Path Planning for Multi-Agent Systems Using Deep Q-Networks Reinforcement Learning

Ibrahim Alispahić[1][0009−0001−7010−9111] and Adnan Tahirović[2][0000−0003−2049−8523]

[1] Faculty of Electrical Engineering, Department of Automatic Control and Electronics, Sarajevo, Bosnia and Herzegovina
ialispahic1@etf.unsa.ba

[2] Faculty of Electrical Engineering, Department of Automatic Control and Electronics, Sarajevo, Bosnia and Herzegovina
atahirovic@etf.unsa.ba

Abstract. This work presents a comprehensive study of the application of multi-agent reinforcement learning ($MARL$) based on deep Q-networks (DQN), aiming to enhance the cooperation and coordination of multiple agents in complex environments. The core problem addressed is the multi-agent traveling salesman problem, i.e. the effective collaboration and target-reaching by multiple agents. This challenge is inherently present in a variety of applications, including inspection and maintenance tasks based on autonomous aerial robotic systems. The research delves into the intricacies of multi-agent systems, emphasizing the dynamic interaction between agents and the environment. Although the proposed algorithm is developed and trained in a planar space, its principles and methodologies are easily extendable to any operational environment. The proposed approach is capable of training on a number of different scenarios, using various numbers of agents and targets, regardless of their initial positions, making it highly adaptable to scenarios like aerial inspections, where the number of agents, the number of inspection sites and their locations are unknown a priori. Furthermore, the work outlines how the enhanced DQN algorithm can be tailored for scenarios with numerous autonomous agents and targets, ensuring efficient path planning and target acquisition in constrained and unstructured spaces. This work also provides a solid foundation for future research in integrating advanced reinforcement learning techniques into the design and operational strategies for solving tasks, which can be described by the multi-agent traveling salesman problem, including inspection and maintenance with an autonomous multi-agent system.

Keywords: Multi-agent reinforcement learning ($MARL$) · Deep Q-networks (DQN) · Path Planning · inspection and maintanance.

1 Introduction

The domain of aerial robotics faces the challenge of efficiently coordinating multiple autonomous agents within dynamic environments. This paper introduces

a novel path planning strategy utilizing Multi-Agent Reinforcement Learning (MARL) combined with Deep Q-Networks (DQN), aimed at enhancing path optimization and coordination among agents.

Our research focuses on the multi-agent traveling salesman problem (mTSP), a complex variant where traditional methods often fall short, particularly in adaptability and real-time decision-making. By implementing MARL and DQN, this work facilitates dynamic and adaptive solutions that are crucial for complex operational environments like aerial inspections and maintenance.

This model extends the foundational framework provided by Busoniu et al. [1] and integrates advancements in deep reinforcement learning for routing problems as discussed by Kool et al. [4]. Our approach showcases a shift from static strategies to adaptive, agent-centric coordination, addressing both theoretical and practical gaps in current methodologies.

The subsequent sections detail our contributions to MARL and mTSP, describe the methodologies employed, and present experimental validations of the effectiveness of our proposed solutions.

2 Contribution

This section delineates the significant contributions of our paper within the interlinked domains of *MARL* and *TSP*, highlighting how our research extends and differentiates from existing works. By comparing our findings with seminal and recent advancements in both fields, we aim to underscore the novelty and applicability of our approaches in enhancing the theoretical and practical aspects of multi-agent systems and complex path optimization challenges.

2.1 *MARL*

The complexities of multi-agent systems in aerial robotics have been substantially explored, with foundational insights provided by Busoniu, Babuska, and De Schutter [1]. Their survey establishes the paradigms of *MARL*, detailing key algorithms and challenges. Drawing from this, our research enhances *MARL* with a *DQN* model for real-time adaptability and decision-making in dynamic environments. This progression illustrates the transition of *MARL* from theoretical frameworks to practical applications in aerial robotics, especially in multi-agent path planning.

