Reinforcement Learning for Collision Avoidance in Autonomous Driving

Nazmus Sakib
Department of Computer Science
University of Colorado Colorado Springs
nsakib@uccs.edu

Abstract

This study assesses the effectiveness of six reinforcement learning (RL) algorithms: Q-Learning, Deep Q-Learning, SARSA, MLP SARSA, DDPG, and PPO, in facilitating dynamic collision avoidance for autonomous vehicles within a simulated urban environment. Utilizing two primary performance metrics: Cumulative Reward per Episode and Number of Steps to Reach Goal, our analysis offers a direct comparison of these algorithms' operational efficiency. The results indicate that SARSA, Q-Learning, and DDPG consistently outperform the others in terms of both maximizing cumulative rewards and minimizing the travel steps required to reach objectives. These algorithms demonstrated a more aggressive approach to navigating complex environments. Conversely, PPO, MLP SARSA, and MLP Q-Learning exhibit more conservative behaviors, potentially compromising path efficiency. This provides a concise synthesis of our findings, suggesting significant distinctions in algorithmic performance that could influence the strategic deployment of RL techniques in autonomous driving systems.

Introduction

A fundamental concept of ensuring the safety of drivers and their vehicles is collision avoidance (Yurtsever et al., 2020). Due to rising vehicle density, autonomous vehicle navigation is becoming a difficult issue. There are two types of obstacles that can be found on roads: static obstacles, such as cars parked there, and dynamic obstacles, which include moving objects like animals that move randomly (Almazrouei et al., 2023). Autonomous driving requires the ability to detect and avoid both static and dynamic obstacles. To ensure safety, various sensors, including vision, ultrasonic radar, and lidar, need to be used. Vision sensors can distinguish and classify obstacles, but they cannot precisely determine their distance for action. Combining ultrasonic radar or lidar data with a vision sensor improves obstacle detection accuracy in various settings. Supervised learning outperforms traditional approaches in detecting objects using vision and ultrasonic radar. However, this strategy requires extensive ground truth data from various situations to train the machine learning model (Arvind and Senthilnath, 2019). To overcome this limitation, combining supervised and reinforcement learning may eliminate the need to manually provide ground truth for obstacles (Babu et al., 2016). Reinforcement learning (RL) is a machine learning technique that allows an agent, such as an autonomous car, to learn its surroundings based on prior actions, states, and rewards (Kaelbling et al., 1996). In recent years, there has been an increase in the use of RL for dynamic collision avoidance in autonomous driving (Kim et al., 2020).

The goal of this research is to assess several reinforcement learning algorithms for dynamic collision avoidance in autonomous driving. We identified six widely used reinforcement learning models: Q-Learning, Deep Q-Learning, SARSA, MLP SARSA, DDPG, and PPO. Thus, the primary objective of this study is to evaluate the effectiveness of these six reinforcement learning models for autonomous driving on a busy road.

Background

Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technique in which an agent functions in a given environment and tries to find a policy (π) that maximizes a cumulative reward function (Wiering and Van Otterlo, 2012). The policy specifies what action should be done, a, in a state, s. The environment will thereafter shift to a new state, s', and yield a reward, r.

Q-Learning

The agent in Q-Learning, a reinforcement learning algorithm, tries to learn $Q^*(s, a)$, the optimal action-value function. The maximum expected return that results from being in a state (s), acting (a), and then according to the best course of action (π) is the definition of this function (Jamshidi et al., 2021).

Deep Q-Learning

Deep Q-Learning is an advanced reinforcement learning technique that integrates deep neural networks with Q-learning, a value-based method for determining the optimal action-selection policy. In Deep Q-Learning, a deep neural network, known as the Q-network, is used to approximate the Q-value function, which estimates the value of taking a given action in a particular state. This approach allows the agent to handle complex, high-dimensional environments that traditional Q-learning methods struggle with due to their tabular nature (Cai et al., 2022). For a state-action pair (s, a), the Q-value is updated as:

$Q_{new}(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a'))$

State Action Reward State Action (SARSA)

The State-Action-Reward-State-Action (SARSA) algorithm is a fundamental approach in reinforcement learning that operates on the principle of learning an action-value function to guide the actions of an agent within an environment (Arvind and Senthilnath, 2019). Unlike Q-Learning, which is based on the principle of maximizing future rewards, SARSA takes a more conservative approach by considering the actual next action the agent will take according to its current policy. In essence, SARSA updates its action-value function, $Q(s, a)$, based on the observed transition from the current state-action pair (s, a) to the next state-action pair (s', a') and the reward received in the process.

