First-Order Optimization (Training) Algorithms in Deep Learning

Oleg Rudenko$^{1[0000-0003-0859-2015]}$, Oleksandr Bezsonov$^{1[0000-0001-6104-4275]}$, Kyrylo Oliinyk$^{1[0000-0001-8536-5217]}$

^{1} Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine

oleh.rudenko@nure.ua
oleksandr.bezsonov@nure.ua
kirill.oleynik.olegovich@gmail.com

Abstract. The use of artificial neural networks (ANN) requires solving structural and parametric identification problems corresponding to the choice of the optimal network topology and its training (parameter settings). In contrast to the problem of determining the structure, which is a discrete optimization (combinatorial), the search for optimal parameters is carried out in continuous space using some optimization methods. The most widely used optimization method in deep learning is the first-order algorithm that based on gradient descent (GD). In the given paper a comparative analysis of convolutional neural networks training algorithms that are used in tasks of image recognition is provided. Comparison of training algorithms was carried out on the Oxford17 category flower dataset with TensorFlow framework usage. Studies show that for this task a simple gradient descent algorithm is quite effective. At the same time, however, the problem of selecting the optimal values of the algorithms parameters that provide top speed of learning still remains open.

Keywords: Convolution, Optimization, Neural Network, Algorithm, Gradient, Training, Image Recognition.

1 Introduction

Deep Learning is a class of Artificial Neural Network (ANN) that has many processing layers. There is huge number of ANN architectures variants in the literature. ANNs can be used as a very effective technology to solving a wide class of problems [1-5]. After the breakthrough result in the ImageNet classification challenge [2], different kinds of neural network, i.e. convolution NN (CNN) architectures have been proposed and the performance is improved year by year [6,7].

The use of ANN requires solving structural and parametric identification problems corresponding to the choice of the optimal network topology and its training (parameter settings). In contrast to the problem of determining the structure, which is a discrete optimization (combinatorial), the search for optimal parameters is carried out in continuous space using classical optimization methods. To train direct distribution networks with a teacher, algorithms are usually used that optimize some objective function. There are a lot of works that aim to improve ANN in different aspects (architecture design, choice and optimization of training algorithms and so on).

The most widely used optimization method in deep learning is the first-order algorithm that based on gradient descent (GD). The BP algorithm is the standard training method for ANN which uses GD. These methods can be split into the following categories: batch gradient decent method, mini-batch gradient decent method, and stochastic gradient decent method (SGD) [8, 9]. The GD method is the earliest optimization method. It often converges with a slower speed. The batch gradient decent method has high computational complexity for scale data.

The using of SGDs is the predominant methodology in training deep learning (CNN).

2 The Structure of the Convolution Neural Network

Initially, the convolution neural network structure was created taking into account the structural features of some parts of the human brain responsible for vision. The basis for the development of such networks is incorporated by three mechanisms:

- local perception;
- forming a set of layers in the shape of the characteristics maps (shared weights);
- sub-sampling (sub-set).

Under the local perception it is understood that the input neuron receives not the whole picture but only some part of it. This helps to keep the image configuration during the transition from layer to layer.

The idea of shared weights means that a large number of connections used a small set of weights, i.e. each area of the image to which it is divided, will be processed by the same set of weights. Such artificial limitation of weights improves network's generalization property.

CNN consists of the convolution layers, sub-sampling and fully connected neural network layers.

3 Convolution Neural Network Layers

CNN got its name from the operator "convolution". The main purpose of convolution in the CNN case is to extract features from the input image.

Convolution keeps spatial relations among pixels, studying the features of the image, using the small batches of the input data.

Each neuron in the plane of the convolutional layer receives its inputs from the certain region of the previous layer (local receptive field).

Subsample layer zooms planes by local averaging of the neurons output values. Subsequent layers extract more common characteristics relied upon the picture distortion.

Each convolutional layer is followed by subsampling or computational layer which produces a reduction of the image dimension by local averaging the values of the neurons output.

The architecture of the convolution network is assumed that evidence of the feature's existence is more important information than its exact location. Therefore, from a plurality of neighboring neurons in the map attributes one neuron with maximum value is chosen to map features of smaller dimension.

