Comparative study of optimization techniques in deep learning: Application in the ophthalmology field.

AATILA Mustapha, LACHGAR Mohamed and KARTIT Ali
LTI laboratory, ENSA, National Road of Azemmour N°1, ELHAOUZIA BP: 1166, Chouaib Doukkali University, El Jadida 24002, Morocco
mu.aatila@gmail.com, lachgar.m@gmail.com, alikartit@gmail.com

Abstract. The optimization is a discipline which is part of mathematics and which aims to model, analyse and solve analytically or numerically problems of minimization or maximization of a function on a specific dataset. Several optimization algorithms are used in systems based on deep learning (DL) such as gradient descent (GD) algorithm. Considering the importance and the efficiency of the GD algorithm, several research works made it possible to improve it and to produce several other variants which also knew great success in DL. This paper presents a comparative study of stochastic, momentum, Nesterov, AdaGrad, RMSProp, AdaDelta, Adam, AdaMax and Nadam gradient descent algorithms based on the speed of convergence of these different algorithms, as well as the mean absolute error of each algorithm in the generation of an optimization solution. The obtained results show that AdaGrad algorithm represents the best performances than the other algorithms with a mean absolute error (MAE) of 0.3858 in 53 iterations and AdaDelta one represents the lowest performances with a MAE of 0.6035 in 6000 iterations. The case study treated in this work is based on an extract of data from the keratoconus dataset of Harvard Dataverse and the results are obtained using Python.

1. Introduction
AI is difficult to define, yet it can be presented as a set of theories and techniques which aim to make computer systems capable to imitate some human behaviours such as reasoning, task planning, decision-making and learning [1].

The expectations of AI in the health sector are promising. Intelligent systems can assist in the diagnosis and detection of several diseases such as keratoconus [2], glaucoma [3] and diabetic retinopathy [4] in ophthalmology for example. These systems are mainly based on the analysis of biomedical images, taking advantage of the benefits of DL tools for the classification, prediction, and treatment of these diseases. Detection and classification of keratoconus using DL requires the analysis of topographic maps of the eye with a high number of features [5], this makes the learning phase of these systems very complicated and slow. Gradient descent (GD) is an optimization algorithm which is widely used in DL [6].

This work represents a comparative study of the stochastic, momentum, Nesterov, AdaGrad, RMSProp, AdaDelta, Adam, AdaMax and Nadam GD algorithms. The paper is organized into 5 sections, section 2 will present GD algorithm and its different variants. Related works are presented in section 3. Section 4 presents a case study. And section 5 presents a discussion of obtained results and finally a conclusion and perspectives of this work in the last section.

2. Gradient descent algorithms
The GD algorithm is an iterative optimization algorithm widely used in DL; this algorithm allows to gradually correct the parameters $\theta$ in order to minimize the cost function $J(\theta)$ [7]. GD algorithm uses all the dataset for each update of a parameter. This approach is the most precise of the gradient algorithms, but the most expensive given the number of calculations to be done. To correct this defect, several variants of this algorithm have been implemented.

2.1. Stochastic gradient descent algorithm (SGD)
In the SGD algorithm, only one randomly selected example of the dataset is used for the update. This solution is faster than the GD, but it does not provide a solution of the same precision [7] [8]. The SGD algorithm is described below:

| Input: initial vector $\theta$, learning rate $\eta$ |
| Repeat until convergence and $k \leq$ maximum number of iterations |
| mix the dataset |
| for $i \leftarrow 1, 2, 3, \ldots, n$ do: |
| Select an observation randomly |
| Calculate the step with the gradient from $\nabla f(\theta)$, scores and $\eta$ |
| $\theta \leftarrow \theta + \text{step}$ |
| Endfor |
| EndRepeat |

2.2. Momentum algorithm
The objective of Momentum algorithm [9] is to accelerate the descent process by adding a velocity vector to the equation of SGD [10]. The basic idea of momentum algorithm is to use the fraction from the previous update for the current one [8]. The algorithm introduces a new hyperparameter $\beta$, called momentum. The equation used to update the parameter $\theta$ is as follows:

$$m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta)$$  \hspace{1cm} (1)

$$\theta \leftarrow \theta + m$$  \hspace{1cm} (2)

