令和 5 年度 修士論文

Traffic Intersection Negotiation Using Multi-Agent Policy Optimization

主指導教員

山崎達也 教授

新潟大学 大学院の所属を書く

KHEANG Kim Ang

Abstract

This study investigates negotiation tasks at intersections for autonomous vehicles and proposes a multi-agent reinforcement learning approach to tackle this challenge.

Negotiating interactions at intersections is crucial for safe and efficient autonomous driving, requiring intelligent decision-making. However, traditional approaches, such as Proximal Policy Optimization (PPO) with Gaussian distribution, face limitations in capturing the complex dynamics of negotiation scenarios. We introduce the utilization of the Beta distribution as an alternative to the Gaussian distribution. The Beta distribution offers several advantages, including greater flexibility in modeling continuous actions and a more accurate representation of the uncertainty associated with negotiation tasks. By leveraging the properties of the Beta distribution, we can enhance the agents' decision-making capabilities, leading to improved negotiation outcomes.

Our approach employs a centralized learning phase and decentralized execution. In the centralized learning phase, agents learn from a global perspective, considering the interactions and dynamics of all vehicles at the intersection. This enables the agents to learn effective negotiation strategies and understand the impact of their actions on the overall traffic flow. During decentralized execution, agents make decisions independently in real time, allowing for adaptive negotiation behaviors based on local observations.

Moreover, we focus on a dual-channel architecture that predicts both the alpha and beta values of the Beta distribution. This architecture provides fine-grained control over the agents' behavior, allowing them to dynamically adjust their negotiation strategies based on the specific context of the intersection and the behavior of other vehicles. By accurately predicting the alpha and beta values, the agents can make informed decisions and exhibit more nuanced negotiation behaviors.

To further improve negotiation performance, we introduce a shared value advantage function that incorporates the joined actions and states of the agents. This shared value function captures the collective impact of all agents, promoting coordination and cooperation in negotiation tasks. By considering the holistic effects of the agents' actions, our

approach leads to better negotiation outcomes and improved traffic flow efficiency.

The experiment is carried out through simulation which is a low-cost option to train autonomous vehicles. Through extensive experimentation and comparisons, we demonstrate that our proposed approach outperforms PPO with Gaussian distribution in terms of learning rate and overall performance. The utilization of the Beta distribution, combined with the dual-channel architecture and the shared value advantage function, enables our agents to exhibit interesting emergent behaviors at intersections. These behaviors include slowing down to let other vehicles pass or strategically cutting through intersections, showcasing the agents' adaptability and effective negotiation skills.

Overall, this thesis presents a comprehensive multi-agent reinforcement learning approach for negotiation tasks at intersections for autonomous vehicles. By utilizing the Beta distribution, employing a centralized learning decentralized execution framework, adopting a dual-channel architecture, and incorporating a shared value advantage function, we achieve superior negotiation performance. The emergent behaviors exhibited by the agents highlight the efficacy and adaptability of our approach in complex negotiation scenarios.

Our findings contribute to the advancement of negotiation tasks in intersection scenarios for autonomous vehicles, facilitating safer and more efficient traffic flow. By leveraging multi-agent reinforcement learning techniques and addressing the limitations of traditional approaches, we pave the way for the development of intelligent and socially aware autonomous vehicles capable of effectively navigating intersections and engaging in successful negotiation protocols.

目 次

図 目 次

表 目 次

第1章 Introduction

1.1 Background

Reinforcement learning (RL) has emerged as a transformative paradigm in the field of control engineering, enabling agents to learn how to act in complex environments through observation and interaction. This learning approach has seen significant advancements in recent years, largely driven by the exponential growth in computing capability. The integration of deep learning with RL has further accelerated progress, allowing agents to be trained end-to-end by directly processing raw sensor data through neural networks to predict appropriate actions [1–5].

Within the realm of RL, various algorithms have been developed [4, 6–8], each dictating how an agent learns from its environment. These algorithms can broadly be categorized as model-based and model-free methods. In the context of autonomous vehicles(AVs) navigating intersections, agents must acquire the skills to negotiate interactions effectively. They should be capable of adapting their behavior to be either collaborative, cooperative, or competitive, depending on the situation, in order to safely and efficiently cross intersections.

However, a notable limitation of conventional RL algorithms is that they were primarily designed with the assumption of single-agent environments, where one agent interacts with the environment in isolation. In contrast, intersection scenarios involve multiple agents (vehicles) interacting with one another, necessitating the development of multi-agent reinforcement learning (MARL) techniques [9].

Efforts have been made to extend model-free methods for MARL purposes, aiming to enable agents to learn in collaborative and competitive multi-agent environments. Nev-

ertheless, many of these MARL methods have tended to simplify policy architectures, overlooking essential aspects such as the choice of probability distribution density function or the design of agents' policy architectures.

1.2 Motivation

The motivation behind this research stems from the urgent need to address the challenges associated with negotiation tasks at intersections for AVs. As autonomous driving technology continues to advance rapidly, intersections remain one of the most complex and critical environments for AVs to navigate safely and efficiently. Traditional approaches, built on single-agent assumptions and Gaussian distributions, fall short in capturing the intricacies of multi-agent negotiation scenarios.

The potential benefits of effective intersection negotiation are significant, including enhanced traffic flow, reduced congestion, and increased road safety. By enabling AVs to negotiate intersections collaboratively, competitively, or in mixed settings, we can create a more harmonious and efficient traffic ecosystem. Furthermore, successful negotiation protocols at intersections are crucial for the widespread adoption and acceptance of autonomous driving technology.

The adoption of multi-agent reinforcement learning and the utilization of the Beta distribution present promising avenues to revolutionize intersection negotiation. The ability to fine-tune negotiation strategies, consider the joint impact of all agents, and accurately model uncertainty using the Beta distribution are all factors that can greatly improve the negotiation capabilities of AVs.

The research seeks to push the boundaries of existing techniques and bridge the gap between single-agent and multi-agent reinforcement learning. By designing a robust and adaptable negotiation framework, we aim to empower AVs to make well-informed and context-aware decisions at intersections, leading to safer, smoother, and more efficient traffic flow.

Ultimately, this study's motivation lies in contributing to the development of socially-aware and intelligent AVs that can effectively interact with other agents and adapt to various negotiation scenarios at intersections. By overcoming the limitations of conventional approaches, we aspire to pave the way for the widespread integration of AVs in urban environments, revolutionizing transportation and advancing the future of mobility.

1.3 Problem Statement

The problem of negotiating interactions at intersections for AVs poses significant challenges in the field of autonomous driving. Traditional RL algorithms, primarily designed for single-agent environments, struggle to address the complexities of multi-agent negotiation scenarios. While some efforts have been made to extend model-free methods for MARL purposes, these approaches often overlook crucial aspects, such as the choice of probability distribution functions and the design of policy architectures. Additionally, conventional RL algorithms typically rely on Gaussian distributions, which may not adequately capture the complexities of negotiation tasks. As a result, there is a need for a novel and sophisticated approach that leverages the advantages of MARL, centralized learning with decentralized execution, and the Beta distribution to effectively train autonomous agents to negotiate intersections collaboratively, competitively, or in a mixed setting. Addressing these challenges will significantly advance the capabilities of AVs, leading to safer and more efficient traffic flow at intersections.

1.4 Objectives and Scope of the Study

The primary objective of this study is to develop and evaluate a novel MARL approach for negotiating interactions at intersections for AVs. The specific objectives are as follows:

- To design a robust and efficient multi-agent reinforcement learning framework that enables AVs to navigate intersections safely and efficiently by learning collaborative, competitive, and mixed negotiation strategies.

- To explore the advantages of the Beta distribution over traditional Gaussian distributions in modeling continuous actions and uncertainty during negotiation tasks, thereby enhancing the adaptability and decision-making capabilities of the autonomous agents.

- To propose a dual-channel architecture for predicting both the α and β values of the Beta distribution, empowering the agents to fine-tune their negotiation strategies based on the specific context of the intersection and the behavior of other vehicles.

- To develop a shared value advantage function that incorporates joint actions and states of the agents, promoting coordination and cooperation among the AVs during negotiation scenarios, leading to improved negotiation outcomes and overall traffic flow efficiency.

- To conduct extensive experiments and comparisons with conventional PPO algorithms using Gaussian distributions to demonstrate the superior performance and learning rate achieved by our proposed methodology.

