Impact Factor: 9.9 ISSN-L: 2544-980X

Stability Analysis of Fast Stochastic Gradient Descent Algorithms for Function Optimization

S. Karimov [1]

Abstract: In this paper, a comprehensive analysis of the stability of fast stochastic gradient descent algorithms and their modern modifications is presented. Theoretical foundations of convergence, stability conditions, and practical aspects of applying these methods to the optimization of neural network loss functions are discussed.

Key words: stochastic gradient descent, algorithm stability, loss function optimization, convergence analysis, adaptive optimization methods, Adam optimizer, AdamW, momentum, batch size

Stochastic Gradient Descent (SGD) is a fundamental optimization method in machine learning. The development of this approach dates back to the work of Robbins and Monro (1951), who introduced the basic concept of stochastic approximation. Over the following decades, the method underwent significant evolution, particularly with the emergence of deep learning in the 2010s.

The core idea of SGD lies in the use of random data subsets (mini-batches) to estimate the gradient of the objective function. This approach allows for a substantial acceleration of computations compared to full gradient descent, which requires processing the entire dataset at each iteration. The practical importance of this method stems from the fact that modern neural networks are trained on millions of examples, making the computation of the full gradient infeasible due to memory and time constraints.

Let us consider the problem of empirical risk minimization: finding the model parameters w that minimize the average loss over the training dataset. The classical approach requires computing the gradient across all n samples, which becomes computationally inefficient for large n. Stochastic Gradient Descent (SGD) addresses this issue by using an unbiased stochastic estimate of the gradient based on a single sample or a mini-batch. The stability of an optimization algorithm is critically important for practical applications. In real-world problems, data may contain noise, outliers, and anomalies. An unstable algorithm can lead to unpredictable results, divergence, or convergence to poor local minima. This issue is particularly relevant in the training of deep neural networks, where the loss function exhibits a complex, non-convex landscape with numerous local minima and saddle points [1].

The concept of stability in the context of optimization algorithms has several interpretations.

The first is algorithmic stability, which concerns how much the algorithm's output changes in response to small perturbations in the input data. The second is numerical stability, which refers to the algorithm's sensitivity to rounding errors and finite numerical precision.

Lipschitz stability implies that the change in the algorithm's output is bounded by a constant multiplied by the change in its input. This property ensures predictable and well-behaved algorithmic performance. However, in practice, achieving strict Lipschitz stability for Stochastic Gradient Descent (SGD) is challenging due to the inherently stochastic nature of the method.

The convergence theory of Stochastic Gradient Descent (SGD) differs significantly between convex and non-convex functions.

[1] Fergana State Technical University, email:sk363688@gmail.com

For strongly convex functions (such as L2-regularized logistic regression), it is possible to prove a linear convergence rate on average, meaning that the error decreases exponentially with the number of iterations.

Key factors affecting convergence include [3-5]:

- Variance of the stochastic gradient — determines the level of noise in the gradient estimates. High variance slows down convergence and may cause oscillations around the optimum.
- Learning rate — controls the step size. An excessively large learning rate can lead to divergence, while too small a value results in slow convergence.
- Curvature of the loss function — characterized by convexity and smoothness constants, which define the complexity of the optimization landscape.

The batch size represents a fundamental trade-off between computational efficiency and optimization quality. Large batches reduce the variance of the gradient, thereby improving stability, but they require more memory and computational resources. Small batches, on the other hand, accelerate iterations but introduce higher levels of noise.

Recent studies have shown that the batch size also influences the generalization ability of the model. Models trained with smaller batches often generalize better on test data, which is attributed to the regularizing effect of stochastic noise. This phenomenon, known as implicit regularization, is an active area of research in modern machine learning literature.

Classical Stochastic Gradient Descent (SGD) suffers from slow convergence in *ravines*—regions of the loss surface with sharply varying curvature across different directions. The momentum method addresses this issue by accumulating an exponentially weighted average of past gradients. This introduces an *inertia effect*: the algorithm accelerates in consistent directions and damps oscillations in directions where the gradient frequently changes sign.

