Comparative Study of Advantage Actor-Critic (A2C) and Deep Q-Network (DQN) Algorithms in Autonomous Car Racing Game Environments

Syed Ali Vijdaan Naqvi
Faculty of Computer Science and Engineering
GIK Institute of Engineering Sciences and Technology
Topi, Pakistan
u2022560@giki.edu.pk

Hamza Amin
Faculty of Computer Science and Engineering
GIK Institute of Engineering Sciences and Technology
Topi, Pakistan
u2022378@giki.edu.pk

***Abstract*—This research investigates the application of two prominent reinforcement learning (RL) algorithms, Advantage Actor-Critic (A2C) and Deep Q-Network (DQN), in the context of autonomous car racing. The car racing environment provides a challenging and dynamic setting where agents must navigate complex trajectories, make real-time decisions, and adapt to changing conditions in pursuit of optimal performance. The objective of this study is to compare the performance and learning efficiency of A2C and DQN in an autonomous racing task, highlighting the strengths and weaknesses of each approach. The results of this study provide valuable guidelines for selecting appropriate RL algorithms for real-time autonomous systems, such as self-driving cars, where decision-making efficiency and real-time performance are critical. Ultimately, the research contributes to advancing the understanding of RL algorithms' applicability to complex, real-world problems in the domain of autonomous vehicles.**

***Index Terms*—reinforcement learning, deep q-learning, actor-critic, gaming**

I. Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for training agents to make sequential decisions in complex, uncertain environments. Among the earliest breakthroughs was the Deep Q-Network (DQN), which combines Q-learning with deep convolutional neural networks to approximate the optimal action-value function directly from high-dimensional sensory inputs. DQN demonstrated human-level performance on a suite of Atari 2600 games, stabilizing training through experience replay and target networks [1]. However, its value-based nature makes extending DQN to environments with continuous or large action spaces (common in robotics and autonomous driving) challenging.

Actor–critic methods offer an alternative by explicitly learning both a policy (actor) and a value function (critic). A particularly effective variant is the Advantage Actor-Critic (A2C), a synchronous implementation of the Asynchronous Advantage Actor-Critic (A3C) algorithm. A2C trains multiple parallel workers synchronously to reduce variance and improve sample efficiency, enabling stable policy learning even in high-dimensional, continuous domains [2]. By learning a stochastic policy directly, A2C naturally handles continuous actions—such as steering and acceleration in a racing scenario—while benefiting from on-policy updates that adapt quickly to changing dynamics.

II. Realted Work

Reinforcement learning (RL) has been extensively explored in the domain of autonomous vehicle control, with significant advancements in both value-based and policy-based approaches. The Deep Q-Network (DQN) algorithm, introduced by Mnih et al., demonstrated human-level performance on Atari games by combining Q-learning with deep convolutional neural networks, stabilizing training through experience replay and target networks [1]. This success spurred interest in applying DQN to autonomous driving tasks.

In the context of autonomous car racing, several studies have leveraged DQN and its variants. For instance, Yuwono et al. applied DQN to the OpenAI Gymnasium CarRacing environment, integrating ResNet and LSTM models to capture complex spatial and temporal dynamics, resulting in improved performance over baseline DQN implementations [3]. Similarly, Jalali's work on the CarRacing-v0 environment demonstrated that enhancements such as Prioritized Experience Replay and Double DQN significantly improved the agent's performance, achieving higher average rewards and stability [4]. arXiv GitHub

However, DQN's applicability to continuous action spaces, which are prevalent in autonomous driving scenarios, is limited due to its discrete action formulation. To address this, policy-based methods like the Advantage Actor-Critic (A2C) algorithm have been explored. A2C combines policy and value estimation, enabling it to handle continuous action spaces effectively. McEllhiney developed an autonomous driving system using A2C, training the agent to navigate tracks without human input, demonstrating resilience to sensor noise and track variations. They achieve this by using the Advantage Actor Critic reinforcement system that trains the car based on continuously adapting the model to minimize the penalty received by the car. [5].

Furthermore, Jaritz et al. employed an Asynchronous Advantage Actor-Critic (A3C) framework for end-to-end race driving, using only RGB images from a forward-facing camera. Their approach led to faster convergence and robust driving policies capable of generalizing to unseen tracks [6].

