Deep reinforcement learning under uncertainty for autonomous driving

Deep Reinforcement Learning unter Unsicherheit im autonomen Fahren

Master-Thesis von Rong Zhi aus Beijing

November 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Deep reinforcement learning under uncertainty for autonomous driving
Deep Reinforcement Learning unter Unsicherheit im autonomen Fahren

Vorgelegte Master-Thesis von Rong Zhi aus Beijing

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Heinz Koeppl
3. Gutachten: Dr. Joni Pajarinen, Dr. Adrian Šošić

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Rong Zhi, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Errata

Incorrect clarifications and equations
(This thesis report has been modified by the following changes)

- **p.21, paragraph 3**

  The sentence "The update rule can be obtained by using V^{ret} as the target value minimize the MSE loss in Euqationb 3.7" should be removed.

- **p.21, paragraph 4**

  The third sentence "COPOS has shown that natural gradient leads to premature convergence as it always reduces the entropy of the distribution, especially for POMDP problems, where the agent needs more exploration to find the optimal policy" should be replaced by "We also adopt the entropy bound as hard constraint to balance the trade-off between exploration and exploitation."

- **p.22, Section 3.3.3** The update rule is missing. The update rule for parameters of final control policy: $\Theta^f_{new} = \Theta^f + \alpha \nabla_{\Theta^f} D_{KL}$, where $\Theta^f = (\theta^f, \beta^f)$ should be introduced after Equation 3.14. Add the explanation "Applying this auxiliary update step will lead to an approximated solution for the final control policy." directly after the new Equation. The order of second sentence and third sentence has to be changed.

- **p.23, Algorithm 1, line 14**

  The reference Equations are missing. The original sentence "Use $\tilde{F}^{\pi_{gold}}_{w_g}(s,h,a)$ and $\tilde{F}^{\pi_{fold}}_{w_f}(h,a)$ to solve Lagrangian multipliers using the dual to equation (??) and (??)" should be reformulated as "Compute $\tilde{F}^{\pi_{gold}}_{w_g}(s,h,a)$ and $\tilde{F}^{\pi_{fold}}_{w_f}(h,a)$ by Equation (3.9) and apply them into Equation (3.11) and (3.12) to compute $\eta^g, \omega^g, \eta^f, \omega^f$"

- **p.21, Equation 3.10**

  The importance weight $\bar{\rho}_t$ are missing. It should be changed from $\delta^{ret}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ to $\delta^{ret}_t = r_t + \gamma \bar{\rho}_t V(s_{t+1}) - V(s_t)$.

- **p.23, Equation 3.17**

  It should be changed from $\Theta_{fnew} = \Theta_{fold} + \alpha \nabla_{\Theta_f} \mathcal{L}(\pi_{\{})$ to $\Theta_{fnew} = \Theta_{fold} + \alpha \nabla_{\Theta_f} \mathcal{L}(\pi_f)$

- **p.33, Section 4.2.4**

  The historical observation and action pair should be changed from $(o_{t-2^T} - a_{t-2^T})$ to (o_{t-2^T}, a_{t-2^T})

Abstract

Compatible policy search (COPOS) [1] is a model-free policy search method that combines the idea of using compatible function approximation and employing the well-known Kullback–Leibler (KL) divergence as constraint for a trust region policy update. In addition, COPOS adds an entropy regularization constraint to trade off between exploration and exploitation and to prevent premature convergence. However, model-free policy search methods require more training data and long training time compared to model-based methods. Consequently, it is difficult for model-free policy search methods to solve challenging partially observed tasks where the agent can only perceive partial observations of the environment. Guided policy search (GPS)[2] and its variants [3, 4] combine the advantages of trajectory optimization and supervised learning, namely, high sample efficiency, and complex nonlinear policy representations that can cope with high-dimensional state and action spaces. However, the known model or the learned model in the GPS family may be inaccurate and potentially lead to failures of the final learned control strategies. In this thesis, we aim to solve partially observable Markov decision processes via the combination of policy search and supervised learning. We propose a novel method called guided-COPOS that combines a model-free guided framework with COPOS. Specifically, we use two agents to generate samples during the training phase, one of the agents selects actions based on partial observations, while the other agent can take actions by perceiving the full state observations. We decompose the policy optimization into two steps, a constrained optimization step embedded in COPOS such that both policies update toward the direction of achieving better long-time rewards, and an unconstrained optimization step by minimizing the KL divergence between the two policies such that they converge to the same behavior. In our customized challenging partially observable environment (LunarLander-POMDP), where we have successfully learned the policy and achieves good empirical results, outperforming other well-known policy search methods —TRPO, PPO, and COPOS. We then test our guided-COPOS for the challenging partially observable autonomous driving task. The results show that our guided-COPOS is able to stabilize the training process, and has fewer collisions with pedestrians and cars at test time compared to COPOS.

Zusammenfassung

Compatible Policy Search (COPOS) [1] ist eine modellfreie Policy-Suchmethode, welche die Ideen kombiniert, die kompatible Funktionsapproximation zu verwenden und dabei die bekannte Kullback-Leibler (KL) Divergenz als Nebenbedingung für ein Trust Region Policy Update zu nutzen. Darüber hinaus fügt COPOS eine Einschränkung der Entropie-Regulierung hinzu, um zwischen Exploration und Ausnutzung zu unterscheiden und eine vorzeitige Konvergenz zu verhindern. Im Vergleich zu modellbasierten Methoden erfordern modellfreie Methoden der Policyoptimierung jedoch mehr Trainingsdaten und eine lange Trainingszeit. Folglich ist es für modellfreie Methoden der Policyoptimierung schwierig, anspruchsvolle, partiell beobachtbare Aufgaben zu lösen, bei denen der Agent die Umwelt nur teilweise wahrnehmen kann. Guided Policy Search (GPS)[2] und seine Varianten [3, 4] kombinieren die Vorteile von Trajektorienoptimierung und überwachtem Lernen, nämlich hohe Sample-Effizienz und komplexe nichtlineare Policy-Repräsentationen, die mit hochdimensionalen Zustands- und Aktionsräumen umgehen können. Das bekannte oder gelernte Modell in der GPS-Familie kann jedoch ungenau sein und möglicherweise zu Fehlern der endgültig gelernten Policy führen. In dieser Arbeit zielen wir darauf ab, partiell beobachtbare Markov decision processes durch die Kombination von Policyoptimierung und überwachtem Lernen zu lösen. Wir schlagen eine neuartige Methode namens Guided-COPOS vor, die ein modellfreies Guided Framework mit COPOS kombiniert. Konkret verwenden wir zwei Agenten, um Stichproben während der Trainingsphase zu generieren, wobei einer der Agenten Aktionen basierend auf partiellen Beobachtungen auswählt, während der andere Agent Aktionen durchführen kann, indem er die vollständigen Zustaende beobachtet. Wir zerlegen die Policyoptimierung in zwei Schritte, einen Optimierungsschritt mit Nebenbedingungen, der in COPOS eingebettet ist, sodass beide Politicies in Richtung besserer Langzeitbelohnungen aktualisiert werden, und einen nicht beschraenkten Optimierungsschritt, indem wir die KL-Divergenz zwischen den beiden Policies minimieren, sodass sie sich auf das gleiche Verhalten einigen können. Wir evaluieren unsere Methode zunächst in unserem maßgeschneiderten, anspruchsvollen, partiell beobachtbaren Umfeld (LunarLander-POMDP), in dem wir die Policy erfolgreich gelernt und gute empirische Ergebnisse erzielt haben, die andere bekannte Methoden der Policysuche - TRPO, PPO und COPOS - übertreffen. Anschließend testen wir unser Guided-COPOS für die anspruchsvolle, partiell beobachtbare Aufgabe des autonomen Fahrens. Die Ergebnisse zeigen, dass unser Guided-COPOS den Trainingsprozess stabilisieren kann und zu Testzeiten weniger Kollisionen mit Fußgängern und Autos hat als COPOS.

Acknowledgments

First of all, I would like to express my sincere gratitude to Dr. Joni Pajarinen and Dr. Adrian Šošić for the interesting topic, their patient supervision, helpful guidance, and continuous support for this thesis. I would also like to thank Prof. Dr. Jan Peters and Prof. Dr. Heinz Koeppl for offering me the opportunity to work in their groups: "Intelligent Autonomous Systems" and "Bioinspired Communication Systems" and leading me working on this exciting thesis. Besides, I would like to thank Yunlong Song, Susanne Trick, Onur Celik, and Philipp Becker for their helpful suggestions and discussions through the process of researching and writing this thesis.

Finally, I must express my very profound gratitude to my parents for providing me with continuous support and encouragement throughout my life. This accomplishment would not have been possible without them.

Contents

Figures and Tables

List of Figures

List of Tables

1 Introduction

Nowadays, artificial intelligence techniques are experiencing a resurgence of interests in the application of computer vision, natural language understanding, robotics, etc., and have become a core part of the technology industry. AlexNet surprised both the academy and the industry in 2012 by showing for the first time that convolutional neural networks performed so well on a historically difficult ImageNet dataset [7]. Two years later, a research group from Facebook created a face recognition system called Deepface that achieves a face recognition accuracy of 97.35% which rivals human performance in 2014 [8]. In 2016, Google introduced a Google Neural Machine Translation (GNMT) system that translates text from one language to another using deep neural networks and achieves competitive results to state-of-the-art [9]. All these achievements depend heavily on a mathematical framework called the deep neural network that provides powerful function approximation and generalization abilities for finding the underlying features of high-dimensional data, such as raw image pixels. NVIDIA group [10] proposed an end-to-end training framework that use CNNs to map the camera images into steering control commands. They trained the network with human driving data via supervised learning and shown a satisfying solution for highway driving. Despite the fact that deep neural networks have shown its capacity in a variety of contexts, many applications use supervised learning methods which require a huge amount of manual labor from humans. Hence, supervised learning or deep learning alone is impractical to solving problems in which humans' supervision is intractable, e.g., autonomous driving under uncertainty.

Autonomous driving under uncertainty is a challenging problem—the states and actions in real-world environments are in high-dimensional space. The behavior of pedestrians on the sidewalk or other vehicles on the road and weather conditions can be high uncertainty and not be labeled, and hence, such uncertainty makes the problem even harder. Intersecting with supervised learning and unsupervised learning, reinforcement learning (RL) is concerned with how to make optimal sequential decisions in an environment through trial-and-error learning scheme. RL algorithms are commonly picked out to solving problems in a stochastic environment where humans' supervision is not possible. The method that combines the trial-and-error learning scheme of RL and the powerful generalization ability of deep neural network is called deep reinforcement learning (deep RL). For example, DeepMind introduced the Deep Q-Network (DQN) [11] that was the first method used deep learning in RL and achieved "human level" performance on playing Atari games in 2013. In October 2015, AlphaGo beat Lee Sedol, which was the first time that an AI had beaten a human professional player at the game Go, and in 2017, AlphaGo beat the world No.1 ranked player Ke Jie [12].

1.1 Contribution

In this thesis, we aim to solve the autonomous driving under uncertainty problem using deep reinforcement learning methods. In order to do that, we propose a novel method called guided-compatible policy search (guided-COPOS) that combines a model-free guided framework with the compatible policy search (COPOS) method. We introduce a new samples generation way for guided policy search, where the guiding agent and final control agent iteratively interact with the same environment. We update the two policies via compatible feature approximation embedded in COPOS such that the final control policy update towards the direction of a policy with better long-time rewards. We further introduced an additional update step for final control policy by minimizing the Kullback–Leibler divergence between the two policy distributions such that the final control policy converges to the same behavior as guiding policy. We first evaluate our method and other well-known policy search methods: trust region policy optimization (TRPO), proximal policy optimization (PPO), and COPOS in our customized partially observable environment—LunarLander-POMDP, the results of which can be used as a baseline for future research. Then, we introduce our customized urban autonomous driving environment based on CARLA simulator and create three different autonomous driving tasks with increasing difficulty: a goal-directed task without dynamic traffic users, a goal-directed task with full state observations of dynamic traffic users, and a goal-directed task with partial observations of dynamic traffic users. We also test COPOS and guided-COPOS in the autonomous driving tasks as a baseline for future research.

1.2 Outline

The thesis is structured as follows:

- Chapter 2: We give a brief introduction for several key concepts in reinforcement learning, including the mathematical framework of Markov decision process and partially observable Markov decision process and show several

methods to solve them. Then, we introduce deep learning and highlight its combination with reinforcement learning algorithms by reviewing several state-of-the-art deep RL algorithms. Finally, we introduce the traditional framework of the autonomous driving system and show several challenges the framework faced with.

- Chapter 3: We introduce our model-free guided-compatible policy search (guided-COPOS) that adapt COPOS into a new guided policy search framework. We compare our guided-COPOS with other well-known policy search methods: TRPO, PPO, and COPOS in our customized POMDP environment.

- Chapter 4: We create our customized urban autonomous driving environment based on CARLA simulator and design three tasks with increasing difficulty based on this environment. Then, we test COPOS with all environment as baselines. And finally, we compare guided-COPOS and COPOS in the challenging partially observable task.

- Chapter 5: We summarize the work of this thesis, discuss the current results, and present future work to further improve the performance of our proposed guided-COPOS.

2 Background

In this chapter, we first briefly introduce several key concepts in reinforcement learning (RL) fundamentals, including a mathematical framework of the Markov decision process (MDP) and function approximations of the policy and the value function. When the state of the environment is fully observable by the agent, an RL problem can be modeled as an MDP, however, when agents cannot directly observe the underlying state, the problem is defined as a partially observable Markov decision process (POMDP). To show the differences between MDPs and POMDPs, we describe the decision process via a typical agent-environment interface that shows how the agent interacts with its surrounding environment. There are two main categories of RL algorithm to solving the MDP problem: value-based methods, which are based on an estimation of the environment's dynamics via value functions, and policy-based methods, which are based on the direct mapping from states to actions. In order to solve the more challenging POMDP problems, we focus our discussion on the other two classes: brief-based methods and model-free methods. In the second part of this chapter, we introduce deep learning methods that serve as the basis and provide a powerful function approximation for our deep RL algorithms. Methods that combines deep neural networks with RL algorithms is called deep RL. We highlight several deep RL methods, such as trust region policy optimization (TRPO), compatible policy search (COPOS), and proximal policy optimization (PPO), to express important properties in deep RL. For example, deep RL methods enable agents to learn optimal control policies in the problem with high-dimensional raw states input, e.g., raw pixel data. Finally, we talk about the automated level of self-driving cars and underlying technical challenges, which are long-standing problems for the traditional algorithms. In contrast to naive approaches, we show that deep RL methods are the promising ways in quest of full automation systems.

2.1 Reinforcement learning

The interaction between the agent and the environment can be modeled formally as a Markov decision process (MDP) in terms of states, actions, and rewards. However, in reality, the agent can only perceive the environment via noisy sensors and must make decisions under uncertainty of true environment states. Such decision-making problems under uncertainty are well-known as the partially observable Markov decision processes (POMDPs). We introduce mathematical frameworks for both MDPs and POMDPs in order to discuss several key notations and explain the standard learning process in RL. More importantly, we talk about the function approximation, a problem that has been well-studied in machine learning and plays a crucial role in most RL methods for representing value functions and policies. We show that deep neural networks compare favorably to linear function methods in terms of learning high-dimensional states representations. In addition, we discuss compatible value function approximation.

2.1.1 Markov decision process (MDP)

Markov decision process is the fundamental formalism of sequential decision making [5], where the agent takes states as input and generates actions as output that influence immediate rewards, and subsequently, future states and rewards.

A Markov decision process can be defined as a tuple $<\mathcal{S}, \mathcal{A}, \mathcal{P}, r>$, where

- $\mathcal{S}$ is the state space that contains all possible states $\mathbf{s} \in \mathcal{S}$ of the environent,
- $\mathcal{A}$ is the action space that contains all possible actions $\mathbf{a} \in \mathcal{A}$ that the agent can excute,
- $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ represents the conditional probability of reaching state s_{t+1} given that action a_t is taken at state s_t . In MDP, we assume the P satisfies a Markov property, $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, ...) = \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$,
- $r(\mathbf{s_t}, \mathbf{a_t})$ describes the immediate reward received for taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$,

In different kinds of literature, definitions of MDPs vary slightly in notations and components of the tuple representation, depending on different tasks. For example, some definitions include an initial state μ_0 that specifies the probability distribution of the initial state of the environment, some include the horizon T describes the maximum time steps of an episode. For a continuous task, the horizon is infinite $T = \infty$, and thus, others include the discount factor $0 < \gamma \leq 1$ to trade off between the immediate reward and the future reward.

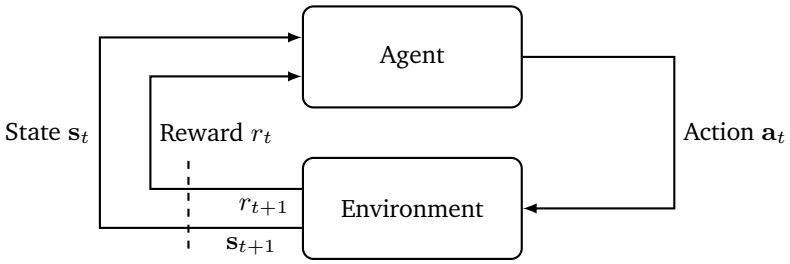

Figure 2.1: Typical agent-environment interaction in a Markov decision process. [5]

The interaction between the agent and the environment is illustrated in Figure 2.1, where an agent takes action $\mathbf{a}_t$ based on the current state $\mathbf{s}_t$ and then the environment responses to the agent's action with a new state $\mathbf{s}_{t+1}$ and a scalar signal r named rewards. The interaction repeats iteratively until the horizon T is reached. The goal of the RL agent is to find an optimal policy that maximizes the sum of the reward, which also called returns. One important concept in MDP is the discount rate γ, which is used for reducing the value of future rewards by γ^{k-1}, where k is the time difference between current state s_t and the future state s_{t+k}.