The Momentum gradient algorithm is described as follows:

| Input: learning rate $\eta$, momentum $\beta$, initial $\theta$, initial velocity $m$ |
| While stop criteria is not met do: |
| Compose a minibatch $\theta^{(i)}$ from the dataset and corresponding targets $y^{(i)}$ |
| Calculate gradient estimate: $g \leftarrow \frac{1}{n} \nabla_{\theta} \sum_{i} J \left( f(\theta^{(i)}; \theta), y^{(i)} \right)$ |
| Calculate velocity update: $m \leftarrow \beta m - \eta g$ |
| Apply the update: $\theta \leftarrow \theta + m$ |
| EndWhile |

2.3. Nesterov Accelerated Gradient
The idea of Nesterov method is to add an intermediate displacement to each iteration, which makes it possible to shift the point according to the last direction of displacement [11]. This will correct the Momentum jump to eliminate the risk of skipping the overall minimum. The equation of Nesterov algorithm is as following:

$$m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta + \beta m)$$  \hspace{1cm} (3)

$$\theta \leftarrow \theta + m$$  \hspace{1cm} (4)

The Nesterov algorithm is depicted as follows:
**Input**: learning rate $\eta$, momentum parameter $\beta$, initial $\theta$, initial velocity $m$

**While** stop criteria is not met **do**:

Compose a minibatch $\theta^0$ from the dataset and corresponding targets $y^0$

Apply an intermediate update: $\tilde{\theta} \leftarrow \theta + \eta m$

Calculate gradient estimate: $g \leftarrow \frac{1}{n} \nabla_{\theta} \sum_j J\left(f\left(\theta^j; \tilde{\theta}\right), y^j\right)$

Calculate velocity update: $m \leftarrow \beta m - \eta g$

Apply update: $\theta \leftarrow \theta + m$

**EndWhile**

These algorithms use the same learning rate for all parameters. However, approaching a minimum using a bad learning rate produces fluctuations around the minimums.

2.4. **Adaptive Gradient algorithm (AdaGrad)**

AdaGrad algorithm [12] proposes to adjust the learning rate for each parameter during the learning phase based on historical information [13]. The objective of this adaptation is to improve the convergence of the algorithm and its prediction accuracy [14]. The equation of the AdaGrad algorithm is as follows:

$$s \leftarrow s + \nabla_{\theta} J\left(\theta\right) \otimes \nabla_{\theta} J\left(\theta\right)$$

$$\theta \leftarrow \theta - \frac{\eta \nabla_{\theta} J\left(\theta\right)}{\sqrt{s + \beta}}$$

(5)

(6)

Where $\eta$ is the initial learning rate, $S$ is the history of the square gradient of the $i$-th dimension of all the previous gradients for this dimension and $\beta$ is a smoothing constant. The AdaGrad algorithm is structured as follows [13]:

**Input**: $\eta$, a constant $\beta$ (about $10^{-7}$ for the numeric stability), initial $\theta$.

Initialize the gradient accumulation variable $r \leftarrow 0$

**While** stop criteria is not met **do**:

Compose a minibatch $\theta^0$ from the dataset and corresponding targets $y^0$

Apply an intermediate update: $\tilde{\theta} \leftarrow \theta + \eta m$

Calculate the gradient: $g \leftarrow \frac{1}{n} \nabla_{\theta} \sum_j J\left(f\left(\theta^j; \tilde{\theta}\right), y^j\right)$

Accumulate the squares of gradients: $r \leftarrow r + g \cdot g$

Calculate update: $m \leftarrow -\frac{\eta}{\sqrt{r + \beta}} \otimes g$

Apply update: $\theta \leftarrow \theta + m$

**EndWhile**

As the sum of squares of the gradients accumulates, the learning rhythm becomes slow, which is a weakness for the AdaGrad algorithm.

