Proceedings of CONF-SEML 2026 Symposium: Importance of Machine Learning Methods and Analysis in Engineering
DOI: 10.54254/2755-2721/2026.AS32514

Comparative Study on the Synergy Between Deep Learning Optimization Algorithms and Hardware Acceleration Architectures

Longjie Hu

New Channel, Shenzhen, China
15974210677@163.com

Abstract: With the continuous scaling of deep learning models, increasingly stringent requirements have been imposed on computing efficiency, energy consumption control and computing architectures during the training and inference phases. The synergistic relationship between optimization algorithms and computing hardware has gradually become a pivotal research direction in the field of artificial intelligence. This paper conducts a systematic comparative study on deep learning optimization algorithms and hardware acceleration architectures, focusing on analyzing the performance discrepancies between gradient descent-based optimization methods and Adam and its improved variants across different computing platforms. By adopting the methodologies of literature review and comparative analysis, this paper explores typical optimization algorithms in terms of their convergence characteristics, stability, computational complexity and memory access patterns, and further integrates the analysis with the high parallel computing capability of GPUs as well as the dataflow computing paradigm and energy efficiency advantages of FPGAs. Special attention is paid to the applicability of different optimization algorithms in large-scale model training, convolutional neural networks, as well as embedded and energy-constrained application scenarios. The results demonstrate that adaptive optimization algorithms such as Adam exhibit faster convergence rates and stronger robustness in complex models and large-scale training tasks, whereas gradient descent-based methods still retain distinct advantages in application scenarios with relatively simple structures or limited computing resources.

Keywords: Deep learning optimization algorithm, Gradient Descent algorithm, Adam optimizer, Heterogeneous computing architecture, Software-hardware co-design

1. Introduction

As deep learning models grow in scale and complexity, the requirements for computing efficiency, resources, and energy management during training and inference are escalating. Thus, traditional computing paradigms are progressively encountering fundamental performance limitations. And the performance of deep learning models depends on optimization algorithms, which control factors like convergence speed, stability, and computational efficiency. Meanwhile, computing acceleration hardware represented by GPUs (Graphics Processing Units) and FPGAs (Field-Programmable Gate Arrays) is playing an increasingly indispensable role in artificial intelligence computing. Existing studies show that Gradient Descent (GD)-based optimization algorithms are widely adopted in deep learning training processes due to their well-established theoretical foundations and straightforward implementation. In contrast, Adam and its improved variants, leveraging the adaptive learning rate mechanism, exhibit better convergence in complex models and high-dimensional spaces. Moreover, GPUs have become the mainstream platform for large-scale model training by virtue of their high parallel computing capability, while FPGAs exhibit unique advantages in low-power, low-latency and embedded inference scenarios. By employing a literature review and comparative analysis, this paper compares the performance of GD-based algorithms and the Adam optimizer across different application scenarios and hardware platforms, and explores the relationship between optimization algorithms and computing architectures. Besides, it analyzes the intrinsic differences among various optimization algorithms in terms of convergence, resource consumption, and suitable scenarios. This study may enhance the understanding of the synergy between optimization algorithms and hardware acceleration, offering both theoretical and practical insights for algorithm selection and hardware design in deep learning systems..

2. Comparison of deep learning optimization algorithms

2.1. Different gradient descent optimization algorithms

In deep learning training, optimization algorithms differ mainly in convergence speed, stability, and computational cost. In contrast, GD-based methods have long served as fundamental optimization strategies owing to their simple implementation and mature theoretical framework. However, with the expansion of model and data scales, the discrepancies in training performance among different algorithms have become increasingly pronounced, necessitating a comprehensive trade-off between computing efficiency and optimization behavior [1]. In particular, Batch Gradient Descent (BGD) calculates gradients based on the entire training dataset, featuring a stable update direction and well-defined theoretical convergence properties. Nevertheless, its computational overhead rises markedly with the growth of data scale, leading to low training efficiency for large-scale datasets. In contrast, Stochastic Gradient Descent (SGD) approximates gradients using mini-batches of samples, which reduces the cost of a single iteration and accelerates the decline of the objective function in the early stage of training. However, its randomness tends to cause oscillations near the optimal solution and renders it highly sensitive to hyperparameters such as the learning rate. To enhance the stability of SGD, the momentum mechanism can be introduced, which uses historical gradients to stabilize the update direction, reduce high-frequency oscillations, and speed up convergence along the primary descent path. Additionally, the adaptive learning rate mechanism adopts differentiated step sizes for different parameter dimensions and dynamically adjusts the update amplitude according to gradient variations, improving the adaptability of the algorithm in complex training environments. Existing studies show an inherent trade-off between convergence speed, stability, and resource consumption in different GD algorithms. While simple methods work well for small models or limited resources, advanced algorithms with momentum or adaptive learning rates perform better in large models [2].