Difference between subsample layer and convolution layer is that in the convolution layer neighboring neurons overlap, which does not occur in the subsampling layer.

Thus, CNN is constructed by alternating of convolution and subsampling layers. At the output of the network several layers of fully connected neural network are usually installed. The input for these layers is the final feature's map. Each neuron of the output layer is the perceptron, which has a non-linear activation function.

4 Training Methods of Convolution Neural Network

For convolutional neural network training a standard backpropagation algorithm and its various modifications can be used. The basis of this method is a stochastic gradient descent algorithm (Stochastic Gradient Descent).

4.1 Training Based on Stochastic Gradient

Stochastic gradient descent (SGD) and its variants are the most widely-studied algorithms for optimization problems in machine learning and stochastic approximation.

The usual gradient descent is described by the following relation

$$\theta(k+1)=\theta(k)-\eta\nabla_{\theta}J(\theta(k)), \tag{1}$$

where θ – network parameter $N\times 1$, $J(\theta(k))$ – the loss function; η – training speed parameter (learning rate).

Algorithm (1) convergence is generally not guaranteed, but it is proved [9, 10] that in the case of a convex function $J(\theta(k))$ and under the following conditions:

$$\lim \eta(k)=0;\ \sum_{k=1}^{\infty}\eta(k)=\infty;\ \sum_{k=1}^{\infty}\eta^{2}(k)<\infty, \tag{2}$$

gradient descent process will converge.

There are two main approaches to implementing gradient descent:

- Batch - when the training sample is viewed entirely at each iteration, and only after this θ is changed. This requires large computational cost.

- Stochastic (online) - where at each iteration of the algorithm from the training set some (random) object is selected. Thus, the vector θ is configurable for each newly selected object.

The following disadvantages are inherent to this algorithm:

- stuck in local minima and saddle points of the minimized functional.
- Slow convergence due to difficult terrain of the objective function when the plateau regions alternate with strong nonlinearity (the derivative of the plateau is almost zero, and sudden fall, on the contrary, can change the parameter estimation).
- Some of the parameters are updated less often than others, especially when in the data some informative but rare features are found. This has a bad effect on the nuances of the network rules generalization. On the other hand, giving too much importance to all rarely seen features can lead to overtraining.
- Too small value of η parameter leads to slow convergence and stucking in local minima, while too large value of η leads to "overshooting" the narrow global minima or no divergence at all.

Using the second-order methods discussed above requires calculating the Hessian - derivative matrix for each pair of parameters, and, for the Newton's method additionally its inverse matrix, i.e. implementation of these methods involves considerable computing effort.

Therefore, in practice, widespread methods are based on the stochastic gradient method which has number of advantages [11].

Consider these methods in more detail.

4.2 Stochastic Average Gradient (SAG)

The stochastic average gradient (SAG) algorithm [12] is a difference decrease strategy proposed to increase the speed of convergence.

The SAG iterations take the form

$$\theta(k+1) = \theta(k) - \frac{\alpha(k)}{n}\sum_{i=1}^{n} y_i(k), \qquad y_i = \begin{cases} \nabla_\theta J_i(\theta(k)) & if \quad i = i_k, \\ y_{i-1} & otherwise. \end{cases} \tag{3}$$

where $\alpha(k)$ is the learning rate and a random index i_k is selected at each iteration during which we set.

Essentially, SAG maintains in memory, the gradient with respect to each function in the sum of functions being optimized. The gradient value of only one such function is computed and updated in memory at each iteration. The SAG method estimates the overall gradient at by averaging the gradient values stored in memory.

However, the SAG technique can be utilized only with the smooth loss function and a convex objective function. The SAG has better convergence comparing to the SGD in tasks like convex linear prediction problems.

