Towards Efficient Deep Reinforcement Learning: A Comprehensive Review of Optimization Techniques

Ziteng Yin

School of Computer and Electronic Information /School of Artificial Intelligence, Nanjing Normal University, Nanjing, Jiangsu, China

Abstract. Deep reinforcement learning (DRL) has achieved remarkable success in various domains, including robotics, game playing, and autonomous systems. However, optimizing DRL models remains a significant challenge due to issues such as sample inefficiency, instability, and high computational costs. To address these challenges, researchers have explored various optimization strategies that enhance learning efficiency and model performance. This paper provides a comprehensive review of optimization methods in DRL, focusing on hyperparameter optimization, structural optimization, and algorithm optimization. It explores common techniques for hyperparameter optimization, including Bayesian optimization, grid search, gradient optimization, and evolutionary algorithms, discussing their strengths, weaknesses, and suitable application scenarios. The paper also examines structural optimization, highlighting key mechanisms such as attention mechanisms and generative adversarial networks (GANs), and their impact on improving model performance in various domains. Additionally, it analyzes algorithm optimization strategies, including Double-Q-learning, Proximal Policy Optimization (PPO), MuZero, and deep evolutionary strategies (DES), comparing their effectiveness in solving complex tasks. Future research directions include the combination of optimization methods, with a focus on generalization and interpretability, as well as exploring real-world applications to further improve existing strategies. This review provides valuable insights for researchers and practitioners aiming to advance DRL technologies

1 Introduction

The field of artificial intelligence is one of the most popular and widely discussed areas of research today, with DRL as a key component of AI that has undoubtedly become a focal point of study. Over the past decade, DRL has made significant progress across various domains, particularly in tasks traditionally considered highly complex, such as gaming, robotic control, and natural language processing. Notable achievements include AlphaGo's victory over the world Go champion using Double-Q-Network (DQN), and OpenAI Five's defeat of professional Dota 2 players. The core idea of DRL is to leverage deep neural

19240722@njnu.edu.cn

networks to automatically extract features from high-dimensional observations, enabling effective learning in complex environments [1].

Today, DRL algorithms are widely applied in fields such as robotic control, autonomous driving, financial decision-making, and natural language processing, with DRL's advantages being especially evident when facing high-dimensional, complex environments. However, despite the numerous achievements of DRL across various fields, there are still several challenges in practical applications. First, model training often faces instability, such as gradient explosion or vanishing gradients, leading to slow convergence or even failure to converge. Second, the performance of DRL is often constrained by hyperparameter selection, neural network structure design, and algorithm optimization. Therefore, how to effectively optimize DRL models to improve training efficiency, stability, and performance has become a key issue that needs to be addressed.

This paper reviews model optimization strategies in DRL, focusing on parameter optimization, model optimization, and algorithm optimization. By reviewing recent literature and research findings, it analyzes the effectiveness of different optimization strategies in various application scenarios, systematically comparing their strengths and weaknesses, and providing references for future research.

2 Hyperparameter Optimization

In DRL, the choice of hyperparameters plays a crucial role in the training process and final performance. Some of the most important hyperparameters include the learning rate, discount factor, batch size, and ε value. Common optimization methods for these hyperparameters include Bayesian optimization, grid search, gradient optimization, and evolutionary algorithms.

Bayesian optimization is a probability model-based method that builds a surrogate process to model the objective function and guides the next evaluation of hyperparameters through acquisition functions, allowing it to find an optimal solution with a limited number of trials [2]. However, its major drawbacks are the strong assumptions made about the objective function and the high computational cost required to train the surrogate model. Therefore, Bayesian optimization is most suitable for tasks where the objective function is complex and expensive to evaluate, such as tuning hyperparameters for deep neural networks. It has shown good applications in scenarios requiring large-scale hyperparameter tuning, such as convolutional neural networks and long short-term memory networks [3].

Grid search, on the other hand, is an exhaustive search method where the hyperparameter space is divided into regular intervals, and all possible combinations are explored to select the best hyperparameters. While theoretically, it can find the optimal solution by exploring all combinations, its disadvantage is the computational cost rapidly increases as the hyperparameter space grows, leading to a decline in efficiency. Therefore, grid search is more suitable for tasks with a smaller hyperparameter space [4].