These studies highlight the strengths and limitations of DQN and A2C in autonomous car racing. While DQN provides a solid foundation for discrete action spaces, A2C offers advantages in continuous control tasks, making it more suitable for real-world autonomous driving applications. This research aims to build upon these findings by directly comparing the performance and learning efficiency of DQN and A2C in an autonomous racing context, providing insights into their respective strengths and applicability.

III. Methodology

In this section, we detail the setup, preprocessing, and algorithmic components used to train and evaluate Advantage Actor-Critic (A2C) and Deep Q-Network (DQN) agents on the CarRacing-v3 environment.

A. Environment and Preprocessing

The CarRacing-v3 environment in Gymnasium serves as the experimental testbed for our study. At each discrete time step t, the agent receives an observation in the form of a raw RGB image $F_t \in \mathbb{R}^{96x96x3}$, which encodes the full visual state of the race track and vehicle. The action space $\mathbb{A}$ is continuous and three-dimensional, compromising of steering (left and right) $a^{(1)} \in [-1, 1]$, gas $a^{(2)} \in [0, 1]$ and brake $a^{(3)} \in [0, 1]$. The reward r_t at time t is proportional to the distance traveled along the track, with penalties applied for off-track behavior.

To reduce computational complexity and focus learning on essential visual features, each raw frame F_t undergoes a sequence of pre-processing steps. First, the image is converted to grayscale:

$$G_t = \text{gray}(F_t) = 0.2989F_t^{(R)} + 0.5870F_t^G + 0.1140F_t^B$$

where F_t^R, F_t^G, F_t^B, denote the red, green and blue channels respectively. Next, we crop the resulting grayscale image to remove irrelevant sky and dashboard regions, retaining only the top 84 rows:

$$C_t = G_t[0 : 84, 0 : 96]$$

This cropping focuses the agent's attention on the racetrack ahead, eliminating visual distractions and reducing input dimensionality.

The cropped frame C_t is then resized to a square resolution of 96 x 96 pixels using area interpolation:

$$R_t = \text{resize}(C_t, 96x96)$$

and normalized to lie in the unit interval:

$$f_t = \frac{R_t}{255.0}, f_t \in [0, 1]^{(96x96)}$$

Normalizing by 255.0 ensures that pixel values have zero mean and unit variance, properties that accelerate and stabilize neural network training.

At the beginning of each episode, the stack is initialized by repeating the first processed frame four times, ensuring a fully populated state.

Subsequently, at each time step the oldest frame is discarded and the newest f_t is appended. This temporal embedding provides the networks with short-term motion information—critical for learning steering dynamics and acceleration control in a high-speed racing context. Together, these pre-processing operations transform raw sensory input into a compact, informative state representation suitable for our reinforcement learning agents.

B. Deep Q-Learning

Deep Q-Learning (DQN) is a powerful reinforcement learning algorithm that combines Q-learning with deep neural networks to handle high-dimensional state spaces, such as raw image inputs from the CarRacing-v3 environment. In traditional Q-learning, the agent learns an action-value function $Q(s, a)$ that estimates the expected cumulative reward for taking action aa in state ss and following the optimal policy thereafter. However, in environments with continuous or high-dimensional observation spaces, representing $Q(s, a)$ as a lookup table becomes infeasible. DQN overcomes this by approximating $Q(s, a)$ using a deep neural network, parameterized by , that takes the current state s_t as input and outputs a Q-value for each possible discrete action.

The action-value function is updated using the Bellman equation:

$$Q(s_t, a_t) = r_t + \gamma(s_{t+1}, a')$$

In DQN, this target value is used to compute the loss between the predicted Q-value and the target. Where:

$$y = r_t + \gamma(s_{t+1}, a'; \theta^-)$$

and θ^- are the parameters of a target network, which is a periodically updated copy of the Q-network to stabilize training. DQN uses several techniques to improve learning stability and performance:

- **Experience Replay:** Transitions are stored in a replay buffer D, and the network is trained using mini-batches sampled uniformly from this buffer. This reduces the correlation between sequential data and improves sample efficiency.
- **Target Network:** A separate target network with parameters θ^- is used to compute the target Q-values. This network is updated less frequently to provide a stable target during training.
- **Frame Stacking:** As described in the preprocessing section, the agent observes a stack of the last $K = 4$ frames, allowing it to infer motion and direction from static images.
- ϵ **- Greedy Policy:** To balance exploration and exploitation, the agent selects a random action with probability,

and the action with the highest predicted Q-value otherwise. Over time, decays to favor exploitation as the agent becomes more confident in its policy.