Thus, the selection of optimization algorithms should be made based on the specific task, data size, and available computing resources.

2.2. Adam and its main improved algorithms

Adam and its enhanced variants strike a good balance between convergence speed, stability, and generalization in deep learning training through adaptive gradient updates. And its core mechanism, which applies exponential moving averages and bias correction to the first and second moments of the gradient, dynamically adjusts the update step size [3]. This allows for rapid convergence early in training while ensuring stability in high-dimensional spaces and non-stationary objective functions. However, in some tasks, bias in the second moment estimation can cause unstable convergence or limit generalization. To address the inherent limitations of Adam, several improved algorithms have been proposed. For instance, AMSGrad imposes a non-decreasing constraint on the second moment to prevent the update step size from becoming too large due to small variance, thereby improving convergence stability. AdaGrad accumulates the squared historical gradients to adaptively adjust the learning rate, yielding remarkable effects in sparse feature scenarios, yet the learning rate decays rapidly in the later stage of training. Moreover, RMSProp employs exponential moving average to balance the squared gradients, alleviating the decline of the learning rate during long-term training. On this basis, Adam combines momentum and RMSProp with bias correction to adaptively adjust the update step size, as shown in the following formula:

$$\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{1}$$

where m_t and v_t are the bias-corrected estimates of the first moment and the second raw moment, respectively; η represents the learning rate; and ϵ is a small constant to prevent division by zero. In practice, improved Adam algorithms show excellent performance in training deep neural networks on large-scale datasets. For instance, combined with Dropout regularization for image recognition on MNIST and CIFAR-10, reduces validation error, achieves higher test accuracy, and narrows the training-validation error gap, effectively reducing overfitting [4].

2.3. Computational and resource characteristics of optimization algorithms

The performance of deep learning optimization algorithms depends on convergence, complexity, parameter state, and hardware needs. Also, different algorithms have varying demands on storage, memory access, and parallel computing, affecting their suitability for different hardware platforms and applications [5]. Traditional GD algorithms typically only need to maintain the current gradient, featuring simple parameter states and low occupancy of GPU memory or On-Chip Memory (OCM), and their computing processes are also easily parallelized, making them suitable for resource-constrained or embedded environments. In contrast, Adam and its variants require maintaining both first and second moment estimates during updates, increasing parameter states and demanding more GPU memory, bandwidth, and coordination. Meanwhile, the frequent access to historical gradient statistical information significantly increases the memory bandwidth pressure in large-scale training [6]. In terms of hardware adaptability, high-bandwidth memory and large-scale parallel computing platforms (e.g., GPUs) can fully support the efficient operation of adaptive optimization algorithms, whereas their implementation complexity and resource consumption increase markedly on resource-constrained platforms like FPGAs. In practices, when training convolutional neural networks on the MNIST and CIFAR-10 datasets, Adam converges faster than the standard GD and achieves higher test accuracy, but at the cost of increased computational and storage overhead. In convolutional network inference or embedded applications, GD algorithms can efficiently minimize storage usage and computational cost. In reinforcement learning tasks (e.g., DDPG), fluctuations in optimization algorithm updates or computational delays can lead to slow convergence of policy parameters and impair the overall system performance.

3. Implementation and performance evaluation of GPU and FPGA acceleration

3.1. GPU architecture and deep learning performance characteristics

Within the GPU and FPGA acceleration framework, the GPU is the main platform for deep learning, with its architecture well-suited to the needs of deep learning. With the continuous expansion of the parameter scale and computing density of large models, GPUs provide robust computing power support for deep learning training by virtue of dedicated computing units, highly parallel processing architectures and high-bandwidth storage systems, becoming the mainstream acceleration hardware in the large-scale model training phase [7]. From a hardware structural perspective, modern GPUs adopt a parallel architecture composed of multiple Streaming Multiprocessors (SMs), and each SM integrates a large number of computing cores, enabling the simultaneous execution of large-scale parallel computing tasks. To optimize matrix multiplication and tensor operations in deep learning, GPUs feature Tensor Cores, boosting matrix operation throughput through low-precision parallel computing. In addition, GPUs are equipped with High-Bandwidth Memory (HBM) to enable fast access to large parameters and intermediate feature maps, offering clear performance advantages in deep neural network training. As model size grows in large-scale training, GPUs boost efficiency with high parallel computing and bandwidth, but face challenges as computing demands hit the “computing wall.” The expansion of parameter scale leads to frequent GPU memory access, making memory bandwidth gradually become a critical performance bottleneck, namely the “memory wall.” In multi-GPU training, gradient synchronization and parameter communication overhead create the “communication wall,” limiting GPU efficiency and pushing the development of software-hardware co-optimization and heterogeneous architectures [8].