Our *MARL*-based *DQN* model, designed for the *mTSP*, scales to manage up to four agents and 20 targets, expanding beyond the two-agent scenarios discussed by Tampuu et al. [2]. This model innovates through meticulously calibrated hyperparameters and a nuanced reward function, enhancing strategic coordination and navigation in complex settings. The extensive training periods underscore our commitment to refining the model's robustness and performance in unpredictable scenarios.

This refined *DQN* implementation not only showcases enhanced adaptability but also advances the field by addressing the complexities of *mTSP* in multi-agent systems, demonstrating the practical impact of evolved *MARL* strategies.

2.2 *TSP*

The Traveling Salesman Problem (*TSP*) is fundamental in computational mathematics and operations research, focusing on optimizing the path for visiting each city on a list exactly once. The seminal work by Dantzig, Fulkerson, and Johnson [3] establishes critical strategies for *TSP*, influencing subsequent innovations in this area.

The Multi-Agent Traveling Salesman Problem (*mTSP*) extends *TSP* by incorporating multiple agents, increasing complexity due to the need for inter-agent coordination to optimize overall route efficiency. This variant underlines the challenge of multi-agent systems, especially in scenarios requiring real-time, adaptive path planning.

Our research builds upon these foundational concepts but shifts focus to the application of *TSP* in aerial robotics, addressing the unique challenges posed by dynamic environments and multiple agents. Recent advancements, such as the application of Neural Networks by Kool et al. [4] for combinatorial optimization and policy gradient methods for *TSP* as explored by Deudon et al. [5], further inform our approach. These innovations offer new perspectives and methods that enhance the traditional *TSP* solutions.

Distinctively, our work integrates these advanced computational strategies into multi-agent path planning, emphasizing real-time adaptability and strategic coordination. This focus significantly extends the scope and applicability of *TSP* solutions, positioning our research as a pivotal contribution to the fields of both *TSP* and multi-agent systems.

3 Methodology

3.1 Evolution of Reinforcement Learning in Multi-Agent Systems

Reinforcement Learning (*RL*) has transitioned from a single-agent framework, where the goal is to optimize behavior in a Markov Decision Process (*MDP*), to a more complex Multi-Agent Reinforcement Learning (*MARL*) paradigm. In single-agent *RL*, agents learn to map states to actions to maximize rewards, a concept explored in detail by Sutton and Barto [8]. *MARL* introduces additional complexities as multiple agents interact within the same environment, requiring not only optimal action learning but also adaptation to other agents' behaviors.

This shift adds layers of complexity including coordination and strategic planning among agents, extensively discussed by Busoniu, Babuska, and De Schutter [1]. In *MARL*, the policy function of each agent (π_i) accounts for both individual and collective behaviors, enhancing the capability to navigate dynamic scenarios. Innovative *DQN* strategies by Mnih et al. [17] and advancements in policy optimization by Fujimoto et al. [16] demonstrate how *MARL* can effectively synchronize individual goals with group dynamics.

The following subsections will detail the methodologies of two *MARL* algorithms used in this study: Policy Optimization (*PO*) and Deep Q-Networks (*DQN*), showcasing the current state of *MARL* strategies and their applications in complex environments.

3.2 Policy Optimization in Multi-Agent Systems

PO is a strategic approach in multi-agent systems, particularly effective for complex problems like the multi-agent variant of the *TSP*. At its core, *PO* utilizes a Policy Network, a deep neural network that interprets the environmental state and outputs action probabilities. This network is instrumental in shaping agent behaviors, ensuring that actions are not just executed, but are also based on a thoughtful process involving continuous learning and adaptation from accumulated experiences.

The *PO* algorithm operates through distinct phases within each episode: initialization of the Policy Network, action prediction and execution using an epsilon-greedy strategy for balance between exploration and exploitation, and meticulous analysis of each action's outcomes. Rewards from these actions serve as a direct performance indicator, guiding the crucial phase of Policy Network updates. In this phase, the network weights are adjusted through gradient descent, using calculated losses based on the rewards and action probabilities. Optimizers like the Adam optimizer are crucial in this stage, fine-tuning the network to ensure efficiency and precision in learning, and consequently, in the decision-making process of agents in dynamic, multi-agent environments.