2.5. **Root Mean Square Propagation algorithm (RMSProp)**

To adjust the excessive growth of the cumulative squares of gradients for the AdaGrad algorithm, the RMSProp algorithm proposes to accumulate squares only for the most recent gradients [15]. The RMSProp algorithm equation is structured as follows:

$$s \leftarrow \beta s + (1 - \beta) \nabla_{\theta} J\left(\theta\right) \otimes \nabla_{\theta} J\left(\theta\right)$$

(7)
The RMSProp algorithm is structured as follows:

**Input:** \( \eta \), decay rate \( \beta \), small constant \( \alpha \) (about \( 10^{-7} \)), initial \( \theta \).

**Initialize the gradient accumulation variable:** \( r \leftarrow 0 \)

**While** stop criteria is not met **do**:

- Compose a minibatch \( \theta^{(r)} \) from the dataset and corresponding targets \( y^{(r)} \)
- Apply an intermediate update: \( \hat{\theta} \leftarrow \theta + \eta m \)
- Calculate the gradient: \( g \leftarrow \frac{1}{n} \sum \nabla_{\theta} J \left( f(\theta^{(i)}; \theta), y^{(i)} \right) \)
- Accumulate the squares of gradients: \( r \leftarrow \beta, r + (1 - \beta) g \cdot g \)
- Calculate update: \( m \leftarrow \frac{\hat{U}}{\sqrt{r + \alpha}} \odot g \)
- Apply update: \( \theta \leftarrow \theta + m \)

**EndWhile**

2.6. AdaDelta algorithm

The objective of AdaDelta algorithm is to reduce the rapid decay of AdaGrad learning rate. The idea behind it is to accumulate only the gradient squares of a window of a fixed size \( w \) of gradients [16]. The root mean square error (RMS) of parameter updates is described by the following equation:

\[
RMS[\Delta \theta] \leftarrow \sqrt{E[\Delta \theta^2]} + \hat{U}
\]

Replacing \( \eta \) in the previous update rule with \( RMS[\Delta \theta] \), finally provides the AdaDelta algorithm equation as follow:

\[
\Delta \theta_i \leftarrow \frac{RMS[\Delta \theta]_{i-1}}{RMS[g]_i} g_i
\]

\[
\theta_{i+1} \leftarrow \theta_i + \Delta \theta_i
\]

The AdaDelta algorithm is organized as follows:

**Input:** decay rate \( \rho \), small constant \( \epsilon \) (about \( 10^{-7} \)), \( \theta \) initial.

**Initialize accumulation variables:** \( E[g^2]_0 \leftarrow 0, E[\Delta \theta^2]_0 \leftarrow 0 \)

**For** \( t \leftarrow 1: T \) **do** % Loop over \# of updates

- Calculate gradient: \( g_t \)
- Accumulate gradients squares: \( E[g^2]_t \leftarrow \rho E[g^2]_{t-1} + (1 - \rho) g_t^2 \)
- Calculate update: \( \Delta \theta_t \leftarrow \frac{RMS[\Delta \theta]_{t-1}}{RMS[g]_t} g_t \)
- Accumulate updates: \( E[\Delta \theta^2]_t \leftarrow \rho E[\Delta \theta^2]_{t-1} + (1 - \rho) \Delta \theta_t^2 \)
- Apply update: \( \theta_{t+1} \leftarrow \theta_t + \Delta \theta_t \)

**EndFor**
The learning rate is eliminated from the parameter update expression for the AdaDelta algorithm.

2.7. Adaptive Moment Algorithm (Adam)
Adam algorithm [17] is a combination of Momentum and RMSProp. This algorithm also calculates adaptive learning rates for each parameter. Adam stores an exponentially decaying average of the previous gradient squares $v_t$ like AdaDelta and RMSprop and keeps an exponentially decaying average of the past gradients $m$ as for Momentum. Adam algorithm equation is described as follows:

$$ m_t \leftarrow \beta_1 m - (1 - \beta_1) \nabla \theta J(\theta) $$

$$ v_t \leftarrow \beta_2 v_t + (1 - \beta_2) \nabla \theta J(\theta) \nabla \theta J(\theta) $$

