

Parameter Tuning Using Adaptive Moment Estimation in Deep Learning Neural Networks

Emmanuel Okewu[1(✉)], Sanjay Misra[2], and Fernandez-Sanz Lius[3]

[1] Centre for Information Technology and Systems, University of Lagos, Lagos, Nigeria
eokewu@unilag.edu.ng
[2] Department of Electrical and Information Engineering, Covenant University, Ota, Nigeria
sanjaymisra@covenantuniversity.edu.ng
[3] Department of Computer Sciences, University of Alcala, Alcalá de Henares, Spain
luis.fernandez.sanz@uah.es

Abstract. The twin issues of loss quality (accuracy) and training time are critical in choosing a stochastic optimizer for training deep neural networks. Optimization methods for machine learning include gradient descent, simulated annealing, genetic algorithm and second order techniques like Newton's method. However, the popular method for optimizing neural networks is gradient descent. Overtime, researchers have made gradient descent more responsive to the requirements of improved quality loss (accuracy) and reduced training time by progressing from using simple learning rate to using adaptive moment estimation technique for parameter tuning. In this work, we investigate the performances of established stochastic gradient descent algorithms like Adam, RMSProp, Adagrad, and Adadelta in terms of training time and loss quality. We show practically, using series of stochastic experiments, that adaptive moment estimation has improved the gradient descent optimization method. Based on the empirical outcomes, we recommend further improvement of the method by using higher moments of gradient for parameter tuning (weight update). The output of our experiments also indicate that neural network is a stochastic algorithm.

Keywords: Adaptive moment estimation · Deep learning · Neural networks · Error function · Parameter tuning

1 Introduction

Deep learning is an intelligent software process for extracting features of data using multiple layers of computational neurons [1]. Deep learning (DL) uses deep neural networks (DNN) as its main architecture [2]. Others include DL architectures are deep belief network, deep random forests [3], neural processes [4], deep gaussian processes [5].

The training process of DNN involves nonlinear transformation of input data which creates a statistical model (output) and improvement of the model using partial

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12254, pp. 261–272, 2020.
https://doi.org/10.1007/978-3-030-58817-5_20

derivative (mathematical method). DNN training is characterized by forward transfer of input information and backward transfer of error. In forward transfer process of information, the input data is transferred layer by layer from the input level to the output level. The data reaching the output layer is compared with the expected (actual) data to ascertain if the error is within acceptable limits. Else, the error is transferred backward. The forward pass and backward pass are illustrated in Fig. 1.