4.3 Stochastic Variance Reduction Gradient (SVRG)

The SVRG algorithm [13] calculates gradient $\tilde{\mu}$ in the following way:

$$\tilde{\mu} = \frac{1}{N}\sum_{i=1}^{N} g_i(\tilde{\theta}), \tag{4}$$

where $\tilde{\theta}$ - interval update parameter. SVRG performs gradient updates by using following equation:

$$\theta(k+1) = \theta(k) - \eta(g_i(k)(\theta(k) - g_i(k)(\tilde{\theta}) + \tilde{\mu}). \tag{5}$$

The gradient can be calculated up to two times during each update. After w iterations, parameter $\tilde{\theta}$ is updated and the next w iterations start. Through these update, $\theta(k+1)$ and the interval update parameter $\tilde{\theta}$ will converge to the optimal θ^*, and then $\tilde{\mu} \to 0$, and

$$g_i(k)(\theta(k) - g_i(k)(\tilde{\theta}) + \tilde{\mu} \to g_i(k)(\theta(k) - g_i(k)(\theta^*) \to 0. \tag{6}$$

There are also many variants of such linear convergence stochastic optimization algorithms, such as the SAGA algorithm [13, 14].

4.4 Momentum

Instead of depending on current gradient only for updating weights, the gradient descent algorithm [15] with momentum replaces the current gradient with $v(k+1)$ (v means the velocity), exponential moving average of the current and past gradients (i.e., before the time $k+1$)

$$\theta(k+1) = \theta(k) - \eta v(k+1); \tag{7}$$

$$v(k+1) = \gamma v(k) + (1-\gamma)\nabla_{\theta} J(\theta(k), \tag{8}$$

where $\gamma = 0.9.$

Later this pulse update becomes a standard for the gradient components upgrade.

4.5 NAG (Nesterov Accelerated Gradient)

This algorithm [16] implements the idea of accumulation the pulse by using the information on the change of each parameter in the form of an exponential moving average

$$v(k+1) = \gamma v(k) + (1-\gamma)x. \tag{9}$$

The gradient algorithm accumulates target network functions

$$v(k+1) = \gamma v(k) + \eta \nabla_{\theta} J(\theta(k)), \tag{10}$$

which is used during the parameters correction

$$\theta(k+1) = \theta(k) - v(k). \tag{11}$$

More precise correlation algorithm for NAG has the form

$$\theta(k+1) = \theta(k) - v(k+1); \tag{12}$$

$$v(k+1) = \gamma v(k) + \eta \nabla_{\theta} J(\theta(k) - \gamma v(k)). \tag{13}$$

4.6 SNM (Simplified Nesterov Momentum)

In [17] was offer a new formulation of Nesterov momentum differing from (8) and (9). The main difference from (8) and (9) lies in committing to the "peekedahead" parameters $\Theta(k) = \theta(k) - \gamma(k)v(k)$ and backtracking by the same amount before each update. These new parameters $\Theta(k+1)$ updates become [17]:

$$v(k+1) = \gamma(k)v(k) + \eta(k)\nabla_{\Theta} J(\Theta(k)); \tag{14}$$

$$\begin{aligned} \Theta(k+1) &= \Theta(k) + \gamma(k)v(k) - \gamma(k+1)v(k+1) - v(k+1) = \\ &= \Theta(k) - \gamma(k+1)\gamma(k)v(k) + (1+\gamma(k+1))\eta(k)\nabla_{\Theta} J(\Theta(k)). \end{aligned} \tag{15}$$

4.7 Adagrad

Adagrad (adaptive gradient) [18] takes into account the frequency of neurons activation by storing for each network parameter the sum of its squares updates. It uses a modified formula of renovation

$$M\{g^2(k+1)\} = \gamma M\{g^2(k)\} + (1-\gamma) g^2(k), \tag{16}$$

and correction parameter is carried out according to the rule

$$\theta(k+1) = \theta(k) - \frac{\eta}{\sqrt{G(k)+\varepsilon}} g(k), \tag{17}$$

where $g(k) = \nabla_\theta J(\theta(k);$ $G(k)$ – the sum of the updates squares, ε - smoothing parameter that is required in order to avoid division by 0. The frequently updated last parameter $G(k)$ is large (large denominator in (17)), i.e. the parameter will change slightly. Rarely changed parameters will change substantially. Parameter ε generally selected in range of 10-6 - 10-8.

