MULTI- STAGE MOMENT-BASED OPTIMIZATION: ANALYSIS AND APPLICATION OF THE ADAM ALGORITHM

[1]Muminov E.N.
[2]Tillaboev, A. A.
[3]Qobilov S.Sh.
[1]Teacher of the Department of "Artificial Intelligence" at the Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi
Gaspower@mail.ru
[2]Teacher of the Department of "Artificial Intelligence" at the Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi
emuminov864@gmail.com
[3]Teacher of the Department of "Artificial Intelligence" at the Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi
qobilov.sirojiddin92@gmail.com
https://doi.org/10.5281/zenodo.15662774

In the era of deep learning and large-scale artificial intelligence systems, the importance of efficient optimization algorithms has significantly increased. Neural networks, particularly those with deep and complex architectures, rely heavily on gradient-based iterative methods to update model parameters by minimizing a loss function. Among these methods, the Adam (Adaptive Moment Estimation) algorithm has emerged as a widely adopted solution due to its adaptive learning capability and robust convergence behavior. Originally introduced by D. Kingma and J. Ba in 2015, Adam integrates the advantages of both Stochastic Gradient Descent (SGD) and RMSprop algorithms, addressing several limitations of traditional approaches, such as fixed learning rates, slow convergence, oscillatory updates, and sensitivity to noisy gradients [1][2].

Characterized by its multi-step update structure, Adam does not rely solely on the current gradient but leverages a statistically smoothed history of past gradients. This cumulative memory effect mimics the behavior of momentum-based methods, guiding the optimization trajectory through complex parameter landscapes more effectively. The algorithm's inherent adaptability, low hyperparameter sensitivity, and empirical success across a wide range of deep learning models—such as Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs), and Transformer-based architectures—have solidified its role as a de facto optimization standard in modern machine learning practice [4].

This thesis provides a comprehensive analysis of Adam's mathematical formulation, including moment calculations, bias correction, and the parameter update rule. We also examine the practical advantages of the algorithm through comparative experiments and assess its impact on training efficiency and accuracy. Finally, the application of Adam in real-world programming environments is considered to demonstrate its robustness and flexibility in diverse machine learning scenarios [1].

Advantages of Adam's multi-step nature

- Fast convergence — Moments smooth the gradient direction, which cognitively serves to find the path that can "go down faster" [3].
- Resistant to noisy gradients — Exponential averaging filters out random fluctuations and makes the update stable [5].

➢ Sparse features — For dimensions with low variance, the denominator is small, resulting in a more significant update being applied to them [4].

➢ Hyper-parameters are less sensitive — β_1, β_2, ϵ, and η operate uniformly over a wide range, reducing the need for lengthy manual model tuning [2].

Mathematical formulas of Adam's algorithm

The main strength of the Adam algorithm is that it relies on precise mathematical formulas to stabilize the gradient optimization process and provide adaptive learning rates for individual parameters. This section describes all the mathematical mechanisms of the Adam algorithm in sequence: exponential smoothing, moment calculation, bias correction, and parameter update formulas. Suppose that during the training of a neural network, we calculate the gradient of the loss function at each time point t:

$$g_t = \nabla_{\theta_t} \delta(\theta_t)$$

where θ_t is the set of model parameters at time t, δ is the loss function. The Adam algorithm calculates two statistical measures by smoothing this gradient: the first and second moments [9].

First Moment: Average Gradient

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t,$$

here: m_t — exponentially smoothed average gradient (first moment), $\beta_1 \in [0,1)$ — inertia or momentum coefficient, usually $\beta_1 = 0.9$. This formula is similar to the momentum technique, reducing fluctuations in the gradient and stabilizing the optimization direction [10].

Second Moment: Mean of Squares of Gradient

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

here: v_t — exponential mean of the squares of the gradient (second moment), $\beta_2 \in [0,1)$ — usually $\beta_2 = 0.999$. Learning rate for each parameter [9,11]. These adjustments are especially important in the early iterations, as the gradients change rapidly during the initial stages of training [9].

Parameter update formula: Using bias-corrected moments, the Adam algorithm updates the parameters according to the following formula:

$$\theta_t = \theta_{t-1} - \gamma \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

here: γ — overall learning rate, usually $\gamma = 0.001$, $\epsilon \approx 10{-}8$ is a small number that prevents division by zero.

In this formula, the presence of $\sqrt{\hat{v}_t} + \epsilon$ in the denominator provides a step length that is adjusted according to the gradient variance of each parameter. If the gradient is small, the step is larger, and if the gradient is large, the step is smaller. Thus, the learning rate of the model is automatically adjusted to each parameter [10,12].

If we analyze the formulaic characteristics of Adam, we should list them with special attention to the following.

➢ The momentum element m_t *moves in the general* direction of the spline gradient, which smoothes the convergence compared to SGD.

➢ Through variance scaling, *the* v_t elements control the level of parameter update.

➢ Bias correction balances out initial irregularities and gives the model the desired inertia effect.

➢ The above formulas protect the model from excessive drift, especially on a noisy loss surface.

Multi-step approaches ensure that the model **converges even under stochastic noise**. This is because the model adjusts its direction of motion by storing information about previous gradients. This is especially useful in cases where the loss surface is uneven.