Hyperparameter gradient optimization is an emerging strategy that simultaneously optimizes model parameters and hyperparameters by calculating the gradients of the hyperparameters. This method uses first- or second-order gradient information to update the hyperparameters, enabling dynamic adjustment during training. The main advantage of hyperparameter gradient optimization is that it can adjust hyperparameters in real-time by leveraging information from the model training process. However, it requires computing gradients for the hyperparameters, which adds significant computational overhead, and it relies on the smoothness and differentiability of the objective function [5]. Gradient optimization has shown excellent performance in multi-stage training and complex tasks such as reinforcement learning and meta-learning. Evolutionary algorithms optimize hyperparameters through processes that simulate natural selection, using population,

crossover, mutation, and selection operations. It excels particularly in high-dimensional hyperparameter spaces, finding optimal solutions for very complex and nonlinear tasks [6]. However, it also has the disadvantage of high computational cost, as each generation's fitness evaluation requires substantial computational resources. Therefore, it is typically used when computational resources are not limited and a globally optimal solution is needed.

In summary, tasks with high computational cost and high-dimensional space benefit from Bayesian optimization for optimal solutions within limited trials, whereas grid search is more suitable for lower-dimensional tasks where an optimal solution can be found reliably. For complex problems where both model parameters and hyperparameters need to be optimized, evolutionary algorithms and gradient optimization tend to perform better.

3 Structural Optimization

Structural optimization is an important aspect of DRL, aimed at improving learning efficiency and task-solving ability through the design of appropriate neural network architectures or by enhancing existing structures. Structural optimization covers a wide range of content, including the selection of network architecture, optimization of depth and width, introduction of advanced modules and mechanisms, and techniques for compressing and accelerating models. This section focuses on discussing new modules and mechanisms introduced to the base structure to enhance the network's learning effectiveness and expressive ability, comparing their impacts.

When discussing introduced modules and mechanisms, attention mechanisms are undoubtedly significant. They perform particularly well in natural language processing (NLP) and image processing by mimicking human visual attention, allowing the model to focus on the most relevant parts of input data, thus improving performance. This mechanism is especially effective for long-range dependencies and can process entire input sequences simultaneously, improving performance on long-sequence data or complex datasets, especially in NLP tasks. It is mainly used for sequence data processing and has been widely adopted in computer vision, with the self-attention mechanism in Transformers being a classic example [7]. However, its major drawback is the high computational cost, particularly for long sequences. GANs consist of a generator and a discriminator that compete with each other, allowing the generator to produce realistic data. The generator attempts to create realistic data, while the discriminator tries to distinguish between real and generated data [8]. Its advantage is in generating high-quality samples, especially when real data is scarce, and it can enhance the model's generalization ability through adversarial training, which is widely used in image generation, image repair, and data augmentation. However, it can suffer from instability during training, as adversarial training between the generator and discriminator might cause the model to fail to converge. Therefore, careful training is required to avoid mode collapse. Despite the fact that these two mechanisms are not from the same field, many studies now aim to combine them. The basic idea is that GANs generate realistic data samples through adversarial training, while the attention mechanism can help the model focus on specific regions of the input during generation, thereby improving the quality of the generated samples [9]. A specific example is the self-attention mechanism, which allows each pixel or feature point to consider other pixels or feature points in the entire image during generation, thus improving the global consistency of the image [10].

In summary, combining these two mechanisms can enhance local details, coordinate the global consistency, and improve generation results.

4 Algorithm Optimization

Finally, algorithm optimization, as the core part of overall DRL optimization, typically refers to directly improving the core components of the learning algorithm to enhance model performance, stability, and efficiency. These optimization algorithms can be roughly classified into the following categories: improvements to basic Q-learning, policy gradient method optimization, model-based methods, and hybrid algorithms. This section will select the most representative algorithms from each category to explore their advantages and disadvantages as well as their corresponding application scenarios.