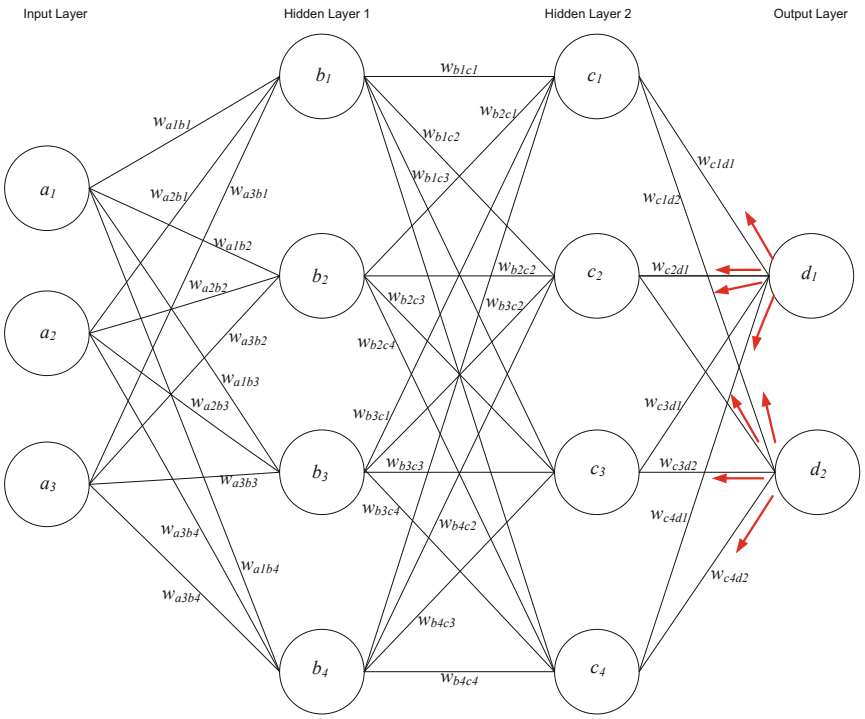

Fig. 1. Forward pass and backward pass in deep neural networks

The forward pass can be vectorized and matrix operations applied as neural network operations are largely matrix operations [6]. During the backward process, the error signal is transferred layer by layer from the output layer to the input layer. The error is distributed to neurons of each layer proportionate to the contributing parameter (weight) values, producing the error signal of each-layer neurons.

Refining and updating link weights, otherwise referred to as parameter tuning, requires a mathematical relationship between the parameters (weights) and errors so that changes made to one entity bring about a change in the other. The aim is to minimize the neural network's error and refine the parameter which is the neural network link weights. One of the commonly used optimization algorithms for achieving the task is Gradient Descent [1]. Gradient descent is an iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function

using gradient descent, steps proportional to the negative of the gradient of the function at the current point are taken.

Given a simple neural network as shown in Fig. 2,

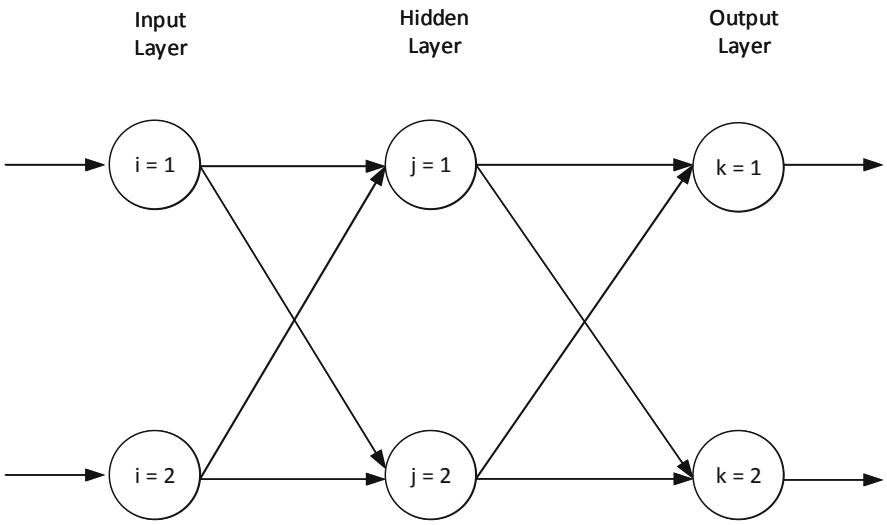

Fig. 2. A simple neural network

the task is to find how error E changes as the weight changes (the slope of the error function) towards a minimum. Mathematically, it is represented as:

$$\frac{\partial E}{\partial w_{jk}}$$

where E = error and W_{jk} = the link weight between layers.

Using activation function (say sigmoid), the slope of the error for link weights (parameters) between the hidden and the output layers as shown in Fig. 2 above is represented as [6]:

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - O_k).\text{sigmoid}\left(\sum jW_{jk}.O_j\right)\left(1 - \text{sigmoid}\left(\sum jW_{jk}.O_j\right)\right).O_j$$

The slope of the error function for any other weights is:

$$\frac{\partial E}{\partial w_{ij}} = -(e_j).\text{sigmoid}\left(\sum jW_{ij}.O_i\right)\left(1 - \text{sigmoid}\left(\sum jW_{ij}.O_i\right)\right).O_i$$

where e_j = recombined back propagated error out of the hidden nodes
W_{ij} = weights into a hidden node j
W_{jk} = weights into an output node k
O_k = output from a node k
O_i = output from a node i

Parameter tuning (refinement of weight) is done using the formula:

$$\text{New } w_{jk} = \text{old } w_{jk} - \eta . \frac{\partial E}{\partial w_{jk}}$$

where η is the learning rate, a factor that moderates the strength of the changes so that loss function does not overshoot the minimum.