Multi-Layer Perceptron (MLP) based State-Action-Reward-State-Action (SARSA)

The Multi-Layer Perceptron (MLP) based State-Action-Reward-State-Action (SARSA) reinforcement learning method integrates the classic SARSA algorithm with the power of neural networks, specifically MLPs, to handle environments with high-dimensional state spaces (Arvind and Senthilnath, 2019). In this approach, the MLP is used to approximate the action-value function $Q(s, a)$, enabling the agent to learn and generalize across a vast number of states and actions. By inputting the state (and possibly the action) into the MLP, the network outputs the estimated values of taking each action in the given state, effectively learning the optimal policy through iterative updates. The SARSA update rule is applied in a way that the target for the MLP's output is adjusted based on the reward received and the estimated value of the next state-action pair, $Q(s', a')$, according to the policy being followed.

Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) algorithm combines aspects of Q-learning and policy gradient methods, using a deterministic policy and an off-policy learning approach (Chen et al., 2022). It utilizes an actor-critic approach, where the actor provides the current policy by mapping states to actions, and the critic evaluates the action by computing the value function. The actor network directly maps states to actions, representing the policy $\pi(s|\theta^{\pi})$, where θ^{π} are the parameters of the actor network. The critic evaluates the policy by computing the value of state-action pairs, essentially approximating the Q-function $Q(s, a|\theta^{Q})$, where θ^{Q} are the parameters of the critic network.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) aims to improve learning stability and efficiency in reinforcement learning by controlling the size of policy updates. It does so through an objective function that encourages the new policy to stay close to the old policy, ensuring that the updates are not too large, which could destabilize training (Wei et al., 2019). For an action a taken in state s, the policy ratio is given by:

$r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)}$

Related Work

We examine several studies on the application of reinforcement learning models to autonomous driving. A couple of related works are listed below.

(Babu et al., 2016) developed an autonomous agent that leverages Q-learning methods to find the shortest path from a current state to a desired state in a fully dynamic environment based on camera data. Similarly, (Hong et al., 2017) utilized a fuzzy Q-learning method to identify obstacles in a dynamic environment utilizing input from ultrasonic sensors. The State Action Reward State Action (SARSA) method was employed by (Rais et al., 2023) to avoid collisions in autonomous vehicles on highways. In an urban dynamic environment scenario, (Arvind and Senthilnath, 2019) employed the State Action Reward State Action (SARSA) method using Multi Layer Perception (MLP) for autonomous vehicle obstacle detection and avoidance. Authors in (Arvind and Senthilnath, 2020) employed a policy-free, model-free Q-learning based reinforcement learning algorithm with a multi-layer perceptron neural network (MLP-NN) to forecast the best course of action for the vehicle in the future, given its present state. Moreover, Unlike previous studies, authors in (Guan et al., 2020) utilized a centralized controller using proximal policy optimization (PPO) to manage all vehicles approaching the intersection, optimizing their trajectories in real time to prevent collisions and improve traffic flow efficiency. Similarly, authors in (Siboo et al., 2023) proposed a Deep Deterministic Policy Gradient (DDPG) based decision-making model to address a sequential decision problem in autonomous vehicles.

In contrast to earlier studies, the objective of this research is to evaluate the performance of reinforcement learning algorithms for driving autonomously over a busy road. In addition, we aim to assess the performance of these popular reinforcement learning methods by training them in a dynamic collision avoidance urban simulation scenario.