This work focuses on model-free unsignalized intersections with no sign where only AVs exist in a homogenous environment. The agents learn together in a mixed cooperative and competitive nature to ensure that the intersection crossing is safe and efficient. This is known as decentralized learning as each agent has its own observation and make decision separately which can ensure emergent behaviors like in the case of multi-agent hide and seek and more [10–13]. However, state-of-the-art MARL methods often overlook the importance of agent policy architecture if it is suitable for environment and try to make the learning generalized. This work focuses on extending the policy gradient method to multi-agent version and using Beta distribution instead of standard Gaussian distribution instead. We also put emphasis on agent's policy architecture designed specifically for Beta distribution. The architecture predicts α, β values to feed to distribution density function in order to sample steering angle and acceleration value to control the vehicle. The research scope encompasses the development of a multi-agent reinforcement learning

approach, specifically centered on Proximal Policy Optimization (PPO) with continuous actions and the utilization of the Beta distribution. The investigation will primarily involve AVs acting as the agents, aiming to negotiate interactions with other vehicles in a dynamic traffic environment. The scope includes the exploration of various policy architectures, with particular emphasis on the dual-channel architecture for predicting the α and β values of the Beta distribution. The shared value advantage function will also be integrated to enable agents to consider the collective impact of their actions and enhance coordination during negotiation. The performance of the proposed methodology will be evaluated through extensive simulations and comparisons with traditional PPO algorithms utilizing Gaussian distributions. The evaluation metrics will include learning rate, negotiation outcomes, and the emergence of interesting behaviors such as slowing down to let other vehicles pass or cutting through intersections. It is important to note that this study focuses specifically on the intersection negotiation tasks for AVs and does not cover other aspects of autonomous driving, such as motion planning or perception. Additionally, while the study aims to achieve superior performance in negotiation scenarios, it may not address all possible scenarios that AVs may encounter in real-world urban environments.

第2章　Literature Review

2.1 Reinforcement Learning in Traffic Intersection

RL has shown great promise in enabling vehicles to make intelligent decisions on the road. RL algorithms allow autonomous vehicles to learn from their interactions with the environment, such as other vehicles, pedestrians, traffic lights, and road conditions. This learning process helps the vehicles adapt to changing situations and make appropriate decisions in real-time.

One of the significant advantages of RL in autonomous driving is its ability to handle complex and uncertain environments. RL agents can navigate through various traffic scenarios, including intersections, merging lanes, and lane-changing maneuvers [14,15], by learning optimal policies based on trial and error. RL also allows autonomous vehicles to adapt to diverse driving conditions, such as different weather conditions and traffic patterns.

Applications of RL in autonomous driving include:

- Trajectory Planning: RL can be used to plan safe and efficient trajectories for autonomous vehicles, considering various factors such as traffic density, speed limits, and road conditions.
- Behavior Prediction: RL algorithms can predict the behavior of other road users, such as pedestrians and human-driven vehicles, to anticipate their actions and plan accordingly.
- Lane Keeping and Lane Changing: RL can help autonomous vehicles maintain their lane position and make safe lane-changing maneuvers based on surrounding traffic.

- Intersection Negotiation: As seen in the context of this study, RL is applied to enable autonomous vehicles to negotiate intersections effectively, making decisions on when to yield or proceed.
- Optimizing Energy Efficiency: RL can be used to optimize the energy consumption of autonomous vehicles, leading to longer battery life and reduced carbon emissions.
- Traffic Flow Optimization: RL can be employed to optimize traffic flow and reduce congestion in urban environments, improving overall transportation efficiency.
- Autonomous Parking: RL can be used to enable autonomous vehicles to park themselves in parking lots or on the street.

The use of RL in autonomous driving continues to evolve rapidly, with ongoing research and development focusing on improving safety, efficiency, and adaptability. As autonomous driving technology matures, RL is expected to play a crucial role in enhancing the capabilities of autonomous vehicles and making them more reliable and intelligent on the road. Many works try to improve transportation safety and efficiency at the intersection by integrating various RL techniques such as intelligent traffic light (sometimes known as adaptive signal control) [16–18]. However, as autonomous vehicles (AV) are gaining ground, problems arose as AV had difficulty trying to recognize the state of traffic signal in occluded situation [19]. AV can also rely on other data such as Lidar points, and other sensors to navigate through the intersection which renders the adaptive signal control unsuitable for Avs. Another approach was to place an intersection controller (IC) somewhere at the center of the intersection. The IC relies on various data such as video footage and lane detectors to coordinate vehicle movement. AVs need to request passage to the IC in a similar fashion to air traffic control [20–23]. As the solution becomes more complex, the infrastructure cost also increased which makes the situation only available in big city. RL brings promising solutions to solve control problems in this ever-growing big- data community. Typically, RL algorithms are divided into 2 schools of thought — model-free and mode-based [24]. Model-based can be thought of when the agent has a

complete understanding of the environment which means the action taken by the action guarantee the next state transition. However, most world problems evolve round uncertainty where agents often have incomplete view of the environment. As agents interaction became inevitable, several MARL algorithms were proposed [9, 10, 25–30].

2.2 Reinforcement Learning Algorithms

Reinforcement Learning (RL) algorithms can be broadly categorized into two main classes: model-free and model-based approaches. Each class has several sub-branches that address specific challenges and requirements in different problem domains.

2.2.1 Model-Free Reinforcement Learning

Model-free reinforcement learning (RL) is a type of reinforcement learning approach where an agent learns to make decisions and take actions in an environment without explicitly building a model of the environment. In traditional RL, an agent typically constructs a model that represents the transition dynamics and rewards of the environment. This model helps the agent to simulate and plan future actions to make decisions. However, in model-free RL, the agent directly learns from experience through trial and error without explicitly constructing a model.

In model-free RL, the agent interacts with the environment over multiple episodes or time steps, and it receives feedback in the form of rewards based on the actions it takes. The primary goal of the agent is to maximize the total accumulated reward over time. To achieve this, the agent uses different algorithms and techniques to learn the best policy, which is a mapping from states to actions that guides the agent in making decisions.

Model-free RL is particularly useful when the environment is complex, and constructing an accurate model is difficult or computationally expensive. Instead of relying on a model, model-free RL agents learn from real interactions with the environment, making them more adaptable and applicable to a broader range of scenarios. However, model-free RL also has

some challenges. Since it does not use a model, learning can be more sample inefficient [24], requiring a large number of interactions with the environment to converge to an optimal policy. Additionally, model-free RL can sometimes struggle with environments that have sparse rewards or deceptive reward structures. Nevertheless, with proper tuning and algorithm selection, model-free RL can be highly effective in a wide range of real-world applications.

Q-Learning

Q-learning is a model-free reinforcement learning algorithm that estimates the action-value function $Q(s, a)$, representing the expected cumulative reward when taking action a in state s. It is a widely used algorithm for solving Markov Decision Processes (MDPs) without requiring a model of the environment's dynamics.

The Q-value is updated iteratively using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

where: $Q(s, a)$ is the Q-value of state-action pair (s, a), r is the immediate reward received after taking action a in state s, α is the learning rate that controls the step size of updates, γ is the discount factor that balances future rewards' importance, s' is the next state reached after taking action a in state s, and $\max_{a'} Q(s', a')$ represents the maximum Q-value over all possible actions in state s'.

The Q-learning algorithm follows the following steps:

1. Initialize the Q-values for all state-action pairs to arbitrary values.
2. Observe the current state s.
3. Select an action a using an exploration policy, such as ϵ-greedy, which balances between exploration and exploitation.
4. Execute the selected action a in the environment.

5. Observe the immediate reward r and the next state s'.

6. Update the Q-value for the current state-action pair using the Bellman equation.

7. Repeat steps 2 to 6 until convergence or a predefined number of iterations.

Q-learning is an off-policy algorithm, meaning that it learns the optimal policy while following a different behavior policy for exploration. The exploration policy allows the agent to explore the environment and learn about different state-action pairs. As Q-learning iteratively updates the Q-values, it converges to the optimal action-value function $Q^*(s, a)$, which represents the maximum expected cumulative reward from each state-action pair under the optimal policy. Q-learning is known to be model-free, which means it does not require knowledge of the environment's dynamics or transition probabilities. Instead, it learns from interacting with the environment and experiencing different state-action-reward sequences. The Q-learning algorithm is widely used due to its simplicity and ability to handle large state spaces. However, it may suffer from slow convergence in complex environments, and variants like Double Q-learning and Deep Q-Networks (DQNs) have been introduced to address these limitations.

Deep Q-Networks (DQNs)

Deep Q-Networks (DQNs) is a deep learning extension of the Q-learning algorithm. DQNs leverage the power of deep neural networks to approximate the action-value function $Q(s, a)$ in complex and high-dimensional environments. It was introduced as a breakthrough in combining deep learning and reinforcement learning, particularly in solving Atari 2600 games [31].

The core idea of DQNs is to use a neural network as a function approximator for the action-value function. The neural network takes the state s as input and outputs a vector of Q-values, one for each possible action. The action with the highest Q-value is then selected as the agent's action in the given state.

DQNs aim to solve two key challenges in Q-learning:

- **Curse of Dimensionality**: In traditional tabular Q-learning, the size of the Q-table grows exponentially with the number of states and actions, making it infeasible for large state spaces. DQNs address this issue by using a neural network to generalize across states and actions.