3.3 Deep Q-Network Algorithm for Multi-Agent Systems

After exploring the *PO* algorithm, the *DQN* approach is also analysed. *DQN* integrates deep learning with Q-learning, offering better generalization and efficiency in multi-agent systems, particularly for complex scenarios involving numerous agents and targets.

Algorithm Structure *DQN* combines Q-networks and a replay buffer method, featuring:

- **Q-Networks and target networks:** Estimating Q-values for each action-state pair.
- **Epsilon-Greedy approach:** Balancing exploration and exploitation for action selection.
- **Replay Buffer:** Storing experiences for stable and efficient learning.
- **Network Updating:** Regularly updating Q-networks based on collected experiences and target Q-values.

The pseudo-code (1) illustrates the learning process, highlighting the interaction of these components.

Algorithm Analysis The initialization involves setting up Q-networks and target networks for each agent, establishing the learning environment, and setting key hyperparameters (Table 1).

Algorithm 1 Deep Q-Network (DQN)

```
Initialize Q-networks and target networks for each agent
Set algorithm parameters (e.g., learning rate, epsilon)
Initialize replay buffer
for each episode do
    Reset environment and receive initial state
    while episode not finished do
        for each agent do
            Select action using epsilon-greedy approach
            Execute action and receive reward and new state
            Store experience in replay buffer
        end for
        Update Q-network using experiences from replay buffer
    end while
    Adjust parameters (e.g., decrease epsilon)
end for
```

Hyperparameter	**Definition**
replay_buffer	Size of the replay buffer (1,000,000)
epsilon_start	Initial value for epsilon (1.0)
epsilon_end	Final value for epsilon (0.05)
epsilon_decay	Decay rate for epsilon (0.9975)
gamma	Discount factor (0.8)
tau	Parameter for soft update of the target network (0.01)
num_episodes	Number of training episodes (5,000)
batch_size	Batch size for training (64)

[1] Number of episodes may vary based on the number of agents and targets.

Table 1: Hyperparameter definitions

Training with *DQN* During each episode, agents select actions using the epsilon-greedy strategy, similar to the PO algorithm. Executed actions lead to new states and rewards, which are then stored in the replay buffer. The Q-network is periodically updated based on samples from the replay buffer, following the update rule based on the Bellman equation:

$$Q_{\text{new}}(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{1}$$

where α denotes the learning rate, r is the reward, γ is the discount factor, and $Q(s',a')$ are the Q-values for the new state s' and actions a'. The update process involves:

1. **Sampling from the Replay Buffer:** A batch of experiences (states, actions, rewards, new states, and episode termination info) is sampled from the replay buffer.

2. **Calculating Target Q-Values:** Target Q-values are computed as:

$$Q_{\text{target}}(s', a') = r + \gamma \max_{a'} Q_{\text{target}}(s', a') \tag{2}$$

3. **Calculating Loss:** The loss is calculated using the Mean Squared Error between the target Q-values and the current Q-values:

$$L = \frac{1}{N} \sum (Q_{\text{target}}(s', a') - Q(s, a))^2 \tag{3}$$

4. **Backpropagation and Optimization:** Backpropagation is used to update the weights in the Q-network, optimizing the network parameters:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L \tag{4}$$

This iterative process ensures the continuous improvement of the Q-network, learning more effective decision-making strategies across various *TSP* scenarios.

The implementation of soft target updates and the replay buffer are key to enhancing the performance of the *DQN* algorithm. These techniques play a pivotal role in stabilizing and improving the learning process.

Soft Update of Target Network To stabilize the learning process, a soft update is applied to the target network after updating the Q-network:

$$\theta_{\text{target}} \leftarrow \tau \theta_{\text{Q}} + (1 - \tau) \theta_{\text{target}} \tag{5}$$

where τ represents the update rate (typically a small fraction, e.g., 0.001), θ_{Q} are the weights of the main Q-network, and θ_{target} are the weights of the target network. This technique ensures gradual updates to the target network, contributing to the stability of the learning targets over time.

