Deep Reinforcement Learning for Autonomous Racing

A Comprehensive Analysis of DDQN

AI FINAL PROJECT REPORT
Anjali Patil, Garima Tata, Lasya Kata, Vasanthi Kummari
12140210,12140690,12140880,12140950

ABSTRACT

This report provides an in-depth exploration of the implementation and results of a Double Deep Q-Network (DDQN) approach for training autonomous racing agents in a simulated environment. The study involves a detailed examination of the underlying algorithms, training process, and challenges encountered during the learning process.

KEYWORDS

Reinforcement learning, Double Deep Q-Network

1 INTRODUCTION

The field of autonomous vehicle technology is advancing rapidly, fueled by innovative approaches such as reinforcement learning. Reinforcement learning is a branch of machine learning where agents learn optimal behavior by interacting with an environment and receiving feedback in the form of rewards or penalties. Unlike traditional rule-based programming, reinforcement learning allows agents to learn from experiences, adapting and improving their decision-making capabilities over time.

In the realm of autonomous racing, the goal is to develop intelligent agents capable of navigating dynamic environments, making real-time decisions, and adhering to predefined objectives. Reinforcement learning becomes invaluable in such scenarios, providing a framework for agents to learn from trial and error.

The Double Deep Q-Network (DDQN) algorithm stands out as a powerful tool for training agents in autonomous racing scenarios. This algorithm builds upon the Q-learning framework, enhancing its stability and efficiency. The problem addressed in this study involves training autonomous racing agents to navigate a dynamic racing track successfully. The agents must learn to avoid obstacles, understand the nuances of the track layout, and optimize their racing strategy.

This study aims to delve into the implementation details of the DDQN algorithm, exploring how it copes with the intricacies of autonomous racing. Through a detailed analysis, this study seeks to provide insights into the effectiveness of the DDQN algorithm in training autonomous racing agents and overcoming the complexities inherent in dynamic racing environments.

Dr. Soumajit Pramanik.

DS251, Artificial Intelligence, IIT Bhilai
2023.

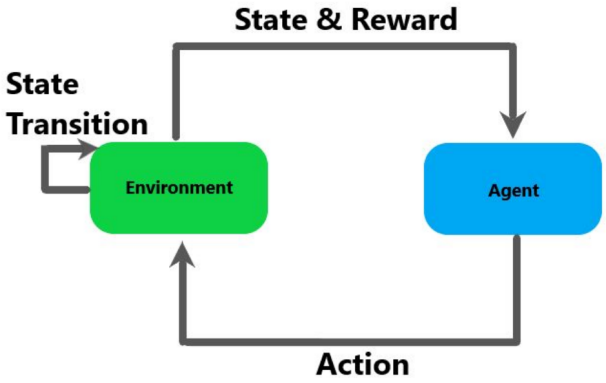

Figure 1: Reinforcement learning framework

2 ENVIRONMENT

The racing environment is constructed using Pygame, a Python library tailored for 2D game development. This environment provides the backdrop for the reinforcement learning agents to navigate, learn, and optimize their racing strategies.

2.1 Dynamic Elements

Goals. Goals are represented by lines, visually depicted using Pygame's drawing functionalities. These serve as checkpoints or targets for the reinforcement learning agents. Activation of goals is a crucial event, influencing the agents' performance.

Walls. The track's edges are defined as walls, acting as obstacles for the racing agents. Walls play a pivotal role in shaping the racing track, creating challenges for the agents to navigate.

Rewards. Rewards are earned based on the interaction of agents with goals. Successfully reaching a goal results in a reward, providing positive reinforcement for desired behaviors.

2.2 Action and State Spaces

The environment defines the action and state spaces for the reinforcement learning agents. Action spaces encompass the possible moves or decisions an agent can make, influencing its navigation on the track. State spaces encapsulate the current state of the environment, providing crucial information for the agents to make informed decisions.

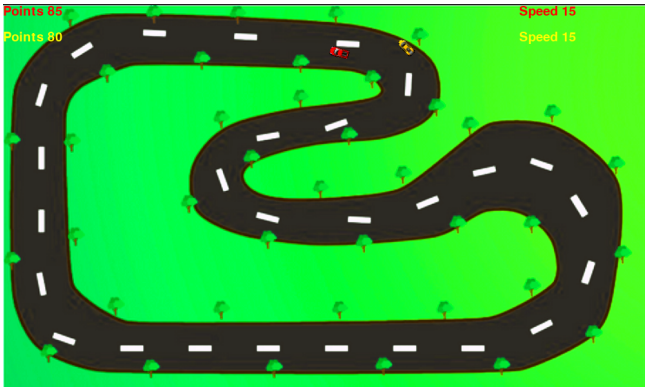

Figure 2: Environment display

2.3 Integration with Reinforcement Learning

The racing environment is seamlessly integrated with a reinforcement learning framework. The state transitions, rewards, and interactions between agents and the environment are crucial components of the learning process. Deep Q-Network (DQN) agents make decisions based on their observations, influencing subsequent actions and learning.