$$ m \leftarrow m \otimes (1 - \beta_1) $$

$$ v_t \leftarrow \frac{v_t}{1 - \beta_2} $$

$$ \theta \leftarrow \theta - \frac{\eta m}{\sqrt{v_t} + \epsilon} $$

The Adam algorithm is structured as follows:

| Input: $\eta$, decay rate $\beta_1$ and $\beta_2$, small constant $\alpha$ (about $10^{-7}$), initial $\theta$. Initialize gradient accumulation variable $r \leftarrow 0$
| $m_0 \leftarrow 0; v_0 \leftarrow 0; t \leftarrow 0.$ (Initialize 1st moment, 2nd moment and time step)
| **While** $\theta$, not converged do:
| $t \leftarrow t + 1$
| Calculate gradients for step $t$: $g_t \leftarrow \nabla \theta J_t(\theta_{t-1})$
| Update biased 1st moment estimate: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
| Update biased 2nd raw moment estimate: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
| Calculate bias-corrected 1st moment estimate: $\hat{m}_t \leftarrow \frac{m_t}{(1 - \beta_1)^t}$
| Calculate bias-corrected 2nd raw moment estimate: $\hat{v}_t \leftarrow \frac{v_t}{(1 - \beta_2)^t}$
| Update parameters: $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \alpha\right)$
| **EndWhile**

2.8. AdaMax Algorithm
AdaMax algorithm is an extension of Adam algorithm based on an infinite norm. In Adam algorithm, the factor $\bar{v}_t$ in the update rule scales the gradient inversely proportional to the $l_2$ norm of past gradients ($v_t$) and current gradient $|g_t|^2$ [17]:

$$ v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) |g_t|^2 $$

The generalization of $l_2$ norm to $l_p$ norm provides:

$$ v_t \leftarrow \beta_2^p v_{t-1} + (1 - \beta_2^p) |g_t|^p $$
Authors of AdaMax [17] prove that $v$ with $l$ converges to a more stable value. To avoid confusion with Adam, $u$ is used to denote the infinity norm $v$:

$$u_t \leftarrow \beta_2^\infty v_{t-1} + \left(1 - \beta_2^\infty\right) |g_{t}|$$

$$u_t \leftarrow \max\left\{\beta_2 \cdot v_{t-1}, |g_{t}|\right\}$$

(19)

The obtained AdaMax update rule is as follows:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{u_t} \hat{m}_t$$

(20)

The AdaMax Algorithm is structured as following:

**Input:** $\eta$, decay rate $\beta_1$ and $\beta_2$, small constant $\alpha$ (about $10^{-7}$), initial $\theta$.

$m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$. (Initialize 1st moment, 2nd moment and time step)

**While** $\theta_t$ not converged** do:**

$t \leftarrow t + 1$

Calculate gradients for step $t$: $g_t \leftarrow \nabla_{\theta_t} J_\theta(\theta_{t-1})$

Update biased 1st moment estimate: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$

Update exponentially weighted infinity norm: $u_t \leftarrow \max\left\{\beta_2 \cdot u_{t-1}, |g_{t}|\right\}$

Update parameters: $\theta_t \leftarrow \theta_{t-1} - \left(\eta / \left(1 - \beta_1^t\right)\right) \cdot m_t / u_t$

**EndWhile**

2.9. Nadam algorithm

Nesterov-accelerated Adaptive Moment (Nadam) algorithm [18] is a fusion of Adam and Nesterov-accelerated algorithms. The idea behind this type of algorithm is to increase and decrease the decay factor $\beta$ over time, a series of parameters $\beta_1, \beta_2, \ldots, \beta_t$ corresponding respectively to steps $1, 2, \ldots, t$ is considered for better clarity. The application of the momentum step in step $t+1$ is applied once updating the step $t$ instead of $t+1$ as follows [18]:

$$g_t \leftarrow \nabla_{\theta_{t-1}} J_\theta(\theta_{t-1})$$

(21)

$$m_t \leftarrow \beta_1 m_{t-1} + \eta g_t$$

(22)

$$\theta_t \leftarrow \theta_{t-1} - \left(\beta_1 m_t + \eta g_t\right)$$

(23)

