2016	0.89 0.85	Cross-entropy over patches	The proposed optimization Algorithm

solution space due to its better gradient approximation and its better adaptation according to varying nature of complex loss surface of deep CNNs.

Conclusions

Deep learning as one of the branches of the machine vision was of interest to researchers since many years ago. CNNs are used as one of the techniques used in deep learning to extract the features automatically. In this study, an analysis of an optimal learning algorithm for the training of a DCNN model was considered. The purposed optimization algorithm was to increase the accuracy and reduce the loss function in the model training process. For this purpose, in the model training process, several simulations of based optimization algorithms were presented to test the loss test compared to the proposed optimization algorithm. Implementing this algorithm based on the BRATS dataset and comparing the accuracy obtained with the algorithms presented in other studies indicate that this algorithm is more accurate than other optimization algorithms.

Future Works

To continue the study, we are going to employ the proposed optimization algorithm on other image processing tasks, such as object detection and recognition. Also, we intend to analyze the sensitivity of the attained results to the architecture and hyper-parameters of the model. For user-friendly processing interfaces, it is desired to develop them with less effort. Because of the limited memory space and the number of parallel GPUs, the use of higher-volume data is difficult to handle. With the advancement of GPUs and the use of libraries that distribute processing across multiple GPUs and multiple machines, this problem can be addressed. In the near future, we look forward to seeing improvements in the programmability and generality of future GPU architectures. The decreased computation time can be immediately attributed to the parallel environment.

References

1. LeCun Y, Bengio Y, Hinton G: Deep learning. Nature 521(7553): 436–444, 2015
2. Schmidhuber J: Deep learning in neural networks: An overview. Neural networks 61:85–117, 2015
3. Krizhevsky A, Sutskever I, Hinton GE: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems:1097–1105, 2012
4. Karpathy A, Toderici G, Shetty S, Leung T, Sukthankar R, Fei-Fei L: Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, (pp. 1725–1732).
5. Sainath TN, Mohamed AR, Kingsbury B, Ramabhadran B: Deep convolutional neural networks for LVCSR. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on IEEE. 2013, pp. 8614–8618
6. Vincent P, Larochelle H, Bengio Y, Manzagol PA: Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning. ACM. 2008pp. 1096–1103
7. Schaul T, Antonoglou I, Silver D: Unit tests for stochastic optimization. arXiv preprint arXiv:1312.6055, 2013
8. Zhang T: Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning. ACM. 2004, p. 116
9. Sutskever I, Martens J, Dahl G, Hinton G: On the importance of initialization and momentum in deep learning. In International conference on machine learning, 2013, (pp. 1139–114)
10. Nesterov Y: 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), 1983
11. Polyak BT: Some methods of speeding up the convergence of iteration methods. USSR Comput Math Math Phys 4(5):1–17, 1964
12. Duchi J, Hazan E, Singer Y: Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12: 2121–2159, 2011
13. Kingma D, Ba J: Adam: A method for stochastic optimization. In 3rd International Conference for Learning Representations, 2014, (pp. 1–13)
14. Hoseini F, Shahbahrami, Bayat P: A hybrid optimization algorithm for learning deep models. Journal of Advances in Computer Research 9(4):1–13, 2018
15. Bottou L, Bousquet O: The tradeoffs of large scale learning. In International Conference of Advances in neural information processing systems, 2008, (pp. 161–168)
16. Saad D,Solla, SA: Exact solution for on-line learning in multilayer neural networks. Phys Rev Lett 74(21):4337, 1995
17. Nesterov Y: Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J Optim 22(2):341–362, 2012
18. Huang FJ, Boureau YL, LeCun Y: Unsupervised learning of invariant feature hierarchies with applications to object recognition. In International Conference of Computer Vision and Pattern Recognition, 2007, (pp. 1–8)
19. Dozat T: Incorporating Nesterov momentum into Adam, 2016
20. Zikic D, Ioannou Y, Brown M, Criminisi A: Segmentation of brain tumor tissues with convolutional neural networks. In International Conference of Proceedings MICCAI-BRATS, 2014, (pp. 36–39)
21. Long J, Shelhamer E, Darrell T: Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3431–3440), 2015
22. Dvořák P, Menze B: Local Structure Prediction with Convolutional Neural Networks for Multimodal Brain Tumor Segmentation. In International MICCAI Workshop on Medical Computer Vision (pp. 59–71). Springer International Publishin, 2015
23. Pereira S, Pinto A, Alves V, Silva CA: Brain tumor segmentation using convolutional neural networks in MRI images. IEEE Trans Med Imaging 35(5):1240–1251, 2016
24. Havaei M, Davy A, Warde-Farley D, Biard A, Courville A, Bengio Y, Pal C, Jodoin PM, Larochelle H: Brain tumor segmentation with deep neural networks. Medical image analysis 35:18–31, 2017
25. Havaei M., Guizard N., Larochelle H., Jodoin PM. (2016) Deep learning trends for focal brain pathology segmentation in MRI. In: Holzinger A Ed. Machine Learning for Health Informatics. Lecture Notes in Computer Science (Vol. 9605). Cham, Springer International Publishing, 2016
26. Hoseini F, Shahbahrami A, Bayat P: An efficient implementation of deep convolutional neural networks for MRI segmentation. J Digit Imaging 1–10, 2018. https://doi.org/10.1007/s10278-018-0062-2