2.1.2 Value functions and policies

Value functions—including state-value functions and action-value functions—and policies are two essential elements in RL, as most RL algorithms require approximations and optimizations of value functions and (or) policies. The state-value function is a function mapping from states to a scalar that indicates the goodness or badness of being in a given state, similarly, action-value functions are applied to represent future rewards that are expected when performing a given action in a given state. The policy, quite differently, is used to define the behavior of an agent. Given a state, we can define a deterministic policy $\mathbf{a} = \pi(\mathbf{s})$ that selects a deterministic action $\mathbf{a}$, on the other hand, we can draw an action from a stochastic policy $\mathbf{a} = \pi(\mathbf{a}|\mathbf{s})$, which is represented as a conditional probability distribution, e.g., a Gaussian distribution. In reality, the optimal policy is generally represented via a stochastic probability distribution, hence, we assume a stochastic policy in our work. The goal of RL is to find the optimal policy that maximizes the cumulative reward in the long run, which is expressed as the expected return. Here, the return is defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k r(\mathbf{s}_t, \mathbf{a}_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ is the discounted factor, $0 =< \gamma <= 1$. Thus, the value of a state when following the policy π is formally defined by

$$V_\pi(\mathbf{s}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | \mathbf{s}_t = \mathbf{s}\right], \tag{2.1}$$

where $\mathbb{E}_\pi$ indicates an estimation given sampled rewards under policy π, and t is a time step. This value is the "ground-truth" number for optimizing the state-value function approximator $V(\mathbf{s})$, or, we find an optimal state-value function by using the estimated value $v_\pi(s)$ for minimizing a the, denoted $\|V_\theta(\mathbf{s}) - v_\pi(\mathbf{s})\|^2$, with respect to the parameter vector θ of the value function approxiamtor $V_\theta(\mathbf{s})$. Similarly, we can also learn the action-value function $Q_\pi(\mathbf{s}, \mathbf{a})$, defined as the expected return when starting in state $\mathbf{s}$, taking action $\mathbf{a}$ and following policy π

$$Q_\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right] \tag{2.2}$$

An important property of value functions is that they satisfy the recursive properties. The relationship between the value of current state $\mathbf{s}$ and its successive state $\mathbf{s}'$ can be described via the following Bellman Equation

$$V_\pi(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \sum_{\mathbf{s}', r} \mathcal{P}(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) \left[r + \gamma V_\pi(\mathbf{s}')\right]$$

2.1.3 Function approximation

For discrete MPDs, we can represent the value function by a lookup table, where every state $\mathbf{s}$ has an entry $V(\mathbf{s})$, or every state-action pair $\mathbf{s}, \mathbf{a}$ has an entry $Q(\mathbf{s}, \mathbf{a})$. However, for continuous MDPs, the state and action space become so large that it is impossible to visit and store each value individually in an array. A common solution for dealing with high-dimensional state and action spaces in continuous MDPs is to use value function approximations as genelarization of state features. Similarly, the polies can be estimated by function approximation as well, where an optimal policy is directly estimated and updated. Both policy and value function are approximated by samples, which can be described as $\boldsymbol{\tau} = (\mathbf{s}_0, \mathbf{a}_0, r_0, \mathbf{s}_1, \mathbf{a}_1, r_1, ...)$ or $\boldsymbol{\tau} = (S_t, A_t, G_t)$, also called trajectories or rollouts, generated by following the policy π and the transition function $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ of the environment.

For simplicity, we use the notation $\boldsymbol{\theta}_v$ to represent the parameter vector of the value function and $\boldsymbol{\theta}_p$ to denote the parameter vector of the policy function.

Linear function approximation:
A linear value function approximator is a function that is linear in the weight vector θ, however, not necessarily linear in the state input s. For example, linear methods approximate the state-value function by a linear inner product between θ and $\psi(s)$

$$V(\mathbf{s}) = \theta_v^T \phi(\mathbf{s}),$$

where ϕ s a feature vector that maps states into features representations. Several well-known features representations include polynomials, Fourier basis, and radial basis functions (RBFs). For example, one type of RBFs is a Gaussian function

$$\phi(\mathbf{s})_i = \exp\left(-\frac{||\mathbf{s} - c_i||^2}{2\sigma_i^2}\right),$$

where c_i is the center state, and σ_i is the feature width. When state space is large, the state can be represented by a feature vector of limited size, hence, linear methods are very efficient in terms of data representation and computational cost in practice [5].

One drawback of linear function approximation is that the features have to be chosen beforehand very carefully and not every value function can be represented as feature vectors. Thus, the domain knowledge may be required here [13].Linear methods may be most useful in cases where the input space is low dimensional, or where the training examples all come from a low-dimensional manifold in a high-dimensional space.

Nonlinear function approximation:

A typical kind of non-linear function approximator is called the deep neural network, which is composed of several layers that are connected by non-linear activation functions, e.g., sigmoid activation functions. The universal approximation theorem claims that the standard multilayer feed-forward networks with a single hidden layer that contains a finite number of hidden neurons, and with arbitrary activation function are universal approximators in $C(R^m)$ [14], or, simply put, simple neural networks can learn very complex functions. We show a multilayer perceptron (MLP) in Figure 2.2 that consists of three layers. For the sake of simplicity, we omit the activation layer that connects the hidden layer to the output layer. We can define an MLP formally using the following function as

$$\mathbf{z} = f^{(k)}\left(\mathbf{W}^{(k)T}\mathbf{x} + \mathbf{b}\right),$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}$ and $\mathbf{b}$ are the corresponding weights and biases, $\mathbf{z}$ is the output of the MLP, and $f^{(k)}$ is a nonlinear activation function, a logistic function $f(\mathbf{x}) = 1/(1 + e^{-\mathbf{x}})$ for instance. TThus, we can represent the MLP depicted in Figure 2.2 as

$$\mathbf{y} = f^{(2)}\left(\mathbf{W}^{(2)T} f^{(1)}(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}\right)$$

In order to optimize the network (through the backpropagation procedure), we have to define a loss function that is differentiated with respect to the weights of the network and update the network in the direction of minimizing the loss function

$$E(\mathbf{W}) = \sum_{i=1}^{N} \frac{1}{2}(\mathbf{y}_n - \mathbf{t}_n)^2. \tag{2.3}$$

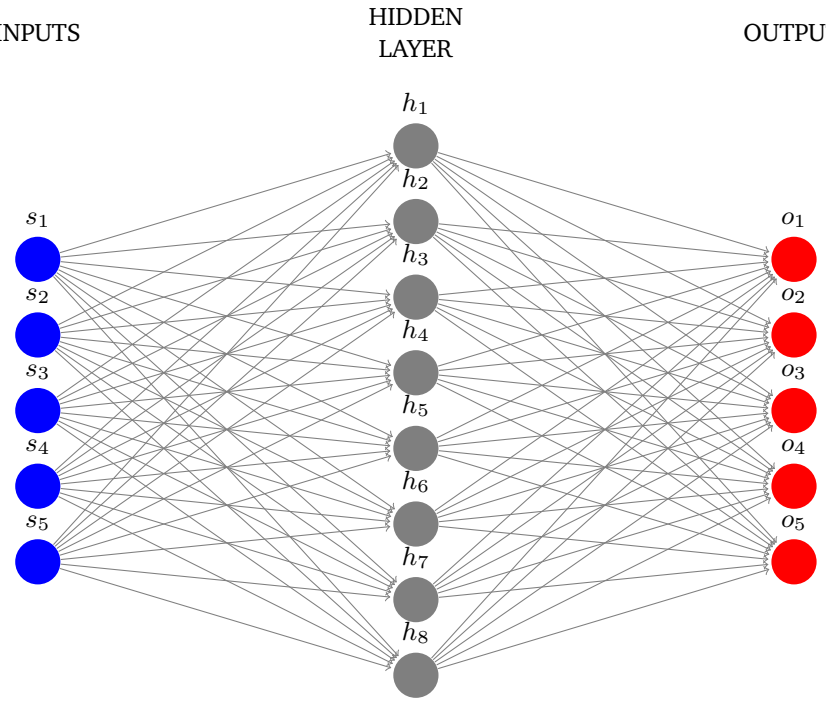

Figure 2.2: A sketch of the multilayer perceptron that has three layers. The prediction is made by using **forward-propagation**: the data is fed into the first layer (blue), the hidden layer (gray) does a linear transformation on input data using network parameters (arrows), the resulting outputs are non-linearly processed by an activation layer, the information moves from layer to layer, and finally the output layer predict outputs.The training is made by using **backward-propagation**: the gradients with respect to network parameters are calculated from the last layer and propagated back to the first layer via the chain rule.

As one does not need to select hand-crafted features with nonlinear value function approximation, one does not know what exactly have the neural networks learned during training either. Furthermore, there are less theoritical convergence guarantees can be given for neural networks. But in general, nonlinear function approximator has shown a better approximation accuracy in practice than linear function approximator [13].

Compatible value function approximation:

Sutton et al. (1999) [15] proposed a policy gradient with function approximation that explicitly represents a parametrized policy by its own parameters, independent of the value function. Such a method is also known as compatible value function approximation, of which one can obtain an unbiased gradient with smaller variance [16]. Specifically, let $F_{\mathbf{w}}$ be the approximation to $Q_\pi(\mathbf{s}, \mathbf{a})$, with parameter w

$$Q^\pi(\mathbf{s}, \mathbf{a}) \approx \hat{F}^\pi_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) = \phi(\mathbf{s}, \mathbf{a})^T \mathbf{w}, \quad \phi(\mathbf{s}, \mathbf{a}) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}), \tag{2.4}$$

where $\nabla_w F(s, a) = \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ indicates that the value function approximator is compatible to the policy.

$$\nabla_{\mathbf{w}} \hat{F}^\pi_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a}).$$

The value function parameters $\mathbf{w}$ minimize the mean-squared error $(Q^\pi(\mathbf{s}, \mathbf{a}) - \phi(\mathbf{s}, \mathbf{a})^T \mathbf{w})^2$, then the policy graident is

$$\mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[(Q^\pi(\mathbf{s}, \mathbf{a}) - \phi(\mathbf{s}, \mathbf{a})^T \mathbf{w})^2 \right]$$

and the corresponding solution is

$$\mathbf{w}^* = \text{argmin} \quad \mathbb{E}_{p(s)\pi(\mathbf{a}|\mathbf{s})} [(Q^\pi(\mathbf{s}, \mathbf{a}) - \phi(\mathbf{s}, \mathbf{a})^T \mathbf{w})^2]$$

the solution of whcih is also known as natural gradient form. Notice that the compatible value function approximation always has zero mean for each state, hence, it is actually the approximation for advantage function $A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s}, \mathbf{a})$ rather than Q-function, where the knowledge of value function is required. We will talk about its implementation with policy search methods in Section 2.3.

2.1.4 Partially obervable Markov decision process (POMDP)

MDP assumes that the agent has the full knowledge about its current state, while in reality, the agent can only gain knowledge of the real states through observations, and thus, does not fit the assumption of MDP. For example, in autonomous driving, agent cars perceive their surroundings through camera, radar, GPS, and odometry. These sensor data are inaccurate and always noisy, and in general, can only provide partial information of the world, e.g. pedestrains behind cars or cars behind corners cannot be detected through current tehcniques.

The partiallly observable Markove decision process (POMDP) model for describing the agent taking actions under uncertanity of the states can be formally defined as tuple $<\mathcal{S}, \Omega, \mathcal{A}, \mathcal{P}, \mathcal{O}, r>$, where:

- $\mathcal{S}$ is the state space, containing all possible real states $\mathbf{s} \in \mathcal{S}$ of the environent,
- Ω is the observation space, containing all possible observations $\mathbf{o} \in \Omega$ that the agent can perceive,
- $\mathcal{A}$ is the action space, containing all possible actions $\mathbf{a} \in \mathcal{A}$ that the agent can excute,
- $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ is the transition function, describing the probability of executing action $\mathbf{a}_t$ from state $\mathbf{s}_t$ and reaching state $\mathbf{s}_{t+1}$,
- $\mathcal{O}(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{a}_t)$ is the observation function, describing the probability of the ovservation $\mathbf{o}_{t+1}$ given that the agent has executed action $\mathbf{a}_t$, reaching state $\mathbf{s}_{t+1}$,
- $r(\mathbf{s_t}, \mathbf{a_t})$ is the reward function, describing the immediate reward received for taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$,

We assume that the agent is acting in a Markovian world, in which the transition function $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ must fulfill Markov property. However, the limited sensor information does not allow the agent to be Markovian with respect to observations since the projection from observation space to state space is not unique. One option to rebuild a Markovian agent is to use the past experience that composes of all interactions between the agent and the environment [17], denoted as $\mathbf{h}_t = (\mathbf{o}_0, \mathbf{a}_1, ..., \mathbf{a}_{t-1}, \mathbf{o}_t)$. It can be easily seen that this history of observation and action pairs satisfies the Markov property: $\mathcal{P}(\mathbf{h}_{t+1} = (\mathbf{o}_0, \mathbf{a}_1, ..., \mathbf{a}_t, \mathbf{o}_{t+1})|\mathbf{h}_t, \mathbf{a}_t) = \mathcal{P}(\mathbf{o}_{t+1}|\mathbf{h}_t, \mathbf{a}_t)$. As a consequence, the POMDP problem is transferred to an MDP problem, also known as *information state* MDP, and can be further solved by using same methods for solving MDP. Other methods like *belief state* MDP, using belief $\mathbf{b}_t$ to represent the probability of the agent being at state $\mathbf{s}_t$, can be updated after observing $\mathbf{o}_t$ and taking action $\mathbf{a}_t$: $\mathbf{b}_{t+1} = p(\mathbf{s}_{t+1}|\mathbf{b}_t, \mathbf{a}_t, \mathbf{o}_t)$, which means that the process over belief states is Markovian.

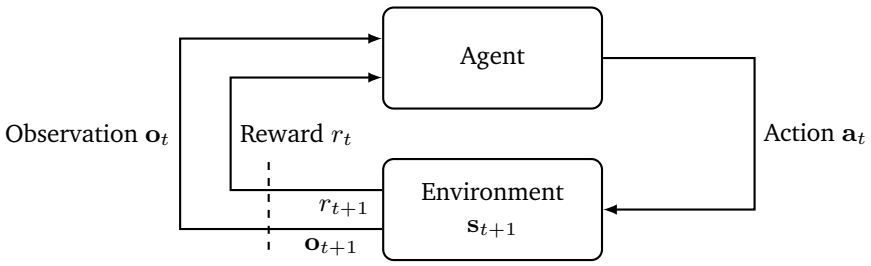

Figure 2.3: Agent-environment interaction in a partially observable Markov decision process. The agent perceive observation, while the real state of the envrionment is hidden from the agent.

The interaction loop between the agent and the envrionment is shown in Figure 2.3 the agent interact with the enrionment iteratively by taking actions after receiving observations, while the real states are hidden from the agent.

2.2 Reinforcement learning methods

Now that we have defined MDP, POMDP, policy and value functions, and function approximation techniques, we will show how to comupte an optimal policy in RL. There are two main classes RL methods—value function methods and policy search methods. Value function methods estimate value functions directly, where the policy is explicitly learnt by the estimated values. Policy search methods learn a parameterized policy directly that can choose actions without the requirement of value function.

2.2.1 Value-based methods

Value function methods learn the estimated state-action values for nonterminal state $\mathbf{s}_t$ from samples of trajectories $\boldsymbol{\tau}$, and further choose actions based on their estimated values. The basic idea of value based methods is to learn the optimal policy by *generalized policy iteration* (GPI), which consists of two steps: *policy evaluation* and *policy improvement*. In the policy evaluation step one estimates the current value and Q-function of the current policy, in the policy improvement step the policy is updated with respect to the current value function. Several well known value-based methods are *Monte-Carlo* (MC) and *temporal difference* (TD) learning methods.

MC methods work only for episodic MDPs, as they estimate the value by empirical mean return from complete episodes. Specifically, MC keep frequency counts on the states and the corresponding returns and average them after each visit to the state. As more visits to states we have, the average return should converge to the true value of the state [5]. A simple *every-visit* MC method is updated by:

$$V(\mathbf{s}_t) \leftarrow V(\mathbf{s}_t) + \alpha \left[G_t - V(\mathbf{s}_t) \right].$$

As the return depends on a large number of random actions, transitions, rewards, MC methods have high variance but are unbiased.

In contrast of MC methods, TD methods update estimates of values based in part on the estimated values of next step, and thus, can learn online without waiting for the end of an episode, this technique is also known as *bootstrapping*. For TD(0) learning, a member of the family of TD learning, the update rule is:

$$V(\mathbf{s}_t) \leftarrow V(\mathbf{s}_t) + \alpha \left[r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) \right],$$

where $r_{t+1} + \gamma V(\mathbf{s}_{t+1})$ is called *TD target*, $\delta_t = r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)$ is called *TD error*, measuring the difference between current estimate and TD target. As we can see from the update rule, TD target depends only on one random action, transition, reward, hence, TD methods has lower variance compared to MC methods.

So far, we have estimated the state-values, where the policy is determined by:

$$\pi(\mathbf{s}_t) = \arg\max_a r(\mathbf{s}, \mathbf{a}) + \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}) V(\mathbf{s}_{t+1}).$$

Note that we still need to know the transition function $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}, \mathbf{a})$ in this case, but if we estimate the state-action values instead, we will remove the need for knowledge of the model, and the policy is turned to:

$$\pi(\mathbf{s}_t) = \arg\max_a Q(\mathbf{s}_t, \mathbf{a}_t).$$

The update procedure for estimating state-action values is same as that for state-values.

Some popular value function methods like *Q-learning* [18] and *state-action-reward-state-action* (Sarsa) [19] are based on TD methods, where Sarsa is an *on-policy* method, which aims to learn the optimal policy by exploring action space using same policy non-optmilally, and Q-learning is an *off-policy* method, which seeks to learn the optimal policy individually from the *behavior policy* (policy executed by the agent).

2.2.2 Policy-based methods

An alternative method of RL is to search directly in the policy space, or in other words, to learn the parameters of a parameterized policy such that the parameters are updated in the direction of a policy with better perfomance. The objective of policy based RL methods is to find parameters $\boldsymbol{\theta}$ that maximizes the expected retrun:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[r(\boldsymbol{\tau}) \right] = \int \pi_\theta(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau},$$

where $\pi_{\boldsymbol{\theta}}$ can be a parameterized stochasitc policy, e.g. Gaussian policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{a}_t|\mathbf{s}_t, \boldsymbol{\theta})$, $\boldsymbol{\tau} = (\mathbf{s}_1, \mathbf{a}_1, r_1, ..., \mathbf{s}_T, \mathbf{a}_T, r_T)$ is the sampled trajectories, and $r(\boldsymbol{\tau})$ is the cummulative reward of $\boldsymbol{\tau}$.

Policy based methods can be divided into gradient-based and graident-free methods, we will mainly focus on gradient-based methods in this thesis. One of the well-known methods is *policy graident method*, also known as REINFORCE [20],

which uses gradient descent to maximize the expected return $J(\pi_\theta)$. The update direction of parameters is given by the gradient of the $\nabla J(\pi_\theta)$, which is proportional to the sum over states weighted by states occurrence frequency following policy $\pi_{\boldsymbol{\theta}}$, and can be approximated using Monte Carlo simulation with samples $\boldsymbol{\tau}$ according to [5]:

$$\begin{aligned}\nabla J(\pi_\theta) &= \int \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau} \\ &= \int \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau} \\ &= \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}) r(\boldsymbol{\tau}) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right]\end{aligned} \tag{2.5}$$

Since the past rewards do not depend on future actions [15], the policy gradient becomes

$$\nabla_{\boldsymbol{\theta}} \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$$

REINFORCE estimates values using returns from complete episodes, and is thus also called Monte Carlo Policy Gradient. As a disadvantage of Monte-Carlo estimates, REINFORCE also suffers from high variance, which can be solved by subtracting a baseline $b(\mathbf{s})$:

$$\nabla J(\pi_{\boldsymbol{\theta}}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) \left(r(\mathbf{s}_t, \mathbf{a}_t) - b(\mathbf{s}) \right)$$

Notice that the baseline $b(\mathbf{s})$ can be any function that vary with state $\mathbf{s}$ but does not depend on action $\mathbf{a}$. One can use a learned state-value function $V_{\boldsymbol{\phi}}(\mathbf{s})$ as the baseline, then, the quantity $r(\mathbf{s}_t, \mathbf{a}_t) - b(\mathbf{s}_t)$ can be seen as an estimate of the *advantage* of action $\mathbf{a}_t$ in state $\mathbf{s}_t$, or $A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V_\phi(\mathbf{s})$. This approach is also known as *actor-critic* method where the policy π is the actor and the baseline b_t is the critice [5, 21]. As a result, the policy gradient becomes to

$$\nabla J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) A^\pi(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{2.6}$$

The general form of updating parameters for policy gradient is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}),$$

where α is the step size, also called learning rate. This update rule is well-known as classical stochastic gradient descent, which is generally sensitive to learning rate α. Fast convergence requires large learning rates but this may bring about numerical instability and even results in catastrophic policy updates. However, small learning rates could potentially lead to slow convergence and require more samples and time for training. Thus, one has to carefully choose an appropriate learning rate upon different learning tasks or use more advanced stochastic gradient descent (SGD) optimization scheme, such as Adam.