- **High-Dimensional Inputs**: Many real-world environments have high-dimensional state spaces, such as images in computer vision tasks. DQNs can handle these inputs by using convolutional neural networks (CNNs) as the function approximator.

The DQN algorithm follows these main steps:

1. **Experience Replay**: DQNs use an experience replay buffer to store transitions (state, action, reward, next state) experienced by the agent. This buffer helps decorrelate experiences and reduce the correlation between consecutive updates, leading to more stable training.

2. **Target Network**: To stabilize the learning process, DQNs use a separate target network with fixed parameters to generate the target Q-values during the update step. The target network's parameters are updated periodically to match the primary Q-network.

3. **Loss Function**: The DQN loss function is the mean squared error between the predicted Q-values and the target Q-values, given by the Bellman equation:

$$Loss = \frac{1}{N} \sum_{i} (Q(s_i, a_i) - (r_i + \gamma \max_{a'} Q_{\text{target}}(s'_i, a')))^2$$

where N is the batch size, Q_{target} represents the target Q-network, and γ is the discount factor.

The DQN algorithm iteratively samples batches of experiences from the replay buffer, computes the loss, and performs stochastic gradient descent to update the primary Q-network's parameters.

DQNs have demonstrated remarkable success in various tasks, including playing Atari games and controlling robotic systems. They can handle high-dimensional inputs and effectively approximate the action-value function in complex environments.

However, DQNs also have some challenges, such as the tendency to overestimate Q-values and difficulties in handling continuous action spaces. Several extensions, such as Double DQNs and Dueling DQNs, have been proposed to address these limitations and further improve performance.

Policy Gradient Methods

Policy Gradient Methods are a class of reinforcement learning algorithms that directly optimize the policy of an agent to maximize the expected cumulative reward. Unlike value-based methods that estimate the action-value function $Q(s, a)$, policy gradient methods aim to learn the policy $\pi(a|s)$, which is a mapping from states s to actions a.

The policy is typically represented as a parametric function, such as a neural network, with trainable parameters θ. The objective of policy gradient methods is to find the optimal parameters θ^* that maximize the expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

where $J(\theta)$ is the objective function, τ is a trajectory (sequence of states, actions, and rewards), π_θ is the policy with parameters θ, r_t is the reward at time step t, γ is the discount factor, and T is the maximum time step in the trajectory.

Policy gradient methods employ gradient ascent to update the policy parameters in the direction that increases the expected reward. The policy gradient is computed using the likelihood ratio trick:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \right) \right]$$

This gradient can be estimated through sampling by interacting with the environment and collecting trajectories. The policy parameters are then updated using stochastic gradient ascent:

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \nabla_\theta J(\theta)$$

where α is the learning rate.

Policy gradient methods offer several advantages, such as handling continuous action spaces and directly optimizing the policy without relying on value functions. They are also well-suited for problems with stochastic policies or environments.

However, policy gradient methods can suffer from high variance in the gradient estimates, which can lead to slow convergence. To address this, variance reduction techniques like baseline subtraction and advantage functions are used to stabilize training.

Additionally, modern policy gradient methods, such as Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO), introduce mechanisms to constrain the policy updates to ensure more stable and controlled learning.

Overall, policy gradient methods have proven to be effective in a wide range of applications, including robotic control, game playing, and natural language processing, where learning directly from high-dimensional observations is crucial.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a state-of-the-art policy gradient algorithm for reinforcement learning. It addresses the challenges of traditional policy gradient methods, such as high variance and instability, by introducing constraints on policy updates to ensure more stable and controlled learning.

PPO operates by iteratively optimizing a surrogate objective function that approximates the policy's performance while avoiding drastic policy changes that could lead to divergence. The objective function is defined as the clipped surrogate objective:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^{\pi_{\theta_{\text{old}}}}(s_t, a_t), \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right) \right]$$

where: θ represents the current policy parameters, θ_{old} represents the old policy parameters (used as a reference), $\pi_\theta(a_t|s_t)$ is the probability of taking action a_t in state s_t under the current policy, $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability of taking action a_t in state s_t under the old

policy, $A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ is the advantage function, representing the advantage of taking action a_t in state s_t under the old policy, ϵ is a hyperparameter that controls the magnitude of the policy change.

The clipped surrogate objective encourages small policy updates by limiting the change in the likelihood ratio between the new and old policies. This constraint prevents the policy from changing too rapidly, which contributes to the algorithm's stability.

The PPO algorithm operates in two phases:

1. **Data Collection**: The agent interacts with the environment to collect trajectories using the current policy π_θ.

2. **Policy Update**: The policy parameters θ are updated using stochastic gradient ascent to maximize the clipped surrogate objective function $L^{CLIP}(\theta)$.

PPO's ability to strike a balance between exploration and exploitation makes it effective in a wide range of tasks. The algorithm has been successfully applied to various domains, including robotic control, game playing, and natural language processing.

PPO also offers a robust and efficient solution for parallelizing training across multiple environments, further accelerating learning in complex environments.

Overall, Proximal Policy Optimization (PPO) has become a popular choice for deep reinforcement learning due to its stable performance, ease of implementation, and excellent results across diverse problem domains.

2.2.2 Model-Based Reinforcement Learning

Model-based reinforcement learning (RL) is an approach where an agent learns and utilizes an explicit model of the environment to make decisions and plan actions. Unlike model-free RL, which directly learns from experience without constructing a model, model-based RL involves creating a representation of how the environment behaves, including its transition dynamics and reward function. This model is then used to simulate possible trajectories and predict the outcomes of different actions.

The typical process of model-based RL can be summarized as follows:

- **Model Learning** : The agent collects data by interacting with the environment and observes the outcomes of its actions. It uses this data to build a model of the environment. The model could be in the form of a transition function, which predicts the next state given the current state and action, and a reward function, which estimates the expected reward for a state-action pair.

- **Planning** : The agent uses its model to simulate possible trajectories and predict the outcomes of different actions. It can perform offline planning to evaluate different sequences of actions and predict the total expected rewards. This process allows the agent to make decisions that are based on the simulations and predictions from the model, rather than relying solely on trial and error in the real environment.

- **Policy Improvement** : After simulating different trajectories and estimating the rewards, the agent can update its policy to improve decision-making. It can use various optimization algorithms to find a policy that maximizes the expected cumulative reward based on the predictions from the model.

Model-based RL has several advantages:

- **Sample Efficiency** : Since the agent can simulate interactions with the environment using its model, it can potentially learn more efficiently and require fewer real interactions to find an optimal policy.

- **Safety** : The agent can use its model to explore the environment safely, avoiding potentially harmful or risky actions by simulating their consequences.

- **Handling Partial Observability** : Model-based RL can be effective in partially observable environments where the agent does not have access to complete information about the state. The model helps the agent predict the underlying state, improving decision-making.

However, model-based RL also has its challenges:

- **Model Accuracy** : The success of model-based RL heavily depends on the accuracy of the learned model. If the model is inaccurate, the agent's decisions and predictions can be suboptimal.
- **Model Complexity** : Constructing an accurate model may be computationally expensive or infeasible in complex environments with large state spaces.
- **Planning Overhead** : Planning using the model can introduce additional computational overhead, which might limit the real-time applicability of the approach in certain scenarios.

Dynamic Programming

Dynamic programming is a powerful optimization technique used in computer science and mathematics to solve problems by breaking them down into smaller subproblems and solving each subproblem only once. The solutions to these subproblems are stored and reused when needed, which avoids redundant calculations and leads to more efficient computations.

The fundamental idea behind dynamic programming is to divide a complex problem into smaller, overlapping subproblems and solve each subproblem just once, storing its solution in a table or array. Then, the solution to the original problem is constructed by combining the solutions of the subproblems.

The process of dynamic programming can be described by the following steps:

- **Define the Recurrence Relation** : Express the solution to a larger problem in terms of the solutions to its subproblems. This recursive relationship defines the optimal value of the larger problem based on the optimal values of its subproblems.
- **Identify Overlapping Subproblems** : Analyze the problem to identify if there are any overlapping subproblems, i.e., if the same subproblems are solved multiple times in the computation. This is the key property that makes dynamic programming effective, as we can avoid redundant computations.

- **Create a Memoization Table** : Use a data structure, such as an array or a dictionary, to store the solutions of the subproblems as they are computed. This is called memoization, and it helps in efficiently retrieving solutions for previously solved subproblems.

- **Bottom-Up or Top-Down Approach** : Dynamic programming can be implemented using either a bottom-up or top-down approach.

  - **Bottom-Up** : In the bottom-up approach, we start solving the smallest subproblems first and gradually build up to the solution of the original problem.

  - **Top-Down** : In the top-down approach, also known as memoization, we start with the original problem and break it down into smaller subproblems.