3.2. FPGA architecture and dataflow computing characteristics

Within the GPU and FPGA acceleration framework, FPGAs stand out for their customizable structure and dataflow model. Unlike general processors, FPGAs optimize algorithms via hardware parallelism and dedicated data paths, making them well-suited for real-time AI inference where low latency and power efficiency are critical.

From an architectural standpoint, FPGAs are made up of numerous Programmable Logic Blocks (PLBs), On-Chip Memory (OCM), and configurable interconnection networks. Through the flexible configuration of logic resources and data paths, FPGAs can map the computing process in deep learning models into a dedicated dataflow computing structure, enabling data to flow continuously in hardware according to a predetermined path, thereby reducing unnecessary control overhead and memory access latency. Research has shown that artificial intelligence accelerators based on FPGA architecture can effectively improve the throughput and energy efficiency ratio of convolutional neural networks in the inference phase through pipeline design and rational configuration of parallel computing units [9]. In certain applications, FPGAs offer significant low-latency advantages in embedded reinforcement learning tasks. Zhu Xiaoqing et al. optimized the key computing modules of the DDPG algorithm by mapping the policy and value networks onto FPGAs using a dataflow computing model, which reduced inference latency and improved system response time. And this endows FPGAs with obvious advantages in applications with high real-time requirements like robot motion control. In convolutional neural network inference, FPGAs accelerate compute-intensive operations utilizing custom convolutional units, On-Chip Memory (OCM) caching, and pipeline structures. This reduces power consumption while maintaining inference accuracy, highlighting their value in edge computing and embedded AI systems [9].

3.3. Applicability of GPUs and FPGAs in deep learning

Due to differences in architecture and computing paradigms, the applicability of GPUs and FPGAs varies across scenarios. Thus, the performance analysis of these hardware platforms clarifies their optimal use in training, inference, and embedded applications.

In high-performance training tasks, GPUs exhibit significant advantages in scenarios with large model parameter scales and high computing density by virtue of their large-scale parallel computing capability and high-bandwidth storage systems. Many studies demonstrate that GPUs offer higher computing throughput and better parallel scalability in large-scale model training, making them ideal for iterative training of deep neural networks and large datasets. However, with the continuous expansion of model scale, GPUs are gradually constrained by factors such as the “computing wall,” “memory wall” and multi-card communication overhead during training [8,10]. In contrast, FPGAs highlight their advantages more prominently in the inference phase and energy efficiency-sensitive tasks. Due to their configurable logic and dataflow architecture, FPGAs offer custom optimization for specific models and tasks, reducing computation latency and energy consumption. Prior studies showed that in convolutional neural network inference and reinforcement learning control tasks, FPGAs reduce system energy consumption and response latency, and maintain computing accuracy, making them particularly suitable for application scenarios with stringent requirements for real-time performance and energy efficiency [9-11]. With their flexible logic and dataflow design, FPGAs enable custom optimization for specific tasks, cutting computation delay and power use. In practical system design, GPU and FPGA acceleration platforms should be rationally selected or combined in different stages such as training, inference and edge deployment according to specific application requirements, to achieve an optimal balance between performance and energy efficiency [8,9,11].

4. Synergy between optimization algorithms and hardware acceleration

4.1. Adaptability of optimization algorithms on GPUs

In the software-hardware co-design of optimization algorithms and hardware acceleration, GPUs are the main computing platform for deep learning model training, due to their parallel architecture and high-bandwidth storage systems. The differences in computing processes and parameter updates of optimization algorithms affect GPU parallel efficiency and training outcomes, requiring an in-depth analysis of their adaptability.