For weights between hidden and output layer, the update is as follows:

$$\text{New } w_{jk} = \text{old } w_{jk} - \eta . \frac{\partial E}{\partial w_{jk}}$$

While parameter tuning for weights between input and hidden layer is as follows:

$$\text{New } w_{ij} = \text{old } w_{ij} - \eta . \frac{\partial E}{\partial w_{ij}}$$

However, some challenges of parameter tuning (weight update) which adversely impact on quality loss and training time are: (1) the step size (learning rate) for increasing weight could be very small, making learning slow and in extreme cases, learning is halted [7, 8]; (2) changes in weights alter the loss function towards the local minimum but in some instances, the movement could be impeded by paths of slow convergence surrounding saddle points [9]; (3) the presence of cost (weight) plateau resulting in the presence of many poor local minima [10]; (4) frequent weight update could lead to fluctuations in loss function which prolongs learning time [11]; (5) some weights are less frequently and scantily updated, resulting in sparse gradient – gradient that approaches slope slowly [12]; and (6) some parameter tuning (weight updates) may not be in the direction of local minimum.

2 Background and Related Work

2.1 State of Optimizers

Deep neural network optimizers like Adam, RMSProp, SGD, Momentum share the common goal of helping DNN models to learn and optimize network parameters like weights and bias after so many iterations. The optimizers also help to minimize errors. This implies that saving the state of an optimizer tantamount to saving the values of learnt network parameters [13]. Also, saving a model means saving the parameters'

values which the model has learnt. In addition, saving a model translates into saving the entire network's architecture and the units in every layer (input, output and hidden), and also the results (loss and accuracy scores of models).

2.2 Moving Averages of Gradients and Learning Rate Adaptability in Deep Neural Networks

One of the strategies for aligning parameter updates with direction of gradient descent (local minimum) is the use of moving averages of gradients. Momentum changes in per-parameter updates as outlined in [14] was used in Adam stochastic optimization algorithm with first and second moments of gradients. The first moment connotes exponentially decreasing mean of past gradients and second moment refers to exponentially decreasing average of past squared gradients. The use of moving averages of gradients by popular deep learning algorithms like RMSProp and Adam has confirmed that the strategy enhances convergence rate and speeds up network training [15].

Moving average is a statistical concept also known as rolling average or running average relevant in deep learning optimization [16, 17]. It is a computation that focuses on analyzing data points through the creation of range of means of different subsets of complete dataset. It is equally known as rolling mean or moving mean (MM). On the other hand, the term 'moment' emanates from physics where the moment of a system of point masses is calculated using a formula to find the center of mass of the points. Also, in statistics, moments measure something relative to the center of the values though the values may not be masses. Moments in mathematical statistics involve basic calculations which are useful in finding probability distribution's mean, variance, and skewness [17].

Given a set of past gradients g_1, g_2, g_3, …, g_n, as data points, the sth moment is given by the formula:

$$\mathrm{Sth} = \left(g_1^s + g_2^s + g_3^s + \ldots + g_n^s\right)/n$$

Hence, first, second, third, fourth and even higher moments of gradients generated during neural network training can be calculated and used for updating individual parameters using momentum changes [19]. This facilitates convergence rate and speeds up neural network training.

2.3 Decay Rates for Deep Neural Networks

Stochastic gradient descent with a simple decay is not suitable for training deep neural networks [20]. This is because aligning parameter updates with the direction of gradient descent (local minimum) takes prolonged time and in extreme cases, the updates may tend in different direction. As a result, convergence may be a mirage. The tactics of gradient descent as a stochastic optimization technique revolves around obtaining optimal parameter values from initial random parameter values that guarantees minimal loss function. To prevent parameters from languishing on cost plateaus, there has been shift from updating parameters with constant and simple decay rate to parameter update adaptation to slope of loss function as in Momentum and Nesterov, adapting updates to

individual parameter by calculating individual learning rates as in Adagrad, AdaDelta, RMSProp, and adapting updates to individual parameter by calculating individual parameter momentum changes as in Adam [14]. The proposed Adum-Aiona algorithm in this study uses higher order exponential decay rates for per parameter updates in a bid to tackle challenges faced by other optimization techniques like vanishing learning rate, parameters languishing on cost plateaus, high variance in parameter updates, sparse gradients, presence of many poor local optima, and saddle points.