2.2.3 POMDP methods

In this section, we will discuss several common methods solving POMDPs. Belief-based methods and model-free methods. Belief-based approaches reformulate POMDPs into belief-state MDP and further learn the optimal policy via MDP methods (learning with a model), while model-free approaches compute the optimal solution without learning the model, and can be classifed into memory-free methods and memory-based methods.

Belief-based methods

As we mentioned before, the belief-state MDP defines a belief state $b(\mathbf{s})$ that represents the probability of an agent being at state $\mathbf{s}$, the axioms of probability require that $0 \leq b(\mathbf{s}) \leq 1$ for all $\mathbf{s} \in S$ and $\sum_s b(\mathbf{s}) = 1$. The update rule for belief-state MDP is:

$$\begin{aligned}b'(\mathbf{s}')_{b,a,o} &= p(\mathbf{s}|\mathbf{o}, \mathbf{a}, b) \\ &= \frac{p(\mathbf{o}|\mathbf{s}', b, \mathbf{a}) p(\mathbf{s}'|b, \mathbf{a})}{p(\mathbf{o}|b, \mathbf{a})} \\ &= \frac{\mathcal{O}(\mathbf{o}|\mathbf{s}', \mathbf{a}) \sum_s b(\mathbf{s}) \mathcal{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})}{p(\mathbf{o}|\mathbf{b}, \mathbf{a})},\end{aligned}$$

where the denominator, $p(\mathbf{o}|\mathbf{b},\mathbf{a}) = \sum_{s' \in S} p(\mathbf{o}|\mathbf{s}',\mathbf{a}) \sum_{s \in S} p(\mathbf{s}'|\mathbf{s},\mathbf{a}) b(\mathbf{s})$, can be treated as a normalizing factor and sums to 1. The policy of a POMDP agent for belief-based MDP is the mapping from belief state into action. Therefore, the POMDP environment can be described as tuple $<\mathcal{S}, \Omega, \mathcal{A}, \mathcal{P}, \mathcal{B}, \mathcal{O}, r, \tau, \rho>$ according to [22]:

- $\mathcal{S}, \Omega, \mathcal{A}, \mathcal{P}, \mathcal{O}$,and r remain the same ,
- $\mathcal{B}$ is the belief states space (infinite), containing all possible belief state $b \in \mathcal{B}$,
- $\tau(\mathbf{b}'|\mathbf{a},\mathbf{b})$ is the belief state-transition function, defined as the probability of starting at belief $\mathbf{b}$, executing action $\mathbf{a}$ and reaching the new belief $\mathbf{b}'$:

$$\tau(\mathbf{b}'|\mathbf{a},\mathbf{b}) = p(\mathbf{b}'|\mathbf{a},\mathbf{b}) = \sum_o p(\mathbf{b}'|\mathbf{a},\mathbf{b},\mathbf{o}) p(\mathbf{o}|\mathbf{a},\mathbf{b}),$$

  where

$$p(\mathbf{b}'|\mathbf{b},\mathbf{a},\mathbf{o}) = \begin{cases} 1, & \text{if} \quad \mathbf{b} = \mathbf{b}' \\ 0, & \text{otherwise,} \end{cases}$$

- $\rho(\mathbf{b},\mathbf{a})$ is the belief reward function of executing action $\mathbf{a}$ at state $\mathbf{s}$:

$$\rho(\mathbf{b},\mathbf{a}) = \sum_s b(\mathbf{s}) r(\mathbf{s},\mathbf{a}).$$

The goal of a POMDP agent is still to find an optimal policy $\pi(\mathbf{b})$, mapping beliefs to actions, such that the expected return is maximized, which is defined as the value function $V_\pi(\mathbf{b})$ starting from belief state $\mathbf{b}$ and following π:

$$V_\pi(\mathbf{b}) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \rho(\mathbf{b}_{t+k}, \mathbf{a}_{t+k}) | \mathbf{b} = \mathbf{b}_t \right].$$

The value of an optimal policy π^* is defined by the optimal value function V^*, which satisfies the Bellman optmiality equation:

$$V^* = H_{\text{POMDP}} V^*,$$

where H_{POMDP} is the *Bellman backup operator* for POMDPs, can be defined according to [13]

$$V^*(\mathbf{b}) = \max_{a \in A} \left[\rho(\mathbf{b},\mathbf{a}) + \gamma \sum_o \tau(\mathbf{o}|\mathbf{b},\mathbf{a}) V^*(\mathbf{b}')_{b,a,o} \right]. \tag{2.7}$$

It has been shown that the value function for a POMDP can be modeled arbitrarily closely as a finite set of linear functions $\Gamma = \{\alpha_1, ..., \alpha_n\}$ and $\alpha^a(\mathbf{s}) = r(\mathbf{s},\mathbf{a})$ [23], and the value at a given belief is:

$$V(\mathbf{b}) = \max_{\alpha \in \Gamma} \mathbf{b} \cdot \alpha,$$

where $\mathbf{b} \cdot \alpha = \sum_{s \in S} b(\mathbf{s}) \cdot \alpha(\mathbf{s})$ is the standard inner product operation in vector space.

Since belief-based MDP provides a compact representation of the historical data, many model-based techniques exploit such method for solving POMDP problems. For example, value iteration algorithm based on belief MDP, also known as *exact value iteration*, [23, 24], they compute the optimal value function V^* by applying H_{POMDP} for each iteration across the entire belief space until the error $|V_t^*(\mathbf{b}) - V_{t+1}^*(\mathbf{b})|$ is less than a threshold ϵ. As the set Γ grows exponentially with every iteration, the overall computational cost of exact value iteration is expensive, which makes the algorithm only feasible for smallest problems [13].

Other methods such as point-based value iteration (PBVI) [25], heuristic search value iteration (HSVI) [26] are point-based methods, which reduce the complexity by computing solutions only for limited set belief states that sampled from the interaction between an agent and the environment.

Model-free methods

Unlike the belief-based method, where either the knowledge of the model is required or the agent has to learn the model, model-free methods learn the policy without learning the model of the environment, and can be divided into memoryless approaches and memory-based approaches.

Memoryless approaches are based on the ideas that replace the state by latest observation without keeping track of past observations, and then apply some methods for fully observable domains to solve it. As we mentioned in the section 2.2.3, the projection from observation space to state space is not unique, using observations directly instead of states might cause the agent suffering from perceptual aliasing [27]. [28] proved theoretical limitations for memoryless policies. Several methods have been proposed to solve aliasing problem [29] or to dedicate to the problems where a memoryless policy exists [30, 31]. But in general, POMDP problems could not be optimally solved by memoryless policies.

In contrast, memory-based approaches use history of observations and actions and reduce POMDP problems to MDP problems. For instances, finite size history methods use the last N observations instead of latest observation solely. It assumes a POMDP that has no noisy sensor or very little noise [17], as for a large noise, it cannot provide optimal solutions. The combination of neural networks and memory-based approaches was first proposed by [32], where a recurrent neural network is used to solve the aliased states problems. The paper [14] showed a recurrent policy gradient method by using long-short-term memory (LSTM) network, which is able to learn memory-based policies for deep memory POMDP effectively. In this thesis, we will use a model-free and memory-based method.

2.3 Deep reinforcement learning

As we discussed in section 2.1.2, we can approximate value functions and policies by neural networks, and many other deep learning architectures. When we say "deep", we actually mean that the architectures like neural networks give the depth of a model [33]. Several remarkable architectures such as convolutional neural networks (CNNs) [34], a feed-forward neural network where a convolution operation is used between concatenated layers, and have been widely used in image processing. Recurrent neural networks (RNNs), another famous architecture, use their internal state (memory) to process sequences of inputs, which makes them applicable to tasks in natural language processing. Moreover, the combination of RNNs with long short-term memory (LSTM) blocks, also called LSTM networks [35], which avoids the vanishing gradient problem of RNNs, and are now widely used in many fields.

With the help of deep learning technology, RL is able to process complex sensory input, such as camera images and Lidar scanning data, and map them into action space. We will discuss several advanced methods in this section that has better data efficient and better convergence properties compared to the REINFORCE method. For example, the first important implementation of natural gradient in RL introduced by [16], uses the Fisher information metric that gives the steepest direction of the objective function, and is extended by Trust region policy optimization (TRPO) [36] to be scalable for large nonlinear networks, e.g. deep neural networks. Proximal policy optimization (PPO) [37] modifies the constraint optimization problem defined in TRPO further into an optimization problem without hard constraint, which can be solved using stochastic gradient descent (SGD) with similar performance and less computational cost. Compatible policy search (COPOS) also starts from the natural policy gradient, extends the compatible value function to deep neural networks. COPOS also avoids the premature convergence that caused by natural gradient by using an additional entropy bound.

Trust region policy optimization (TRPO)

Trust region policy optimization (TRPO) introduced a policy search method for optimizing certain "surrogate" objective function, with guaranteed monotonic policy improvement [36]. Specifically, the "surrogate" objective is defined as (single path)

$$\max_{\boldsymbol{\theta}} \quad J^{\text{surr}}(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_{\pi_{\boldsymbol{\theta}_{old}}} \left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{old}}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}_t \right], \tag{2.8}$$

where $\hat{A}$ is the estimated advantage value, and can be obtained by *generative advantage estimate* (GAE) $\sum_{i=0}^{K-1} \gamma^i r_{t+i} \gamma^k + V(S_{t+k}; \theta_v) - V(S_t; \theta_v)$ [21]. The objective is constraint by Kullback-Leibler (KL) divergence between new and old policy such that the update step is in trust region

$$\text{subject to} \quad \hat{\mathbb{E}} \left[KL[\pi_{\boldsymbol{\theta}_{old}}(\cdot|\mathbf{s}_t), \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)] \right] \leq \delta \tag{2.9}$$

Equation 2.8 and 2.9 together forms the contrained optimization problem, which is similar to natural policy gradients [16], natural actor critic [38], and REPS [39]. The problem can be solved by making linear approximation to $J(\pi_{\boldsymbol{\theta}})$ and quadratic approximation to KL divergence

$$g = \frac{\partial}{\partial \boldsymbol{\theta}} L_{\pi_{\boldsymbol{\theta}_{old}}}(\pi_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{old}}, \quad F = \frac{\partial^2}{\partial^2 \boldsymbol{\theta}} \overline{\mathrm{KL}}_{\pi_{\theta_{old}}}(\pi_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{old}},$$

which forms the Lagrangian function

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) = g \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}_{old}) - \frac{\lambda}{2}\left[(\boldsymbol{\theta} - \boldsymbol{\theta}_{old})^T F(\boldsymbol{\theta} - \boldsymbol{\theta}_{old}) - \delta\right] \tag{2.10}$$

differentiate Equation 2.10 with repect to $\boldsymbol{\theta}$ leads to the update rule:

$$\boldsymbol{\theta} - \boldsymbol{\theta}_{old} = \frac{1}{\lambda} F^{-1} g,$$

which is the steepest direction of the constraint optimizaiton problem, also known as natural gradient, and is invariant to rescaling and transformation of the parameters. The F is also called Fisher information matrix (FIM), which measures how sensitive the probability distribution is to different directions in parameter space, The expansive computational cost of calculating the inverse of FIM makes the natural policy gradient hard to scale to large nonlinear network in deep learning. TRPO solve the problem by computing an approximated solution using conjugate gradient followed by a line search without explicitly forming F^{-1}, which makes it scalable and can optimize nonlinear policies with deep neural networks. TRPO has shown robust performance on several MuJoCo tasks and Atari games.

Proximal policy optimization (PPO)

Proximal policy optimization (PPO) [37] solved the "surrogate" objective from TRPO by clipping it into a certain region and further solved it with first-order optimization using stochastic gradient descent (SGD). Specifically, the clipped "surrogate" objective is defined as

$$\max_{\boldsymbol{\theta}} \quad J^{\text{CLIP}}(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t\left[\min(r_t(\boldsymbol{\theta})\hat{A}_t), clip(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right],$$

where the probability ratio $r_t(\boldsymbol{\theta}) = \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)/\pi_{\boldsymbol{\theta}_{old}}(\mathbf{a}_t|\mathbf{s}_t)$, the advantage estimate $\hat{A}$ is estimated by GAE. Without the hard contraint of KL divergence like TRPO, the update rule becomes

$$\boldsymbol{\theta} = \boldsymbol{\theta}_{old} + \alpha \nabla_{\boldsymbol{\theta}} J^{\text{CLIP}}(\pi_{\boldsymbol{\theta}}),$$

where α is the learning rate. With this scheme, PPO avoids greedy update by including the change in probability ratio only when it makes the objective worse. They also provide an alternative to the clipped surrogate objective, which is to use a penalty on KL divergence with an adaptive coefficient

$$\max_{\boldsymbol{\theta}} \quad J^{\text{KLPEN}}(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_{\pi_{\boldsymbol{\theta}_{old}}}\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{old}}(\mathbf{a}_t|\mathbf{s}_t)}\hat{A}_t - \beta \mathrm{KL}\left[\pi_{\boldsymbol{\theta}_{old}}(\cdot|\mathbf{s}_t, \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t))\right]\right],$$

Likewise, the KL penalty version is also solved by SGD, which leads the update rule

$$\boldsymbol{\theta} = \boldsymbol{\theta}_{old} + \alpha \nabla_{\boldsymbol{\theta}} J^{\text{KLPEN}}(\pi_{\boldsymbol{\theta}}). \tag{2.11}$$

Note that the KL penalty coefficient β is determined by the current estimated KL divergence $d = \hat{\mathbb{E}}\left[\mathrm{KL}\left[\pi_{\boldsymbol{\theta}_{old}}(\cdot|\mathbf{s}_t), \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)\right]\right]$ and the target KL divergence d_{targ}, if $d < d_{targ}/1.5$, $\beta = \beta/2$, if $d > d_{targ} \times 1.5$, $\beta = \beta \times 2$.

Consequently, the computation for optimizing both versions of objective function is much less than that of TRPO's. The emperical results show that PPO's performance is overall better than TRPO in MuJoCo environments, Atari games, and RoboSchool tasks.

Compatible policy optimization (COPOS)

Compatible policy optimization (COPOS) [1] is another policy search method based on the idea of updating the policy within a trust region by using KL divergence as the bound. They proved that the natural gradient and the trust region optimization are equivalent when using a natural parametrization of the distribution in combination with compatible value function approximation. Furthermore, they showed that the standard natural gradient updates may reduce the entropy of the policy and lead to premature convergence eventually, which can be solved by using an additional entropy bound between new and old policy.

Specifically, COPOS formulates the objective function to Equation 2.12, as a trust-region optimization problem by a KL constraint between old and new policy to prevent unstable updates and an additional entropy constraint to avoid premature convergence. The objective is defined as

$$\begin{aligned} \text{argmax}_{\boldsymbol{\theta}} \quad & J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[\int \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s},\mathbf{a}) d\mathbf{a}\right] \\ \text{s.t.} \quad & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[\text{KL}(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})||\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot|\mathbf{s}))\right] \leq \epsilon, \\ & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[H(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot|\mathbf{s}))||H(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}))\right] \leq \beta, \end{aligned} \tag{2.12}$$

COPOS solves the contrained optimization problem via compatible value function approximation. The approximated advantage function $\tilde{F}^{\pi_{\text{old}}}_{\mathbf{w}(\mathbf{s},\mathbf{a})} = \phi(\mathbf{s},\mathbf{a})^T\mathbf{w}$ is composed of two parts: compatible features $\phi(\mathbf{s})$ and compatible weights $\mathbf{w}$. The compatible weights $\mathbf{w} = (\mathbf{w}_{\boldsymbol{\theta}}, \mathbf{w}_{\boldsymbol{\beta}})$ can be computed by

$$\mathbf{w}^* = \mathbf{F}^{-1}\nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}}), \quad \nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}}) = \sum_i \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_i|\mathbf{s}_i) A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_i,\mathbf{a}_i), \tag{2.13}$$

where J_{PG} is the standard policy gradient of Equation 2.5, $\mathbf{w}$ can be approximately solved using conjugate gradient followed by a line search. Then, the contrained optimization problem becomes to

$$\begin{aligned} \text{argmax}_{\boldsymbol{\theta}} \quad & \mathbb{E}_{\mu(\mathbf{s})}\left[\int \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \tilde{F}^{\pi_{\text{old}}}_{\mathbf{w}(\mathbf{s},\mathbf{a})} d\mathbf{a}\right] \\ \text{s.t.} \quad & \mathbb{E}_{\mu(\mathbf{s})}\left[KL(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})||\pi_{\boldsymbol{\theta}_{old}}(\cdot|\mathbf{s}))\right] < \epsilon \\ & \mathbb{E}_{\mu(\mathbf{s})}\left[H(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})) - H(\pi_{\boldsymbol{\theta}_{old}}(\cdot|\mathbf{s}))\right] < \beta \end{aligned} \tag{2.14}$$

COPOS solves the constraint optimization via Lagrangian multipliers method. Applying the weights into dual of the optimization objective, COPOS gives us the full update rule for log-linear parameters and the approximated update rule for non-linear parameters

$$\boldsymbol{\theta} = \frac{\eta\boldsymbol{\theta}_{\text{old}} + \mathbf{w}}{\eta + \omega}, \quad \boldsymbol{\beta}_{\text{new}} = \boldsymbol{\beta}_{\text{old}} + s\frac{\mathbf{w}_{\boldsymbol{\beta}}}{\eta}$$

Empirically, they show that COPOS achieves state-of-the-art results in wide variety of continuous RoboSchool tasks and several challenging POMDP tasks.

2.4 Autonumous driving under uncertainty

An autonomous car (also called a self-driving car, a driverless car) is a vehicle capable of monitoring its surrounding environment and navigating without human intervention. The autonomous driving system (ADS) is a highly complex system that combines a variety of technologies in the fields of signal processing, computer vision, machine learning, reinforcement learning, and automatic control. The Society of Automobile Engineers (SAE) defined five levels of autonomous driving as summarized in Table 2.1. According to SAE's definition, "driving modes" represents the type of driving scenario with characteristic dynamic driving task requirements (e.g., expressway merging, high-speed cruising, low speed traffic jam, closed-campus operations, etc.), "dynamic driving task (DDT)" includes the operational and tactical aspects of the driving tasks, but not the strategic (determining destinations and waypoints) aspect of the driving task. The ADS between level 0 and level 3 requires a human intervention for the fallback performance of DDT, which can be performed by the ADS for level 4 and level 5.

In this thesis, we will focus on autonomous driving for partially observable environment such as bad weather conditions and pedestrians crossing from shadowed portion of the road, where the agent car partially or totally loses its ability to monitor the surrounding environment and has to perform the DDT fallback based on the currently available data to achieve a minimal risk condition (e.g. crash). Since the behavior of drivers in collision imminent situations cannot be tested in real life due to safety concerns, our experiments will be performed on driving simulation software only.