Dynamic programming is widely used in various areas of computer science, such as algorithm design, optimization problems, artificial intelligence, and bioinformatics, to efficiently solve complex problems with overlapping subproblems.

Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a popular algorithm used in decision-making and game playing problems, particularly in games with large state spaces and complex branching factors. It is commonly applied in artificial intelligence for games like Go, Chess, and other strategic board games. MCTS combines elements of random sampling (Monte Carlo) with tree-based search to efficiently explore the game tree and make informed decisions.

The MCTS algorithm can be explained in the following steps:

- **Selection** : Starting from the root of the search tree, the algorithm traverses the tree by selecting child nodes based on a selection policy, often using Upper Confidence Bound (UCB) or variants like UCT (Upper Confidence Bounds for Trees). The selection policy balances between exploring new nodes and exploiting existing knowledge to focus on promising branches.

- **Expansion** : Once a leaf node (a node with unexplored children) is reached, the algorithm expands the tree by adding child nodes representing possible game moves or actions.

- **Simulation (Rollout)** : From the newly added child node, the algorithm performs a Monte Carlo simulation (also known as a rollout) by playing the game randomly until a terminal state (end of the game) is reached. The result of the simulation is a reward or utility value representing the outcome of the game from that state.

- **Backpropagation** : The reward obtained from the simulation is backpropagated up the tree, updating the statistics of each visited node, such as the total reward and the number of visits. This information helps in refining the selection policy and improves the decision-making.

- **Repeated Iteration**: Steps 1 to 4 are repeated for a certain number of iterations or until a predefined computational budget is reached. The more iterations performed, the more refined the search tree becomes, leading to better decisions.

Model Predictive Control (MPC)

Model Predictive Control (MPC) is an advanced control strategy used in various engineering fields, such as process control, robotics, and automotive systems. MPC is particularly well-suited for systems with complex dynamics, constraints, and uncertainties. It involves using a model of the system's behavior to predict future states and optimize a control sequence over a finite time horizon. The control sequence is applied to the system, and the process is iterated in a receding horizon fashion, continually updating the control actions based on the most recent measurements and predictions.

Here's a detailed explanation of Model Predictive Control:

- **Modeling the System** : The first step in MPC is to create a dynamic model of the system being controlled. This model should accurately represent the system's behavior and dynamics. Depending on the complexity of the system and the available

information, the model can be a physics-based model, an empirical model, or even a data-driven model based on machine learning techniques.

- **Prediction Horizon** : MPC works by optimizing a control sequence over a finite prediction horizon, which is a predetermined time interval into the future. The length of the prediction horizon determines how far into the future the control actions are computed. Longer prediction horizons can provide better performance, but they also increase the computational burden.

- **Optimization Objective** : The main objective of MPC is to optimize a cost function over the prediction horizon. The cost function is a measure of how well the system performs, and it typically includes components related to control effort, desired setpoints, and constraints. By minimizing or maximizing the cost function, MPC finds the optimal control sequence that achieves the desired control objectives.

- **Constraints** : MPC can handle both hard and soft constraints on the system's states and inputs. Hard constraints are strict limits that cannot be violated, while soft constraints are desired limits that can be violated if necessary. MPC can also handle probabilistic constraints, which are constraints that must be satisfied with a certain probability.

- **Receding Horizon Control** : Once the optimal control sequence is computed, only the first control action is applied to the system. The process is then repeated, with the optimization being performed again using the most recent measurements and predictions. This is called receding horizon control, as the optimization horizon keeps receding into the future.

- **Real-Time Implementation** : MPC is typically implemented in real-time using a digital computer. The optimization problem is solved at each time step, and the first control action is applied to the system. The process is repeated at the next time step, and so on. The real-time implementation of MPC requires fast and efficient algorithms to solve the optimization problem.

- **Robustness and Adaptation** : MPC is inherently robust to disturbances and uncertainties, as it continually updates the control actions based on the most recent measurements and predictions. MPC can also be extended to handle model uncertainties and unmodeled dynamics by using adaptive MPC techniques.

MPC is widely used in industrial automation, autonomous systems, power systems, and various other applications where precise control and handling of complex dynamics and constraints are essential. It provides robust and adaptive control, making it a valuable tool in controlling complex systems in real-world scenarios.

2.3 Single Agent Reinforcement Learning

Single-agent RL algorithms are designed to solve problems where there is only one agent interacting with the environment. There are advantages and disadvantages to using single-agent RL algorithms in autonomous driving applications:

- Simplicity: Single-agent RL is relatively straightforward to implement and understand since there is only one agent interacting with the environment.
- Model-Based Approaches: Model-based RL algorithms can work effectively in environments where the agent has a complete understanding of the environment's dynamics, enabling accurate predictions.
- Sample Efficiency: Single-agent RL algorithms can be more sample-efficient compared to MARL algorithms since they do not involve interactions among multiple agents.

However, as the number of agents increases, single-agent RL algorithms may struggle to scale effectively, especially when the agents' actions affect each other. So we have some disadvantages:

- Scalability: In complex environments with many agents, single-agent RL algorithms may struggle to scale effectively, especially when the agents' actions affect each other.

- Non-Stationary Environments: Single-agent RL assumes a stationary environment, which may not hold true in dynamic settings with multiple agents.

2.4 Multi-Agent Reinforcement Learning

The advantages of MARL in autonomous driving include:

- Coordination and Cooperation: MARL allows agents to coordinate and cooperate to achieve common goals or address conflicting objectives, making it suitable for problems with multiple agents interacting.
- Dynamic Environments: MARL is well-suited for non-stationary and complex environments, where the behavior of agents affects each other and evolves over time.
- Flexibility: MARL algorithms can handle various scenarios, from collaborative tasks to competitive interactions, making them versatile in real-world settings.

However, MARL also has some disadvantages:

- Curse of Dimensionality: As the number of agents increases, the state space grows exponentially, leading to challenges in scalability and computation.
- Communication Overhead: In some MARL algorithms, communication among agents may be required to coordinate actions, introducing additional overhead and complexity.
- Partial Observability: In scenarios where agents have limited visibility of the environment or other agents' states, coordinating effectively becomes more challenging.

第3章　Methodology

3.1 Preliminary

3.1.1 Markov Decision Process

A Markov decision process (MDP) is a tuple (S, A, P, R, γ), where: S is a set of states. A is a set of actions. $P(s'|s, a)$ is the probability of transitioning from state s to state s' when action a is taken. $R(s, a)$ is the reward received when taking action a in state s. $\gamma \in [0, 1)$ is a discount factor that determines the importance of future rewards. The MDP definition can be interpreted as follows:

- The state s represents the current state of the environment.
- The action a represents the agent's decision in the current state.
- The transition probability $P(s'|s, a)$ represents the probability of the environment transitioning to state s' after the agent takes action a in state s.
- The reward $R(s, a)$ represents the amount of reward the agent receives after taking action a in state s.
- The discount factor γ represents how much the agent values future rewards relative to immediate rewards.

Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) are a class of multi-agent reinforcement learning problems where multiple agents collectively aim to maximize a global objective function while facing uncertainty in a partially observable environment.

Elements of Dec-POMDPs:

1. **Agents:** Dec-POMDPs involve multiple agents denoted by index $i \in \{1, 2, \ldots, N\}$, each with its own local observation space O_i and action set A_i. The agents interact with the environment and communicate with each other to make coordinated decisions.

2. **States:** The environment's underlying state is not directly observable. Instead, each agent i receives partial, private observations o_i that provide probabilistic information about the true state. The state space is denoted as S.

3. **Observations:** Each agent's observation o_i is a partial and noisy representation of the global state. Observations are generated based on the current state s and the agent's local observation function denoted as $O_i(s)$.

4. **Actions:** Agents take actions a_i based on their local observations and communication with other agents. The joint action of all agents affects the evolution of the underlying state. The joint action of all agents is denoted as $\mathbf{a} = (a_1, a_2, \ldots, a_N)$.

5. **Transition Model:** The transition model $P(s'|s, \mathbf{a})$ describes the probability distribution over next states s' given the current state s and joint actions of all agents $\mathbf{a}$.

6. **Observation Model:** The observation model $P(o_i|s)$ describes the probability distribution over observations o_i given the current state s for agent i.

7. **Global Objective Function:** The goal of the agents is to optimize a global objective function $J(\mathbf{a})$ that captures the overall mission or task. This objective function depends on the collective actions of all agents and may involve cooperation and coordination.

3.1.2 Gaussian Distribution

The Gaussian distribution, also known as the normal distribution, is a continuous probability distribution widely used to model random variables in various real-world scenarios. It is characterized by its mean μ and variance σ^2. The probability density function (PDF) of a Gaussian distribution is given by:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.1}$$

where x is a random variable, μ is the mean (expected value), and σ^2 is the variance.