4.8 RMSProp

Disadvantage of Adagrad is that $G(k)$ in (17) can be increased without any limitations. As the result, after short time an update becomes too small. This leads to paralysis of the algorithm. RMSProp and Adadelta designed to solve this problem [19].

Adagrad modifies parameters in such a way that the frequently updated weights are adjusted less frequently. To do this, instead of the full sum of the updates averaged over history gradient square is used, i.e., moving average of the following form

$$M\{g^2(k+1)\} = \gamma M\{g^2(k)\} + (1-\gamma)g^2(k), \tag{18}$$

then instead of (17) we obtain

$$\theta(k+1) = \theta(k) - \frac{\eta}{\sqrt{M\{g^2(k)\} + \varepsilon}} g(k). \tag{19}$$

Since the denominator represents the root of the mean squares gradient

$$RMS\{g(k)\} = \sqrt{M\{g^2(k)\} + \varepsilon}, \tag{20}$$

algorithm got its name RMSProp - root mean square propagation.

4.9 Adadelta

Adadelta is a continuation Adagrad [20], which aims to avoidance of monotonic decrease of training speed. Instead of accumulating all the gradients of the last square, Adadelta bounds the window of collected past gradients to the fixed size.

Adadelta is different from RMSProp because we add to the numerator (17) the stabilizing member proportional to RMS from $\Delta\theta(k)$. In step $k+1$ value of $RMS\{\Delta\theta(k)\}$ is not yet known, so the update of the parameters is implemented in three stages instead of two: at the first stage square of the gradient is accumulated, then θ is updated. And finally $RMS\{\Delta\theta(k)\}$, is updated

$$\theta(k+1) = \theta(k) - \frac{RMS\{\Delta\theta(k)\}}{RMS\{g(k)\}} g(k); \tag{21}$$

$$M\{\|\Delta\theta(k+1)\|^2\} = \gamma M\{\|\Delta\theta(k)\|^2\} + (1-\gamma)\|\Delta\theta(k)\|^2\ ; \qquad (22)$$

$$RMS\{\Delta\theta(k)\} = \sqrt{M\{\|\Delta\theta(k)\|^2\} + \varepsilon}. \qquad (23)$$

For RMSProp, Adadelta and Adagrad there is no need in very accurate choose of the learning curve - just its approximate value is needed. Usually it is advised to start snapping η from 0,1-1, and leave γ 0.9. The closer γ to 1, the longer RMSProp and Adadelta with great $RMS\{\Delta\theta(k)\}$ will much update rarely used weights. If $\gamma \approx 1$ and $RMS\{\Delta\theta(k)\} = 0$, then Adadelta be longer "with a grain of salt" refers to a rarely used weights that can lead to paralysis of the algorithm, and intentionally cause "greedy" behavior, when the algorithm updates the first neurons that encode the best features.

4.10 Adam

Adam (Adaptive moment estimation) – another optimization algorithm. It combines the idea of accumulation of the motion and the idea of a weaker weight updates for typical features [21]. By analogy with (6) we can obtain:

$$m(k+1) = \beta_1 m(k) + (1-\beta_1)g(k). \qquad (24)$$

Nesterov is different from Adam because there is no need to accumulate $\Delta\theta$, and the gradient's value. To obtain information about the gradient's change in [21] it is proposed to estimate additionally an average dispersion:

$$v(k+1) = \beta_2 v(k) + (1-\beta_2)\,|g(k+1)|^2\ ; \qquad (25)$$

$$\theta(k+1) = \theta(k) - \eta\frac{\sqrt{1-\beta_2(k+1)}}{1-\beta_1(k+1)}\frac{m(k+1)}{\sqrt{v(k+1)}+\varepsilon}, \qquad (26)$$

where $\beta_1, \beta_2 \in [0,1)$, $g(k)$ – stochastic gradient off J at $\theta(k+1)$, η – step size, ε – a small constant.

Authors of Adam offered as defaults $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$ and argue that an algorithm performs better or about the same as all previous algorithms on a broad set of datasets due to the initial calibration.