Table 2.1: SAE (J3016) Automation Levels [6]

SAE level	Name	Narrative definition	Execution of steering and acceleration/deceleration	Monitoring of driving environment	Fallback performance of dynamic driving task	System capability (driving modes)
Human driver monitors the driving environment						
0	No automation	The full-time performance by the human driver of all aspects of the dynamic driving task, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver assistance	The driving mode-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the human driver perform all remaining aspects of the dynamic driving task	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial automation	The driving mode-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the human driver perform all remaining aspects of the dynamic driving task	System	Human driver	Human driver	Some driving mdoes
Automated driving system ("system") monitors the driving environment						
3	Conditional automation	The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task with the expectation that the human driver will respond appropriately to a request to intervene	System	System	Human driver	Some driving modes
4	High automation	The driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task, even if a human driver does not respond appropriately to a request to intervene	System	System	System	Some driving modes
5	Full automation	The full-time performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed by a human driver	System	System	System	All driving mdoes

2.4.1 Functional components

Autonomous driving can be divided into three major categories according to [40, 41]: algorithms, client systems, and cloud platform. The algorithms include perception, planning, and control; the client systems consist of hardware and software; and the cloud platform is comprised of high-definition (HD) map, simulation, model training, and data storage. The client system and cloud platform are out of our research scope, hence, we won't discuss them here. Besides, the traditional autonomous driving system uses a hierarchy of scenarios and rules that decomposes the system into different components upon their functionalities and further optimize them from high-level to low-level individually. While the traditional approach is currently the mainstream approach due to safety concerns, learning-based approaches such as reinforcement learning [42, 43], imitation learning [44] and deep learning [45] have been attracted increased interest because of their abilities to solve multiple or even all optimization problems simultaneously and to utilize the historical driving data more efficiently compared to traditional approaches. In this thesis, we will mainly focus on solving planning and control problems via reinforcement learning approaches. However, for a better understanding of the entire autonomous driving system, we will briefly introduce the traditional architecture and comment on the responsibilities of each component in this section.

Perception

The goal of perception module is to extract meaningful information from sensor data, to localize the vehicle, and to build a reliable and detailed representation of the surrounding environment, whereby multiple sensors must be combined to provide a composite information of the world. In autonomous driving, safety is the primary concern, the perception modules must be capable to detect static objects like road surface, surrounding obstacles, traffic signs and lights, to track moving objects such as pedestrians, cyclists and other vehicles, etc.. Therefore, the perception modules, upon its functionality, can be divided into the following three parts:

Sensing
The sensing part perceives internal and external states of the environment. The internal states can be obtained via GPS or IMU system, which provides accurate and real-time updates for both inertial and global position of the vehicle, and odometry. Stereo/- cameras capture both two-dimensional (2D) and three-dimensional (3D) images that can be used for object recognition and tracking tasks. LiDAR/RADAR system can generate point clouds of the environment, and is used for mapping, localization and obstacle avoidance.

Sensor fusion
After collection of the multiple sources of sensory data, the sensor fusion part combines them to construct a representation of the state of the environment. Besides, the sensor fusion part can also be used to perform association and dissociation, which refers to conclude correlating data that refers to the same object and eliminate the data that is most likely to be noise. Finally, the sensor fusion can be used to localize the vehicle with respect to a global map via map

matching algorithms.

Object detection, segmentation and tracking
The goal of this part is to first detect various traffic participants such as pedestrians, cars, cyclists, and obstacles such as lane drivers, then estimate object states such as location, speed, and acceleration over time, and finally track the traffic users and predict their future states and trajectories. The segmentation is an optional enhancement of object detection that parses camera images into meaningful segments so as to give the vehicle an understanding of the environment.

Planning and control

Based on the understanding of the surrounding environment from the perception module, the planning and control module selects a coarse-to-fine route from the road network and executes the corresponding motion to achieve the destination. To be specific, the route planning part firstly selects a route through the road network. Then, the behavioral decision making part chooses an appropriate driving behavior based on the perceived behavior of other traffic participants, road conditions. After that, the motion planning part generates a local trajectory based on higher navigation command. Finally, the vehicle control part executes the planned motion and correct tracking errors. In this section, we will briefly introduce the responsibilities of each of these components and some common methods to solve them.

Route planning
The highest level route planning, as an essential part for both human-driven and self-driving cars, selects a route through the road network based on the vehicle's current position and its destination. The problem can be formulated as finding a minimum-cost path through the road network by selecting lane sequences from the road network graph [46]. The route through network graph can be computed by Dijkstra [47], a typical shortest path algorithm in graph theory, and A* [48], a heuristic based search algorithm. However, in reality, the road network contains millions of elementary connections, which makes both of the algorithms impractical. In order to solve the route planning problems for complex road network efficiently, a wide range of methods have been developed, [49] gave a detailed comparison of practical algorithms upon the requirements of preprocessing effort, query time, and space usage.

Behavior decision making
The behavior decision making is responsible for determining a driving behavior by taking the behavior of other traffic participants, traffic rules, and road conditions into account. Specifically, this part must be able to decide the driving command including cruise-in-lane, follow, change-lane, turn, and stop. Due to the diversity of the local traffic laws, it is hard to formulate the behavior decision making into a uniformed model. Therefore, it is suitable to design a rule-based system that decomposes the surrounding environment into different layers of scenarios, where each scenario focuses on its own domain logic and makes decisions for objects within itself. After all, layers have finished decision making, the final layer will merge the individual decisions and solves conflicts among them. In the DARPA challenge, CMU self-driving car "Boss" [50] and Stanford's self-driving system "Junior" [51] used the rule-based system to compute the behaviors of the self-driving car. Another possible solution for behavior decision making is to use MDP or POMDP in modeling the driving behavior by designing an appropriate reward function, [42] developed an MDP based approach for the behavioral decision, [52] assumed partial observability of pedestrians' intentions and solve the behavior decision-making problem via POMDP methods.

Motion planning
The task of motion planning is to dynamically compute a path or trajectory from the initial configuration to the goal configuration that avoids obstacles and satisfies certain constraints, which could be comfortable for passengers, kinematic constraints, or time constraints [53]. A variety of approaches have been studied, which fall into the framework of path planning or trajectory planning. The former aims to find a geometric path without the concern for dynamics, or constraints on the motion, while the latter tries to find a trajectory, i.e. a path that includes the time along the path, with the constraints on dynamics and several aspects of trajectory performance [54]. An optimal path planning problem can be numerically solved using variational methods, graph search methods, incremental search techniques, etc., which can also be used to solve trajectory planning problems by converting the problem into path planning in a configuration space with an added time-dimension [46].

Vehicle control
Given the motion plan from the last layer, the vehicle control executes the motion for steering, throttle, and brake control, and minimizes the tracking error with respect to the corresponding path or trajectory based on the feedback [46]. Since there is no essential difference between vehicle control with general robotic control, the traditional methods of which can be applied to vehicle control. One of the most widely used lower level control methods is proportional-integral-

derivative (PID) feedback controller [55] due to its simplicity and flexibility. Depending on the different form of the planned motion from the previous layer, the PID controller could track longitudinal/lateral difference along a path or trajectory, angle/curvature difference at the trajectory points or a mixture of vehicle pose variables [41].

2.4.2 Challenges

We have introduced the structure of a traditional autonomous driving system and explained the responsibilities of each component. We would like to discuss several key challenges for some of the components. The first challenge is that when faced with extreme weather conditions, the system must be able to make decisions under uncertainty. Even with good weather conditions, objects are often partially observable due to the noisy sensory data or fully occluded when the view is blocked by other objects, which could lead to failures of object detection or even traffic accidents. Furthermore, for pedestrians and cyclists, its hard to predict their intention, the uncertainty of other traffic participants could also lead to failure prediction or even collision. The second challenge is that the system must be fast enough to process and analyze the sensor data and make decisions and execute actions based on the observations. For a system with learning-based approaches such as reinforcement learning, the processing and decision making might be faster than the traditional system. In addition, the historical data is utilized more efficiently than the traditional system. But the uncertainty of the environment and other traffic participants remains a very challenging problem for reinforcement learning. Besides, how to design the reward function is another challenge. The reward function needs to consider factors such as reaching the destination within a limited time, avoiding collisions, comfortable for passengers, etc.

3 Guided compatible policy search

3.1 Introduction

In the preceding chapter, we talked about several policy search methods that can be used for solving POMDP problems by incorporating truncated historical observation-action pairs into current state-action representations. However, model-free policy search methods require more training data and long training time compared to model-based methods. Therefore, it is difficult for model-free policy search methods to solve challenging partially observed tasks where the agent can only perceive partial observations of the environment. Guided policy search (GPS)[2] and its variants [3, 4] provide us an insight of transforming RL into supervised learning to solve POMDP problems.

In this chapter, We propose a novel method called guided-COPOS that combines a model-free guided framework with COPOS for solving the POMDP problems. Specifically, we use two agents to generate samples during the training phase, one of the agents selects actions based on partial observations, while the other agent can take actions by perceiving the full state observations. We decompose the policy optimization into two steps, a constrained optimization step embedded in COPOS such that both policies update toward the direction of achieving better long-time rewards, and an unconstrained optimization step by minimizing the KL divergence between the two policies such that they converge to the same behavior. Different from traditional GPS methods, our approach does not require the knowledge of the system model or the need to learn the model. In order to evaluate our method, we modify a continuous control task from OpenAI Gym—LunarLanderContinuous-v2 into partially observable environment—LunarLander-POMDP. We first run several state-of-the-art policy search methods (TRPO, PPO, and COPOS) in LunarLanderContinuous-v2 as baselines. Then, we run them as well as our guided-COPOS in LunarLander-POMDP showing the comparison of the learning performance and the testing results. As a result, we show that guided-COPOS outperforms all other methods in the challenging POMDP task.

3.2 Preliminaries

Recall that the goal of policy gradient methods is to find an optimal policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ by repeatedly interacting with the environment and updating its parameters in the direction of maximizing certain measurement of long-term cumulative rewards. The policy gradient is derived from the objective denoted as $J(\pi_{\boldsymbol{\theta}})$.

In this thesis, we focus our research on continuous environments. We determine the stochastic policy as a Gaussian policy, as it is the standard for continuous control tasks. We could either use the Gaussian policy with the constant variance where the mean is parameterized by function approximation, or the variance and the mean that both are parameterized by function approximation. For simplicity, we use the policy with constant covariance Σ where the state-dependent mean is parameterized by a neural network features $\phi_i(\mathbf{s})$ and a mixing matrix $\mathbf{K}$. Together, the stochastic policy can be written according to [1] as

$$\pi(\mathbf{a}|\mathbf{s}) = \mathcal{N}\left(\mathbf{a}\middle|\sum_i \varphi_i(\mathbf{s})\mathbf{k}_i, \boldsymbol{\Sigma}\right), \tag{3.1}$$

where $\mathbf{k}_i$ represents the weights of the last layer of the neural network, $\varphi_i(\mathbf{s})$ represents the non-linear features of the neural network, i.e. all weights and biases of the previous layers. For an agent following policy π in finite-horizon MDP environment, we use the standard definitions of state value function $V^\pi(\mathbf{s})$ from Equation 2.1, state-action value function $Q^\pi(\mathbf{s},\mathbf{a})$ from Equation 2.2. The advantage function $A^\pi(\mathbf{s},\mathbf{a}) = Q^\pi(\mathbf{s},\mathbf{a}) - V^\pi(\mathbf{s})$ can be estimated through the truncated generalized advantage estimation (GAE) scheme [21] with horizon T

$$\begin{gathered}\hat{A}^{\text{GAE}}_\pi(\mathbf{s}_t,\mathbf{a}_t) = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \\ \text{where} \quad \delta_t = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)\end{gathered} \tag{3.2}$$

Applying the Gaussian policy and the GAE into the policy gradient of Equation 2.6. The policy gradient can be approximated as

$$\nabla_{\boldsymbol{\theta}}\hat{J}^{\text{PG}}(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t\left[\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\hat{A}^{\text{GAE}}_{\pi_{\text{old}}}\right], \tag{3.3}$$

TRPO proposed to use a "surrogate" objective in Equation 2.8, based on importance sampling estimator, where the sampling distribution is the old policy distribution. If we differentiate the policy graident objective with old parameters, the resulting objective is same as the "surrogate" objective in Equation 2.8

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \hat{J}^{\text{PG}}(\pi_{\boldsymbol{\theta}}) &= \hat{\mathbb{E}}_t \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) \hat{A}^{\text{GAE}}_{\pi_{\text{old}}} \right] \\ &= \hat{\mathbb{E}}_t \left[\left(\frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}^{\text{GAE}}_{\pi_{\text{old}}} \right) \right] \\ &= \nabla_{\boldsymbol{\theta}} \hat{J}^{\text{surr}}(\pi_{\boldsymbol{\theta}}).\end{aligned} \tag{3.4}$$

As we mentioned in Section 2.3, COPOS solves the contrained optimization problem in combination with compatible function approximation. The compatible function approximation is actually the approximation of the advantage function

$$A^{\pi}(\mathbf{s}, \mathbf{a}) \approx \hat{F}^{\pi}_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) = \phi(\mathbf{s}, \mathbf{a})^T \mathbf{w}, \tag{3.5}$$

where $\boldsymbol{\phi}(\mathbf{s})$ is the compatible feature basis and $\mathbf{w}$ is the compatible weights. In order to embed COPOS in our guided policy search framework, we also need to compute $\boldsymbol{\phi}(\mathbf{s})$ and $\mathbf{w}$.

Applying "surrogate" objective into Equation 2.13, one can compute the approximated solution of compatible weghts $\mathbf{w}$ via conjugate gradient. Applying the property in Equation 2.4 into the Gaussian policy of Equation 3.1, one can obtain the compatible features regarding to log-linear parameters $\boldsymbol{\theta}$ and non-log-linear parameters $\boldsymbol{\beta}$ accordingly

$$\boldsymbol{\phi}_{\boldsymbol{\theta}} = \left[-0.5\mathbf{a}\mathbf{a}^T, \mathbf{a}\boldsymbol{\varphi}(\mathbf{s})^T \right], \quad \boldsymbol{\phi}_{\boldsymbol{\beta}} = \frac{\partial}{\partial \boldsymbol{\beta}} \boldsymbol{\varphi}(\mathbf{s})^T \mathbf{U}\mathbf{a},$$

where $\mathbf{U} = \mathbf{K}^T \boldsymbol{\Sigma}^{-1}$. Note that we compute the gradient with respect to log-linear paramters and non-log-linear parameters separately since COPOS gives us different update rule for them. We will denote all parameters of the neural network as $\boldsymbol{\Theta} = (\boldsymbol{\theta}, \boldsymbol{\beta})$, and all compatible feature basis as $\boldsymbol{\phi}_{\boldsymbol{\Theta}} = (\boldsymbol{\phi}_{\boldsymbol{\theta}}, \boldsymbol{\phi}_{\boldsymbol{\beta}})$.

For a POMDP environment, the projection from observation space to state space is not unique, using single observation directly might cause the agent suffering from perceptual aliasing citearticle. In this chapter, we use the finite size history method that using the last N observations and actions to represent the real states $\mathbf{h}_t = (\mathbf{o}_t, \mathbf{a}_{t-1}, ..., \mathbf{a}_{t-N}, \mathbf{o}_{t-N})$.

3.3 Guided-COPOS

In this section, we first introduce the framework of guided-COPOS, where we use two agents to generate samples during the training phase, one of the agents selects actions based on partial observations, while the other agent can select actions by perceiving the full state observations. Secondly, we describe how to incorporate the samples generated from different policies into policy search. Thirdly, we introduce two optimization steps in guided-COPOS, a constrained policy optimization step that optimizes the two policies independently, and an unconstrained policy optimization step that forces the final control agent to mimic the behavior of the guiding agent. Finally, we present the practical algorithm for continuous action case.

3.3.1 General framework

Similar to MPC-GPS [4], our guided-COPOS also consists of two phases—a training phase and a test phase, and two agents—a guiding agent and a final control agent. The interaction loop of guided-COPOS is illustrated in Figure 3.1. In the training phase, the final control agent can only receive partial observations, while the guiding agent, as the "teacher", is able to perceive both partial observations and full state observations. The final control policy $\pi_f(\mathbf{a}_t|\mathbf{h}_t)$ and the guiding policy $\pi_g(\mathbf{a}_t|\mathbf{h}_t, \mathbf{s}_t)$ are parameterized by Equation 3.1. Both policies are trained to maximize long-time rewards. Moreover, the final control policy is also trained in a supervised way to mimic the behavior of the guiding policy such that it can learn an internal representation of the full states of the environment. However, supervised learning will not, in general, produce a policy with good long-horizon performance [56]. A small error on the part of the final control policy may cumulatively lead to completely different trajectories where the control agent might never see before. To avoid this issue, parts of the training data must come from the final control policy [57]. We achieve this by using π_g and π_f alternately generating the same trajectory. The interaction loop for training phase is illustrated in Figure 3.1 (left). To be specific, the guiding agent perceiving the current state $\mathbf{s}_t$ and the observation $\mathbf{o}_t$, will take the action $\mathbf{a}^g_t$. The environment changes its state to $\mathbf{s}_{t+1}$ and the corresponding observation to $\mathbf{o}_{t+1}$ based on the chosen action.

Then, the final control agent responses the observation with action $\mathbf{a}_{t+1}$. Together, the two agents interact with the same environment repeatedly and iteratively to generate samples, which are recorded externally and denoted as sample set $\mathcal{D} = (\mathbf{a}_0^g, \mathbf{s}_0^g, r_0^g, \mathbf{o}_0^g, \mathbf{a}_1^f, \mathbf{s}_1^f, r_1^f, \mathbf{o}_1^f, \mathbf{a}_2^g, ..., \mathbf{a}_T^g, \mathbf{s}_T^g, r_T^g, \mathbf{o}_T^g)$ with the total length of horizon T. For each update, the guiding policy is trained with the history of observation-action pairs as well as the current state from $\mathcal{D}$ to maximize the certain measurement of long-time reward, denoted objective $J(\pi_g)$. The final control agent updates its policy with the history of observation-action pairs from $\mathcal{D}$ by likewise maximizing certain measurements of long-time reward, denoted as objective $J(\pi_f)$, and followed by a supervised learning procedure to minimize the differences of the two policies with respect to the parameters of final control policy. At test time, shown in Figure 3.1 (right), the final control agent interacts with the environment solely where only observations are available.