Among the improvements to basic Q-learning, Double Q-learning is undoubtedly the most classic and influential, as it uses two independent Q-functions. Specifically, during the update process, one Q-function selects the optimal action, while the other Q-function calculates the value of that action, effectively solving the overestimation problem in Q-learning [11]. Therefore, it is widely used in reinforcement learning problems [12, 13], especially excelling in discrete action spaces. PPO is widely regarded as the most classic and effective policy gradient method, with its core algorithm optimizing a "target function" that updates the policy during each iteration, controlling the magnitude of the updates. Compared to TRPO, PPO uses standard gradient descent and a simple clipping strategy [14]. This makes it easy to implement, and the clipping of the objective function effectively controls the policy update step size, avoiding oscillations during training and providing higher training stability. Given these features, along with the fact that PPO can learn good policies with limited samples, it performs well in high-dimensional state and action spaces [15], though it tends to underperform in handling extremely complex tasks.

For model-based methods, MuZero is a typical example. MuZero's distinguishing feature is that it does not directly use the known environment model but instead constructs an internal representation model that transforms the current state into a latent representation, making decisions based on these representations [16]. This allows MuZero to learn and reason effectively without explicit knowledge of the environment’s transition and reward functions. Additionally, MuZero uses Monte Carlo Tree Search (MCTS) to further improve the quality and efficiency of decision-making. In each decision step, MuZero uses MCTS to simulate potential future rewards and make more reliable decisions. These characteristics allow it to perform excellently in complex tasks, especially in games such as Go and Chess. Moreover, it has proven effective in non-game reinforcement learning tasks [17]. Its limitation lies in its heavy computational requirements and long training times, especially in resource-limited environments, where its convergence speed can be slow.

Finally, for hybrid methods, Deep Evolution Strategies (DES) is considered to perform exceptionally well. DES combines deep learning and evolutionary algorithms to optimize the structure and parameters of deep neural networks [18]. It does not rely on gradient information, instead using black-box optimization methods to improve the network's performance. Since evolutionary algorithms directly evaluate and optimize the output of neural networks [19], DES is less dependent on the environment. This makes it particularly effective in tasks where traditional gradient descent methods are not applicable, such as discontinuous or noisy tasks. Through global search, DES avoids the problem of getting stuck in local optima, and since each individual’s training process can be done independently, DES takes advantage of modern computational resources for parallel computation, significantly improving training efficiency. These features make DES suitable for applications in robot control [20], automated design [21], and image generation [22]. However, DES's limitations include instability across different types of tasks. In environments with dynamic changes, DES may fail to adapt effectively to new requirements and has poor generalization, with slow convergence on complex problems [23].

In summary, value-based methods (such as Double Q-learning), which estimate state-action value functions (Q-values) to guide the agent's decision-making process, are better suited for discrete action space problems, such as games and simple control tasks. Policy-based methods like PPO, which directly optimize the policy function through gradient ascent, are more suitable for high-dimensional continuous spaces, especially for tasks requiring complex decision-making optimization (e.g., robot control, autonomous driving). Model-based methods like MuZero, which build an environment model to predict future states and use that model for planning and optimization, are ideal for tasks requiring long-term planning and decision-making (e.g., robot control, autonomous driving, optimization in simulated environments). Evolutionary algorithms and hybrid methods, such as DES, are particularly effective in tasks where gradient information is unavailable or for high-dimensional complex optimization problems, especially those related to deep neural network architecture optimization (e.g., neural architecture search). Future research and applications should select appropriate optimization algorithms for the best results.

5 Conclusions

This paper provides a comprehensive review of optimization methods in DRL from three key perspectives: hyperparameter optimization, structural optimization, and algorithm optimization.

First, this paper introduced the main hyperparameter optimization techniques, including Bayesian optimization, grid search, gradient optimization, and evolutionary algorithms, discussing their applicability and limitations in different tasks. Next, this paper discussed key structural optimization techniques, particularly attention mechanisms and GANs, and analyzed their contributions and shortcomings in enhancing model performance. Finally, this paper explored core strategies for algorithm optimization, such as Double Q-learning, PPO, MuZero, and deep evolutionary strategies, and compared their performance in complex tasks. Despite providing a thorough theoretical analysis, this paper has certain limitations. Firstly, the review is primarily based on existing literature and does not include experimental validation or systematic comparisons of different optimization methods in practical tasks.