4.11 AdaMax

AdaMax algorithm [22] is a Adam's algorithm modification, wherein the dispersion is used instead of the inertial moment of the distribution of arbitrary degree gradients p.

While this may lead to the calculation of volatility, in practice the case $p \to \infty$. It works surprisingly well

$$v(k+1) = \beta_2^p v(k) + (1-\beta_2^p)\left|g(k)\right|^p . \tag{27}$$

4.12 Adapg

Adapg combins Adadelta and Adam[6]

$$M\{\left\|\Delta\theta(k+1)\right\|^2\} = \gamma M\{\left\|\Delta\theta(k)\right\|^2\} + (1-\gamma)\left\|\Delta\theta(k)\right\|^2 . \tag{28}$$

4.13 HAdam (High-order Adam)

Using the core update law of Adam [23] it can be rewritten using induction as following

$$m(k+1) = (1-\beta_1)\sum_{i=0}^{k} \beta_1^i g(k-i); \tag{29}$$

$$v(k+1) = (1-\beta_2)\sum_{i=0}^{k} \beta_2^i g^2(k). \tag{30}$$

It is easy to note that $m(k+1)$ and $v(k+1)$ are different because of the exponential moving average utilization. In paper [23] was extend the second moment to high-order moment.

4.14 Nadam

Nadam algorithm (Nesterov-accelerated Adaptive Moment Estimation) [22] is a modification of Nesterov algorithm with pulse parameter adaptation

$$\theta(k+1) = \theta(k) - \frac{\eta}{\sqrt{G(k)+\varepsilon}}\left(\beta_1\hat{g}(k) + \frac{1-\beta_1}{1-\beta_2}\nabla_\theta J(\theta(k)\right); \tag{31}$$

$$\hat{g}(k) = \frac{g(k)}{1-\beta_1}; \tag{32}$$

$$\hat{G}(k) = \frac{G(k)}{1-\beta_2}; \tag{33}$$

$$g(k+1) = \beta_1 g(k) + (1-\beta_1)\nabla_\theta J(\theta(k); \tag{34}$$

$$G(k+1) = \beta_2 G(k) + (1-\beta_2)\left[\nabla_\theta J(\theta(k)\right]^2; \tag{35}$$

α= 0.002; β_1= 0.9; β_2 = 0.999; $\varepsilon = 10^{-7}$.

4.15 AMSGrad

Another version of Adam algorithm is AMSGrad [24]. This version revises the components of the adaptive learning rate in Adam and changes it to ensure that the current G is always greater than at the previous time step

$$\theta(k+1) = \theta(k) - \frac{\eta}{\sqrt{\hat{G}(k)+\varepsilon}} g(k); \tag{36}$$

$$\hat{G}(k+1) = \max\left(\hat{G}(k), G(k+1)\right); \tag{37}$$

$$g(k+1) = \beta_1 g(k) + (1-\beta_1)\nabla_\theta J(\theta(k); \tag{38}$$

$$G(k+1) = \beta_2 G(k) + (1-\beta_2)\left[\nabla_\theta J(\theta(k)\right]^2; \tag{39}$$

$$G(k+1) = \beta_2 G(k) + (1-\beta_2)\left[\nabla_\theta J(\theta(k)\right]^2; \tag{40}$$

α= 0.001; β_1= 0.9; β_2 = 0.999; $\varepsilon = 10^{-7}$.

4.16 WNGrad

In WNGrad algorithm (weight normalization Grad) [25] the method of dynamic update of the learning rate is used in accordance with the obtained gradients

$$\theta(k+1) = \theta(k) - \frac{1}{b(k)} \nabla_\theta J(\theta(k)); \tag{41}$$

$$b(k+1) = b(k) + \frac{1}{b(k)} \left\|\nabla_\theta J(\theta(k))\right\|. \tag{42}$$

According to the numerical experiments results, WNGrad is a rival to the simple stochastic gradient descent algorithm in terms of sustainability and generalization error in the training of neural networks. In [15] WNGrad modifications are proposed which use pulse (WN-Adam and WNGrad-Momentum).