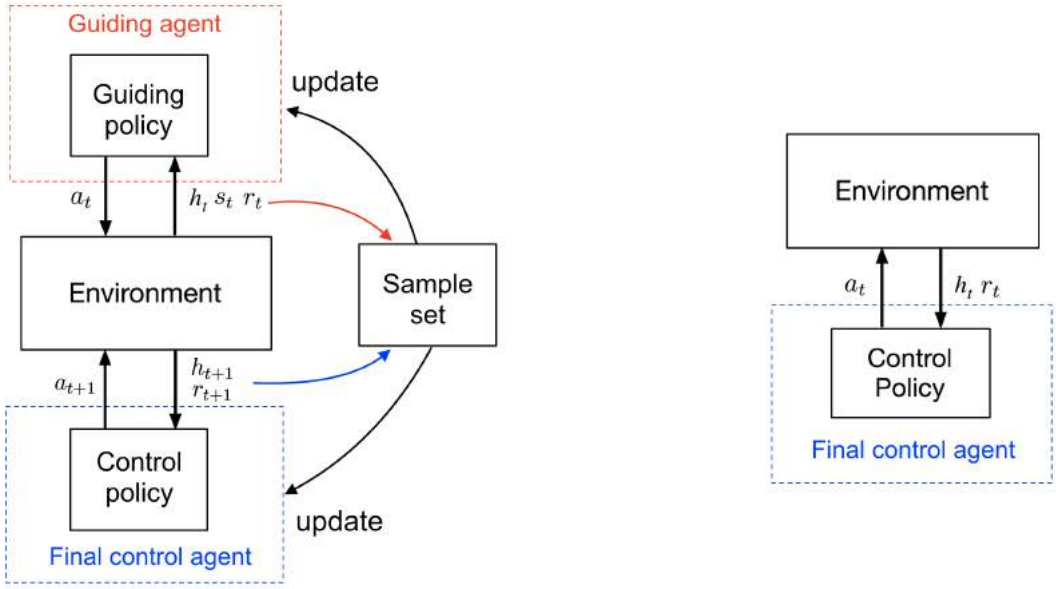

Figure 3.1: Interaction loop of training phase (left) and testing phase(right).

3.3.2 Constrained policy optimization

We begin with defining the objective for calculating the compatible weights $\mathbf{w}$. As we mentioned in Section 3.2, the "surrogate" objective is the same as the policy gradient objective when we take differentiate at the $\boldsymbol{\theta}_{\text{old}}$. We similarly define the "surrogate" objective for guiding agent and final control agent as

$$\begin{aligned} \text{Guiding agent:} \quad J(\pi_{\boldsymbol{\theta}_g}) &= \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_g}(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_t)}{\pi_{\boldsymbol{\theta}_{g_{\text{old}}}}(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_t)} \hat{A}^{\text{GAE}}_{\pi_{g_{\text{old}}}} \right], \\ \text{Final control agent:} \quad J(\pi_{\boldsymbol{\theta}_f}) &= \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_f}(\mathbf{a}_t|\mathbf{h}_t)}{\pi_{\boldsymbol{\theta}_{f_{\text{old}}}}(\mathbf{a}_t|\mathbf{h}_t)} \hat{A}^{\text{GAE}}_{\pi_{f_{\text{old}}}} \right]. \end{aligned} \tag{3.6}$$

However, the sampling distribution is not exactly generated from the old policy, but rather a distribution with mixture samples generated by the old guiding policy $\pi_g(\mathbf{a}_t|\mathbf{h}_t, \mathbf{s}_t)$ and the old control policy $\pi_f(\mathbf{a}_t|\mathbf{h}_t)$. Such samples mixture way could lead to an extremely large or extremely small importance sampling ratio (closer to zero). In order to use samples that generated from different policy distributions, we have to ensure that the update step of the new policy is inside the trust region. Meanwhile, the two policy distributions should be identical, which means the differences between the two policies have to be minimized for each update. We could set the second term as an additional constraint to the final control policy, which will however lead to different update rule as COPOS. For simplicity, we decompose the optimization problem into a constrained optimization step and an unconstrained optimization step. We will discuss the second step in the following section.

Now, we can use the objective of Equation 3.6 to compute the compatible weights $\mathbf{w}$ based on the sampled trajectories set D. In order to comupte the GAE, we need to estimate the state-value function $V(\mathbf{s}_t)$. We adopt an MLP network with

parameters $\boldsymbol{\phi}$ that ouputs the estimate $V_{\boldsymbol{\phi}}(\mathbf{s}_t)$. We use $V^{\text{targ}}(\mathbf{s}_t) = \sum_{l=1}^{T} \gamma^l r(\mathbf{s}_{t+l})$ as the target in a mean squared error (MSE) loss and update the parameters $\boldsymbol{\phi}$ of value network using SGD

$$\min_{\boldsymbol{\phi}} L = \hat{\mathbb{E}}_t[(V^{\text{targ}}(\mathbf{s}_t) - V_{\boldsymbol{\phi}}(\mathbf{s}_t))^2]. \tag{3.7}$$

Applying the objective of Equaion 3.6 into Equation 2.13, one can obtain the optimal solution of compatible weights $\mathbf{w}_g$ and $\mathbf{w}_f$ as

$$\mathbf{w}_g^* = \mathbf{F}^{-1} \nabla_{\boldsymbol{\theta}_g} J(\pi_{\boldsymbol{\theta}_g}), \quad \mathbf{w}_f^* = \mathbf{F}^{-1} \nabla_{\boldsymbol{\theta}_f} J(\pi_{\boldsymbol{\theta}_f}). \tag{3.8}$$

We compute the weights approximately by using conjugate gradient followed by a line search. Then, the approximated advantage function $\tilde{F}^{\pi_{\boldsymbol{\Theta}}}$ can be obtained as

$$\tilde{F}^{\pi_{\boldsymbol{\theta}_g}} = \boldsymbol{\phi}_{\Theta_g}^T \mathbf{w}_g, \quad \tilde{F}^{\pi_{\boldsymbol{\theta}_f}} = \boldsymbol{\phi}_{\Theta_f}^T \mathbf{w}_f. \tag{3.9}$$

In addition, Actor-critic with experience replay (ACER) [58] indicates a variant way to estimate the Q-value function by Retrace [59], which can significantly reduce bias in the estimation of the policy gradient by truncating the importance sampling ratio at a constant. We also adopt Retrace estimation method, but to estimate the state-value function V, then use it to compute the advantage function, which can be expressed recursively as

$$\begin{aligned} \hat{A}_t^{\text{ret}} &= \delta_t^{\text{ret}} + (\gamma\lambda)\delta_{t+1}^{\text{ret}} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1}^{\text{ret}}, \\ \delta_t^{\text{ret}} &= r_t + \gamma\bar{\rho}_t V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t), \end{aligned} \tag{3.10}$$

where $\bar{\rho_t} = \min(c, \rho_t), \rho_t = \pi(\mathbf{a}_t|\mathbf{s}_t)/\pi_{\text{old}}(\mathbf{a}_t|\mathbf{s}_t)$ is the truncated importance weight, V is the current state-value estimate of V^π. The resulting objective can be obtained by replacing $\hat{A}^{\text{GAE}}$ in Equation 3.6 into $\hat{A}^{\text{ret}}$. We will compare the peformance of two different advantage function estimation methods in our experiments in Section 3.5.2.

As introduced in Section 2.3, COPOS formulated the constraint optimization problem as finding the optimal policy which maximizes the expected reward while satisfying the KL constraint and the entropy bound. In guided-COPOS, we similarly bound the difference between the new and old policy into a trust region to prevent unstable update. We also adopt the entropy bound as hard contraint to balance the trade-off between exploration and exploitation.

Applying the compatible weights to Equation 3.5, one can obtain approximated advantage function F^{π_g} and F^{π_f}. Then we can reformulate the objective of the guiding agent based on COPOS as

$$\begin{aligned} \text{argmax}_{\pi_{\theta_g}} \quad & \mathbb{E}\left[\int \pi_g(\mathbf{a}|\mathbf{h},\mathbf{s})\tilde{F}^{\pi_{g_{\text{old}}}}(\mathbf{h},\mathbf{s},\mathbf{a})d\mathbf{a}\right] \\ s.t. \quad & \mathbb{E}\left[\text{KL}(\pi_g(\cdot|\mathbf{h},\mathbf{s})||\pi_{g_{\text{old}}}(\cdot|\mathbf{h},\mathbf{s}))\right] \leq \epsilon, \\ & \mathbb{E}\left[\text{H}(\pi_{g_{\text{old}}}(\cdot|\mathbf{h},\mathbf{s})) - \text{H}(\pi_g(\cdot|\mathbf{h},\mathbf{s}))\right] \leq \beta \end{aligned}, \tag{3.11}$$

the objective of final control agent can be obatined as

$$\begin{aligned} \text{argmax}_{\pi_{\theta_f}} \quad & \mathbb{E}\left[\int \pi_f(\mathbf{a}|\mathbf{h})\tilde{F}^{\pi_{f_{\text{old}}}}(\mathbf{h},\mathbf{a})d\mathbf{a}\right] \\ s.t. \quad & \mathbb{E}\left[\text{KL}(\pi_f(\cdot|\mathbf{h})||\pi_{f_{\text{old}}}(\cdot|\mathbf{h}))\right] \leq \epsilon, \\ & \mathbb{E}\left[\text{H}(\pi_{f_{\text{old}}}(\cdot|\mathbf{h})) - \text{H}(\pi_f(\cdot|\mathbf{h}))\right] \leq \beta \end{aligned} \tag{3.12}$$

Now we can update the log-linear parameters and non-log-linear parameters of both policies based on the update rule from COPOS as

$$\begin{aligned} \boldsymbol{\theta}^g &= \frac{\eta\theta_{old}^g + w_{\boldsymbol{\theta}}^g}{\eta^g + \omega^g} \quad \beta^g = \beta_{\text{old}}^g + s^g \frac{w_{\boldsymbol{\beta}}^g}{\eta^g}, \\ \boldsymbol{\theta}^f &= \frac{\eta\theta_{old}^f + w_{\boldsymbol{\theta}}^f}{\eta^f + \omega^f} \quad \beta^f = \beta_{old}^f + s^f \frac{w_{\boldsymbol{\beta}}^f}{\eta^f}, \end{aligned} \tag{3.13}$$

where η and ω are Lagrangien multipliers and can be obtained by using, for example, Broyden-Fletcher-Goldfarb-Shannon (BFGS) method [60].

3.3.3 Uncontrained policy optimization

After updating the guiding policy and the control policy individually, we perform an auxiliary update step to supervise the control policy updating towards to the guiding policy, which has more knowledge about the environment. As we have explained in the preceding section, in order to use samples generated from different policies, the two policy distribution should be similar to each other. Therefore, this auxiliary supervised learning update step is necessary for a stable update as it prevents the final control policy $\pi_f(\mathbf{a}_t|\mathbf{h}_t, \mathbf{s}_t)$ and the guided policy $\pi_g(\mathbf{a}_t|\mathbf{h}_t, \mathbf{s}_t)$ being too far from each other.

We achieve the supervised learning process by minimizing the KL divergence of the two policy distributions with respect to control policy's parameters $\boldsymbol{\theta}_f$, and perform the update to the final control policy

$$\min_{\theta_f} \quad \mathcal{L}(\pi_f) = \mathbb{E}[KL(\pi_f(\cdot|h)||\pi_g(\cdot|h)] \tag{3.14}$$

We use stochastic gradient descent (SGD) to minimize the KL divergence with respect to the parameters of the control policy based on the sample set D and the updated policy from the preceding section. The resulting update rule becomes

$$\Theta^f_{new} = \Theta^f + \alpha \nabla_{\Theta^f} D_{KL}, \quad \text{where} \quad \Theta^f = (\theta^f, \beta^f) \tag{3.15}$$

Applying this auxiliary update step will lead to an approximated update solution for the final control policy. One might wonder why we don't have such KL loss for guiding policy, it is because the guiding agent is aware of both real states and partial observations of the environment and will make the decision based on them. In fact, we find that the resulting algorithm is able to achieve good empirical results and is robust to random seeds in variety POMDP tasks.

3.3.4 Practical algorithm

In this section, we will present a practical algorithm for guided-COPOS. We use an actor-critic training method, where independent MLP is used to represent the Gaussian policy and value function for each agent, and denoted as $\boldsymbol{\Theta}_{\pi_g}$, $\boldsymbol{\Theta}_{v_g}$, $\boldsymbol{\Theta}_{\pi_f}$, $\boldsymbol{\Theta}_{v_f}$ respectively. After collecting trajectories with horizon T generated by $\pi_{\boldsymbol{\Theta}_g}$ and $\pi_{\boldsymbol{\Theta}_f}$, the advantage function in Equation 3.2 is updated using GAE. As a comparison, we also calculate the advantage function using Retrace method of Equation 3.10. Then, the compatible weights can be computed via Equation 3.8, which can be solved by the conjugate gradient followed by a line search.

The update way for guiding policy is the same as COPOS. The log-linear parameters $\boldsymbol{\theta}_g$ and non-linear parameters $\boldsymbol{\beta}_g$ are updated via Equation 3.13. The value network for the guiding agent is updated using SGD to minimize 3.14. The update rule for final control policy is a little different from COPOS, where an auxiliary KL loss has to be minimized so as to ensure that the final control policy distribution is identical to the guiding policy distribution. We split the update process into two steps, the first step is the same as COPOS and the second step is to minimize the KL divergence of two policy distributions with respect to $\boldsymbol{\Theta}_f$. The value network parameters for final control agent is updated using SGD to minizine the target error of Equation 3.7.

3.4 Connections with previous algorithms

So far, one might wonder why do we need guided-COPOS if we already have COPOS. It is because we aim to solve POMDP problems which are extremely hard for any model-free approaches due to the lack of surrounding information. However, training COPOS in a supervision way affords the advantage that the control agent can learn an internal representation of the real states with the supervision of the guiding policy, which in turns, helps to improve the performance significantly at test time. One might wonder why don't we simply train the final control policy with supervised learning. It is because training the final control policy with supervised learning solely cannot provide a good long-horizon performance. Since a small error of the learned control policy at the beginning of the episode may lead to a completely different trajectory that the agent never seen before. Therefore, using the long-time cumulative reward objective for final control policy will ensure a good long-horizon performance.

COPOS[1]: The constrained optimization part of our approach is based on COPOS. In other words, we adapt COPOS within our guided policy search framework by first mixing samples generated from the guiding policy and the final control policy. Then, we minimize the KL divergence of the guiding policy distribution and the final control distribution with respect to the parameters of the control policy. With the help of the supervision from the guiding policy, we show that

Algorithm 1 Guided COPOS

Initialize final control policy network $\pi_{(\boldsymbol{\theta}_f,\boldsymbol{\beta}_f)}$ with non-linear parameters $\boldsymbol{\beta}_f$ and linear parameters $\boldsymbol{\theta}_f$ and $\boldsymbol{\Theta}_f = (\boldsymbol{\beta}_f, \boldsymbol{\theta}_f)$, value function network $v_{\boldsymbol{\phi}_f}$ with parameters $\boldsymbol{\phi}_f$
Initialize guiding policy network $\pi_{(\boldsymbol{\theta}_g,\boldsymbol{\beta}_g)}$ and its value function network $v_{\boldsymbol{\phi}_g}$ by copy correspoinding values of parameters in $\pi_{(\boldsymbol{\theta}_f,\boldsymbol{\beta}_f)}$, initialize the rest parameters with zeros
for iteration $i = 0$ to $MaxIteration$ **do**

Run the current guiding policy $\pi_{(\theta_g,\beta_g)}$ and current final control policy $\pi_{(\theta_f,\beta_f)}$ iteratively and collect a set of trajactories $\mathcal{D} = (a_0^g, s_0^g, r_0^g, h_0^g, a_1^f, s_1^f, r_1^f, h_1^f, a_2^g, ...)$ of length T

Build subset $\mathcal{D}_f$ from $\mathcal{D}$ with historical obseration and action pairs, build subset $\mathcal{D}_g$ from $\mathcal{D}$ with real states and historical data

Estimate the advantages $\hat{A}_g$ and $\hat{A}_f$ using GAE or Retrace compute $w_g = (w_{\theta_g}, w_{\beta_g})$ and $w_f = (w_{\theta_f}, w_{\beta_f})$ via conjugate gradient to solve

$$\begin{aligned} w_g &= F_g^{-1}\nabla_{\boldsymbol{\Theta}_g} J(\pi_{\boldsymbol{\Theta}_g}) \quad J(\pi_{\boldsymbol{\Theta}_g}) = \hat{\mathbb{E}}_t\left[\frac{\pi_{\boldsymbol{\Theta}_g}(\mathbf{a}_t|\mathbf{s}_t,\mathbf{h}_t)}{\pi_{\boldsymbol{\Theta}_{g_{\text{old}}}}(\mathbf{a}_t|\mathbf{s}_t,\mathbf{h}_t)}\hat{A}^{\text{GAE}}_{\pi_{g_{\text{old}}}}\right], \\ w_f &= F_f^{-1}\nabla_{\Theta_f} J(\pi_{\boldsymbol{\Theta}_f}) \quad J(\pi_{\boldsymbol{\Theta}_f}) = \hat{\mathbb{E}}_t\left[\frac{\pi_{\boldsymbol{\Theta}_f}(\mathbf{a}_t|\mathbf{h}_t)}{\pi_{\boldsymbol{\Theta}_{f_{\text{old}}}}(\mathbf{a}_t|\mathbf{h}_t)}\hat{A}^{\text{GAE}}_{\pi_{f_{\text{old}}}}\right] \end{aligned} \tag{3.16}$$

Compute $\tilde{F}_{w_g}^{\pi_{gold}}(s,h,a)$ and $\tilde{F}_{w_f}^{\pi_{fold}}(h,a)$ by Equation (3.9) and apply them into Equation (3.11) and (3.12) to compute $\eta^g, \omega^g, \eta^f, \omega^f$

Apply updates for the new policy

$$\begin{aligned} \boldsymbol{\theta}^g_{new} &= \frac{\eta\boldsymbol{\theta}^g_{old} + w^g_{\boldsymbol{\theta}}}{\eta^g + \omega^g} \quad \boldsymbol{\beta}^g_{new} = \boldsymbol{\beta}^g_{old} + s^g\frac{w^g_{\boldsymbol{\beta}}}{\eta^g} \\ \boldsymbol{\theta}^f_{new} &= \frac{\eta\boldsymbol{\theta}^f_{old} + w^f_{\boldsymbol{\theta}}}{\eta^f + \omega^f} \quad \boldsymbol{\beta}^f_{new} = \boldsymbol{\beta}^f_{old} + s^f\frac{w^f_{\boldsymbol{\beta}}}{\eta^f} \end{aligned} \tag{3.17}$$

where s^g and s^f is rescaling factor found by line search

for epoch $n = 0$ to N **do**

Minimize the KL loss (3.14) using SGD optimizer with minibatch size K

Apply updates for the new policy

$$\Theta_{fnew} = \Theta_{fold} + \alpha\nabla_{\Theta_f}\mathcal{L}(\pi_f) \tag{3.18}$$

end for

for epoch $m = 0$ to M **do**

Update value network parameters $\boldsymbol{\phi}_f$ and $\boldsymbol{\phi}_g$ by minimizing target error (3.7) using SGD optimizer with minibatch size K

end for

end for

Return the final control policy π_f

the resulting algorithm is capable to achieve better empirical results.

GPS [2] and GPS with unknown dynamics[3]: GPS and GPS with unknown dynamics are similar approaches, the former requires the knowledge of dynamics while the latter using trajectory-centric RL method to approximate the dynamics. However, an error of the model will lead to failures of the learned control strategies. Compared to their approaches, our guided-COPOS does not require known dynamics or the requirement to approximate it, we learn the policy directly based on samples. Moreover, the objective of our final control policy is trained not only to match the guiding policy distribution but also to maximize the long-time cumulative rewards, which inherently guarantees a good long-horizon performance.