Replay Buffer The replay buffer is a crucial learning enhancement mechanism in *DQN*. It stores and later randomly samples experiences to form training mini-batches, described by:

$$P(\textit{sample}) = \frac{1}{\textit{number of stored experiences}} \tag{6}$$

where $P(\textit{sample})$ indicates the selection probability of each experience. This random sampling reduces correlations between sequential experiences, promoting more robust and generalized learning outcomes. Advanced replay buffers also prioritize experiences with higher temporal difference (TD) errors, enhancing the efficiency of learning from significant experiences.

The adoption of *DQN* over *PO* notably enhanced stability, quality, and speed of training, even in simple one agent-one target scenarios. In more complex setups involving multiple agents and targets, adjustments such as hyperparameter tuning, replay buffer enhancements, and soft target updates have been crucial for achieving effective training convergence.

3.4 Environment architecture

This subsection outlines the network and environment architecture of our simulations.

The *QNetwork*, crucial for *DQN*, features an input layer matching the environment's state dimensions, two hidden layers with 128 neurons each using ReLU activation, and an output layer generating Q-values. This design balances efficient learning and strategy optimization while preventing overfitting, enhanced by double precision weight conversion for accuracy in complex scenarios.

The *MaTsEnvironment* simulates the *TSP* in a multi-agent context, with a straightforward graphical interface that displays agents as blue dots and targets in red. This layout is designed to maximize movement space and strategic flexibility while avoiding edge constraints.

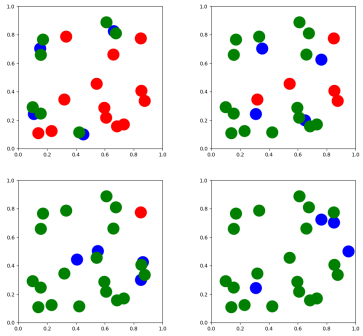

Fig. 1: The image depicts the progression of four agents (in blue) as they locate 20 targets. Unvisited targets are marked in red, while visited targets are highlighted in green.

Visual monitoring within the *MaTsEnvironment* is vital for evaluating agent interactions and strategy effectiveness, as shown in Fig. 1. It supports dynamic, adaptable simulations by initializing agents and targets, defining parameters like agent count and target locations, and randomizing positions to ensure variability.

The environment employs a detailed observation vector that provides agents with extensive insights into the state, including positions, velocities, and distances, reducing computational demands on agents. The `step` function translates actions into environmental changes and manages reward calculations, focusing on target proximity and efficient navigation. This streamlined reward system, refined from earlier complex versions, directly links actions to outcomes, improving performance and learning efficiency.

3.5 Algorithm Validation

Validating the algorithm is crucial to verify its efficacy in multi-agent environments. A Python script, utilizing pre-trained Q networks, rigorously evaluates the algorithm's performance. The script employs *softmax* action selection, which, by considering the Q values, significantly reduces the likelihood of agents getting stuck in local minima, especially in complex scenarios with numerous agents and targets.

The validation involves initializing the *MaTsEnvironment*, loading the Q networks, and executing multiple episodes. During these episodes, the agents' actions and the environmental changes are closely monitored. The *softmax* function selects actions based on a probabilistic model, enhancing the decision-making process. Validation provides insights into the algorithm's performance and pinpointing areas for potential enhancement. This comprehensive approach ensures a deep understanding of the algorithm's operational efficiency and its capability in effectively navigating and solving the multi-agent variant of complex problems.

4 Experimental Results

This section presents an analysis of the experimental outcomes from testing the *PO* and *DQN* algorithms on the *mTSP*. Special emphasis is placed on the *DQN* algorithm due to its notable performance, analyzing how different hyperparameters and modifications affect its efficiency and training quality.