Momentum and gradient steps, here, depend on the current gradient. The same applications on Adam algorithm give the following equations [18]:

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\beta_1 m_{t-1} + \left(1 - \beta_1\right) g_t}{1 - \prod_{i=1}^{t-1} \beta_i}$$

(24)

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\beta_1 m_t + \left(1 - \beta_1\right) g_t}{1 - \prod_{i=1}^{t-1} \beta_i}$$

(25)
The Nadam algorithm is described as follows:

**Input:** learning rates $\eta_1, \eta_2, ..., \eta_t$, decay rates $\beta_1, \beta_2, ..., \beta_t$; small constant $\epsilon$; initial $\theta$; hyperparameter $\nu$.

$m_0 \leftarrow 0; n_0 \leftarrow 0$. \textit{(Initialize 1st and 2nd moments)}

While $\theta_t$ not converged do:

- Calculate gradients for step $t$: $g_t \leftarrow \nabla_{\theta_t} J_t (\theta_{t-1})$
- Update biased 1st moment estimate: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- Update biased 2nd moment estimate: $n_t \leftarrow \nu n_{t-1} + (1 - \nu) g_t^2$

$$\hat{m} \leftarrow \beta_1 m_t / \left(1 - \prod_{i=1}^{t} \beta_i \right) + \left(1 - \beta_1 \right) g_t / \left(1 - \prod_{i=1}^{t-1} \beta_i \right)$$

$$\hat{n} \leftarrow \nu n_t / \left(1 - \nu \right)$$

- Update parameters: $\theta_t \leftarrow \theta_{t-1} - \frac{\eta_t}{\sqrt{n_t + \epsilon}} \hat{m}_t$

**EndWhile**

3. Related works

Several research works are realized to improve performances of gradient descent algorithm and compare its different variants. The authors in [19] provided a comparative study of stochastic algorithms with momentum, Adam, AdaGrad, AdaDelta and RMSProp of optimization. In this study, authors compared the advantages and disadvantages these approaches considering the convergence time, number of fluctuations and the update rate of features while selecting specific test functions. In [15], the authors proposed an analysis of the RMSProp algorithm for training deep neural networks and suggest two variants of this algorithm, SC-Adagrad and SC-RMSProp for which they show logarithmic regret limits for strongly convex functions. The authors in [20] proposed a study to prove that ADAM and RMSProp algorithms are guaranteed to reach criticality for smooth non-convex objectives, authors studied by experiments the convergence and generalization properties of RMSProp and ADAM against Nesterov’s Accelerated Gradient method on a variety of common autoencoder setups. Through these experiments we demonstrate the interesting sensitivity. In [21], the authors realized a comparative study for all already studied algorithms, which were evaluated in terms of convergence speed, accuracy, and loss function. In [22], the authors propose a comparative experimental analysis of different stochastic optimization algorithms for recording images in the spatial domain. Searchers in [12] and [23] provide an analytical study for GD algorithms and offer improvements to increase their performance. This work is an experimental comparative study of the nine variants of gradient descent algorithms already detailed because few are the works that have compared them all in the state of the art.

4. Case Study

The following case study is implemented in Python in order to compare the performances of the stochastic, momentum, Nesterov, AdaGrad, RMSProp, AdaDelta, Adam, AdaMax and Nadam GD algorithms in terms of speed of convergence and mean absolute error for the different generated solutions. The implementation is based on an extract of the keratoconus dataset of Harvard Dataverse [24]. The dataset used is composed of two columns and 96 rows. Structured in a csv file, this dataset represents the relationship between flat and steep corneal meridians as shown in figure 1 bellow:
The visualization of the used dataset shows a linear correlation between the steep and the flat corneal meridians. As the value of the steep corneal meridian increases, so does the flat corneal meridian value.

4.1. Simulation results

Figures 2, 3, 4, 5, 6, 7, 8, 9 and 10 illustrate the simulation results obtained by the application of the different gradient descent algorithms with a learning rate of 0.001. For the stochastic gradient algorithm, the number of iterations is fixed at 85 iterations. All algorithms were implemented in Python.

Figure 1. Original data (b) and Normalized data (b).