3.5 Experiments

Before we use our proposed guided-COPOS for solving complex autonomous driving tasks, we choose a simpler environment from OpenAI Gym and modify it into a partially observable environment for further test. Then, for comparison, we are mainly interested in model-free RL methods, especially gradient based policy search methods. We choose COPOS, TRPO, and PPO, as introduced in Section 2.3 and compare the learning performance and testing results of the final learned policy. We design our experiments to investigate the following questions:

1. How does guided-COPOS perform compared to other model-free RL methods? We are particularly interested in its comparison with COPOS since our approach adapts COPOS to a guided framework. Does guided-COPOS show better performance at test time?

2. As a comparison of estimating the advantage function by GAE, we alternatively estimate it with Retrace method, however, how does this modification affect the training procedure and the final policy performance?

3.5.1 Lunar lander environment

In the following sections, we will execute all experiments based on LunarLanderContinuous-v2, a moderate difficult RL task from OpenAI Gym [61]. As shown in Figure 3.2, LunarLanderContinuous-v2 is a simulated 2D world environment with height H and width W. The lander agent attempts to land on the landing pad at coordinate (0,0) without collision.

Figure 3.2: LunarLanderContinuous-v2 (left) and our modified LunarLander-POMDP (right) envrionment.

The reward function encourges the lander moving towards the landing pad with less feul spent. The episode finishes if the lander crashes, flies out of screen, or comes to rest. Specifically, the reward function is defined as

$$\begin{aligned} r_t = &- 100\left(\sqrt{x_t^2 + y_t^2} - \sqrt{x_{t-1}^2 + y_{t-1}^2}\right) - 100\left(\sqrt{v_{x_t}^2 + v_{y_t}^2} - \sqrt{v_{x_{t-1}}^2 + v_{y_{t-1}}^2}\right) \\ &- 100\left(|\theta_t| - |\theta_{t-1}|\right) + 10\left(g_{l_t} - g_{l_{t-1}}\right) + 10\left(g_{r_t} - g_{r_{t-1}}\right) - 0.3(p_{m_t} - p_{m_{t-1}}) \\ &- 0.3(p_{s_t} - p_{s_{t-1}}) + 100a_t - 100c_t, \end{aligned} \tag{3.19}$$

where x and y represent the lander poistion, v_x and v_y represent the lander velocity, θ and ω represent the angle and the angular velocity, g_l and g_r represent the ground contact, p_m and p_s represent the main power and side power, a means if the agent turns off the engine (1) or not (0), c means if the agent crashes (1) or not (0). The observation space and

Observation	Acronym	Type	Range
position	x,y	continuous	(-∞,∞)
velocity	v_x, v_y	continuous	(-∞,∞)
angle	θ	continuous	(-∞,∞)
angular velocity	ω	continuous	(-∞,∞)
ground touch	g_l, g_r	discrete	0 or 1

Action	Acronym	Type	Range
main engine	p_m	continuous	(-1,1)
orientation engine	p_s	continuous	(-1,1)

Table 3.1: The observation space (left) and action space (right) of LunarLanderContinuous-v2

the state space is summarized in table 3.1.

Since our ultimate goal is to use guided-COPOS solving a specific autonomous driving task where the sight of the agent car is blocked by surrounding objects temporarily. We therefore modify the lunar lander environment into partially observable environment by adding a "blind" area in the middle of the starting point and the target landing pad, we denote the new environment as LunarLander-POMDP. As illustrated in Figure 3.2, the gray band represents the "blind" area, with height of $H/8$. When the lander agent enters into this area, it cannot perceive any information of the environment but rather an alarm signal a, which equals to -1 when the lander enters into this "dangerous" area and $+1$ when the lander is in the "safe" area. The reward function remains the same as Equation 3.19.

3.5.2 Implementation and results

In the following experiments, we adopt an actor-critic style where two independent neural networks are used to approximate the Gaussian policy and value function. The architecture of the neural network is shown in Figure 2.2, but with 2 hidden layers of 32 units, and tanh nonlinearities, further detailed setup, and parameters will be discussed individually for each experiment.

Comparison among TRPO, PPO, and COPOS in LunarLanderContinuous-v2

We first execute three policy search methods in LunarLanderContinuous-v2—TRPO, PPO, and COPOS, such that we could establish a standard baseline of their learning performance in the MDP lunar lander environment. The input of the two networks are real states of the environment, the output of the policy network is the action that will be executed by the agent, and the output of the value network is the target value that will be used to calculate the advantage function. Note that we only evaluate the policy of COPOS based on Equation 3.1. As for TRPO and PPO, the neural network directly specifies the mean, while the constant diagonal covariance matrix is parameterized with log standard deviations. We perform TRPO and PPO based on the open-source code from the OpenAI baseline [62] and perform COPOS based on [1]. We execute each of the algorithms for 10 seeds, with a total number of 2 million game timesteps for each seed. We focus the comparison on the learning curve and the entropy curve and plot the results across ten runs of each algorithm, as shown in Figure 3.3. Moreover, we also test the final learned policies for 10 random seeds with 20 episodes for each seed and show the averaged total return and the averaged episode length in Table 3.2. All algorithms show promising results, while COPOS outperforms all other methods in MDP LunarLanderContinuous-v2 task. COPOS shows an excellent control of the drop rate of the entropy, which could be beneficial for solving challenging POMDP tasks.

Comparison among TRPO, PPO, COPOS, and guided-COPOS in LunarLander-POMDP

Then, we carry out the three algorithms in our customized LunarLander-POMDP environment, in comparison with our proposed guided-COPOS. The input for TRPO, PPO, and COPOS are truncated historical observation and action pairs of length 4. In guided-COPOS, the input of the control agent is the same historical data as that for other methods, while the input of the guiding agent is the concatenation of the current state and the historical data. The network structure for all algorithms is the same as that in preceding section. We execute each algorithm for 10 seeds, with a total number of 2 million game timesteps for each seed. Figure 3.4 demonstrates the comparison of the learning curve and entropy among all algorithms in LunarLander-POMDP. The results show that our guided-COPOS outperforms all other approaches in the

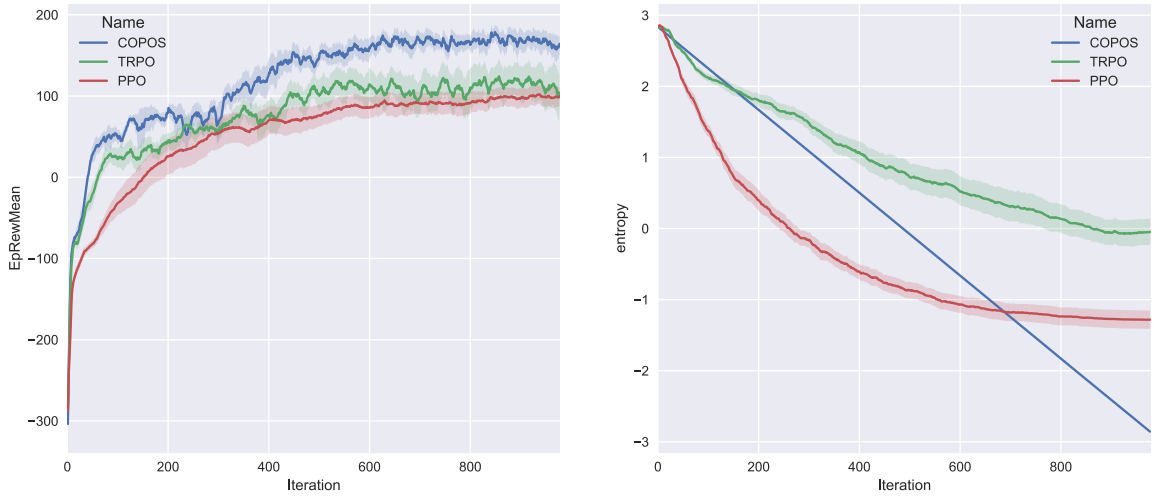

Figure 3.3: Learning curve among TRPO, PPO, and COPOS in LunarLanderContinuous-v2 over 10 seeds. The solid line is the mean, while the shaded area represents the standard deviation. The left is comparison of the averaged cumulative rewards. The right is the comparison of current entropy of the learned policy.

Envrionment	TRPO	PPO	COPOS
Average episode lengh	450.9 ± 206.3	644.4 ± 177.3	$\mathbf{411.9 \pm 238.4}$
Average return	112.3 ± 142.5	95.7 ± 86.3	$\mathbf{136.8 \pm 125.8}$

Table 3.2: Comparison of averaged cumulative reward and averaged episode length of the learned policy after training for 2 million game timsteps. The compariosn is among TRPO, PPO, and COPOS in LunarLanderContinuous-v2 over 10 seeds (same seed for each algorithm), and 20 episodes for each seed.

training phase. If we look at the entropy curve, we can see that the entropy regularization proposed by COPOS helps to avoid premature convergence at the beginning of the training, and forces a convergence after finding a good policy. Since our guided-COPOS is trained with supervised-learning fashion, we also compare the performance of final learned policies over 10 seeds, with 20 episodes for each seed. As we can see from the testing results in Table 3.3, our guided-COPOS also shows promising results at test time.

Envrionment	TRPO	PPO	COPOS	guided-COPOS
Average episode lengh	575.3 ± 304.3	923.1 ± 208.1	603.7 ± 343.1	$\mathbf{253.2 \pm 90.4}$
Average return	76.3 ± 166.4	-83.3 ± 109.4	59 ± 176.9	$\mathbf{175.4 \pm 98.5}$

Table 3.3: Comparison of averaged cumulative reward and averaged episode length of the learned policy after training for 2 million game timsteps. The compariosn is among TRPO, PPO, COPOS, and guided-COPOS in LunarLander-POMDP over 10 seeds (same seed for each algorithm), and 20 episodes for each seed.

Comparison between GAE and Retrace for guided-COPOS in LunarLander-POMDP

Finally, we demonstrate our guided-COPOS with distinct advantage function estimation strategies—generative advantage estimation (GAE) and Retrace method. We adopt the same experimental setup and network structure as guided-COPOS in preceding section. It is worth mentioning that during our experiment, we found that the importance ratio can become large occasionally, causing instability. To further safeguard against unstable update, we also try out a variant combination of Retrace method with the clipped objective introduced in PPO in Section 2.3 for practical implementation. We run guided-COPOS with GAE, Retrace, and Retrace+clip in LunarLander-POMDP for 10 seeds, with a total number of 2 million game timesteps. The comparison of the learning curve and entropy curve are presented in Figure 3.5. We also

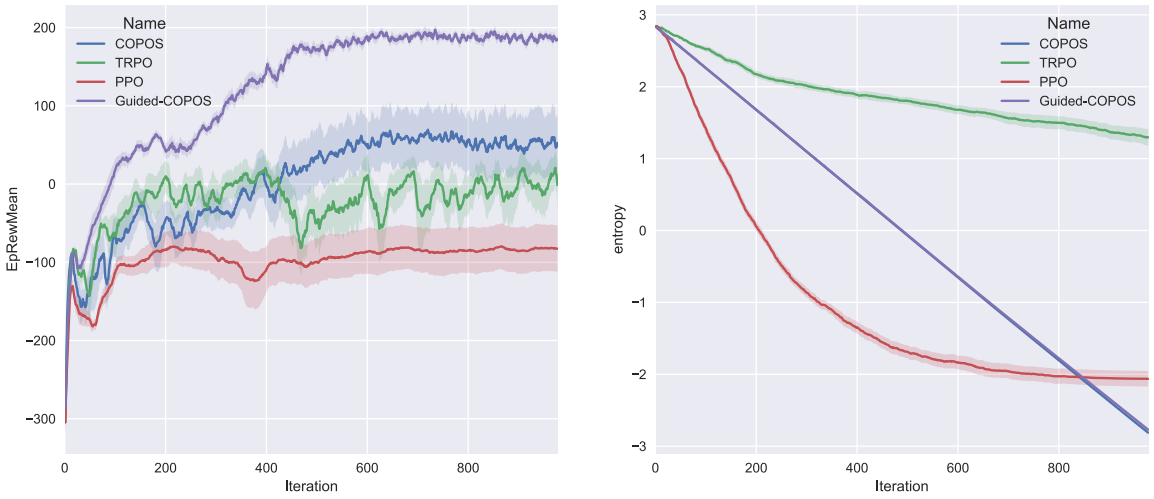

Figure 3.4: Learning curve among TRPO, PPO, COPOS, and guided-COPOS in LunarLander-POMDP over 10 seeds. The solid line is the mean, while the shaded area represents the standard deviation. The left is comparison of the averaged cumulative rewards. The right is the comparison of current entropy of the learned policy. The truncated history length is 4.

test the final learned policies compared for 10 seeds, with 20 episodes for each seed, the results of which are summarized in Table 3.4.

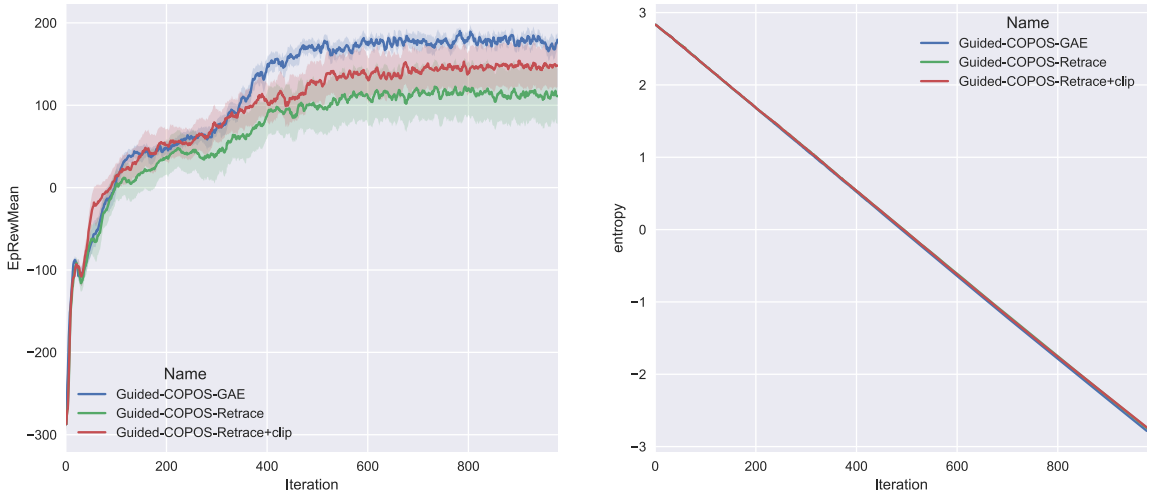

Figure 3.5: Learning curve among guided-COPOS with GAE, Retrace, and Retrace+clip in LunarLander-POMDP over 10 seeds. The solid line is the mean, and the shaded area is the standard deviation. The left is the comparison of the averaged cumulative rewards. The right is the comparison of current entropy of the learned policy. The truncated history length is 4.

In summary, all variations of guided-COPOS show significantly better performance than other policy search methods from the preceding section, the results prove that our implementation of COPOS within the guided scenario can help to accelerate the learning process, and ultimately leading to better performance with the same amount of training data. However, the Retrace method does not help to improve the learning performance or testing results of the final learned policy. The reason is that Retrace assumes the denominator of the importance ratio to be the sampling distribution, which in our case, the denominator distribution is not strictly approximated by sampling distribution but rather a mixture samples generated from the two different policy distributions. Our samples mixture method in combination with GAE shows the best learning performance and the best test results. More importantly, our guided-COPOS shows similar performance over different seeds, which means it is robust to random seeds.

Envrionment	GAE	Retrace	Retrace+clip
Average episode lengh	$\mathbf{253.2 \pm 90.4}$	446.4 ± 275.6	385.43 ± 254.4
Average return	$\mathbf{175.4 \pm 98.5}$	98.1 ± 164.4	133.2 ± 141.7

Table 3.4: Comparison of averaged cumulative reward and averaged episode length of the final learned policy after training for 2 million game timsteps. The compariosn is among guided-COPOS wih GAE, Retrace, and Retrace+clip in LunarLander-POMDP over 10 seeds (same seed for each algorithm), and 20 episodes for each seed.

3.6 Discussion

In this chapter, we proposed a model-free guided policy search method—guided-COPOS for solving partially observable tasks. We suggested a new agent-environment interaction way to generate training samples, where the guiding agent and final control agent iteratively interact with the same environment. The final control agent selects actions based on partial observations, while the guiding agent can take actions by perceiving the full state observations and partial observations. The policies are represented by MLP networks. We use the truncated historical observation-action pairs as input to the final control policy, real states concatenated with the truncated historical data as input to the guiding policy. Then, we decomposed the problem into a constrained policy optimization procedure and an unconstrained policy optimization procedure. The former aims to maximize the long-time cumulative rewards for both agents, which are solved by COPOS in our case. The latter aims to ensure the two policy could converge to the same behavior, which is solved by minimizing the KL divergence of them by SGD with respect to the parameters of the control policy. We evaluated our method in our customized challenging partially observable environment—LunarLander-POMDP, where we have successfully learned the policy and achieved good empirical results, outperforming other well-known policy search methods—TRPO, PPO, and COPOS. Moreover, the idea of our adaptation of COPOS to guided policy search framework ensures a good long-horizon performance, which indicates a new way to solve POMDP tasks, where only model-free RL methods are required.

4 Autonomous driving under uncertainty

4.1 Introduction

Autonomous driving is a challenging problem as it generally consists of several complex subsystems, e.g., a perceptron model and an action-making model. While solving autonomous driving by splitting the system into perception, path planning, and control is still the mainstream method, learning-based approaches are becoming increasingly popular in recent years. The main advantage of learning-based methods is that the methods are capable of optimizing all processing steps simultaneously, which reduces the entire processing and analyzing time. Pioneering work has proven the possibility of solving parts or all of autonomous driving tasks via learning-based approaches [10].

In this chapter, we focus on solving three different autonomous driving tasks with incresing difficulty: a simple goal-directed task where walking pedestrians and moving cars are completely removed from the driving road, a standard goal-directed task where a number of random pedestrians and cars aremoving or driving on the road and true observations of the environment's states are provided to the agent, and a challenging goal-directed POMDP task where the states of the pedestrians are partially observed. We first peform all tasks in a simulated urban driving environment—CARLA. Then, we run COPOS for all tasks as baselines for furture experiments. Finally, we run guided-COPOS in the most challenging partially observable autonomous driving task and compare the result with COPOS.

4.2 Environment setup and implementation

4.2.1 CARLA simulator

CARLA (Car Learning to Act) is an autonomous driving simulator that is implemented as an open-source layer over Unreal Engine 4 (UE4) and provides a challenging realistic urban driving environment. The content provided by CARLA includes a variety of urban layouts, vehicle models, buildings, pedestrians, street signs, etc. [63]. In addition, CARLA also provides different weather conditions, traffic settings, and pedestrian behaviors. For example, Figure 4.1 illustrates several different environmental conditions in CARLA. The simulation supports different sensor suites, including camera raw image, depth map, semantic segmentation image, and Ray-cast based Lidar. Figure 4.2 visualizes a list of sensor data that can be perceived directly by agents. Furthermore, CARLA also defines a built-in function for high-level navigation commands, including go straight, turn left/right, follow the road. Moreover, CARLA provides direct measurements such as forward speed, GPS coordinates, orientation, and detailed data on collisions and other infractions of the agent car. It is also possible to get access to non-player agents' information, such as GPS coordinates, forward speed, velocity, and orientation.