In terms of parallel efficiency and convergence performance, Stochastic Gradient Descent (SGD) and its variants exhibit high adaptability on GPUs. Due to their simple parameter update processes and few computational dependencies, SGD can fully utilize the parallel computing capability of GPUs and demonstrate good stability in the training of large-scale data and models. In contrast, although adaptive optimization algorithms such as Adam generally outperform SGD in convergence speed, especially in the early stage of training where they can reduce the loss function more rapidly, their parameter update processes require the simultaneous calculation and maintenance of first and second moment estimates, which in turn increases computational and synchronization overhead to a certain extent [12,13]. This also places higher demands on GPU memory usage and memory access bandwidth. The frequent access to historical gradient statistical information during the parameter update process gradually makes memory bandwidth a critical performance bottleneck in training. In comparison, SGD-based algorithms occupy relatively low GPU memory resources, thus being more easily deployed in environments with limited GPU memory or high communication overhead [14].

In large-scale model training, the high-throughput parallel capability of GPUs determines their adaptability with optimization algorithms. For tasks with large parameter scales and high computing density, GPUs can effectively support algorithms like Adam through parallel computing and high-bandwidth storage, improving the overall training efficiency [8]. This further illustrates that there exists a strong synergistic relationship between optimization algorithms and GPU architectures in the large-scale model training phase.

4.2. Adaptability of optimization algorithms on FPGAs

In the software-hardware co-design of optimization algorithms and hardware acceleration, FPGAs offer unique advantages over GPUs in certain applications, thanks to their programmable logic and dataflow computing features. However, the implementation effect of optimization algorithms on FPGAs depends not only on the intrinsic performance of the algorithms themselves but also on the constraints of hardware resource scale, design complexity and implementation cost.

The implementation of complex adaptive optimization algorithms on FPGAs presents significant challenges. For example, Adam and its improved algorithms need to maintain multiple sets of state variables during the parameter update process and execute multiple multiply-accumulate (MAC) and exponential weighted operations, which impose high requirements on the logic resources, OCM and design complexity of FPGAs. As a result, implementing such complex optimizers on FPGA platforms typically requires simplifying or approximating the algorithm to balance resource usage and computing performance [9,11]. However, FPGAs excel in small-scale models and low-latency applications. By mapping the computation to hardware via a tailored dataflow, FPGAs can greatly reduce computation paths and memory access delays during inference. Experiments show that in embedded reinforcement learning and real-time control tasks, FPGAs achieve ultra-low latency for model inference and parameter updates, making them ideal for fast-response environments [9,11]. In addition, compared with general-purpose computing platforms, FPGAs can usually achieve lower power consumption and more stable latency performance while maintaining model inference accuracy. Relevant comparative studies demonstrate that FPGAs offer significant advantages in energy efficiency and real-time performance for typical convolutional neural network inference tasks, highlighting their value in edge computing and energy-constrained scenarios [10,12].

4.3. Synergistic characteristics of algorithm-hardware integration

In deep learning systems, optimization algorithms and hardware are closely linked. While algorithm processes and memory access patterns define the need for computing cores, GPU memory, and caching, hardware's parallelism and storage structure limit algorithm design. For instance, GD-based algorithms, with simple parameter states and regular memory access, efficiently leverage high-bandwidth GPU memory for high-throughput computing. In contrast, adaptive optimization algorithms such as Adam, which frequently access historical gradients, are more sensitive to GPU memory bandwidth and latency. On FPGAs, customized dataflow structures and OCM caching can alleviate memory bottlenecks but increase logic resource usage and design complexity [8,11].

The architectural characteristics of different computing platforms directly affect the design and selection of optimization algorithms. Relying on strong parallel computing capability and a mature software ecosystem, GPUs efficiently support compute-intensive, hardware-aware optimization algorithms with many state variables, excelling in large-scale model training. Due to limited on-chip resources, FPGAs are more suitable for optimization methods with simple structures and high customizability, or for the simplified implementation of complex optimizers, which also drives the in-depth research on hardware-aware optimization algorithms [15]. The optimal combination of algorithms and hardware depends on specific application scenarios. In large-scale model training, the combination of GPUs and adaptive optimization algorithms such as Adam can fully exploit the advantages of high throughput and fast convergence. In the inference phase or energy efficiency-sensitive tasks, FPGAs integrated with simple optimization strategies and customized dataflow computing can effectively reduce latency and power consumption. In edge computing or embedded systems, low latency and low power consumption are prioritized over extreme computing power, which further highlights the unique value of FPGAs [8,9,11].