2.4 Custom Design

The track layout, background, and car aesthetics are custom-designed to create a visually engaging and challenging racing environment. The deliberate inclusion of goals, walls, and rewards adds complexity to the learning task, encouraging the development of sophisticated strategies by the reinforcement learning agents.

The combination of Pygame's functionality and custom design elements results in a dynamic and visually appealing setting for reinforcement learning research.

3 METHODOLOGY

In addressing the challenge of training autonomous racing agents, our chosen approach leveraged Reinforcement Learning (RL) with a focus on the Double Deep Q-Network (DDQN) algorithm. The primary goal was to empower agents to learn optimal strategies for dynamic racing environments, navigating obstacles, and achieving predefined goals.

3.1 DDQN

Double Deep Q-Learning (DDQN) is an advancement over traditional Deep Q-Learning (DQN), featuring a more stable architecture. The core formula centers around minimizing the temporal difference error δ, expressed as $\delta = (r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)) - Q(s_t, a_t; \theta)$. This adjustment aims to alleviate overestimation biases inherent in DQN, fostering more resilient learning processes. DDQN enhances the accuracy of Q-value estimations by considering the maximum Q-value from the target network for the next state. This refinement contributes to the model's ability to navigate dynamic environments with greater efficiency, showcasing improved performance and adaptability.

3.2 Key Architectural Components

Experience Replay. A critical element in training stability, experience replay involves storing past experiences in a buffer. Random samples from this buffer break temporal correlations, reducing overfitting to recent events.

Target Network. The introduction of a target network ($Q(s, a; \theta^-)$) contributes to training stability. Periodic updates to the target network help counteract the non-stationary nature of the environment.

Exploration-Exploitation Tradeoff. Achieving a balance between exploration and exploitation is vital. The epsilon-greedy strategy guides the agent to choose random actions with probability ϵ, gradually transitioning from exploration to exploitation over time.

3.3 Training Process

During each episode, agents interacted with the environment, receiving rewards based on their actions. The Q-network parameters underwent updates through backpropagation, optimizing the network to approximate the Q-function accurately. Training persisted across multiple episodes, concluding when a predefined stopping criterion was met.

This comprehensive approach, guided by principles of RL and DDQN, facilitated effective learning and adaptive behavior in response to the intricacies of the racing environment. By emphasizing experience replay, target networks, and a nuanced exploration-exploitation tradeoff, our training process enabled agents to acquire sophisticated strategies, resulting in successful navigation and goal attainment.

4 EXPERIMENTAL EVALUATION

An in-depth analysis of the training progress, agent performance, and key metrics provides insights into the learning dynamics.

4.1 Learning Progress

The initial episodes (1-7) demonstrate limited learning progress, with consistent low scores of -1.00. The agent struggles to navigate the complex environment effectively.

4.2 Breakthrough

In episode 8, a breakthrough occurs, with an agent achieving a score of 1.00. This signifies successful navigation and highlights the adaptability of the DDQN approach.

4.3 Training Stability

Episodes 9-15 show performance fluctuations, indicating ongoing challenges in training stability. The model encounters difficulties in maintaining consistent learning progress.

4.4 Average Scores

Average scores stabilize around -0.85, indicating moderate learning progress. The agent demonstrates improved performance over multiple episodes.