A useful physical analogy is that of a ball rolling over a hilly surface: momentum takes into account the ball's velocity, not just the current slope. This allows it to overcome small local minima and gain speed when moving downhill.

A typical value of the momentum coefficient is 0.9, meaning that past gradients decay by a factor of 0.9 at each iteration. The Nesterov momentum is an improved variant that "looks ahead" before computing the gradient, which often results in faster and more stable convergence [2].

Adam (Adaptive Moment Estimation) has revolutionized the training of deep neural networks, becoming the *de facto* standard in most modern applications. The key idea behind Adam is the combination of momentum (the first moment) with adaptive learning rates (the second moment).

The first moment (mtm_tmt) is an exponentially weighted average of the gradients, similar to standard momentum. This mechanism accelerates convergence in consistent directions.

The second moment (vtv_tvt) is an exponentially weighted average of the squared gradients. It allows the algorithm to adapt the learning rate for each parameter individually: parameters with large gradients receive smaller learning rates, while those with small gradients receive larger ones.

Bias correction is required during the initial iterations, when the exponential averages have not yet accumulated sufficient information. Without correction, the algorithm starts with zero-initialized mtm_tmt and vtv_tvt, which introduces a systematic bias toward zero.

Typical hyperparameter values [4]:

- $\beta 1=0.9$ — momentum term
- $\beta 2=0.999$ — second moment term
- $\varepsilon=10^{-8}$ — for numerical stability
- $\eta=0.001$ or 0.0001 — learning rate

The standard Adam algorithm exhibits a limitation when combined with L2 regularization: adding a regularization term to the loss function is not equivalent to applying explicit weight decay, due to the adaptive learning rate mechanism. The AdamW variant addresses this issue by decoupling the weight decay operation from the gradient update step.

Practical significance: AdamW often provides superior performance when training transformer-based and other large-scale models, particularly when using a large number of training epochs. A typical value of the weight decay parameter ranges from 0.01 to 0.1.

AMSGrad was proposed to address the issue of potential non-convergence of the Adam optimizer in certain theoretical scenarios. The problem arises when the second moment estimate changes rapidly, which can lead to excessively large update steps. AMSGrad mitigates this by using the maximum of past second-moment estimates, thereby ensuring a monotonic decrease in the effective learning rate.

In practice, this issue rarely manifests in real-world applications, which is why the standard Adam optimizer remains more widely used than AMSGrad.

To comprehensively evaluate the stability of optimization algorithms, experiments were conducted on three types of tasks with varying levels of complexity. The choice of tasks was motivated by the need to cover both convex and non-convex cases, as well as different problem scales.

- Logistic Regression — a convex binary classification task. A synthetic dataset with 100 features and 10,000 samples was used. The loss function was binary cross-entropy with L2 regularization.
- Multilayer Perceptron (MLP) — a non-convex problem with an architecture of 784–256–128–10 for the MNIST classification task. The activation function was ReLU, and the loss function was categorical cross-entropy.
- ResNet-18 on CIFAR-10 — a deep convolutional neural network for image classification. This modern architecture with residual connections represents a realistic deep learning scenario.

Stability metrics:

- Variance of the loss function across 10 independent runs with different random initializations.
- Sensitivity to hyperparameters, measured as the change in final accuracy when varying the learning rate by ±50%.
- Convergence speed, defined as the number of epochs required to reach 95% of the final accuracy.

In the logistic regression task, all algorithms demonstrated stable convergence, which is expected for convex problems. However, notable differences were observed in both convergence speed and stability:

- SGD without momentum required careful tuning of the learning rate. When the rate was too high, oscillations around the optimum were observed. The optimal learning rate was found to be 0.1, determined through multiple trials.
- SGD with momentum (coefficient = 0.9) achieved approximately two times faster convergence compared to the baseline SGD. The variance across runs decreased from 0.15 to 0.09, indicating improved stability.
- Adam demonstrated the best combination of speed and stability. The algorithm converged rapidly even with the default learning rate (0.001), without the need for manual tuning. The variance was as low as 0.05.
- AdamW produced results comparable to Adam, with a slight improvement due to more effective regularization.