5. Conclusion

This paper explores the synergistic relationship between deep learning optimization algorithms and hardware acceleration architectures, analyzing the computing properties and adaptive performance of GD-based algorithms as well as Adam and its improved variants on platforms such as GPUs and FPGAs. Through an in-depth study of algorithm principles, computing and resource requirements, and hardware architectural features, this paper reveals the mutual constraints and synergistic effects between algorithm design and hardware platforms. The results reveal that GPUs efficiently support adaptive optimizers such as Adam in large-scale training, thanks to their parallel processing, fast storage, and solid software ecosystem, hence leading to high throughput and rapid convergence. In contrast, in inference, energy efficiency-sensitive or low-latency applications, FPGAs demonstrate outstanding performance in power consumption control and real-time performance via customized dataflow computing structures and flexible hardware resource configuration. In addition, intrinsic differences in parameter state variables and memory access patterns across various optimization algorithms directly impact their execution efficiency on different hardware platforms. However, the current research is mainly based on existing literature and typical experimental results, lacking large-scale quantitative comparisons under a unified experimental platform, and failing to conduct in-depth analysis of emerging accelerator architectures and hybrid computing platforms, which presents certain limitations. Future research should develop a unified framework for quantitatively comparing the performance and energy efficiency of different optimization algorithms on various hardware. Meanwhile, it is necessary to design hardware-aware optimization algorithms for specific hardware architectures and realize in-depth software-hardware co-optimization, further improving the overall performance of deep learning systems.

References

[1] Shen, L., Sun, Y., Yu, Z., et al. (2024). On efficient training of large-scale deep learning models. ACM Computing Surveys, 57(3), 1-36.

[2] Sun, R. Y. (2020). Optimization for deep learning: An overview. Journal of the Operations Research Society of China, 8(2), 249-294.

[3] Zhou, P., Xie, X., Lin, Z., et al. (2024). Towards understanding convergence and generalization of AdamW. IEEE Transactions on Pattern Analysis and Machine Intelligence, 46(9), 6486-6493.

[4] Ogundokun, R. O., Maskeliunas, R., Misra, S., et al. (2022). Improved CNN based on batch normalization and Adam optimizer. In International Conference on Computational Science and Its Applications, 593-604.

[5] Xu, M., Xiang, L., Cai, X., et al. (2024). No more adam: Learning rate scaling at initialization is all you need. arXiv preprint arXiv: 2412.11768.

[6] Tang, Y., Qiao, L., Yin, L., et al. (2025). Training large-scale language models with limited GPU memory: A survey. Frontiers of Information Technology & Electronic Engineering, 26(3), 309-331.

[7] Hanindhito, B., & John, L. K. (2024). Accelerating ML workloads using GPU Tensor Cores: The good, the bad, and the ugly. In Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (pp. 178-189). New York: ACM Press.

[8] Chen, Z., Tang, Y., Zhang, L., et al. (2025). Co-evolution of large models and GPUs: A new AI industry ecosystem driven by software-hardware co-innovation. Artificial Intelligence, (03), 22-33.

[9] Chen, Y. (2025). Design and implementation of AI accelerators based on FPGA architecture. Software, 46(01), 59-61.

[10] Vaithianathan, M., Patil, M., Ng, S. F., et al. (2023). Comparative study of FPGA and GPU for high-performance computing and AI. ESP International Journal of Advancements in Computational Technology, 1(1), 37-46.

[11] Zhu, X., Bi, L., Gong, W., et al. (2026). Hardware mapping analysis of the DDPG algorithm based on FPGA and robot motor skill learning. Journal of Harbin Institute of Technology, 58(01), 24-34.

[12] Yang, G., Yang, J., Li, S., et al. (2018). Improved CNN algorithm based on Dropout and ADAM optimizer. Journal of Huazhong University of Science and Technology (Natural Science Edition), 46(07), 122-127.

[13] Haji, S. H., & Abdulazeez, A. M. (2021). Comparison of optimization techniques based on gradient descent algorithm: A review. PalArch's Journal of Archaeology of Egypt/Egyptology, 18(4), 2715-2743.

[14] Liu, M., Yao, D., Liu, Z., et al. (2023). An improved Adam optimization algorithm combining adaptive coefficients and composite gradients based on randomized block coordinate descent. Computational Intelligence and Neuroscience, 4765891.

[15] Bao, Y. (2025). Analysis of the impact of artificial intelligence on the technological development of chip design. Microelectronics & Computer, 42(10), 1-8.