TABLE I
HYPERPARAMETERS FOR DEEP Q-NETWORK (DQN) IMPLEMENTATION

Hyperparameter	Value
Discount factor (γ)	0.99
Learning rate	1×10^{-4}
Optimizer	Adam
Replay buffer size	1×10^{6}
Batch size	32
Target network update frequency	Every 1000 steps
Exploration strategy	ϵ-greedy
Initial ϵ	1.0
Final ϵ	0.01
ϵ decay schedule	Linear over 1M steps
Update frequency	Every 4 steps
Frame stack size	4
Huber loss coefficient	1.0
Double DQN	Enabled

C. Actor Critic

Actor-Critic (AC) methods combine the benefits of value-based and policy-based reinforcement learning. In contrast to pure policy gradient methods (like REINFORCE) that suffer from high variance, and value-based methods (like DQN) that struggle with continuous action spaces, actor-critic frameworks use two separate components: the actor, which learns a policy $\pi_\theta(a|s)$, and the critic, which estimates the value function $V^\pi(s)$ or the action-value function $Q^\pi(s, a)$. This dual structure allows AC methods to provide low-variance updates while maintaining stability in high-dimensional or continuous control tasks like CarRacing-v3.

The actor is responsible for selecting actions according to a stochastic policy$\pi_\theta(a|s)$, where θ are the policy parameters. The critic learns to evaluate the current policy by estimating either the state-value function $V(s)$ or the advantage function $A(s, a)$, which helps guide the actor's updates.

The policy is updated by taking the gradient of the expected cumulative reward with respect to the actor parameters. This policy gradient is scaled by the advantage to encourage actions that lead to higher-than-expected returns and discourage those that do not.

The critic minimizes the temporal difference (TD) error between the predicted value and the observed return:

$$L_{\text{critic}}(\phi) = (r_t + \gamma V(s_{t+1}; \phi) - V(s_t; \phi))^2$$

where ϕ are the parameters of the critic network. This loss is used to train the critic using standard gradient descent.

Actor-critic architectures are especially useful for environments like CarRacing-v3 that have continuous or hybrid action spaces. Instead of discretizing the action space as in DQN, actor-critic methods can directly output continuous values for steering, acceleration, and braking using a Gaussian policy.

In this work, the actor and critic share a convolutional encoder that processes the stacked image frames (96×96×496×96×4), followed by separate multilayer perceptrons (MLPs) for the policy and value functions. This parameter sharing improves sample efficiency and speeds up convergence.

TABLE II
HYPERPARAMETERS FOR ACTOR-CRITIC IMPLEMENTATION

Hyperparameter	Value
Discount factor (γ)	0.99
Learning rate (actor)	1×10^{-4}
Learning rate (critic)	1×10^{-3}
Optimizer	Adam
Entropy regularization coefficient	1×10^{-3}
Value loss coefficient	0.5
Action standard deviation (σ)	0.1 (fixed)
Batch size	64
Policy update frequency	Every 5 steps
Frame stack size	4
Shared encoder between actor/critic	Yes
Action space	Continuous $[a, s, b]$

Entropy regularization is added to the actor's loss to encourage exploration by preventing premature policy convergence. The continuous output of the actor enables more natural and precise control over the vehicle, which is critical in tasks requiring smooth navigation and fast response to track changes.

IV. RESULTS

The performance of the two reinforcement learning algorithms—Deep Q-Network (DQN) and Advantage Actor-Critic (A2C)—was evaluated in the CarRacing-v3 environment. Both models were trained for a significant number of steps and episodes, respectively, and their performance was assessed based on the average return over time.

A. DQN Performance

The DQN agent was trained over approximately 482,370 steps, with an average training speed of 3.14 steps per second. As shown in Fig. 1, the average return fluctuated significantly throughout training, with a peak value above 20 in the early phase but quickly deteriorating. In later stages, the performance oscillated between -40 and -100, with no clear upward trend. The final reported average return was approximately -71.53, indicating that the DQN agent struggled to maintain consistent performance and did not converge to a robust policy.

B. A2C Performance

The A2C agent was trained over 100 episodes. Early in training (episode 25), the average reward over the last 100 episodes was -65.16. By episode 100, this average had slightly worsened to -72.40, suggesting limited learning progression. The agent's reward in the final episode was -80.33, as shown in Fig. 2, demonstrating a similar trend to DQN, with performance stagnating below the threshold generally considered indicative of competent driving in this environment.