The comparative analysis benchmarks our *MARL* approach against other methods, highlighting its relative efficiency, convergence speed, and adaptability in dynamic environments. The results are substantiated through performance monitoring with `TensorBoard` from the `PyTorch` library, centering on metrics such as *Total Reward*. Notably, the metric *Episode Length* is consistently employed across all figures presented in the paper, offering a uniform standard for evaluating and comparing the performance outcomes.

Detailed implementation specifics, experimental environment nuances, and the algorithms' testing and evaluation methods are streamlined to offer a clear, focused view of the experimental results. The subsequent content will detail these findings, shedding light on the impact of various parameters and the overall effectiveness of the algorithms in tackling the complexities of the *mTSP*.

4.1 Comments on the *PO* algorithm

The experimental outcomes using the *PO* algorithm, though insightful, revealed its limitations in solving the *mTSP*. Despite various adjustments in hyperparameters, reward system modifications, and state vector experimentation, the enhancements in training quality and algorithm efficiency were not as significant as desired. Notably, changes in the learning rate and reward adjustments showed marginal improvements in agent behavior and episode length reduction.

However, the complexity of scenarios, especially with an increased number of targets, posed challenges that the *PO* algorithm struggled to overcome efficiently. The detailed experiments demonstrated that while the *PO* algorithm provided a foundational understanding and some progress in simpler scenarios, its performance plateaued, prompting the exploration of a new approach - the *DQN* algorithm, to better address the intricacies and dynamics of the environment in multi-agent scenarios.

4.2 Comparison of the results

The transition from *PO* to the *DQN* algorithm resulted in significantly improved performance. The initial trials with *DQN* showed remarkable outcomes, with a substantial decrease in average episode length and execution time compared to *PO*, as depicted in Fig. 2. However, the initial *DQN* algorithm's training quality decreased with an increase in the number of agents or targets.

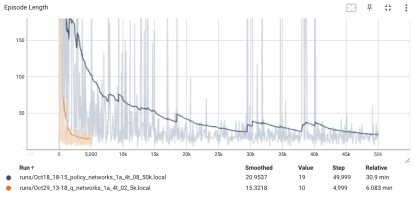

Fig. 2: Comparison of *PO* (gray) and *DQN* (orange) algorithm for the case of 1 agent 4 targets

Efforts to enhance the *DQN* algorithm's performance for larger agent or target numbers included network target adjustments and addressing local minima issues. Modifications such as expanding the environment and adjusting the target network update frequency were explored, with mixed results. The introduction of a more complex reward system and soft target updates (Fig. 3) showed promise in improving training stability and efficiency, yet challenges with local minima persisted.

The application of softmax action selection during the validation phase significantly reduced local minima occurrences, resulting in more stable and improved validation results, as shown in Fig. 4. Additionally, the use of weight decay in the Adam optimizer led to better training outcomes, especially in complex scenarios with more agents and targets (Fig. 5).

Further enhancements were explored by conducting targeted training for specific configurations, which expectedly proved to be significantly faster and more focused compared to general model training. This approach, illustrated in Fig. 6, demonstrates the potential for achieving optimal paths for fixed positions, after model is partly trained on general scenario. The progress of four agents in

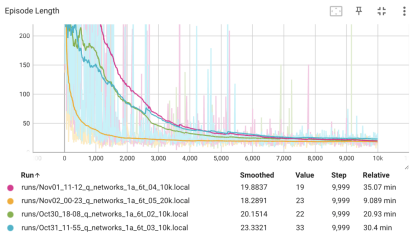

Fig. 3: Comparison of different experiments with the *DQN* algorithm for 1 agent 6 targets, including the effect of soft target network updates (orange)

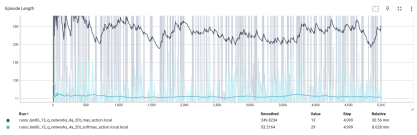

Fig. 4: Comparison of validation average episode length of the *DQN* algorithm for 4 agents and 20 targets, with (blue) and without (gray) *softmax* action selection

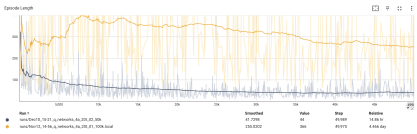

Fig. 5: Comparison of training the *DQN* algorithm for 4 agents and 20 targets, with (gray) and without (yellow) *weight decay* factor

finding 20 targets is visually documented, showcasing the practical application of the *DQN* algorithm in complex multi-agent scenarios.