Proposed Methodology and Environment Setup

The objective of our research is to develop and refine the capabilities of an autonomous car agent, enabling it to navigate through complex urban environments, specifically across densely populated highways, while avoiding any collisions with other vehicles. This shows the autonomous agent's ability to dynamically adjust its speed and positioning on the driving, leveraging a series of discrete actions tailored to navigate towards its designated destination location with optimal efficiency. To achieve this, we have implemented six model-free reinforcement learning algorithms (e.g., Q-Learning, MLP Q-Learning, SARSA, MLP SARSA, DDPG, and PPO). These algorithms are instrumental in guiding the agent towards developing an optimal action-selection policy. Through a process characterized by both exploration of the environment and exploitation of acquired knowledge using neural networks, the agent incrementally maximizes its cumulative reward. This is achieved by continuously updating its understanding and valuation of state-action pairings (Q-values) across a multitude of simulation episodes.

Environment Setup:

For the purpose of our experiments, we create a grid world simulation using a popular Python library called 'pygame' (Gym and Sanghi, 2021) that mirrors the complexities of an urban traffic system, filled with dynamic obstacles. The model takes the form of a 6x20 grid, conceptualized to emulate a highway scenario where each column represents a lane on the highway and each row symbolizes different positions within those lanes. For our simulation, we have assumed that all traffic flows unidirectionally, and we have placed various other cars (acting as dynamic obstacles) throughout this virtual highway to simulate real-life driving conditions. Figure 1 illustrates our environmental setup to train RL agent car.

States: The state space is defined by a three-dimensional grid representing the environment in which the agent car operates. Each state is a tuple (row, column, speed), where: row and column define the agent's position in the grid, with the grid having 20 rows and 6 columns and speed represents the agent's speed level, discretized into three levels: low (0), medium (1), and high (2). The total state size is the product of the number of rows, columns, and speed levels, resulting in a 360 distinct states (20 rows * 6 columns * 3 speed levels).

Actions: The agent can perform one of five actions at any given state:

- **Forward:** Move forward in the grid, with the distance moved dependent on the current speed level.
- **Lane Change Left:** Move one column to the left, if not already in the leftmost column.
- **Lane Change Right:** Move one column to the right, if not already in the rightmost column.
- **Accelerate:** Increase the speed level by one, unless already at the maximum speed.
- **Decelerate:** Decrease the speed level by one, unless already at the minimum speed.

Rewards: The reward structure is designed to guide the agent car towards the goal while avoiding obstacles:

- **Moving Forward:** A default reward of -1.0 is assigned to encourage the agent to reach the goal with as few steps as possible.
- **Collision:** A significant penalty of -200.0 is assigned if the agent collides with an obstacle (other cars) at any speed, discouraging collisions.
- **Reaching the Goal:** A reward of 200.0 is given for reaching the goal state, incentivizing the agent to navigate towards the goal.

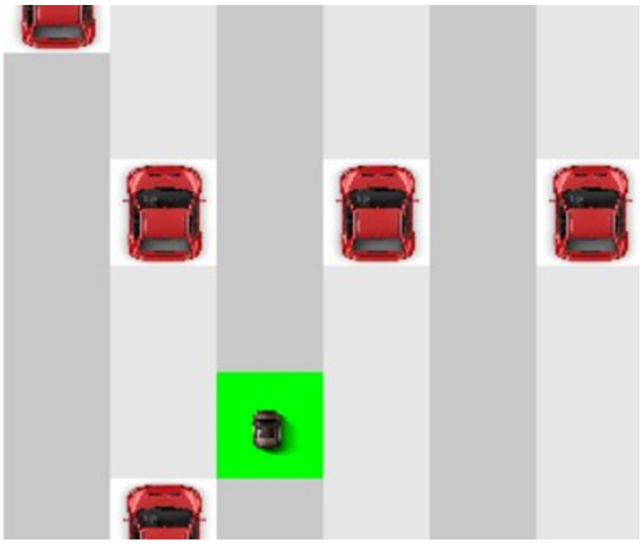
Figure 1: Experimentation Setup and Implementation. Here, red cars denote the dynamic obstacles while the green background car denotes the trained agent car and six shaded lines denote six lanes in the highway.