2.4 Extensions and Variants of Stochastic Gradient Descent

Gradient descent is used for obtaining the minimum of a loss function. In a bid to obtain the local minimum, steps proportional to approximate gradient (or the gradient's negative) at the present point are taken. Conversely, gradient ascent refers to taking steps proportional to the gradient's positive which leads to the function's local maximum.

Though gradient descent is equally referred to as steepest descent, it is not the same as the steepest descent method which approximates integrals. Since optimization of the error function in DNN is a minimization problem, we use gradient descent.

There have been several proposals on the improvement of the basic gradient descent algorithm with particular focus on learning rate annealing and robust parameter updates for faster convergence. In machine learning specifically, setting a learning rate (step size) is a difficult task as setting the configuration parameter (learning rate) very high makes the algorithm not to converge while a very low learning rate ensures slow convergence.

For an iteration number t, the learning rate is essentially made its decreasing function η_t in order to fast-track convergence. This has been the focus of extensions of stochastic gradient descent. Hence, learning rate schedule should ensure first set of iterations cause large changes in the parameters just as fine-tuning is done by later set of iterations until the local minima and optimal parameter values are attained [21].

2.5 Per-Parameter Tuning Using Adaptive Moment Estimation

Per-parameter tuning using adaptive learning rate has proven to be beneficial for improved loss quality and training time. To further enhance convergence rate and network training speed, algorithms like Adam and RMSProp use adaptive moment estimation for updating parameters. The compute and separately store per parameter momentum changes. While RMSProp uses first moment of gradient, Adam uses first and second moments of gradient [8, 14].

Based on the outcome of software simulation conducted in the course of this study, we confirm that both adaptive moment estimation techniques (RMSProp and Adam) offer completive results in terms of loss quality (accuracy) and training time.

2.6 Related Works

Previous works on parameter tuning (weight update) in neural networks and adaptive moment estimation in deep learning are as follows.

In the work of Kim and Fessler [11], discussion centered on first-order iterative methods like gradient descent. The authors stressed that first-order techniques usually escape saddle points by following gradient direction (negative curvature). However, there are instances where plateaus surrounding saddle points known as regions of small curvature slow down first order methods, creating false impression of local minimum. In a bid to solve the problem of saddle point, some researchers have suggested a second order method that tackles the issue with Newton's method. Nonetheless, owing to computational requirements, second order methods are unfit for the training of large-scale models like DNNs. As a result, this study does not consider these approaches but focuses on first order methods which rely solely on gradient information.

Mei [22] focused on the development of stochastic gradient descent (SGD) as a variant of gradient descent for solving the issue of delayed training time synonymous with basic gradient descent. Prior to executing parameter update, basic (or batched) gradient descent computes gradient for entire training examples. This accounts in part for prolonged training time which incremental gradient descent or SGD was designed to address. In essence, gradient descent optimization is stochastically approximated. Because data instances are chosen at random or shuffled, it is referred to as stochastic. This is in contrast to standard gradient descent in which samples constitute single group or better still, according to their arrangement in training dataset. Parameter is tuned (updated) for each training instance. SGD is faster than standard gradient descent owing to the fact that it performs update per training instance. However, frequent updates exhibit high variance, stimulating fluctuation to different intensities by the loss function. Also, due to frequent updates, it can overshoot local minimum. Therefore, a technique like adaptive moment estimation that aligns parameter tuning (update) in the direction of gradient is required.

As part of efforts to align parameter tuning in the direction of gradient and tackle the problem of dropping learning rates for improved loss quality (accuracy) and training time, Tieleman and Hinton [8] developed RMSProp. The algorithm is an extension of SGD that focuses on per parameter learning. It uses momentum on rescaled gradient to compute parameter updates. The mean of past values of gradients is used to adapt the learning rate. This process involves calculating the running mean of recent gradient for a parameter and dividing the parameter's learning rate by the running mean. In many applications, the algorithm has demonstrated good adaptation of learning rate. It also works well on non-stationary and online problems like noisy problems on account of how fast it is changing. However, there is no bias-correction term in RMSProp.