The Gaussian distribution is symmetric around its mean, and its shape is determined by the variance. When the variance is small, the distribution is more concentrated around the mean, while a larger variance leads to a wider distribution.

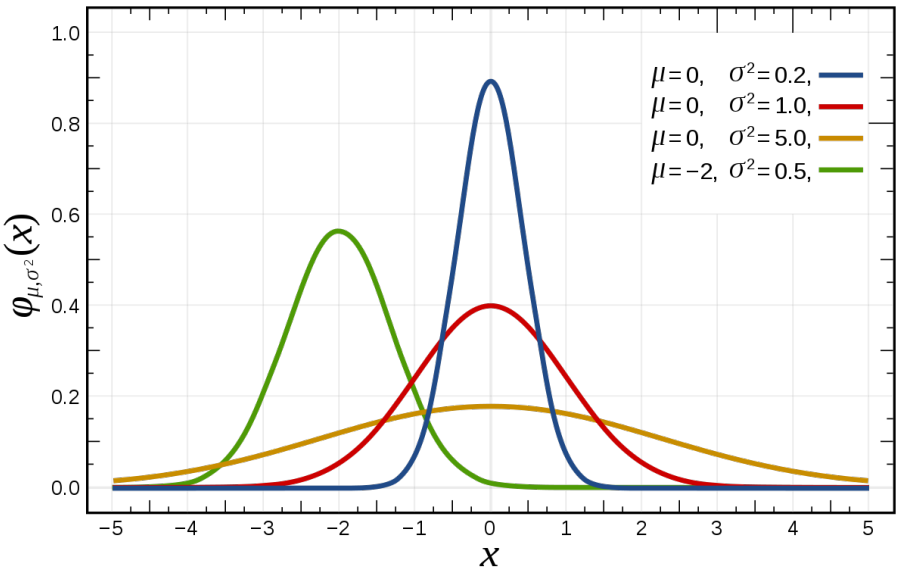

図 3.1: Normal Distribution

Policy Gradient Method with Gaussian Approximation:

In Reinforcement Learning (RL), the policy gradient method is a popular approach to learning the policy of an agent in a continuous action space. The policy gradient method

directly optimizes the policy parameters to maximize the expected cumulative reward.

Parameterized Policy:

To handle continuous actions, the policy is often parameterized as a Gaussian distribution with a mean vector $\boldsymbol{\mu}$ and a variance vector $\boldsymbol{\sigma}^2$, or alternatively, using the standard deviation $\boldsymbol{\sigma}$. The policy function is defined as:

$$\pi_{\boldsymbol{\theta}}(a|s) = \mathcal{N}(\boldsymbol{\mu}(s;\boldsymbol{\theta}), \text{diag}(\boldsymbol{\sigma}(s;\boldsymbol{\theta}))), \tag{3.2}$$

where $\pi_{\boldsymbol{\theta}}(a|s)$ is the policy function with parameters $\boldsymbol{\theta}$, and $\text{diag}(\cdot)$ extracts the diagonal elements to form a diagonal covariance matrix.

Prediction of Mean Using Neural Network:

In practice, the mean $\boldsymbol{\mu}(s;\boldsymbol{\theta})$ is often predicted using a neural network with state s as input and the policy parameters $\boldsymbol{\theta}$ as weights. The neural network learns to map states to action means, allowing the agent to approximate actions based on observations from the environment.

Adjusting Standard Deviation Over Time:

During the training process, the standard deviation $\boldsymbol{\sigma}(s;\boldsymbol{\theta})$ can be adjusted over time. It is often initialized with some initial value and then decayed towards a minimum value [5, 24, 32]. This adjustment allows the agent to explore more at the beginning of training (high uncertainty) and become more confident in its actions as training progresses (low uncertainty).

3.1.3 Beta Distribution

The Beta distribution is a continuous probability distribution defined on the interval [0, 1]. It is commonly used to model random variables that represent probabilities or

proportions. The Beta distribution is parameterized by two shape parameters, denoted as α and β, which control the shape of the distribution. The probability density function (PDF) of the Beta distribution is given by:

$$f(x|\alpha,\beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{3.3}$$

where x is a random variable, α and β are the shape parameters, and $B(\alpha,\beta)$ is the Beta function:

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \tag{3.4}$$

where $\Gamma(\cdot)$ is the gamma function.

The Beta distribution is asymmetric and its shape is determined by the values of α and β. When both α and β are greater than 1, the distribution is unimodal and skewed. The mode of the distribution is at $\frac{\alpha-1}{\alpha+\beta-2}$.

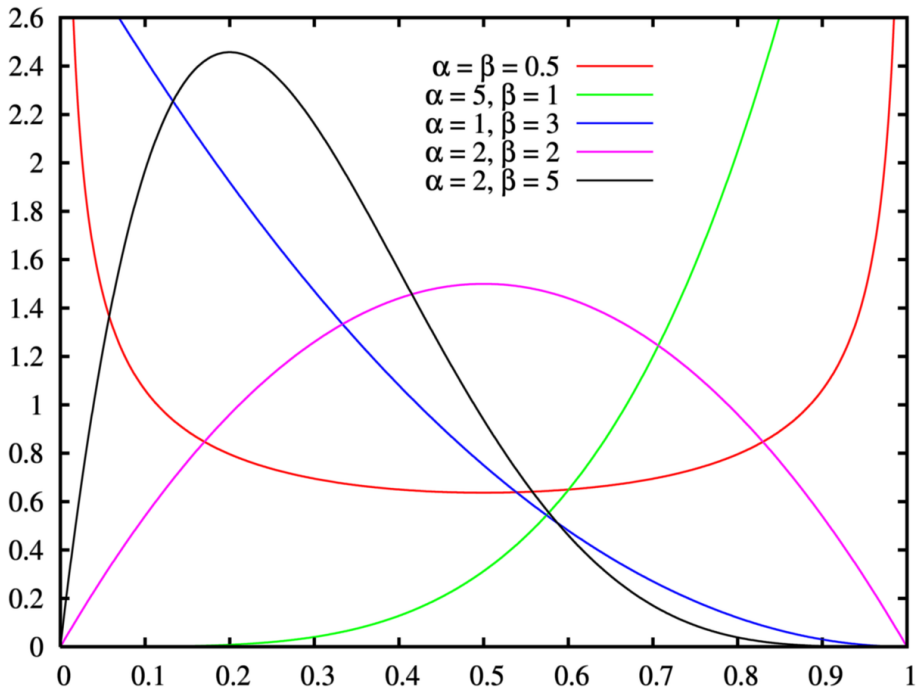

図 3.2: Beta Distribution

Policy Gradient Method with Beta Distribution:

In the policy gradient method for Reinforcement Learning (RL), the policy is often parameterized as a Beta distribution to handle continuous actions within the range [0, 1].

Predicting Alpha and Beta Values from Neural Network:

To approximate the Beta distribution for policy generation, a neural network is used to predict the α and β parameters. The neural network takes the state s as input and produces the α and β values as output. These values are then used to define the Beta distribution for sampling actions.

Interpretation of 1 - α and 1 - β:

The α and β values can be interpreted as counts of successes and failures, respectively. Specifically, if $\alpha > 1$ and $\beta > 1$, then $\alpha - 1$ and $\beta - 1$ can be thought of as the number of successes and failures, respectively. This interpretation is useful when modeling probabilities or proportions in the context of the policy gradient method.

3.1.4 Beta Policy

We propose a straightforward neural network architecture, specifically a Multilayer Perceptron (MLP), with 2 hidden layers each having a size of 256 neurons. This neural network is then divided into two separate channels, each having a hidden layer with 256 neurons. These channels are designed to predict values for steering angle and acceleration, respectively, in order to control the vehicle.

The implementation of the neural network is carried out using PyTorch [33], a popular deep learning library. The input to the network consists of ego states, which include relevant information about the vehicle in relation to its environment, such as velocity, checkpoint positions, time, and distance to the destination. Additionally, LiDAR data from the left, right, and front sides of the vehicle is collected and fed into the network.

LiDAR is a sensing technology used in many autonomous vehicles, and its utilization allows the proposed algorithm to be easily adapted and applied in real-world scenarios.

The proposed neural network architecture employs an MLP with two hidden layers, each with 256 neurons. It is divided into two channels for predicting steering angle and acceleration values, respectively. The input includes ego states and LiDAR data, making it suitable for controlling the vehicle in an autonomous driving context. The implementation is carried out using PyTorch, a popular deep learning framework known for its efficiency and flexibility.