Parameter Selection, Running Experiments, Simulation Results, and Discussion

Parameter Selection:

Parameter selection is a crucial step in setting up reinforcement learning algorithms because the chosen parameters significantly influence the learning rate, the balance between exploration and exploitation, input, hidden, and output layers in the neural network, and the convergence towards an optimal policy. For our autonomous car navigating a grid-like representation of a highway, the following parameters are considered:

- **Learning Rate:** We consider 0.1 as a starting learning rate, allowing the agent to update its Q-values by considering 10% of new learning while retaining 90% of the old value.
- **Discount Factor:** We consider 0.95 as a discount factor.
- **Exploration Rate and Exploration Decay Rate:** Initially we consider 1.0 as a exploration rate but later we introduce a decay rate of 0.001 which ensures a gradual shift from exploration to exploitation.
- **Neural Network Architecture:** Three input layer (corresponding to the normalized values of row, column, and speed), two hidden layers; 128 neurons in each hidden layer, and seven output layer (corresponding to the number of possible actions). Additionally, ReLU (Rectified Linear Unit) for hidden layers activation function, Adam as an optimizer, Learning Rate: 0.001, Loss Function: Mean Squared Error (MSE).
- **Actor-Critic Network Architecture:** Similar to Neural Network Architecture: Input Layer Size: 3 (corresponding to the normalized values of row, column, and speed),

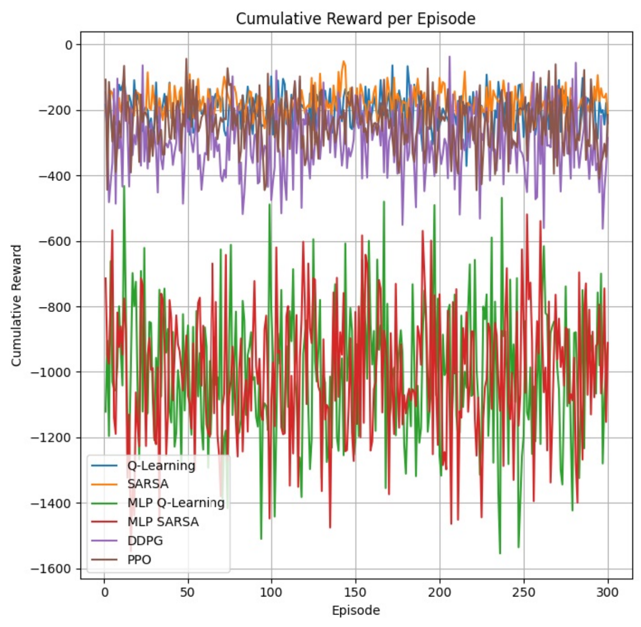

Figure 2: Comparisons result between Q-Learning, MLP Q-Learning, SARSA, MLP SARSA, DDPG, and PPO based on Cumulative Reward per Episode.

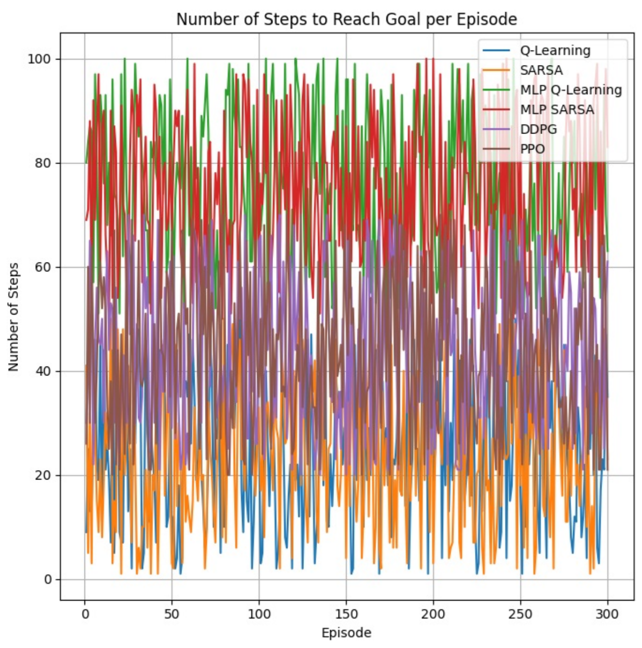

Figure 3: Comparisons result between Q-Learning, MLP Q-Learning, SARSA, MLP SARSA, DDPG, and PPO based on Number of Steps to Reach Goa State.