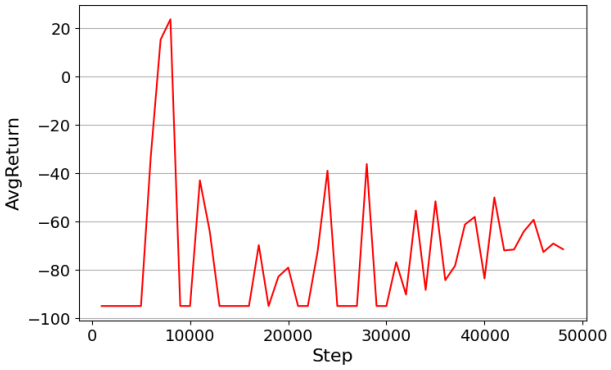

Fig. 1. DQN training performance: average return vs. training steps.

Fig. 2. A2C training performance: Total reward vs. Episodes.

C. Comparison and Analysis

Both algorithms exhibited suboptimal performance in the CarRacing-v3 environment, failing to achieve stable control policies. The DQN model's value-based nature and limited action discretization likely hindered its ability to fine-tune continuous actions such as steering and acceleration. In contrast, although A2C's policy gradient formulation is better suited for continuous control, the shallow performance gain indicates potential challenges in exploration or inadequate hyperparameter tuning.

Training time for both methods was significant, with DQN taking over an hour of continuous training and A2C completing 100 episodes with similarly lengthy computational demand. These results highlight the need for more sophisticated exploration strategies, reward shaping, or hybrid approaches when applying RL to high-dimensional, real-time control problems like autonomous racing.

Conclusion

In this work, we conducted a head-to-head comparison of two widely used deep reinforcement learning algorithms—Deep Q-Network (DQN) and Advantage Actor-Critic (A2C)—within the challenging CarRacing-v3 environment. Our experiments revealed that neither algorithm achieved consistent, high-performance driving policies under the current settings.

The DQN agent exhibited large fluctuations in average return, with transient peaks early in training but no sustained improvement, ultimately converging to an average return of approximately –71.5. This behavior underscores the limitations of a value-based, discrete-action approach when applied to a continuous control task. The A2C agent, though naturally suited for continuous action spaces, also stagnated around a similar performance level (average return around –72.4 after 100 episodes), suggesting possible shortcomings in exploration efficiency or hyperparameter configuration.

Both methods demanded substantial computational resources and training time, highlighting the practical challenges of deploying RL in real-time autonomous systems. These findings suggest several avenues for future work:

- **Enhanced Exploration:** Incorporating intrinsic motivation or parameter noise to encourage broader state-space coverage.
- **Hybrid Architectures:** Combining value- and policy-based components (e.g., DDPG, TD3, SAC) to leverage the strengths of both paradigms.
- **Reward Shaping and Curriculum Learning:** Designing intermediate tasks or auxiliary rewards to guide the agent through progressively complex driving behaviors.
- **Model Ensembles and Transfer Learning:** Leveraging pre-trained perception modules or ensemble strategies to improve sample efficiency and robustness.

Overall, our study provides empirical insights into the practical trade-offs between DQN and A2C for autonomous racing. By identifying their respective limitations, we lay the groundwork for more advanced RL techniques tailored to continuous, real-world driving scenarios.

Acknowledgment

The authors would like to thank Mr. Syed Qasim for his invaluable support and guidance. We also extend our gratitude to the Ghulam Ishaq Khan Institute for providing computational resources. Finally, we acknowledge the open-source software and technologies—particularly Python, Gymnasium, TensorFlow, and PyTorch—that made our experiments possible.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

[2] Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. arXiv. https://arxiv.org/abs/1602.01783

[3] F. Yuwono, G. P. Yen, and J. C. Jason, "Self-Driving Car Racing: Application of Deep Reinforcement Learning," *arXiv preprint arXiv:2410.22766*, 2024.

[4] H. Jalali, "OpenAI-Gym-CarRacing-RL," GitHub repository, 2021. [Online]. Available: https://github.com/hadi-jalali/OpenAI-Gym-CarRacing-RL

[5] E. McEllhiney, "Self-Training Autonomous Driving System Using An Advantage-Actor-Critic Model," M.S. project, Dept. Elect. Eng. Comput. Sci., Univ. Kansas, Lawrence, KS, USA, 2023.
[6] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-End Race Driving with Deep Reinforcement Learning," *arXiv preprint arXiv:1807.02371*, 2018.