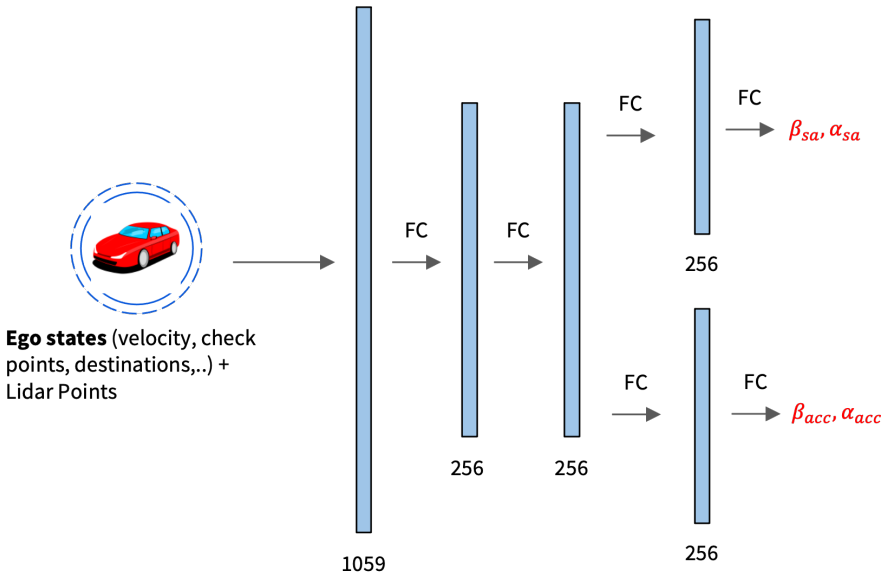

図 3.3: Beta Policy

The final layer that output these values uses softplus activation function and 1 is added to make sure that the output is always positive. The final layer is $f(x) = 1 + \log(1 + e^x)$. As the environment accepts action with $a = [sa, acc]^T = [-1, 1]^2$, the output from Beta's PDF is then mapped with the following equation:

$$sa = 2h(\beta_{sa}, \alpha_{sa}) - 1 \tag{3.5}$$

$$acc = 2h(\beta_{acc}, \alpha_{acc}) - 1 \tag{3.6}$$

where $h(\beta, \alpha)$ is the mean of the Beta distribution with parameters β and α, sa stands

for steering angle, and *acc* stands for acceleration. It should be noted that we only consider the case where $\alpha > 1$ and $\beta > 1$.

3.1.5 Advantage Function (Experience Sharing)

The advantage function is a key component of the policy gradient method. It is used to estimate the advantage of taking an action a in state s over taking an action sampled from the policy π in state s. The advantage function is defined as:

$$A^{\pi_{\theta_k}}(s_t, s_{t+1}, A_j, A_{j+1}, R_G) = R_G + Q(s_{t+1}, A_{j+1}) - Q(s_t, A_j) \tag{3.7}$$

where R_G is the global reward defined as $R_G = \frac{1}{k}\sum_{k}^{i=0} R_k$ whose k is the individual reward calculated by Monte Carlo Method [24]. $Q(\star)$ is the critics part which is popularized by Asynchronous Advantage Actor Critic (A3C) [6] and also adopted by PPO [7] , $Q(s_t, A_j)$ is the Q-value of taking action A_j in state s_t, and A_{j+1} is the action sampled from the policy π in state s_{t+1}.

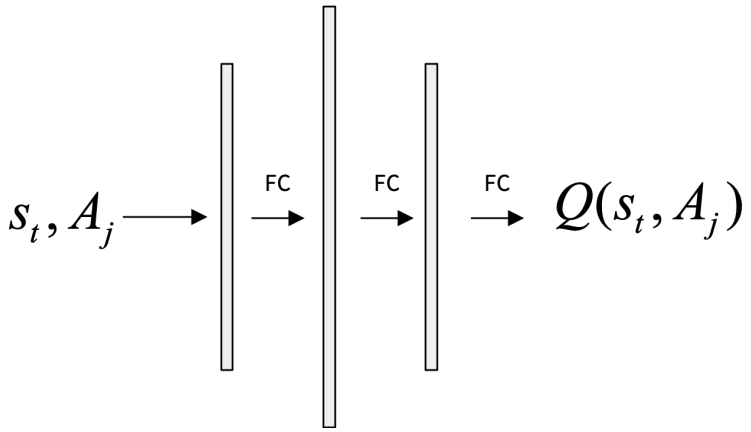

図 3.4: Q value Approximator (Critics)

3.1.6 Objective Function

The objective function is used to optimize the policy parameters θ. The goal of the agent is to maximize the culminative reward over time by updating the parameterized

policy with:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=1}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}, g(\epsilon, A^{\pi_{\theta_k}})\right) \tag{3.8}$$

where D_k is the set of trajectories collected by the agent at iteration k, T is the length of the trajectory, π_θ is the policy parameterized by θ, and $g(\epsilon, A^{\pi_{\theta_k}})$ is the clipped surrogate objective function defined as:

$$g(\epsilon, A^{\pi_{\theta_k}}) = \begin{cases} (1+\epsilon)A^{\pi_{\theta_k}} & \text{if } A^{\pi_{\theta_k}} \geq 0 \\ (1-\epsilon)A^{\pi_{\theta_k}} & \text{otherwise} \end{cases} \tag{3.9}$$

3.1.7 Centralized Training and Decentralized Execution

CTDE starts with several agents being placed in the environment. The agents are allowed to run for several predefined steps (2000 timestamps in our case). In the process, samples are collected in a rollout buffer, the buffer then performs some preprocessing tasks such as arranging data and splitting data into experience batches. Those baches are then fed into the central agent ' s actor-critics network to update the policy by calculating loss and through back propagation. The updated policy is then distributed back to the individual agent. The cycle repeats until the reward feedback converges.

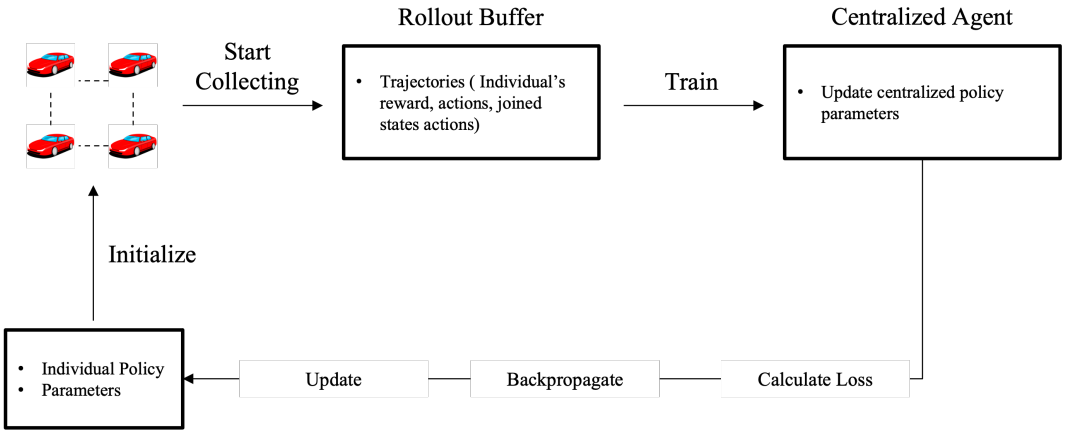

図 3.5: Centralized Training and Decentralized Execution

第4章 Result and Analysis

4.1 Experimental Setup

The following table 4.1 describe computer specifications used in the experiment.

PC Specifications	**Details**
Processor	Intel Core i7 8700K 6 cores / 12 threads
RAM	32GB
Graphics Card	NVIDIA GeForce GTX 1070 8GB VRAM
Operating System	Ubuntu 22.10 Linux 5.13
Python Version	Python 3.7 Anaconda distribution

表 4.1: Experiment Setup with PC Specifications

In this section, we conducted experiments that compare our PPO-Beta and standard PPO-Gaussian implementation. We used MetaDrive [34] 4.1 for traffic simulator which support both single agent and multi-agent training with integrated reinforcement learning interface (Gym) [35]. We proposed 3 evaluation metrics such as training rewards, safety rate, and success rate. Training reward is the feedback given by the simulator as the score. When the agent drives and reach the destination, 130 score (total reward) is given but we can set the solving threshold as 80 as average solving score as it implies that the agent already passes through the intersection safely. For safety and success rate, they are

collected in evaluation stage which we loaded the pretrained model into the agent as they are in real situation and let the agent run without updating the policy. The success is the ratio of how many times the agent reaches the destination over the number of agents participating in the evaluation. Safety is the ratio of the agents successfully driving on the road without crashes to other objects or vehicles. We run the traning fpr both algorithms for 5 tryouts and plot the average for each evaluation metric. The PPO hyperparameters are shown in table 4.2.