Hidden Layers: Two hidden layers: 128 neurons in each hidden layer, and Output Layer Size: 7 (corresponding to the number of possible actions). Additionally, Activation Function: ReLU (Rectified Linear Unit) for hidden layers and Softmax for output layer to generate a probability distribution over actions, Learning Parameters: Optimizer: Adam, Learning Rate: 0.001, Loss Function: Mean Squared Error (MSE).

Running Experiments:

Running experiments involves using the selected parameters to train the agent over a series of episodes, allowing it to learn from its environment and adjust its policy accordingly. Here is how we typically unfold:

- **Number of Episodes:** The total number of episodes for training significantly impacts the agent's performance. Each episode provides an opportunity for the agent to explore the environment, make decisions, and learn from the outcomes. For the given setup, we consider 1,000 episodes which allow the agent sufficient interactions to learn an effective policy for navigating from the start to the goal while avoiding obstacles.
- **Per-Episode Process:** Within each episode, the agent starts from the predefined starting point and makes decisions at each step based on either exploration (random choice) or exploitation (choosing the best-known action). The outcomes of these actions (new state and reward) are then used to update the Q-values.
- **Q-Value Update:** After each action, the Q-value for the state-action pair is updated, incorporating the immediate reward and the discounted value of the next best/randomly selected action. This process iteratively improves the agent's strategy.

Simulation Results and Discussion:

To evaluate the performance of the above mentioned six RL algorithms, we consider the following two performance evaluation matrices:

Cumulative Reward per Episode: This metric sums up all the rewards obtained by the agent in each episode. By plotting the cumulative reward per episode for each algorithm, we assess how effectively each algorithm maximizes rewards over time.

Number of Steps to Reach Goal: This metric counts the number of steps the agent takes to reach the goal in each episode. Plotting the number of steps per episode for both algorithms highlights their efficiency improvements or declines over time.

To visually compare the performance efficiency between these six algorithms, we create two Plots that show the Cumulative Reward Over Episodes and Number of Steps to Reach Goal. Figure 2 illustrates the performance comparisons between algorithms based on Cumulative Reward Over Episodes whereas Figure 3 illustrates the performance comparisons between algorithms based on the Number of Steps to Reach Goal.

From the results (based on Figure 2 and Figure 3), it can easily be seen that in terms of Cumulative Reward, SARSA, Q-Learning, and DDPG demonstrate a more aggressive approach towards maximizing rewards, possibly outperforming MLP SARSA, MLP Q-learning, and PPO in achieving higher cumulative rewards over episodes. Additionally, when evaluating the Steps to Reach Goal, similarly, SARSA, Q-Learning, and DDPG potentially find shorter or more efficient paths to the goal compared to PPO, MLP SARSA, and MLP Q-Learning. PPO, being more conservative due to its on-policy nature, takes into account the current policy's actions, which may result in slightly longer paths as it avoids potential penalties more cautiously.

Limitations of Our Experiments :

Thus far, we have only implemented six RL algorithms in our experiments. Other algorithms need to be implemented (e.g., A3C, TRPO) (Kiran et al., 2021). Our next goal is to implement other algorithms along with these six algorithms and analyze the results. In addition, we aim to train agent cars in a more complex urban simulation environment with dynamic obstacles (e.g., bidirectional traffic with different speeds).