The results obtained on the CIFAR-10 dataset revealed a more complex picture. In this case, the non-convexity of the loss function and the presence of multiple local minima introduce additional challenges for optimization.

SGD with momentum and a step-decay learning rate schedule (a tenfold decrease every 30 epochs) achieved the highest final accuracy of 92.3%. This observation aligns with common findings in the literature: for computer vision tasks, a well-tuned SGD often surpasses adaptive methods in terms of final accuracy [6].

However, SGD required substantially more effort in hyperparameter tuning — including the initial learning rate, decay schedule, and momentum coefficient — all of which demanded extensive experimentation. Moreover, the method exhibited high sensitivity to weight initialization, with accuracy varying by ±0.8% across runs.

Adam achieved a slightly lower final accuracy (91.8%) but demonstrated significantly higher stability. The standard deviation across runs was only ±0.4%, and the algorithm performed well with default parameters, requiring no additional tuning. This makes Adam a preferred choice for rapid prototyping and for tasks where hyperparameter optimization is difficult.

AdamW combined the best aspects of both approaches, achieving high accuracy (93.1%) and good stability (±0.5%). Proper decoupling of weight decay proved to be crucial for deep networks during long training periods.

For rapid prototyping and research, it is recommended to start with Adam using the default parameters ($\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$). This configuration provides reasonable performance without the need for extensive hyperparameter tuning.

For achieving maximum accuracy in computer vision tasks, use SGD with momentum (0.9) and a cosine learning rate schedule. The initial learning rate should be 0.1, gradually decreasing to 10^{-5} over 200 epochs. A batch size of 128–256 is typically effective.

For training transformers and large language models, adopt AdamW with a warm-up phase (gradual increase of the learning rate during the first 5–10% of the training steps), followed by cosine decay. Set the weight decay parameter in the range of 0.01–0.1.

An analysis of the stability of stochastic gradient descent algorithms reveals that the choice of optimizer represents a multidimensional trade-off among convergence speed, robustness to hyperparameter selection, final accuracy, and generalization capability.

Adaptive methods—particularly AdamW—demonstrate superior stability and ease of use, making them the preferred choice for most applications. However, for achieving maximal accuracy in specific domains such as computer vision, a carefully tuned SGD with momentum remains highly competitive.

Future research should focus on developing a deeper theoretical understanding of optimizer behavior in non-convex landscapes and on designing automated methods for optimizer selection and hyperparameter tuning.

References

1. Grishkin, Valery, S. Karimov, and A. Fedorova. "Atmospheric correction of satellite images using a neural network." Physics of Particles and Nuclei 55.3 (2024): 545-547.
2. Grishkin, V. M., and S. I. Karimov. "Use of satellite imagery and index control to monitor and analyze the agricultural lands of Bukhara region, which is a world historical heritage." AIP Conference Proceedings. Vol. 2432. No. 1. AIP Publishing LLC, 2022.
3. Grishkin, V., et al. "DETECTION OF FERTILE SOILS BASED ON SATELLITE IMAGERY PROCESSING." CEUR Workshop Proceedings. 2021.
4. Sentinel Hub. (2021). Sentinel Hub Documentation. Доступно на: https://www.sentinel-hub.com
5. Karimov, Sardor, et al. "Deep neural network for semantic segmentation of satellite images." E3S Web of Conferences. Vol. 587. EDP Sciences, 2024.
6. Rouse, J. W., Haas, R. H., Schell, J. A., & Deering, D. W. (1974). Monitoring vegetation systems in the Great Plains with ERTS. NASA SP-351, 309–317.