As the actual application scenarios of DRL are highly complex, the effectiveness of optimization methods may vary depending on task characteristics, datasets, and hardware environments. Therefore, the paper does not fully capture the real-world performance of these methods. Secondly, while several optimization methods were discussed, the integration of these methods and their suitability for multi-task, dynamic environments were not sufficiently explored. Future research can explore the combination of different optimization methods, particularly in multi-task learning and complex environments. Additionally, optimization strategies should not only focus on model performance but also consider interpretability and generalization, which will be key areas for future study. By introducing more experiments and real-world scenarios, further validation and improvement of existing optimization methods can be achieved.

This paper contributes a comprehensive overview of optimization methods in DRL, analyzing their strengths, weaknesses, and applicable scenarios, while providing theoretical foundations and practical guidance for future research directions. Through this systematic summary, the paper aims to promote the further development and application of DRL technologies.

References

1. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529–533, 2015.
2. J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in Adv. Neural Inf. Process. Syst., 2012.
3. J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in Proc. 30th Int. Conf. Mach. Learn. (ICML), 2013, pp. 115–123.
4. F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in Proc. 5th Int. Conf. Autom. Mach. Learn. (AutoML), 2011, pp. 1–10.
5. L. Franceschi, P. Frasconi, and C. W. Omlin, "Bilevel programming for hyperparameter optimization and meta-learning," in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 1563–1572.
6. T. Salimans, J. Ho, X. Chen, S. Sidor, and P. Abbeel, "Evolution strategies as a scalable alternative to reinforcement learning," in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 1806–1814.
7. A. Vasani, N. Shazeer, N. Parmar et al., "Attention is all you need," in Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 5998–6008.
8. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza et al., "Generative adversarial nets," in Adv. Neural Inf. Process. Syst., vol. 27, pp. 2672–2680, 2014.
9. H. Zhang, T. Xu, H. Li, S. Zhang, and X. He, "StackGAN++: Realistic image synthesis with stacked generative adversarial networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 8, pp. 1947–1962, 2018.
10. J. Zhang and J. Zhu, "Self-attention generative adversarial networks," in Proc. 36th Int. Conf. Mach. Learn. (ICML), 2019, pp. 7354–7363.
11. H. Van Hasselt, A. Guez, and D. Silver, "Double Q-learning," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 23, pp. 2613–2621, 2010.
12. Z. Wang and J. Zhang, "Double Q-learning for robotic control," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2017, pp. 1629–1634.
13. L. Li and Y. Cheng, "Application of double Q-learning for adaptive traffic signal control," IEEE Trans. Intell. Transp. Syst., vol. 21, no. 9, pp. 3761–3769, 2020.
14. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.
15. J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in Proc. 32nd Int. Conf. Mach. Learn. (ICML), 2015, pp. 1889–1897.
16. D. Silver, J. Schrittwieser, K. Simonyan et al., "MuZero: Mastering Go, Chess, Shogi and Atari without rules," Nature, vol. 591, no. 7851, pp. 591–596, 2020.
17. D. Hafner, M. Noroozi, and M. A. Kitani, "Applications of model-based reinforcement learning: MuZero as a case study," IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 4, pp. 1451–1464, 2021.
18. T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 1–10.
19. D. Wierstra, T. Schaul, T. Glasmachers et al., "Natural evolution strategies," Neural Comput., vol. 26, no. 3, pp. 671–702, 2014.

20. T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Robust imitation of diverse behaviors," in Proc. 31st Int. Conf. Mach. Learn. (ICML), 2018, pp. 2823–2832.
21. B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in Proc. 5th Int. Conf. Learn. Represent. (ICLR), 2017, pp. 1–12.
22. Z. Zhang, L. Xie, and X. Zeng, "Learning to generate with social GANs," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 10256–10265.
23. N. Hansen, T. Salimans, and A. Salimans, "A generalized recurrent evolutionary strategy," J. Mach. Learn. Res., vol. 20, no. 122, pp. 1–48, 2019.