4.17 Padam

Padam (Partially adaptive momentum estimation method) [26] unifies Adam/Amsgrad and SGD with momentum by a partially adaptive parameter

$$\theta(k+1) = \theta(k) - \frac{\eta}{\hat{G}^{p}(k)} g(k); \tag{43}$$

$$\hat{G}(k+1) = \max\left(\hat{G}(k), G(k+1)\right), \tag{44}$$

where $p \in (0,1/2]$ is the partially adaptive parameter (1/2 is the largest possible value for p and a larger p will result in non-convergence in the proof). When $p \to 0,$ Padam reduces to SGD with momentum and when $p = 1/2$ it is exactly Amsgrad.

It is empirically shown in [26] that Padam achieves the highest training speed while generalizing just as SGD. These outcomes recommend that a developer should get adaptive gradient methods by and by for faster adjustment of CNN weights.

4.18 AdaShift

The key difference between Adam and AdaShift (ADAptive learning rate method with temporal SHIFTing) [27] is that the latter temporally shifts the gradient $g(k)$ for n-step, i.e., using $g(k-n)$ for calculating $v(k)$ and using the kept-out n gradients, which makes $v(k)$ and $g(k)$ temporally shifted and hence decorrelated:

$$v(k+1) = \beta_2 v(k) + (1-\beta_2)\phi(|g(k-n)|^2), \tag{45}$$

where ϕ is a function (spatial operation). There is no restriction on the choice of ϕ (in [27] $\phi(x) = \max_i x(i)$).

4.19 SWATS

SWATS (Switching from Adam to SGD) [28] is a special method that Switches from Adam to SGD when a triggering condition is satisfied.

While the focus of [28] has been on Adam, the strategy proposed is generally applicable and can be analogously employed to other adaptive methods such as Adagrad and RMSProp. A viable research direction includes exploring the possibility of switching back-and-forth, as needed, from Adam to SGD.

4.20 Parallelizing SGD

A strength of SGDs is that they are simple to implement and also fast for problems that have many training examples. However, SGD methods have many disadvantages

[29]. Recently, several approaches [30][31][32] [33][34] towards an effective parallelization of the SGD optimization have been proposed.

A theoretical framework for the analysis of SGD parallelization performance has been presented in [30]. In [31] was introduced an update scheme called Hogwild that allows performing SGD updates in parallel on CPUs. In [32] was proposed an algorithm called weighted parallel SGD (WP-SGD). Other methods of parallelizing SGD were introduced in [33] [34].

5 Modeling

We train and evaluate our CNN model on the Oxford17 category flower dataset [35]. It contains 17 categories of common flowers in the UK with 80 images for each class. Some of the pictures from the dataset are present at Figure 1.

Proposed CNN based application is implemented using TensorFlow [36] and is trained with using Nvidia GeForce-2080 GPU. Performance and accuracy of different first-order optimization algorithms shown in Figure 2 and Figure 3. From presented results it can be seen that the considered training methods perform differently. Adamax, adagrad and SGD converge faster than the other methods. Performance of Adadelta is also acceptable.

6 Conclusion

A comparative analysis of gradient learning algorithms of convolutional neural networks in solving visual recognition problem was held in this report. Studies show that for this task quite effective is a simple gradient descent algorithm. Pulse usage in the considered modifications led to some improvement in the recognition process, but it also increased the computation cost. In the considered problems the most effective algorithm is Adamax. In [24] it is recommended to always start with the Adam optimizer, regardless of the architecture of the neural network and problem areas in which it is used. However, in our opinion, at the decision of problems of recognition the Adamax algorithm should be used. At the same time, however, the problem of selecting the optimal values of the algorithms parameters that provide top speed of learning still remains open.