Adam stochastic algorithm was proposed by Kingma and Ba [14]. The method is guided by the realization that the usefulness of computing per parameter learning rate as demonstrated in the SDG algorithms examined above is an eye opener that the computation and separate storage of per parameter momentum changes could be helpful in the further improvement of quality loss and training time of neural network. The meaning of Adam is Adaptive Moment Estimation. A popular algorithm for training DNN, it incorporates the merits of RMSProp and Adagrad. Its strategy involves using running mean of previous gradients in determining descent's direction (direction of local minimum) while simultaneously modifying learning rate using the moving average of previously squared gradients. Adam is an improvement on the

RMSProp optimizer. Whereas RMSProp uses only the first moment, Adam utilizes moving averages of second and first moments. Also, its bias-correction term helps it to outperform RMSProp tending towards the end of optimization when gradients get smaller. However, there is room for further improvement [15].

3 Methodology

Though optimization methods for machine learning include gradient descent, simulated annealing, genetic algorithm and second order techniques like Newton's method, we focused on gradient descent algorithms which have been adjudged as best suitable for deep neural networks [14]. We primarily investigated deep neural networks as a popular machine learning technique. Also, the availability of deep learning libraries is a motivating factor.

Our problem focus is ascertaining the performances of existing stochastic optimizers in terms of loss quality and training time. Additionally, we intend to verify if those that use adaptive moment estimation (such as Adam and RMSProp) for parameter tuning have gains over others. Our motivation is to leverage on our findings to propose a new stochastic gradient descent algorithm which outperforms existing ones.

Series of experiments were conducted using selected deep learning neural network optimizers such as Adam, RMSProp, [23]. Python deep learning libraries (Keras and TensorFlow) were used while the MNIST database of images of handwritten digits was used as dataset. The model (architecture) used is convolutional neural networks (CNN). For the purpose of the experiments, the database was broken into two datasets – training dataset and testing dataset. While the training data was used to fit (train) the model, the testing dataset was used to evaluate the model. Further analysis of the MNIST database as used for our experiments is as follows:

Training Data

Total examples (images) = 60,000 (images of handwritten digits)
Pixels per image = 28 x 28 = 784
Size (shape) of training (image) data = [60000,784]

Testing Data

Total instances (images) = 10,000
Pixels per image = 28 x 28 = 784
Size (shape) of training data = [10000, 784]

Label Data

Total examples (images) = 60,000
Total values in label = 60,000
Size (shape) of label data = [60,000]

4 Results and Interpretations

A total of seven experiments were performed using Python deep learning libraries (Keras API and Tensorflow). In experiments 1 and 7, the optimizer tested was Adam, a stochastic optimization method that uses adaptive moment estimation. In experiments 2 and 3, same optimizer (Adagrad) was tested. In experiment 4, we examined Adadelta while experiment 5 focused on RMSProp, another stochastic method that uses adaptive moment estimation. In experiment 6, sgd was examined. The essence of testing some optimizers twice was to empirically verify the stochastic (non-deterministic) behaviour of gradient descent algorithms. Though the output showed metrices like loss, accuracy and training, our major metric focus is accuracy. Each experiment is considered to be a study case and below is a case-by-case analysis.

Experiment 1

Optimizer = Adam
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.98209 = 98.2%

Experiment 2

Optimizer = Adagrad
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.97599 = 97.6%

Experiment 3

Optimizer = Adagrad
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.97739 = 97.7%

Experiment 4

Optimizer = Adadelta
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.98363 = 98.4%

Experiment 5

Optimizer = RMSProp
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.98199 = 98.2%

Experiment 6

Optimizer = sgd
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.94943 = 94.9%

Experiment 7

Optimizer = Adam
Training data shape (dimension) = (60000, 28,28,1)
Number of images in training dataset = 60,000
Number of images in testing dataset = 10,000
Number of epochs (iterations) = 10
Mean Accuracy = 0.98286 = 98.3%

Experiments 2 and 3 confirmed that deep learning is a stochastic (non-deterministic) optimization process in the sense that for the two different experiments, same optimizer (Adagrad) was used on the same dataset of MNIST handwritten images, yet different accuracy outcomes (97.6% and 97.7%) were obtained. In the same vein, experiments 1 and 7 that used Adam as sole optimizer produced different results (98.2% and 98.3%) even with the same dataset. Conversely, deterministic optimizers produce same outputs regardless of the number of times the experiment is performed using same dataset.