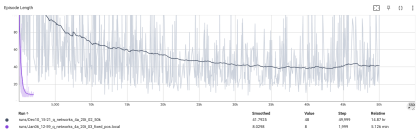

Fig. 6: Comparison of training the *DQN* algorithm for 4 agents and 20 targets for general scenario (gray) and fixed positions of agents and targets (purple)

5 Conclusion and Future Work

This research has significantly advanced multi-agent path planning by addressing a specifically designed multi-agent *TSP* (*mTSP*) challenge, focusing on en-

hancing the *DQN* algorithm. It tackled the dynamic complexities of multi-agent, multi-target environments, providing a detailed analysis and experimental evaluation of *DQN* aspects like training speed, scalability, and adaptability. Our modifications led to marked improvements in performance, particularly in scenarios involving many agents and targets, demonstrating clear benefits over traditional *TSP* solutions.

The findings propose several directions for future research to enhance path optimization and reinforcement learning applications:

1. Implement the Twin Delayed DDPG (*TD3*) algorithm to improve learning and solution quality in complex *mTSP* scenarios, inspired by the advancements in continuous control by Lillicrap et al. [14].
2. Apply Curriculum Learning for more efficient training in complex settings with multiple agents and targets.
3. Innovate neural network structures by exploring varied layers, activation functions, and regularization techniques, guided by the deep learning foundations set by Ioffe and Szegedy [15].
4. Extend these methodologies to other optimization problems or similar dynamic domains, following the innovative approaches by Kool et al. [4] for solving routing challenges.
5. Continuously integrate the latest advancements in path optimization and reinforcement learning to improve the solutions developed in this research.

These strategic directions are expected to propel forward the theoretical and practical capabilities in addressing complex *mTSP* and related challenges, thereby contributing to the ongoing evolution in the field.

Acknowledgment

This work has been supported in part by the scientific project "Strengthening Research and Innovation Excellence in Autonomous Aerial Systems - AeroSTREAM," supported by the European Commission HORIZON WIDERA-2021-ACCESS-05 Programme through the project under G.A. number 101071270.

References

1. L. Busoniu, R. Babuska, and B. De Schutter, "A Comprehensive Survey of Multi-Agent Reinforcement Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.
2. A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, et al., "Multi-agent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, e0172395, 2017.
3. G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

4. Wouter Kool, Herke van Hoof, and Max Welling, "Attention, Learn to Solve Routing Problems!" in *International Conference on Learning Representations*, 2018, url=https://api.semanticscholar.org/CorpusID:59608816
5. Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau, "Learning Heuristics for the TSP by Policy Gradient," in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2018)*, pp. 170–181, 2018.
6. R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," *MIT Press*, 2018.
7. Y. Zhang and M. Yang, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," *Handbook of Reinforcement Learning and Control*, Springer, 2021.
8. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 2018.
9. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Pearson, 2010.
10. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*, Springer, 1986, pp. 396–404.
11. M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.
12. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.
13. A. L. C. Ottoni, E. G. Nepomuceno, M. S. d. Oliveira, et al., "Reinforcement learning for the traveling salesman problem with refueling," *Complex Intelligent Systems*, vol. 8, pp. 2001–2015, 2022. doi:10.1007/s40747-021-00444-4.
14. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," *Google DeepMind*, London, UK, 2016.
15. S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Google*, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, 2015.
16. S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," 2018.
17. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
18. L.-J. Lin, "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, Springer, 1992.