To evaluate the practical effectiveness of the Adam optimization algorithm, a comparative experiment was conducted using a simple Convolutional Neural Network (CNN) on the MNIST dataset, which consists of 28×28 grayscale images representing handwritten digits across ten classes. Two identical CNN models were trained: one using the Adam optimizer and the other using the classical Stochastic Gradient Descent (SGD). Both models were trained over 25 epochs, and the validation accuracy was recorded after each epoch to analyze performance trends.

The results of this experiment are illustrated in Figure 1 and Table 1. As shown in the figure, Adam consistently achieves higher validation accuracy compared to SGD at every epoch. From the very first epoch, Adam leads with a significantly higher accuracy (95.28% vs. 86.20%) and maintains this advantage throughout the training process. While SGD shows a gradual improvement in accuracy across epochs, Adam converges much faster and reaches a stable high-performance plateau around 98% accuracy from epochs 8 onward.

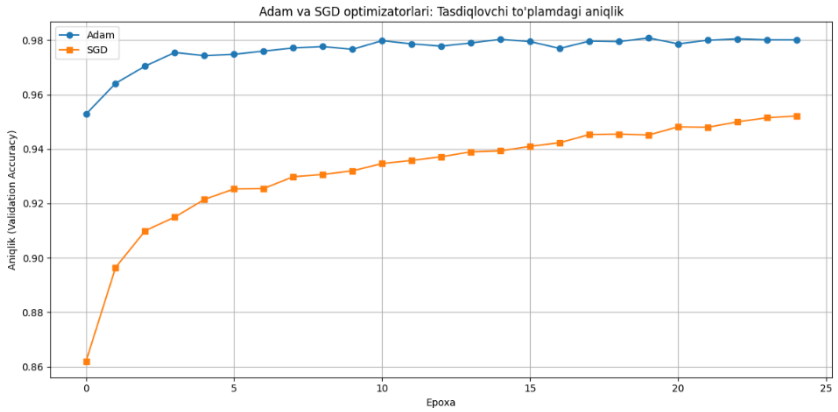

Figure 1. Validation accuracy over 25 epochs for Adam and SGD optimizers on the MNIST dataset.

Table 1. Validation accuracy per epoch for Adam and SGD optimizers (25 epochs).

Epoch	Adam Accuracy	SGD Accuracy
1	0.9528	0.8620
2	0.9642	0.8965
3	0.9705	0.9100
4	0.9755	0.9150
5	0.9743	0.9215
6	0.9748	0.9253
7	0.9760	0.9255
8	0.9785	0.9298
9	0.9777	0.9360

10	0.9798	0.9347
11	0.9798	0.9360
12	0.9798	0.9385
13	0.9778	0.9372
14	0.9790	0.9372
15	0.9803	0.9393
16	0.9795	0.9410
17	0.9800	0.9423
18	0.9797	0.9453
19	0.9800	0.9452
20	0.9883	0.9467
21	0.9800	0.9480
22	0.9800	0.9480
23	0.9800	0.9505
24	0.9802	0.9515
25	0.9802	0.9521

These results strongly highlight the advantages of Adam's multi-step update strategy, which uses exponentially decaying averages of both the first moment (mean) and second moment (squared gradient) to adaptively adjust learning rates for each parameter. In contrast, SGD applies a uniform learning rate, which limits its responsiveness to gradient dynamics and often requires manual tuning to achieve similar performance.

In addition to faster convergence, the Adam optimizer also demonstrates low sensitivity to hyperparameter configuration. In this experiment, a standard learning rate of 0.001 was used without further tuning, yet Adam still outperformed SGD across all epochs. Although Adam requires slightly more memory to store additional moment estimates per parameter, this cost is negligible on modern computing hardware (e.g., GPUs, TPUs) and is offset by the significant gains in both learning speed and final accuracy.

In conclusion, the combination of adaptive parameter updates, fast and stable convergence, and robustness to noisy or sparse gradients makes Adam a superior optimizer for training deep learning models. Its empirical success across a wide range of neural architectures justifies its widespread adoption in modern AI systems.

References:
Используемая литература:
Foydalanilgan adabiyotlar:

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
2. Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Neural Networks: Tricks of the Trade (pp. 437–478). Springer.
3. Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. 2017.
4. Tieleman, T., & Hinton, G. (2012). Lecture 6.5 – RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.
5. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning

and stochastic optimization. Journal of Machine Learning Research, 12, 2121–2159.
6. LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient backprop. In Neural Networks: Tricks of the Trade (pp. 9–48). Springer.
7. Zhang, J., & Mitliagkas, I. (2020). Lookahead optimizer: k steps forward, 1 step back. In Advances in Neural Information Processing Systems, 32.
8. Bock, C., & Gumbsch, P. (2020). Optimization of deep neural networks: Recent advances and applications. Journal of Computational Science, 45, 101182.
9. Li J., Li X., & Hoi, S.C. (2018). Learning to optimize: A primer and a benchmark.
10. Reddi, S.J., Kale, S., & Kumar, S. (2019). On the convergence of Adam and beyond. In International Conference on Learning Representations (ICLR).