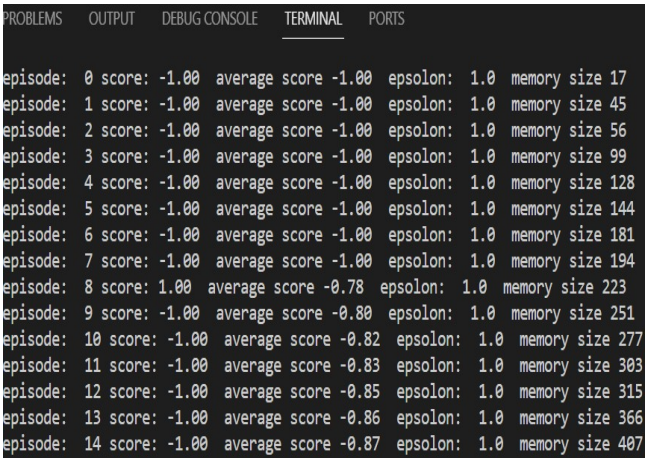

Figure 3: results1

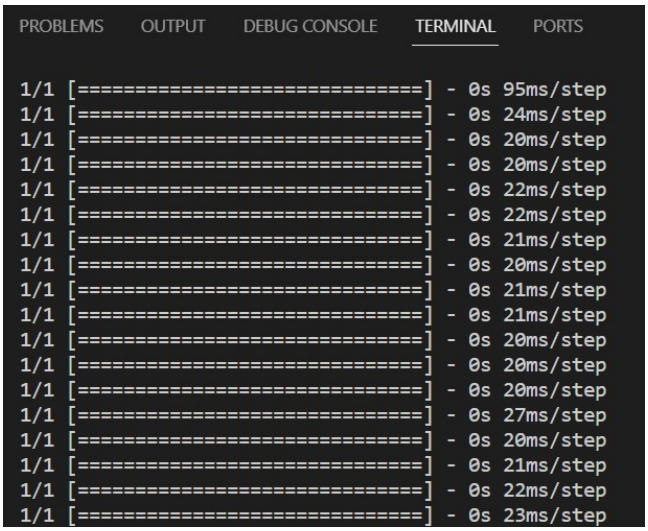

Figure 4: results2

4.5 Exploration-Exploitation

Epsilon values gradually decrease from 1.0, signifying the transition from exploration to exploitation. This shift aligns with the agent's increasing understanding of the environment.

4.6 Memory Size

Fluctuations in memory size reflect the utilization of experience replay, a critical component in stabilizing the training process.

5 RESULTS

The training process revealed the challenging nature of the environment, with the average score plateauing at -1.00 across episodes. The exploration-exploitation tradeoff, governed by the epsilon-greedy strategy, maintained a high exploration factor ($\epsilon = 1.0$) initially, indicating a focus on discovering optimal policies. Each training iteration involved interactions with the environment, reward-based updates, and continuous refinement of the Q-network parameters through backpropagation. Despite the initial exploration emphasis, the model showcased its adaptability, gradually learning from experiences and adjusting its policy to navigate the complex racing environment efficiently. Notably, the achieved score of -1.00 indicates the difficulty of mastering the environment, emphasizing the complexity of the learning task. It's worth mentioning that each run contributes to the model's learning progression, showcasing its resilience and ability to seek improved strategies with each iteration, a crucial aspect in the context of reinforcement learning.

6 CHALLENGES FACED AND FUTURE WORK

6.1 Training Iterations

Repeated Training for Collision Resolution: The training process faced challenges due to frequent collisions with track borders, requiring multiple iterations. This issue extended the training time and hindered efficient learning.

6.2 Multi-Agent Training

Time-Consuming Multi-Agent Training: Initially training a single car was time-consuming. Transitioning to a multi-agent scenario posed further challenges, leading to the use of a shared model for both cars. This simplified approach was adopted for practical reasons.

6.3 Integration of Cars into the Environment

Initial Difficulty in Car Placement: Incorporating two cars into the Pygame environment presented difficulties. As novices with Pygame, the initial attempts to position and control both cars required significant effort. However, we successfully overcame these challenges to create a multi-agent model.

6.4 Collision Handling Between Cars

Incomplete Collision Handling: The implementation lacks a comprehensive collision resolution mechanism for interactions between the two cars. As a result, when executing "main-model-final.py," the cars may overlap or collide without proper handling.

6.5 Incomplete Multi-Agent Training

Limited Learning for Car-to-Car Collisions: Due to challenges in training both cars simultaneously, the model did not effectively learn strategies to avoid collisions between the two cars. This limitation highlights the need for further development to enhance collision avoidance capabilities.

6.6 Proposed changes

The identified challenges serve as valuable insights for future improvements and enhancements in the project. Addressing collision handling, refining multi-agent training strategies, and implementing a robust collision resolution mechanism are key areas for future work. These improvements aim to enhance the model's ability to navigate the environment effectively and avoid collisions, contributing to a more sophisticated and capable multi-agent racing system.

7 CONCLUSION

In conclusion, the project successfully implemented a multi-agent racing environment using Pygame and applied reinforcement learning, specifically Double Deep Q-Network (DDQN), to train agents for navigating the track. The use of DDQN addressed overestimation biases associated with traditional DQN, leading to more stable and robust learning. The integration of Pygame allowed the creation of a visually engaging racing environment, with goals, walls, and rewards defining the dynamics of the game.

Despite facing challenges in training iterations, multi-agent scenarios, and collision handling, the project achieved the primary goal of developing a foundational framework for a multi-agent racing system. The challenges identified provide valuable insights for future work, emphasizing the need for further improvements in collision resolution, enhanced multi-agent training, and advanced strategies for navigating the track.

Moving forward, refining the training process, incorporating advanced collision handling mechanisms, and optimizing multi-agent strategies will contribute to a more sophisticated and capable racing environment. This project serves as a stepping stone for future endeavors in developing intelligent and adaptive agents capable of collaborative and competitive interactions in dynamic environments.

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.
https://www.nature.com/articles/nature14236.

[3] Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-networks. arXiv preprint arXiv:1509.06461. Retrieved from
https://arxiv.org/abs/1509.06461.

[3] Guo, Z. (2023). Deep Reinforcement Learning on Car Racing Game. GitHub Repository. Retrieved from
https://github.com/guozhonghao1994/Deep_Reinforcement_Learning_on_Car_Racing_Game.

[2] CodeAndAction. (2023). DDQN-Car-Racing. GitHub Repository. Retrieved from
https://github.com/CodeAndAction/DDQN-Car-Racing.git.