PPO Method	**Beta (Ours)**	**Gaussian**
Clipping	0.02	0.02
Gamma	0.99	0.99
Action decay rate	N/A	0.005
Action initial Std	N/A	0.6
Minimum decay	N/A	0.1
K Epoch	5	5
Actor learning rate	0.0003	0.0003
Critic learning rate	0.001	0.001

表 4.2: PPO Configuration

Here is the explanation of the hyperparameters: Clipping is the clipping value for the PPO loss function. Gamma is the discount factor for the reward. Action decay rate is the decay rate for the action standard deviation. Action initial Std is the initial standard deviation for the action. Minimum decay is the minimum decay rate for the action standard deviation. K Epoch is the number of epochs for the PPO update. Actor learning rate is the learning rate for the actor network. Critic learning rate is the learning rate for the critic network.

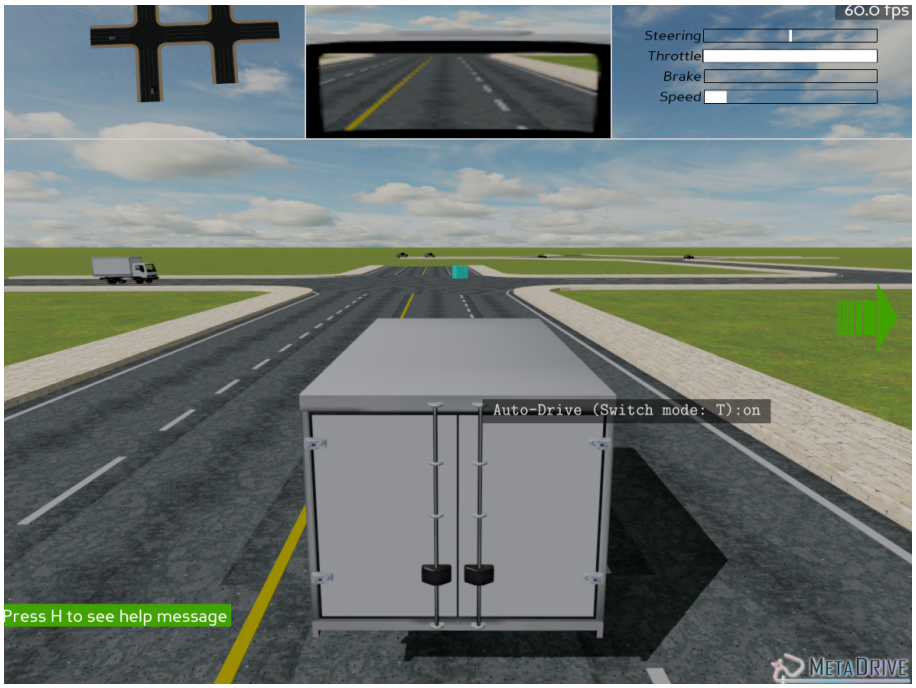

図 4.1: MetaDrive Traffic Simulator

4.2 Reward Shaping

We designed the reward funtion R as:

$$R = c_1(d_t - d_{t-1}) + c_2\frac{v_t}{v_{max}} + R_T \tag{4.1}$$

where d_t is the distance between the agent and the destination at time t, v_t is the speed of the agent at time t, v_{max} is the maximum speed of the agent, R_T is the termination reward, and c_1 and c_2 are the coefficients for the distance and speed reward respectively. We set $c_1 = 1$ and $c_2 = 1$. The design is to encourage the agent to drive faster and reach the destination as soon as possible. R_T is the termination reward which can be one of the following:

- $R_T = 10$ if the agent reaches the destination.
- $R_T = -5$ if the agent drives out of the road.
- $R_T = -5$ if the agent collides with other vehicles.

- $R_T = -5$ if the agent collides with other objects.

4.3 Single Agent Result

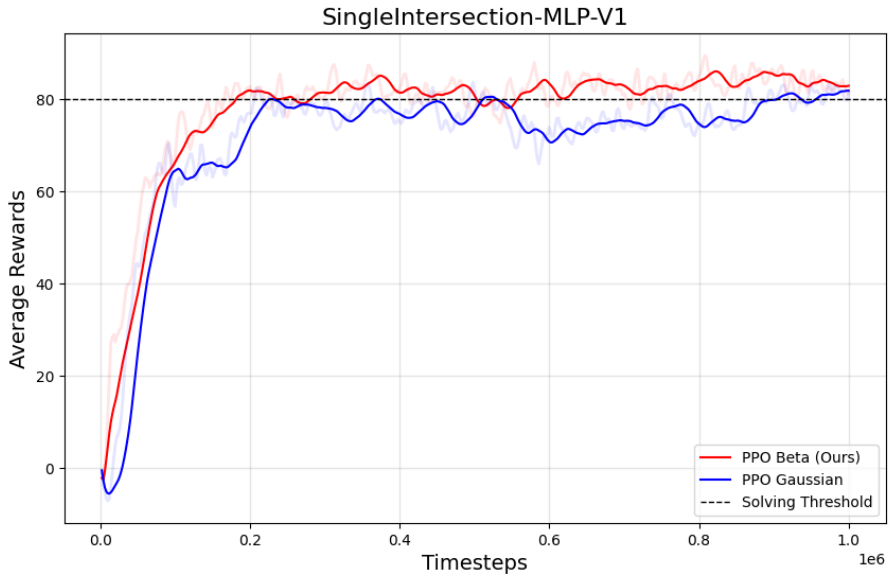

図 4.2: Comparision of single agent traning by average reward betwwen PPO Beta (Ours) and standard PPO Gaussian.

The Gaussian distribution implemented in standard PPO has an adjustment nature that can lead to a drop in training performance after the update of the value function. This is because the Gaussian distribution is centered around the mean, and when the mean is updated, the distribution must shift to accommodate the new mean. This shift can cause a temporary drop in performance as the agent adjusts to the new distribution. The Beta distribution, on the other hand, does not suffer from this problem. The Beta distribution is more flexible than the Gaussian distribution, and it can accommodate changes in the mean without causing a significant drop in performance. This is because the Beta distribution is not centered around a single point, but rather it is spread out over a range of values. This makes it more robust to changes in the mean.

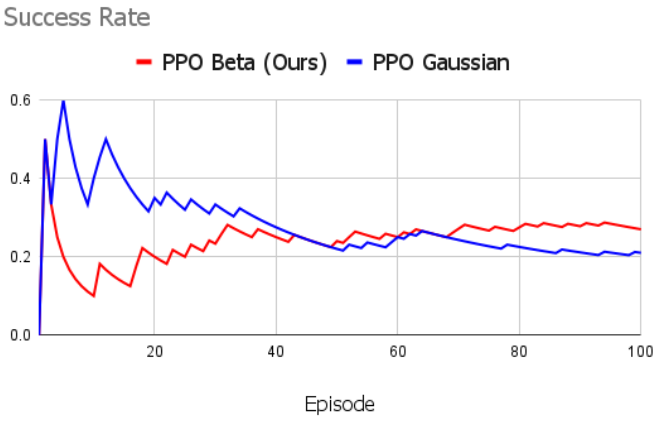

図 4.3: Comparision of single agent traning by average success rate betwwen PPO Beta (Ours) and standard PPO Gaussian.

In our experiments, we found that the Beta policy learned to reach the solving threshold (80) faster than the Gaussian policy. Additionally, the average score of the PPO-Beta agent was slightly better than the PPO-Gaussian agent. These results suggest that the Beta distribution is a more effective choice for continuous control tasks than the Gaussian distribution.

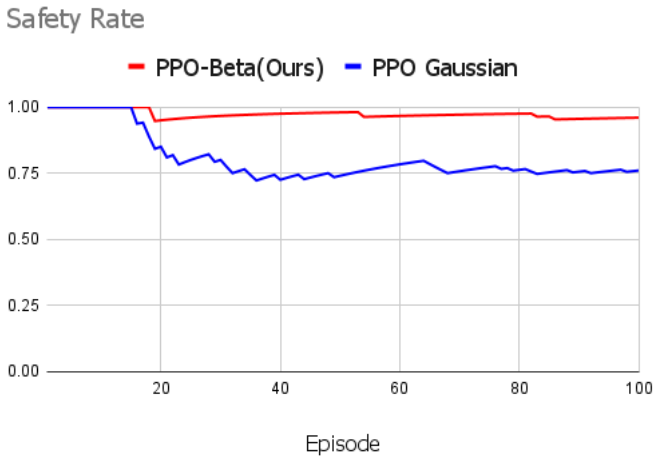

図 4.4: Comparision of single agent traning by average safety rate betwwen PPO Beta (Ours) and standard PPO Gaussian.

We evaluated the performance of PPO-Beta and PPO-Gaussian in a single-agent environment where the training agent had difficulty predicting the actions of the other agents, which were controlled by non-player characters (NPCs). We found that PPO-Beta outperformed PPO-Gaussian on both the success and safety rates after running for 100 episodes. As shown in 4.4, both success rates were below 50because the single-agent environment is inherently difficult, as the training agent cannot coordinate its actions with the other agents. Despite the difficulty of the environment, PPO- Beta was able to learn to dodge the incoming vehicle better than PPO-Gaussian. This is evident in the safety rate, which was 10% higher for PPO-Beta than for PPO-Gaussian. These results suggest that PPO-Beta is a more effective choice for single-agent environments where the training agent has difficulty predicting the actions of other agents.