Figure 2. Stochastic solution (a) and Global cost error function (b).

Figure 3. Momentum solution (a) and Global cost error function (b).

Figure 4. Nesterov solution (a) and Global cost error function (b).
Figure 5. AdaGrad solution (a) and Global cost error function (b).

Figure 6. RMSProp solution (a) and Global cost error function (b).

Figure 7. Adam solution (a) and Global cost error function (b).

Figure 8. AdaDelta solution (a) and Global cost error function (b).
Generally, all generated solutions are close as much as possible to all points of the dataset and after a certain number of iterations, the global cost error functions stabilize, this stability indicates the convergence of different studied algorithms.

5. Results discussion

Figures 2, 3, 4, 5, 6, 7, 8, 9 and 10 represent respectively the optimization solutions and error cost functions obtained by the stochastic, Momentum, Nesterov, AdaGrad, RMSProp, Adam, AdaDelta, AdaMax and Nadam gradient descent algorithms. The table 1 below summarizes the performance of the different algorithms studied in terms of the number of iterations and the mean absolute error (MAE) of each solution:

| Algorithm       | Number of iterations | MAE   |
|-----------------|----------------------|-------|
| Stochastic      | 85 (fixed)           | 0.7221|
| Momentum        | 160                  | 0.3673|
| Nesterov        | 70                   | 0.3856|
| AdaGrad         | 53                   | 0.3858|
| RMSProp         | 210                  | 0.3788|
| Adam            | 90                   | 0.3863|
| AdaDelta        | 6000                 | 0.6035|
| AdaMax          | 95                   | 0.3857|
| Nadam           | 70                   | 0.3856|

The obtained results show that the Stochastic and AdaDelta gradient algorithms, figures 2 and 8 respectively, present the lowest performances with the largest mean absolute errors, 0.7221 and 0.6035 respectively, as well as the greatest number of iterations 6000 for AdaDelta algorithm (the number of iterations is fixed at 85 for the stochastic gradient). The algorithms of Nesterov in Figure 4, Adam in Figure 7, AdaMax in Figure 9, AdaGrad in Figure 5 and Nadam in Figure 10 represent almost similar
performances for the mean absolute error which is of the order of 0.38. On the other hand, a remarkable difference for the number of iterations carried out by each algorithm with a distinction of the AdaGrad algorithm having done the least iterations with only 53 iterations, then the algorithms of Nesterov, AdaMax, Adam and Nadam with respectively 70, 95, 90 and 70 iterations. The algorithms of RMSProp in figure 6 and Momentum in figure 3 represent the best MAE which are of the order of 0.3788 and 0.3673 respectively, with a growth in the number of iterations, 210 and 160 iterations for RMSProp and Momentum respectively. Among the studied algorithms, the AdaGrad algorithm represents a great interest for a later use in a project of detection and classification of keratoconus given its good performances in terms of convergence speed and MAE.

This work comes in order to provide a more detailed and in-depth study of the different gradient descent algorithms, applied to ophthalmological data.

6. Conclusion
This work represents a comparative study of Stochastic, Momentum, Nesterov, AdaGrad, AdaDelta, RMSProp, Adam, AdaMax and Nadam gradient descent algorithms of optimization in terms of convergence speed and mean absolute error of the generated solutions. Among the already cited algorithms, AdaGrad algorithm represents the best solution with a MAE of 0.3858 and 53 iterations, this algorithm is interesting considering the approximation of minimization which it provide, its speed of convergence, this makes it possible to use it in futur works using a large datasets. Stochastic and AdaDelta algorithms present the lowest performances with MAE of 0.7221 and 0.6035 respectively, and 6000 iterations for AdaDelta. This study was realized to compare these algorithms in order to facilitate the choice of the most efficient algorithm for later use in a project of keratoconus detection through eye topographic images analysis.