4.2.2 Customized environment

In order to test our algorithm, we developed a task-specific environment based on the driving benchmark and evaluation protocol of [63]. The protocol set up several goal-directed navigation tasks with four increasing difficulty: driving straight, driving through a single turn, navigating through the town taking several turns and navigating through the town with dynamic cars and pedestrians. The tasks are designed for the purpose of general autonomous driving, while in this thesis, we are mainly interested in specific dynamic driving tasks. To be specific, in an urban driving environment, the surrounding objects are often partially observable due to noisy sensor data or even fully occluded when the view of the car is blocked by other objects. For simplicity, we design a goal-directed task where pedestrians try to cross the road on the driving way of the agent car. The behavior of pedestrians cannot be observed when they are too far from the agent car. When they are walking on the shadowed portion of the road (walking behind the parked cars), the agent car cannot observe their behavior in reality. However, in our simulated environment, we can either enable or disable the observability of their behavior in order that the agent car can receive the true measurements in training time.

CARLA provides different urban road conditions, such as T-intersections, four-way intersections, straight line, curve line, etc.. Since the navigation task is not the focus of this thesis, we therefore select a straight road from Town 1, in which

Figure 4.1: A street in Town 1, shown from a third-person view in four weather conditions. Clockwise from top left: clear day, daytime rain, clear sunset, sunset shortly after rain.

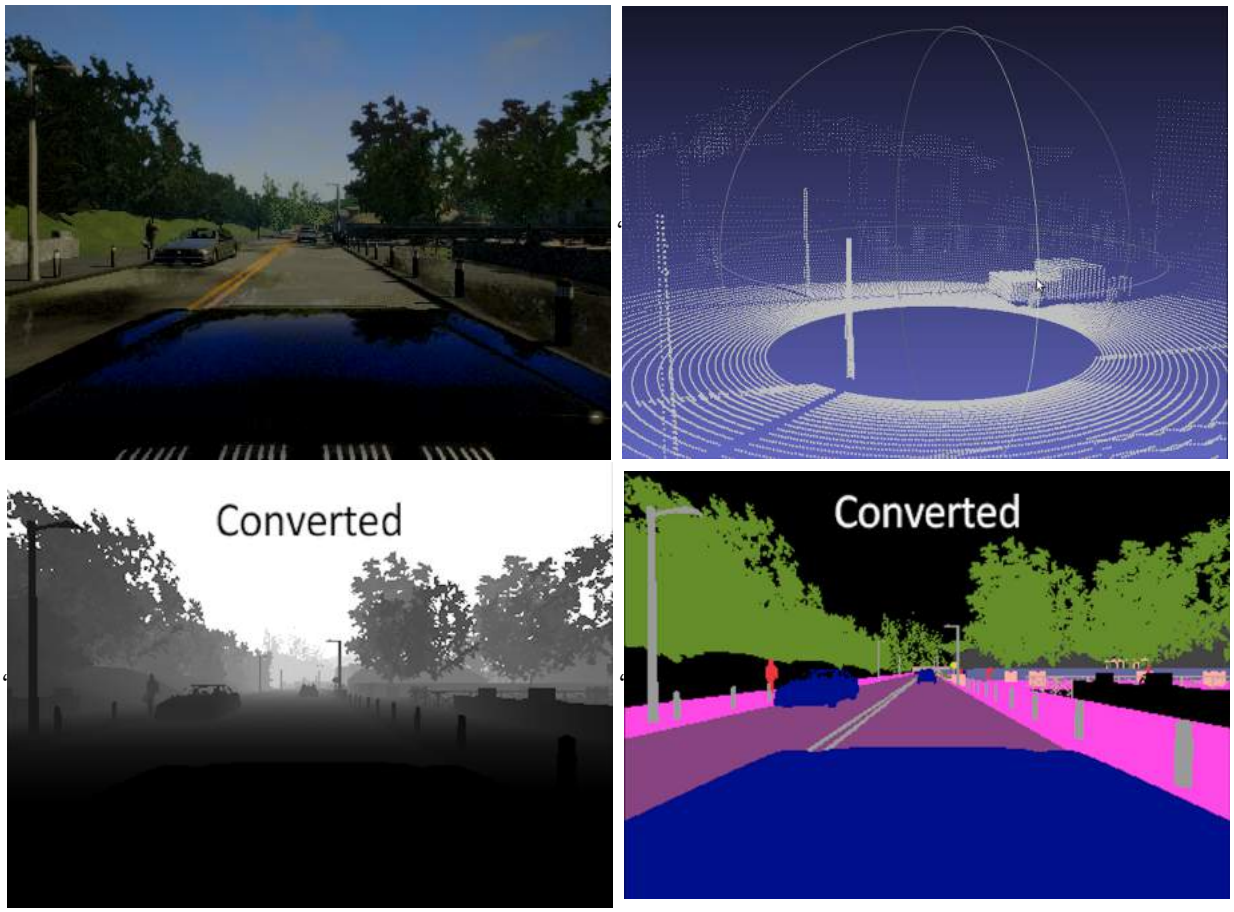

Figure 4.2: The raw sensor data provided by CARLA. Clockwise from top left: camera sence, Ray-cast based Lidar, depth map, semantic segmentation.[64]

the agent car is initialized with location $(299.4, 55.8)$ and has to reach the destination $(207.4, 55.8)$ with a total length of 92 meters. The map and the road conditions are shown in Figure 4.3. We put 5 parked cars on each side of the road in the region $250 < x < 265$, denoted as the "blind" area. We consider the parked cars as static obstacles throughout all experiments. When dynamic traffic users are enabled for this environment, the pedestrians are generated randomly from 10 spawner points, all of which are behind the parked cars on both sides of the road. Meanwhile, the dynamic vehicles are randomly generated from 10 spawner points in the same street and in several nearby streets. The behavior of dynamic pedestrians and cars are determined by in-game AI and can be set via different seeds.

Figure 4.3: The left figure is a street in Town 1, shown from a third-person view, annotated as "START" and "END" in the right figure. The pedestrians are temporarily blocked by the parked car along the sidewalk. When they cross out of the shadowed portion, the agent car can again receives their full state observations.

CARLA provides us with a variety of sensory data and measurements (full state observations of cars and pedestrians). We could directly use the raw sensor data as input in an end-to-end training way. However, such implementation would lead to a long training time and high computational cost due to large amounts of the policy network (CNN). In reality, the measurements can be obtained easily via certain perception technologies in the fully observable environment. Therefore, to simplify the learning tasks, we assume the knowledge of true measurements in the fully observable environment. As for partially observable cases, we hide the measurements completely from the agent when the pedestrians are out of the sight of the car. If we look at the Figure 4.3, when the car drives inside the "blind" area, it can still perceive the full state observations of its own measurements as well as other dynamic vehicles, but cannot observe any pedestrians information when pedestrians walking behind the parked cars. All available observations and actions are summarized in Table 4.1 and Table 4.2. Note that our task is simply a straight driving task where high-level navigation command is not required. We will not use the navigation command in the following experiments. However, in future, it is also interesting to test our algorithm for different driving tasks with the input as raw sensory data (camera images) within the benchmark and evaluation protocol of [63].

4.2.3 Reward function

Next, we define the reward function based on [63], as a weighted sum of five terms: distance traveled towards the goal: $d = \sqrt{(x_{a_t} - x_d)^2 + (y_{a_t} - y_d)^2}$, forward speed v in km/h, collision damage (caused by pedestrians, dynamic cars, and other obstacles) c, intersection with the sidewalk s, and intersection with the opposite lane o. Different from their definition, we weigh the collision impaction caused by pedestrians, cars, other objects separately with weights in descending order. The intuition behind such variation is that, in real life, the last thing we want is to hit pedestrians, then collisions with other vehicles and other objects. In order to achieve a minimal risk condition while trying to reach the target position, we still have to adjust the coefficients for solving our customized challenging partially observable autonomous driving task. We will conduct a series of experiments for fully observable tasks on 3 reward functions as

$$\begin{aligned} r_t^{\text{small}} = {} & 100(d_t - d_{t-1}) + 0.05(v_t - v_{t-1}) - 0.05 * (c_{p_t} - c_{p_{t-1}}) - 0.005 * (c_{c_t} - c_{c_{t-1}}) \\ & - 0.005 * (c_{c_o} - c_{o_{t-1}}) - 5(s_t - s_{t-1}) - 5(o_t - o_{t-1}), \end{aligned} \tag{4.1}$$

Agent	Observation	Acronym	Definition	Range
Agent car	position	x_a, y_a	GPS coodinates	$(-\infty, \infty)$
	forward speed	v_a	forward speed in km/h	$(0, \infty)$
	angle	$\theta_{x_a}, \theta_{y_a}$	driving orientation in Cartesian coordinates	$(-1, 1)$
	collision damage	c_p, c_v, c_o	cumulative collision damage caused by pedestrians, vehicles, other objects	$(0, \infty)$
	intersection	o_s, o_o	intersection with sidewalk, intersection with the opposite lane	$(0, 1)$
Pedestrian	position	x_p, y_p	GPS coodinates	$(-\infty, \infty)$
	forward speed	v_p	forward speed in km/h	$(0, \infty)$
	angle	$\theta_{x_p}, \theta_{y_p}$	walking orientation in Cartesian coordinates	$(-1, 1)$
Other dynamic car	position	x_c,y_c	GPS coodinatese	$(-\infty, \infty)$
	forward speed	v_c	forward speed in km/h	$(0, \infty)$
	angle	$\theta_{x_c}, \theta_{y_c}$	driving orientation in Cartesian coordinates	$(-1, 1)$

Table 4.1: The summary of observation space in CARLA.

Action	Acronym	Type	Range
steering	s	continuous	$(-1,1)$
throttle	t	continuous	$(0,1)$
braking	b	continuous	$(0,1)$

Table 4.2: The summary of action space in CARLA.

$$r_t^{\text{medium}} = 100(d_t - d_{t-1}) + 0.05(v_t - v_{t-1}) - 0.5 * (c_{p_t} - c_{p_{t-1}}) - 0.1 * (c_{c_t} - c_{c_{t-1}}) \\ - 0.1 * (c_{c_o} - c_{o_{t-1}}) - 5(s_t - s_{t-1}) - 5(o_t - o_{t-1}), \tag{4.2}$$

$$r_t^{\text{large}} = 100(d_t - d_{t-1}) + 0.05(v_t - v_{t-1}) - 5 * (c_{p_t} - c_{p_{t-1}}) - (c_{c_t} - c_{c_{t-1}}) \\ - (c_{c_o} - c_{o_{t-1}}) - 5(s_t - s_{t-1}) - 5(o_t - o_{t-1}). \tag{4.3}$$

We will analyze the learning performance and testing results based on the three reward function, and finally draw conclusions of our decision on the coeffients in Section 4.3.2.

4.2.4 Representations of states

In Section 3.5, we used the last N observation-action pairs as a representation of the current state of the agent in the LunarLander-POMDP environment and achieved excellent results in the end. For partially observable autonomous driving tasks, the agent car has to reach the goal without collisions with any static obstacles or any other dynamic traffic users (vehicles and pedestrians). The behavior of dynamic objects changes over time and is unpredictable, while the agent car has to take action with partial observations of pedestrians, which leads to a more challenging autonomous driving task. Consequently, a longer length of history data is required such that we can approximate the current state with more state information. Otherwise, the car might completely lose the information of pedestrians when it drives into the "blind" area. However, the dimension of the observation space in CARLA environment is 55, which is much greater than the state's dimension in LunarLander-POMDP. Feeding a long length of truncated history to the MLP network will require a deep network that has a large number of parameters, and lead to high computational cost. In order to reduce the computational cost while preserving as much information as possible, we alternately represent the history data as

$$\mathbf{h}_t = \{(\mathbf{o}_t), (\mathbf{o}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{o}_{t-2}, \mathbf{a}_{t-2}), (\mathbf{o}_{t-4}, \mathbf{a}_{t-4}), ..., ...(\mathbf{o}_{t-2^T}, \mathbf{a}_{t-2^T})\},$$

where o_t is the observation at the current time, and (o_{t-2^T}, a_{t-2^T}) is the historical observation-action pair that was sampled at time step $t - 2^T$ before current time. Such truncated way provides us the historical obervations and the chosen actions for long time ago within a compact representation, which in turns, reduces the total computational cost.

4.2.5 Network structure

For fully observable autonomous driving tasks where the full state observations are given, we will adopt the same network structure as in Section 3.5. However, for partially observable autonomous driving tasks where the real states are hidden from the agent car, the neural network policy is hard to find the time dependencies from the history data. In order to exploit the temporal dependencies of the history data, we adopt the convolutional neural network structure as the policy and the value function representations, similar to this method [1]. As illustrated in Figure 4.4, the historical data is reordered into a 2D array, the kernel filter of the CNN has the same width as the width of the input 2D array and is convolved with the input array to find the inherent temporal dependencies from the input data. In guided-COPOS, the guiding policy also receives the full states where we will use an additional fully connected layer to process it. The output features are then concatenated with the hidden layer.

4.3 Experiments

We showed in Section 3.5 that guided-COPOS is capable of solving the POMDP problem in a relatively low-dimensional environment (LunarLander-POMDP) and achieving outstanding performance comparing to non-guided policy search methods. In order to further evaluate the performance of guided-COPOS and compare the performance to COPOS, we now present our experiments in our customized challenging environment—CARLA. We design several autonomous driving tasks with increasing difficulty: a simple goal-directed task without dynamic traffic users, a standard goal-directed and collision avoidance task with full state observations of dynamic traffic users, and a challenging goal-directed and collision avoidance task with partial observations of dynamic traffic users. We run COPOS to solve the first two tasks and take the result as a baseline for future comparisons, and test guided-COPOS on the challenging POMDP environment. Ultimately, we analyze the results to investigate the following questions:

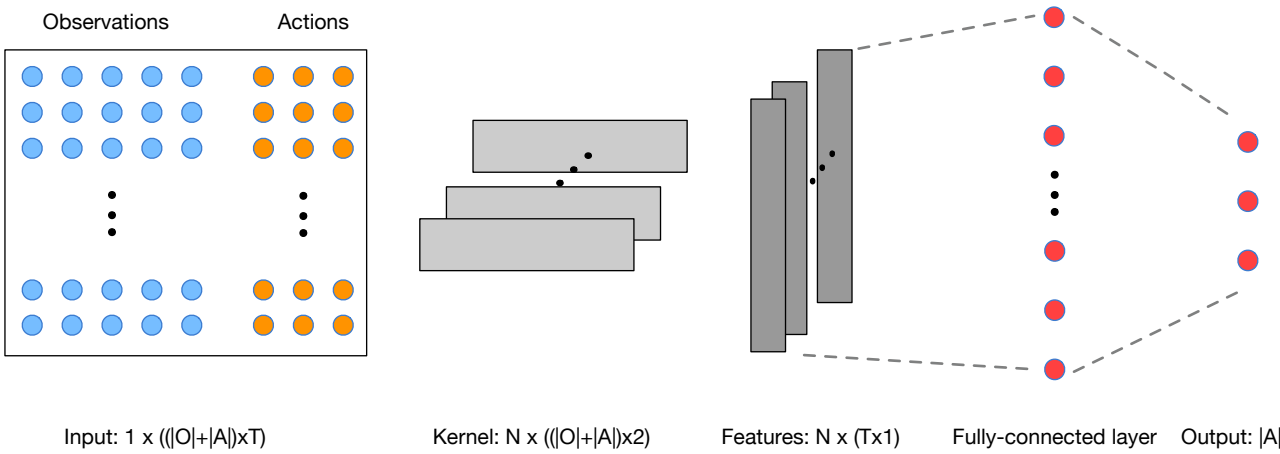

Figure 4.4: Sketch of a convolutional neural network as a policy representation for the experiment in CARLA. The input is a concatenated historical data matrix. |O| represents the dimension of observations, |A| represents the dimension of the actions.

1. Can model-free RL method, e.g. COPOS, solve the autonomous driving task when there's no other traffic users? How does COPOS perform when there are other pedestrians and vehicles and their full states are given? How does COPOS perform when receiving only partial observations of the dynamic users?

2. How does guided-COPOS perform compared to COPOS on the challenging partially observable autonomous driving task?

4.3.1 Autonomous driving without other dynamic traffic users

We start with a straight driving goal-directed task and disable other vehicles or pedestrians. We execute COPOS here such that we could have a baseline for the following experiments. We use the true measurements of the agent car as the input of the policy and value network, including the current GPS coordinate, forward speed, orientation, intersection with the other lane, and intersection with the sidewalk. The output control signals generated by the policy network include steering control, throttle pedal control and brake pedal control as summarized in Table 4.2. We adopt the same network structure as in Section 3.5.2 and use the reward function defined in Equation 4.1. We run COPOS with a total number of 2 million timesteps with a built-in frame skipping technique and set fps $= 2$. We can see the learning curve and the entropy curve in Figure 4.5 that COPOS converges to the reward 9200, which means it is able to achieve the goal without collisions.

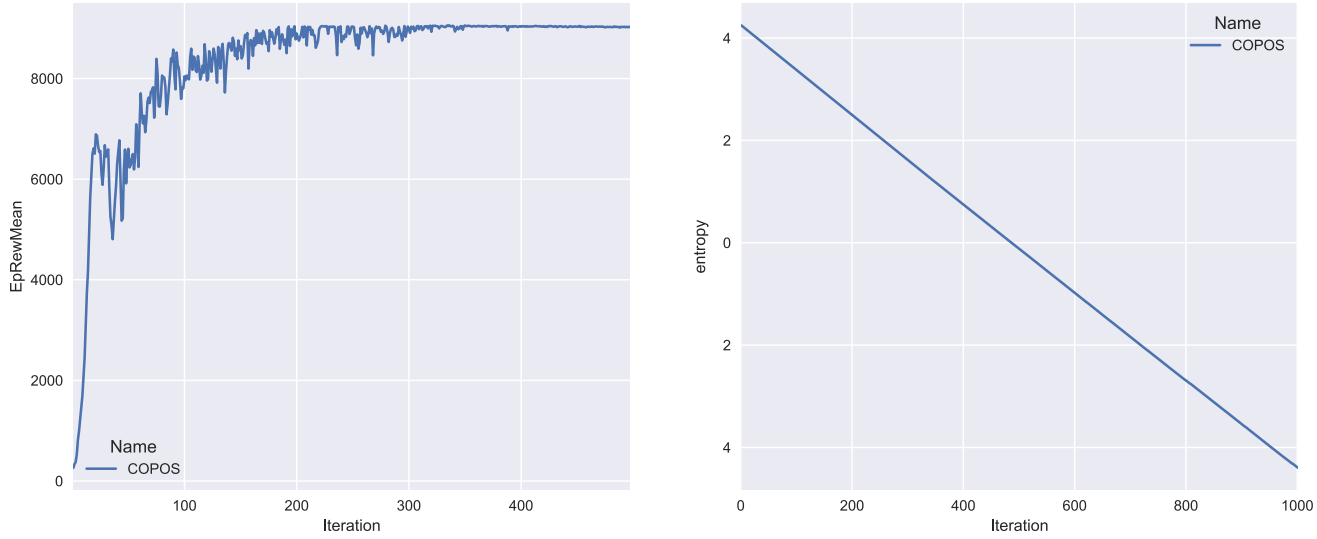

Figure 4.5: The learning curve and entropy curve of COPOS in our customized environment for 2 million timesteps. The dynamic traffic users (pedestrians and other vehicles) are disabled.