Conclusion

This research provides a comprehensive evaluation of six RL algorithms in the context of dynamic collision avoidance for autonomous driving. The findings reveal that SARSA, Q-Learning, and DDPG excel in maximizing cumulative rewards and minimizing the number of steps to the goal, indicating their potential for aggressive and efficient navigation strategies. In contrast, PPO, MLP SARSA, and MLP Q-Learning show a tendency towards more conservative decision-making, which might affect their path efficiency. These insights suggest that the choice of RL algorithm can significantly influence the performance of autonomous vehicles in complex urban settings. Future work could explore the integration of these algorithms into real-world autonomous systems to further validate and refine their applicability and effectiveness.

References

[Almazrouei et al., 2023] Almazrouei, K., Kamel, I., and Rabie, T. (2023). Dynamic obstacle avoidance and path planning through reinforcement learning. *Applied Sciences*, 13(14):8174.

[Arvind and Senthilnath, 2019] Arvind, C. and Senthilnath, J. (2019). Autonomous rl: Autonomous vehicle obstacle avoidance in a dynamic environment using mlp-sarsa reinforcement learning. In *2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR)*, pages 120–124. IEEE.

[Arvind and Senthilnath, 2020] Arvind, C. and Senthilnath, J. (2020). Autonomous vehicle for obstacle detection and avoidance using reinforcement learning. In *Soft Computing for Problem Solving: SocProS 2018, Volume 1*, pages 55–66. Springer.

[Babu et al., 2016] Babu, V. M., Krishna, U. V., and Shahensha, S. (2016). An autonomous path finding robot using q-learning. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–6. IEEE.

[Cai et al., 2022] Cai, P., Wang, H., Sun, Y., and Liu, M. (2022). Dq-gat: Towards safe and efficient autonomous driving with deep q-learning and graph attention networks. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):21102–21112.

[Chen et al., 2022] Chen, Y., Han, W., Zhu, Q., Liu, Y., and Zhao, J. (2022). Target-driven obstacle avoidance algorithm based on ddpg for connected autonomous vehicles. *EURASIP Journal on advances in signal processing*, 2022(1):61.

[Guan et al., 2020] Guan, Y., Ren, Y., Li, S. E., Sun, Q., Luo, L., and Li, K. (2020). Centralized cooperation for connected and automated vehicles at intersections by proximal policy optimization. *IEEE Transactions on Vehicular Technology*, 69(11):12597–12608.

[Gym and Sanghi, 2021] Gym, O. and Sanghi, N. (2021). *Deep reinforcement learning with python*. Springer.

[Hong et al., 2017] Hong, J., Tang, K., and Chen, C. (2017). Obstacle avoidance of hexapod robots using fuzzy q-learning. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6. IEEE.

[Jamshidi et al., 2021] Jamshidi, F., Zhang, L., and Nezhadalinaei, F. (2021). Autonomous driving systems: Developing an approach based on a* and double q-learning. In *2021 7th International Conference on Web Research (ICWR)*, pages 82–85. IEEE.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

[Kim et al., 2020] Kim, M., Lee, S., Lim, J., Choi, J., and Kang, S. G. (2020). Unexpected collision avoidance driving strategy using deep reinforcement learning. *IEEE Access*, 8:17243–17252.

[Kiran et al., 2021] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926.

[Rais et al., 2023] Rais, M. S., Boudour, R., Zouaidia, K., and Bougueroua, L. (2023). Decision making for autonomous vehicles in highway scenarios using harmonic sk deep sarsa. *Applied Intelligence*, 53(3):2488–2505.

[Siboo et al., 2023] Siboo, S., Bhattacharyya, A., Raj, R. N., and Ashwin, S. (2023). An empirical study of ddpg and ppo-based reinforcement learning algorithms for autonomous driving. *IEEE Access*, 11:125094–125108.

[Wei et al., 2019] Wei, H., Liu, X., Mashayekhy, L., and Decker, K. (2019). Mixed-autonomy traffic control with proximal policy optimization. In *2019 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE.

[Wiering and Van Otterlo, 2012] Wiering, M. A. and Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729.

[Yurtsever et al., 2020] Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. (2020). A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469.