References
[1] Stadie B C, Abbeel P and Sutskever I 2017 Third-Person Imitation Learning (arXiv:1703.01703).
[2] Lavric A and Valentin P 2019 KeratoDetect: Keratoconus Detection Algorithm Using Convolutional Neural Networks Comput. Intell. Neurosci 2019 p 1–9.
[3] Chai Y, Liu H and Xu J 2018 Glaucoma diagnosis based on both hidden features and domain knowledge through deep learning models Knowledge-Based Syst 161 p 147–156.
[4] Gargeya R and Leng T 2017 Automated Identification of Diabetic Retinopathy Using Deep Learning Ophthalmology 124 p 962–969.
[5] Kamiya K, Ayatsuka Y, Kato Y, Fujimura F, Takahashi M, Shoji N, Mori Y and Miyata K 2019 Keratoconus detection using deep learning of colour-coded maps with anterior segment optical coherence tomography: A diagnostic accuracy study BMJ Open 9 p 1–7.
[6] Ruder S 2016 An overview of gradient descent optimization algorithms (arXiv:1609.04747) p 1–14.
[7] Aatila M, Lachgar M and Kartit A 2020 An Overview of Gradient Descent Algorithm Optimization in Machine Learning: Application in the Ophthalmology Field International Conference on Smart Applications and Data Analysis Springer Marrakech Morocco p 349-359.
[8] Mandt S, Hoffman M D and Blei D M 2017 Stochastic Gradient Descent as Approximate Bayesian Inference (arXiv:1704.04289) p 1–35.
[9] Botev A, Lever G and Barber D 2017 Nesterov’s Accelerated Gradient and Momentum as approximations to Regularised Update Descent International Joint Conference on Neural Networks (IJCNN) IEEE Anchorage AK USA p 1899–1903.
[10] Qian N 1999 On the Momentum Term in Gradient Descent Learning Algorithms Acknowledgments Neural Networks 12 p 145–151.
[11] Nesterov Y 1983 A method of solving a convex programming problem with convergence rate o (1/k²) Soviet Mathematics Doklady 27 p 372–376.
[12] Duchi J, Hazan E and Singer Y 2010 Adaptive subgradient methods for online learning and stochastic optimization Conf. Learn. Theory p 257–269.
[13] Zhang N, Lei D and Zhao J F 2019 An Improved Adagrad Gradient Descent Optimization Algorithm Proc 2018 Chinese Automation Congress p 2359–2362.
[14] Hadgu A T, Nigam A and Diaz-Aviles E 2015 Large-scale learning with AdaGrad on Spark Proc. 2015 IEEE Int. Conf. Big Data 2015 Santa Clara USA p 2828–2830.
[15] Chandra M and Matthias M 2017 Variants of RMSProp and Adagrad with Logarithmic Regret Bounds (arXiv:1706.05507).
[16] Zeiler M D 2012 ADADELTA: An Adaptive Learning Rate Method (arXiv:1212.5701).
[17] Kingma D P and Ba J L 2015 Adam: A method for stochastic optimization 3rd International Conference on Learning Representations San Diego p 1–15.
[18] Dozat T 2016 Incorporating Nesterov Momentum into Adam ICLR Workshop p 2013–2016.
[19] Yazan Y and Talu M F 2017 Comparison of the stochastic gradient descent based optimization techniques International Artificial Intelligence and Data Processing Symposium IEEE Malatya Turkey p 1–5.
[20] De S, Mukherjee A and Ullah E 2018 Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration (arXiv:1807.06766).
[21] Dogo E M, Afolabi O J, Nwulu N I, Twala B and Aigbavboa C O 2018 A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks International Conference on Computational Techniques, Electronics and Mechanical Systems IEEE Belgum India p 92–99.
[22] Voronov S, Voronov I and Kovalenko R 2018 Comparative analysis of stochastic optimization algorithms for image registration IV International Conference on Information Technology and Nanotechnology.
[23] Hui Z, Zaiyi C, Chuan Q, Zai H, Vincent W Z, Tong X and Enhong C 2020 Adam revisited: a weighted past gradients perspective Front. Comput. Sci 14.
[24] Yousefi S, Yousefi E, Takahashi H, Hayashi T, Tampo H, Inoda S, Arai Y and Asbell P 2018 Replication Data for the "Keratoconus severity identification using unsupervised machine learning" PLOS One 2018.