Also, though all the variants and extensions of gradient descent as the main optimization method for deep learning neural networks have steadily contributed to improved loss quality (accuracy) and reduced training time, the results in experiments 1,5 and 7 indicate that methods that incorporate adaptive moment estimation like RMSProp and Adam posted impressive and competitive outcomes. This indicates that adaptive moment estimation could be further explored in enhancing the loss quality (accuracy) and training time of deep learning.

5 Conclusion

In this study, neural network algorithms for deep learning were identified and experimentally evaluated using Python deep learning libraries and the MNIST database of handwritten images. Though these extensions of gradient descent as the cardinal

optimization method for deep learning neural networks have shown significant improvement in loss quality and training time overtime, there is room for further improvement [15]. In particular, experimental outcomes showed that optimizers like RMSProp and Adam that use adaptive moment estimation are posting improved results.

In future work, we shall propose a fresh stochastic optimization algorithm called Adum-Aiona that uses adaptive moment estimation but with higher (four) moments of gradients for per-parameter tuning (update). Adum-Aiona shall be simulated and benchmarked against established optimizers evaluated in this work to ascertain any gains in terms of loss quality (accuracy) and training time.

References

1. Brownlee, J.: How to choose loss functions when training deep learning neural networks. In: Deep Learning Performance (2019)
2. Shridhar, K.: A beginners guide to deep learning (2017)
3. Zhou, Z., Feng, J.: Deep forest: towards an alternative to deep neural networks. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 3553–3559. AAAI Press (2017)
4. Garnelo, M., Schwarz, J., Rosenbaum, D., Rezende, V.F., Eslami, S.M., Teh, Y.W.: Neural processes, arXiv preprint arXiv:1807.01622 (2018)
5. Damianou, A., Lawrence, N.: Deep Gaussian processes. In: Artificial Intelligence and Statistics, pp. 207–215 (2013)
6. Pandey, P.: Demystifying neural networks: a mathematical approach (Part 2) (2018)
7. Zeiler, M.D.: Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701 (2012)
8. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
9. Dauphin, Y.N., Pascanu, R., Caglar, G., Kyunghyun, C., Ganguli, S., Bengio, Y.: Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Advances in Neural Information Processing Systems, pp. 2933–2941 (2014)
10. Kawaguchi, K.: Deep learning without poor local minima. In: Advances in Neural Information Processing Systems (NIPS) (2016)
11. Kim, D., Fessler, J.A.: Optimized first-order methods for smooth convex minimization. Math. Prog. **151**, 8–107 (2016)
12. Aji, A.F., Heafield, K.: Combining global sparse gradients with local gradients. In: ICLR Conference (2019)
13. Walia, A.S.: Types of optimization algorithms used in neural networks and ways to optimize gradient descent (2017)
14. Kingma, D.P., Ba, J.L.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations (2015)
15. Koushik, J., Hayashi, H.: Improving stochastic gradient descent with feedback. In: Conference Paper at ICLR (2017)
16. Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging (PDF). SIAM J. Control Optim. **30**(4), 838–855 (1992)

17. Zhang, S., Choromanska, A., LeCun, Y.: Deep learning with elastic averaging SGD. In: Neural Information Processing Systems Conference (NIPS) (2015)
18. Davies, C., Dembinska, A.: Computing moments of discrete order statistics from non-identical distributions. J. Comput. Appl. Math. **328**(15), 340–354 (2018)
19. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural Netw. Official J. Int. Neural Netw. Soc. **12**(1), 145–151 (1999)
20. Lockett, A.: What is the most popular learning rate decay formula in machine learning? The University of Texas at Austin (2012)
21. Darken, C., Chang, J., Moody, J.: Learning rate schedules for faster stochastic gradient search. In: Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop (1992)
22. Mei, S.: A mean field view of the landscape of two-layer neural networks. In: Proceedings of the National Academy of Sciences (2018)
23. Okewu, E., Adewole, P., Sennaike, O.: Experimental comparison of stochastic optimizers in deep learning. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11623, pp. 704–715. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24308-1_55