4.4 Multi Agent Result

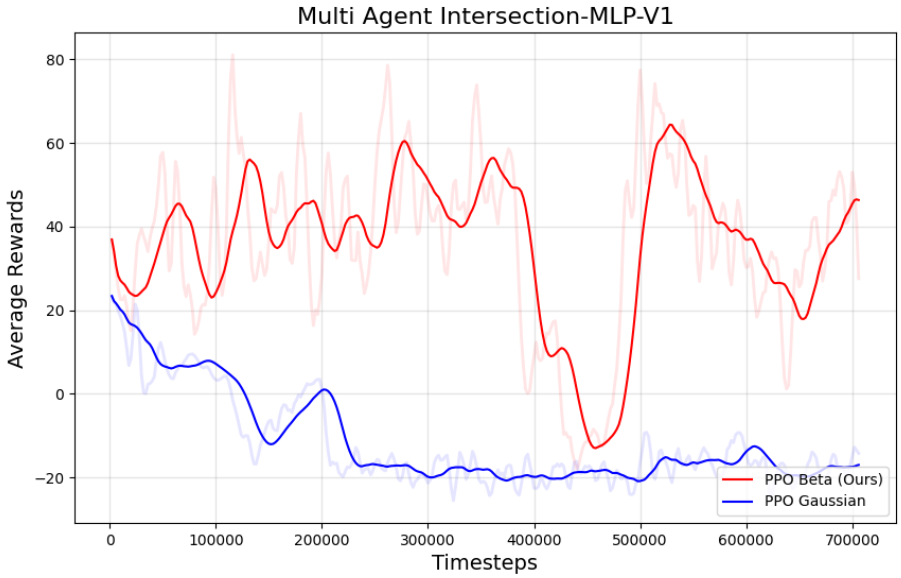

図 4.5: Comparision of multi-agent traning by average reward

We train the multi-agent environment with 4 agents. Here is the interpretation of the result:

Multi-agent PPO Beta:

- **Average Reward:** The average reward obtained by the agents over all episodes is approximately 120. This indicates that, on average, the agents perform reasonably well in the environment when using the PPO beta approach.
- **Reward Fluctuations:** The rewards fluctuate throughout the training process. This fluctuation might be caused by various factors, such as the complexity of the environment, the number of agents interacting, and the training dynamics of PPO beta. It's normal to observe some variance in rewards during training.
- **Learning Stability:** The training curve doesn't show any significant spikes or crashes, which suggests that the training process using PPO beta is stable.

Multi-agent PPO with Gaussian Distribution:

- **Average Reward:** The average reward is lower compared to the multi-agent PPO beta. It indicates that using the Gaussian distribution approach might not be as effective in achieving high rewards as the PPO beta in this specific environment.
- **Reward Fluctuations:** Similar to PPO beta, the rewards fluctuate, but it seems to have higher variance with this approach. It's possible that the Gaussian distribution has a higher sensitivity to certain parameters or initial conditions.
- **Learning Stability:** As with PPO beta, there are no apparent stability issues with this approach.

Overall, from this data, it seems that the multi-agent PPO beta performs better in this particular environment compared to the multi-agent PPO with Gaussian distribution. It's worth noting that RL experiments can be sensitive to hyperparameters and initial conditions, so it's essential to perform multiple runs with different seeds to assess the

robustness of the results. In this case, we run each experiment with 5 tryouts and plot the average reward over all runs. This helps to reduce the variance in the results and provides a more accurate representation of the performance of each approach.

第5章　Conclusion

We proposed PPO-Beta with dual channel architecture to predict both alpha and beta values for environment with 2 bounded continuous action space. We chose PPO as RL algorithm for training the agent. After testing in traffic intersection environment, the agent can learn to be more careful when crossing the intersection than agent trained with standard Gaussian implementation. We also see that the training performance hardly drops in our implementation. In the future, we are going to extend to multi-agent version with more complex agents ' interaction and testing with real street data capture from drone footage such as the data provided by German transportation companies [36] to see how the agents deal with real situations.

References

[1] J. Chen, S.E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," Jan. 2020. http://arxiv.org/abs/2001.08726

[2] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," 2019.

[3] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," 2016.

[4] B. Peng, Q. Sun, S.E. Li, D. Kum, Y. Yin, J. Wei, and T. Gu, "End-to-end autonomous driving through dueling double deep q-network," Automotive Innovation, vol.4, pp.328–337, Aug. 2021. https://link.springer.com/10.1007/s42154-021-00151-3

[5] A.P. Capasso, P. Maramotti, A.Dell'Eva, A. Broggi, "End-to-end intersection handling using multi-agent deep reinforcement learning," 2021.

[6] V. Mnih and etal., "Asynchronous methods for deep reinforcement learning," 2016.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[8] J. Schulman, S. Levine, P. Moritz, M.I. Jordan, and P. Abbeel, "Trust region policy optimization," 2015.

[9] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2019.

[10] M.B. Johanson, E. Hughes, F. Timbers, and J.Z. Leibo, "Emergent social learning via multi-agent reinforcement learning," 2021.

[11] B. Baker and etal., "Emergent tool use from multi-agent autocurricula," 2019.

[12] M.B. Johanson, E. Hughes, F. Timbers, and J.Z. Leibo, "Emergent bartering behaviour in multi-agent reinforcement learning," 2022.

[13] S.A. Baby, L. Li, and A. Pokle, "Analysis of emergent behavior in multi agent environments using deep reinforcement learning," 2022.

[14] W. Zhou, D. Chen, J. Yan, Z. Li, H. Yin, and W. Ge, "Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic," Autonomous Intelligent Systems, vol.2, no.1, p.5, 2022. https://doi.org/10.1007/s43684-022-00023-5

[15] S. Wang, H. Fujii, and S. Yoshimura, "Generating merging strategies for connected autonomous vehicles based on spatiotemporal information extraction module and deep reinforcement learning," Physica A: Statistical Mechanics and its Applications, vol.607, p.128172, 2022. https://www.sciencedirect.com/science/article/pii/S0378437122007300

[16] S. Yang and B. Yang, "An inductive heterogeneous graph attention-based multi-agent deep graph infomax algorithm for adaptive traffic signal control," Information Fusion, vol.88, pp.249–262, Dec. 2022.

[17] T. Wang, J. Cao, and A. Hussain, "Adaptive traffic signal control for large-scale scenario with cooperative group-based multi-agent reinforcement learning," Transportation Research Part C: Emerging Technologies, vol.125, p.103046, 2021. https://www.sciencedirect.com/science/article/pii/S0968090X21000760

[18] M. Essa and T. Sayed, "Self-learning adaptive traffic signal control for real-time safety optimization," Accident Analysis & Prevention, vol.146, p.105713, 2020. https://www.sciencedirect.com/science/article/pii/S0001457520305388

[19] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," 2018 IEEE International Conference on Robotics and Automation (ICRA), pp.2034–2039, 2018.

[20] S. Li, K. Shu, C. Chen, and D. Cao, "Planning and decision-making for connected autonomous vehicles at road intersections: A review," Chinese Journal of Mechanical Engineering, vol.34, p.133, 2021.

[21] U. Gunarathna, S. Karunasekara, R. Borovica-Gajic, and E. Tanin, "Intelligent autonomous intersection management," 2022.

[22] X. Qian, F. Altché, J. Grégoire, and A. deLa Fortelle, "Autonomous intersection management systems: criteria, implementation and evaluation," IET Intelligent Transport Systems, vol.11, no.3, pp.182–189, 2017. https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-its.2016.0043

[23] W.-L. Chen, K.-H. Lee, and P.-A. Hsiung, "Intersection crossing for autonomous vehicles based on deep reinforcement learning," 2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), pp.1–2, 2019.

[24] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.

[25] Z. Peng, Q. Li, K.M. Hui, C. Liu, and B. Zhou, "Learning to simulate self-driven particles system with coordinated policy optimization," 2021.

[26] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," 2018.

[27] Y.C. Tang, "Towards learning multi-agent negotiations via self-play," 2020.

[28] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," 2019.

[29] J. Siekmann and W. Wahlster, Agent and Multi-Agent Systems: Technologies and Applications, Springer, 1998.

[30] M. Zhou and etal., "Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving," 2020.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[32] V. Mnih and etal., "Human-level control through deep reinforcement learning," Nature, vol.518, no.7540, pp.529–533, 2015.

[33] A. Paszke and etal., "Pytorch: An imperative style, high-performance deep learning library," 2019.

[34] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou, "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning," 2022.

[35] G. Brockman and etal., "Openai gym," 2016.

[36] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein, "The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections," 2019.