4.3.2 Comparison of different reward functions with full observations of dynamic traffic users

Then, we generate a random number (0-8) of dynamic vehicles on the same street, and a random number (0-15) of pedestrians either walking on the sidewalk or crossing the road. In reality, some of the pedestrians and cars might outside the view of the agent car. However, in this experiment, we assume the ground truth measurements of all other traffic users in order to choose a proper reward function and to establish a baseline for partially observable tasks. We use the neural network structure with 2 hidden layers of 64 units as the policy and value network, the input of which includes measurements of the agent car, measurements of the nearest 7 pedestrians, and measurements of the nearest 3 dynamic vehicles. We focus the comparison on the learning curve with a total number of 2 million timesteps. As shown in Figure 4.5, the smaller the coefficient is, the more stable the learning curve is. Furthermore, we compare the testing results based on the final learned policy for 100 episodes. We record the averaged return, the counts of completed tasks (reach the goal), collision with pedestrians, collision with other vehicles, collision with static obstaacles, intersection with the other lane, and intersection with the sidewalk, shown in Table 4.3. Note that we only count the occurrence of each event, i.e. no matter how many pedestrians does the agent car collide in the same episode, we only count it once. The results indicate that even though the reward function with a small coefficient has better learning performance and larger average return, it causes more collisions and more intersections. The reward function with large penalty shows minimal collisions, but a tremendously unstable learning process. After a comprehensive comparison of the learning performance and testing results, we decide to use the medium penalty reward function for future experiments, as it presents a more general performance over all aspects.

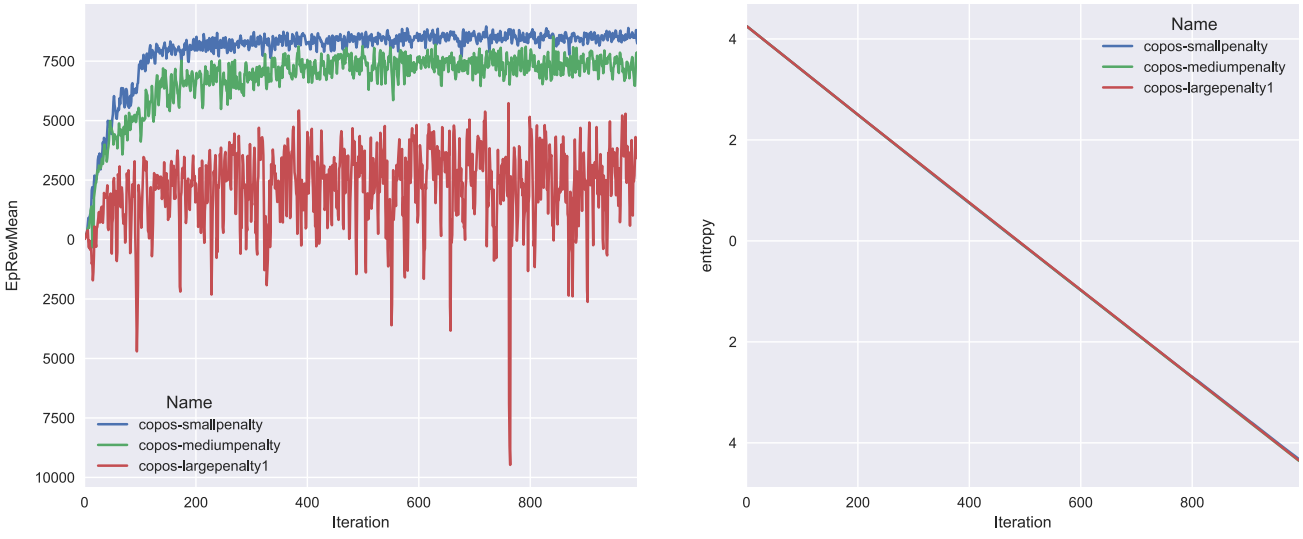

Figure 4.6: The comparison of learning curve and entropy curve among the reward function with small, medium, and large collision penalty. All learning curve are based on COPOS in our customized environment for 2 million timesteps. The dynamic traffic users (pedestrians and other vehicles) are enabled with full state observations.

Penalty level	Averaged return	Complete episodes	Car-collision-episode	Pedestrian-collision-episode	Other-collision-episode	Intersect otherlane	Sidewalk
Small	$\mathbf{8524.6 \pm 1068.3}$	**79**	21	23	29	**23**	**1**
Medium	7061.7 ± 2897.4	68	20	22	14	65	3
Large	2610.6 ± 8902.4	22	**7**	**16**	**10**	60	**1**

Table 4.3: A comparison of the final policy performances obtained by using COPOS in the same environment however with different reward functions. In total, we have three different reward functions of different penalty levels, and then, obtained three optimal policies. The first row contains several key entities that have to be compared and the corresponding results are listed below. The count number represents the total number of episodes of which a certain event occurred. For example, the total number of car-collision episodes is 7/100 when the penalty level is high compared to total episodes of 21/100 when the penalty is low.

4.3.3 Comparison of COPOS and guided COPOS with partial observations of dynamic traffic users

Finally, we generate a random number of vehicles and pedestrians on the same street again. However, the observations are different from previous experiments. Specifically, if the pedestrians walk outside of the shadow points, we always assume the agent car can obtain the true measurements. When the pedestrians are walking behind the parked cars (inside the shadow points), meanwhile the agent car drives into the "blind" area, we will hide the measurements of these pedestrians completely to the agent car. We adopt the network structure introduced in Section 4.2.5, the input of which is the history array of observation-action pairs. For timestep t, we use the observation-action pairs of timestep $t-1$, $t-2$, $t-4$, $t-8$, $t-16$, $t-32$, and $t-64$. If the current timestep is less than 64, we simply stack the lastest observation-action pair on the top of the array. In guided-COPOS, the guiding agent receives the full state measurements as well. We use those measurements as input to an additional hidden layer of 64 units and concatenate the output features with the hidden layer of the convolutional neural network. We focus the comparison on the learning curve and the entropy curve between COPOS and guided-COPOS with a total number of 2 million timesteps for each algorithm. As shown in Figure 4.7, our guided-COPOS is able to stabilize the training process, and had fewer collisions with pedestrians and cars at test time compared to COPOS. However, the learned policy is not the optimal solution due to the lack of training samples an training time. We believe that with a sufficient amount of samples and training time, both of the approaches are able to achieve much better performance.

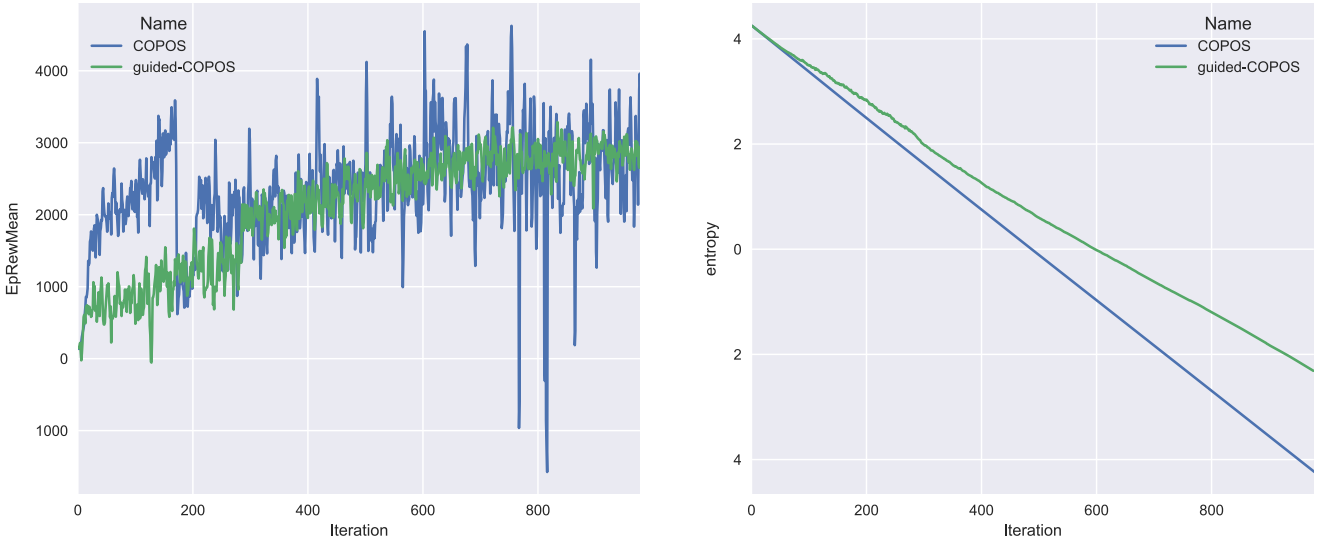

Figure 4.7: The comparison on the learning curve and the entropy curve between COPOS and guided-COPOS in our customized partially observable envrionment. The total training time is 2 million timesteps. The dynamic traffic users (pedestrians and other vehicles) are enabled but are partially observable.

Method	Averaged return	Complete episodes	Car-collision-episode	Pedestrian-collision-episode	Other-collision-episode	Intersect otherlane	Sidewalk
COPOS	$\mathbf{2639.9 \pm 2614.4}$	**3**	56	26	**44**	44	**38**
guided-COPOS	2236.5 ± 823.1	0	**13**	**15**	64	**1**	85

Table 4.4: A comparison of the final policy performances obtained by using COPOS and guided-COPOS in the same environment. The first row contains several key entities that have to be compared and the corresponding results are listed below. The count number represents the total number of episodes of which a certain event occurred. For example, the total number of car-collision episodes is 7/100 when the penalty level is high compared to total episodes of 21/100 when the penalty is low.

4.4 Discussion

In this chapter, we have introduced a customized urban autonomous driving environment, which is based on the CARLA simulator, for the evaluation of our guided-COPOS. We created three different autonomous driving tasks with increasing difficulties: a simple goal-directed task where walking pedestrians and moving cars were completely removed from the driving road, a standard goal-directed task where a number of random pedestrians and cars were moving or driving on the road and full state observations of the environment were provided to the agent, and a challenging goal-directed POMDP task where the states of the dynamic traffic users were partially observed. We first evaluated how well COPOS performed on those tasks and took the results as our baseline for future comparisons. The results indicated that COPOS found an optimal policy that successfully reached the goal state in the first simple task and avoided moving objects with high probabilities in the second task when fully observable states were given. COPOS failed to solve the challenging POMDP task in the end. Next, we evaluated our guided-COPOS on the challenging goal-directed POMDP task, but it did not find the optimal policy either. We found the reason for the bad learning performance as follows: The large number of parameters in the network requires more training samples and longer training time. However, running experiments on CARLA simulator is extremely time-consuming—approximately 15 hours for 1 million timesteps even though we accelerated with frame skipping techniques. If we compare the performance of guided-COPOS with COPOS based on our current results, we found that our guided-COPOS stabilized the training process, and showed more stable performance at test time. More importantly, guided-COPOS reduced the risk of hitting pedestrians compared to COPOS. We believe that with a sufficient amount of samples and training time, both of the approaches are able to achieve much better performance.

5 Conclusion and discussion

In this thesis, we first briefly introduced several key concepts in reinforcement learning fundamentals, including the mathematical framework of Markov decision processes and partially observable Markov decision processes. We gave a short introduction of deep learning and highlighted its combination with RL algorithms by reviewing several state-of-the-art deep RL algorithms. Furthermore, we introduced a traditional framework of the autonomous driving system and explained its key components, followed by showing several major challenges the traditional framework faced with where some of the challenges can be solved via RL approaches.

Our main contribution is to propose a new model-free guided policy search framework combined with COPOS, termed guided-COPOS, for solving partially observable tasks. We have shown how to derive it by first suggesting a new agent-environment interaction way to generate training samples, where the guiding agent and final control agent iteratively interact with the same environment. We update the two policies via compatible feature approximation embedded in COPOS such that the final control policy update towards the direction of a policy with better long-time rewards. We further introduced an additional update step for final control policy by minimizing the KL divergence between the two policies such that the final control policy converges to the same behavior as guiding policy. We evaluated our method in our customized challenging partially observable environment – LunarLander-POMDP, where we have successfully learned the policy and achieved good empirical results, outperforming other well-known policy search methods –TRPO, PPO, and COPOS. The testing results confirmed that our guided-COPOS is capable of generating good long-horizon performance from the final control policy at test time.

Finally, we introduced our customized urban autonomous driving environment based on CARLA simulator. We created three different autonomous driving tasks with increasing difficulty: a goal-directed task without dynamic traffic users, a goal-directed task with full state observations of dynamic traffic users, and a goal-directed task with partial observations of dynamic traffic users. We also adopted the CNN policy for the partially observable task in order to exploit the inherent temporal dependencies of the history data. We have seen COPOS achieved satisfactory results for the two easier tasks, but bad performance for the partially observable task. We have also tested our guided-COPOS for the challenging partially observable task. The results showed that our guided-COPOS was able to stabilize the training process, and had fewer collisions with pedestrians and cars at test time compared to COPOS. However, the learned policy was still not the optimal solution due to the lack of training samples an training time. We believe that with a sufficient amount of samples and training time, both of the approaches are able to achieve much better performance.

In guided-COPOS, we adopted an additional update step for the final control policy, which will lead to an approximate solution of the update. For future work, it would be interesting to integrate an additional constraint to the objective of the final control policy that forces the two policy distribution to be identical, such that we could have a full update rule rather than an approximate solution. Moreover, it is also interesting to adopt recurrent neural network for the policy in future, which could help the policy to learn more from the temporal dependencies.

Bibliography

[1] H. L. Thai, *Deep Reinforcement Learning for POMDPs*. PhD thesis, 2018.

[2] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1–9, PMLR, 17–19 Jun 2013.

[3] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 1071–1079, Curran Associates, Inc., 2014.

[4] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," *CoRR*, vol. abs/1509.06791, 2015.

[5] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[6] S. of Automotive Engineers (www.sae.org).

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[8] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[9] Wikipedia contributors, "Google neural machine translation — Wikipedia, the free encyclopedia," 2018. [Online; accessed 14-November-2018].

[10] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[13] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Springer Publishing Company, Incorporated, 2014.

[14] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving deep memory pomdps with recurrent policy gradients," in *Artificial Neural Networks – ICANN 2007* (J. M. de Sá, L. A. Alexandre, W. Duch, and D. Mandic, eds.), (Berlin, Heidelberg), pp. 697–706, Springer Berlin Heidelberg, 2007.

[15] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, (Cambridge, MA, USA), pp. 1057–1063, MIT Press, 1999.

[16] S. Kakade, "A natural policy gradient," 01 2001.

[17] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based pomdp solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, pp. 1–51, Jul 2013.

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, pp. 279–292, 1992.

[19] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," tech. rep., 1994.

[20] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, pp. 99–134, May 1998.

[23] E. J. Sondik, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs," *Oper. Res.*, vol. 26, pp. 282–304, Apr. 1978.

[24] H.-T. Cheng, *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

[25] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, (San Francisco, CA, USA), pp. 1025–1030, Morgan Kaufmann Publishers Inc., 2003.

[26] T. Smith and R. G. Simmons, "Heuristic search value iteration for pomdps," *CoRR*, vol. abs/1207.4166, 2012.

[27] L. Chrisman, "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach," 10 1994.

[28] M. L. Littman, "Memoryless policies: Theoretical limitations and practical results," in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3: From Animals to Animats 3*, SAB94, (Cambridge, MA, USA), pp. 238–245, MIT Press, 1994.

[29] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act by trial and error," *Machine Learning*, vol. 7, pp. 45–83, Jul 1991.

[30] J. Loch and S. P. Singh, "Using eligibility traces to find the best memoryless policy in partially observable markov decision processes," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, (San Francisco, CA, USA), pp. 323–331, Morgan Kaufmann Publishers Inc., 1998.

[31] L. C. B. III and A. W. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems 11* (M. J. Kearns, S. A. Solla, and D. A. Cohn, eds.), pp. 968–974, MIT Press, 1999.

[32] L.-J. Lin and T. M. Mitchell, "Reinforcement learning with hidden states," in *Proceedings of the Second International Conference on From Animals to Animats 2 : Simulation of Adaptive Behavior: Simulation of Adaptive Behavior*, (Cambridge, MA, USA), pp. 271–280, MIT Press, 1993.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[34] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[36] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[38] J. Peters, S. Vijayakumar, and S. Schaal, "Natural actor-critic," in *Proceedings of the 16th European Conference on Machine Learning*, ECML'05, (Berlin, Heidelberg), pp. 280–291, Springer-Verlag, 2005.

[39] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search."

[40] S. Behere and M. Torngren, "A functional architecture for autonomous driving," in *2015 First International Workshop on Automotive Software Architecture (WASA)*, pp. 3–10, May 2015.

[41] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating autonomous vehicle systems," vol. 6, no. 1, pp. i–186, 2017. Exported from https://app.dimensions.ai on 2018/09/19.

[42] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *CoRR*, vol. abs/1610.03295, 2016.

[43] S. Shalev-Shwartz, N. Ben-Zrihem, A. Cohen, and A. Shashua, "Long-term planning by short-term prediction," *CoRR*, vol. abs/1602.01580, 2016.

[44] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, and V. Koltun, "End-to-end driving via conditional imitation learning," *CoRR*, vol. abs/1710.02410, 2017.

[45] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[46] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *CoRR*, vol. abs/1604.07446, 2016.

[47] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, pp. 269–271, Dec. 1959.

[48] N. J. Nilsson, "A mobius automation: An application of artificial intelligence techniques," in *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, (San Francisco, CA, USA), pp. 509–520, Morgan Kaufmann Publishers Inc., 1969.

[49] H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," *CoRR*, vol. abs/1504.05140, 2015.

[50] C. Urmson, C. R. Baker, J. M. Dolan, P. Rybski, B. Salesky, W. R. L. Whittaker, D. Ferguson, and M. Darms, "Autonomous driving in traffic: Boss and the urban challenge," *AI Magazine*, vol. 30, pp. 17–29, June 2009.

[51] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, "Junior: The stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, pp. 569–597, Sept. 2008.

[52] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps," 10 2014.

[53] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. New York, NY, USA: Cambridge University Press, 1st ed., 2017.

[54] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, p. 65, Nov 2009.

[55] Wikipedia contributors, "Pid controller — Wikipedia, the free encyclopedia," 2018. [Online; accessed 3-October-2018].

[56] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *CoRR*, vol. abs/1504.00702, 2015.

[57] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," *CoRR*, vol. abs/1011.0686, 2010.

[58] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *CoRR*, vol. abs/1611.01224, 2016.

[59] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," *CoRR*, vol. abs/1606.02647, 2016.

[60] R. Fletcher, *Practical Methods of Optimization; (2Nd Ed.)*. New York, NY, USA: Wiley-Interscience, 1987.

[61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[62] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." `https://github.com/openai/baselines`, 2017.

[63] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[64] CARLA, "Carla documentation," 2018.

[65] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[66] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017.

[67] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the darpa grand challenge: Research articles," *J. Robot. Syst.*, vol. 23, pp. 661–692, Sept. 2006.

[68] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *CoRR*, vol. abs/1205.4839, 2012.

[69] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, (USA), pp. 3111–3119, Curran Associates Inc., 2013.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.