7 Acknowledgement

This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Co-funded by the Erasmus+ Programme of the European Union

Fig. 1. Example images from the Oxford flower dataset

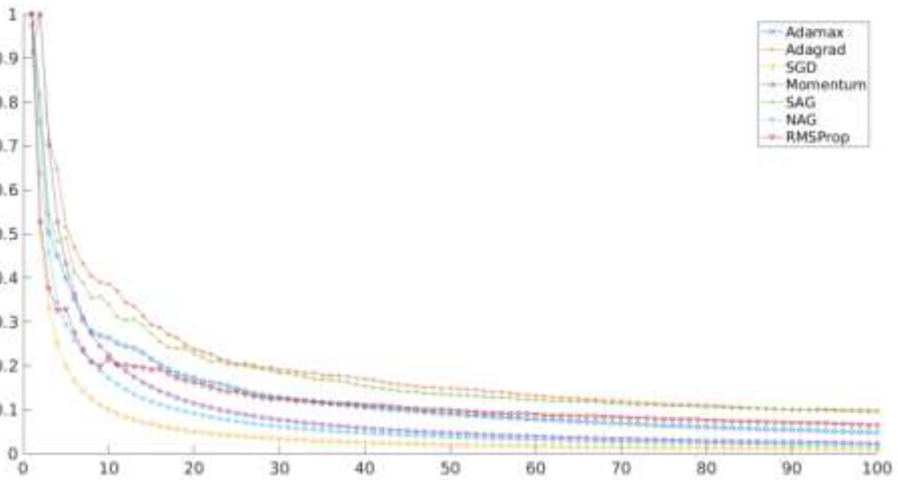

Fig. 2. Performance of different first-order optimization algorithms

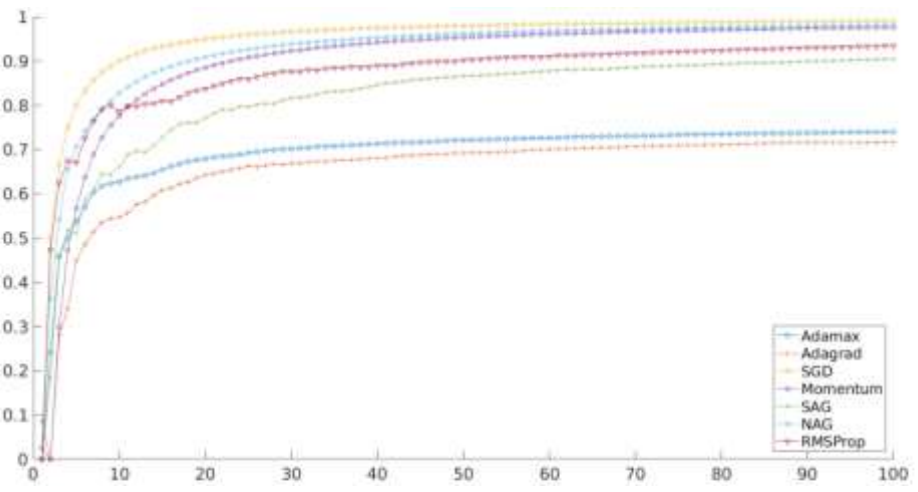

Fig. 3. Accuracy of different first-order optimization algorithms

References

1. Goodfellow, I., Bengio, Y. and Courville, A.: Deep learning. MIT Press (2016).
2. Krzhevsky, A., Sutshever, I. and Hinton, G.: ImageNet classi_cation with deep convolutional neuralnetworks. In NIPS (2012).
3. Long, J., Shelhamer, E. and Darrell, T.: Fully convolutional networks for semantic segmentation. In CVPR (2015).
4. Girshick, R.: Fast R-CNN. In ICCV (2015).
5. Ren, R., Girshick, K., He, and Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Trans. PAMI, 39:1137-1149 (2016).
6. Li, P.: Optimization Algorithms for Deep Learning, http://lipiji.com/docs/li2017optdl.pdf
7. Tan, H.H. and Lim, K.H.: Review of second-order optimization techniques in artificial neural networks backpropagation. IOP Conf. Series: Materials Science and Engineering 495 (2019).
8. Ruder, S.: An overview of gradient descent optimization Algorithms, https://arxiv.org/abs/1609.04747
9. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat., 22(3), pp. 400-407 (1951).
10. Wasan, M.: Stochastic Approximation. Cambridge University Press (1969).
11. Oppermann, A.A.: Optimization Algorithms in Deep Learning, https://www.deeplearning-academy.com/p/ai-wiki-optimization-algorithms
12. Schmidt, M., Le Roux, N. and Bach, F.: Minimizing finite sums with the stochastic average gradient. Technical report, INRIA, hal-0086005 (2013).
13. Johnson, R. and Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In Advances in Neural Information Processing Systems, pp. 315–323 (2013).
14. Defazio, A., Bach, F., Lacoste-Julien, S.: SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives https://arXiv:1407.0202v3

15. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, vol. 4, no. 5, pp. 1-17 (1964).
16. Nesterov, Y.: A method of solving a convex programming problem with convergence rate O(1/sqr (k)). Soviet Mathematics Doklady, 27, pp. 372-376 (1983).
17. Bengio, Y., Boulanger-Lewandowski, N., Pascanu, R.: Advances in optimizing recurrent networks. Proc. ICASSP, pp. 8624-8626, (2013).
18. Duchi, J., Hazan, E., Singer, Y.: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. J. of Machine Learning Research, 12, pp. 2121-2159 (2011).
19. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recentmagnitude. COURSERA: Neural Networks Mach. Learn., 4, 26–31 (2012).
20. Zeiler, M.D.: ADADELTA: An Adaptive Learning Rate Method, http://arxiv.org/abs/1212.57012012.
21. Kingma, D.P., Ba, J.L.: Adam: a Method for Stochastic Optimization. 2nd Int. Conf. Learning Representations, ICLR 2014, Banff, AB, Canada, pp.1-13, April 14-16, 2014.
22. Dozat, T.: Incorporating Nesterov Momentum into Adam. Technical report, Stanford University, Tech. Rep., (2015).
23. Jiang, Z., Balu, A., Tan, S.Y., Lee, Y.M., Sarkar, C.S.: On Higher-order Moments in Adam, https://arXiv:1910.06878v1
24. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. Int. Conf. Learning Representations (ICLR), Vancouver, Canada, 23 p., Apr -May 2018.
25. Wu, X.. Ward, R., Bottou, L.: WNGrad: Learn the Learning Rate in Gradient Descent, https://arXiv:1803.02865v1.
26. Chen, J., Gu, Q.: Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks, https://arXiv:1806.06763v1
27. Zhou, Z., Zhang, Q., Lu, G., Wang, H., Zhang, W., Yu Y.: AdaShift: decorrelation and convergence of adaptive learning rate methods, https://arXiv:1810.00143v4
28. Keskar, N.S., Socher, R.: Improving Generalization Performance by Switching from Adam to SGD, https://arXiv:1712.07628v1
29. Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Le, Q.V. and Ng, A.Y.: On optimization methods for deep learning. In Proc. of the 28th Int. Conf. on Machine Learning (ICML-11), pp. 265–272 (2011).
30. Zinkevich, M., Weimer, M., Smola, A.J., Li, L.: Parallelized stochastic gradient descent., Advances in neural information processing systems 23 (23), pp. 2595–2603 (2010).
31. Recht, B., Re, C., Wright, S. and Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 693–701 (2011).
32. Daninga, C., Shiganga, Li, Yunquana, Z.: Weighted parallel SGD for distributed unbalanced-workload training system, https://arXiv:1708.04801v1
33. Keuper, J., and Pfreundt F.-J.: Asynchronous Parallel Stochastic Gradient Descent A Numeric Core for Scalable Distributed Machine Learning Algorithms, https://arXiv:1802.09941v2
34. Chu, C.T., Kim, S.K., Lin, Y.A., Yu, Y., Bradski, G.R., Ng, A.Y., and Olukotun, K.: Map-reduce for machine learning on multicore. In Proc. of Neural Information Processing Systems Conf. NIPS '06, . MITPress, pp. 281–288 (2006).
35. Flower Datasets, http://www.robots.ox.ac.uk/~vgg/data/flowers/
36. An end-to-end open source machine learning platform, https://www.tensorflow.org/