creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

심층 갤러킨 방법 기반 강화 학습

Reinforcement Learning Based on the Deep Galerkin Method

2020년 2월

서울대학교 대학원

기계항공공학부

최 진 원

ABSTRACT

Reinforcement Learning Based on the Deep Galerkin Method

by

Jinwon Choi

Department of Mechanical and Aerospace Engineering
Seoul National University

This thesis proposes a deep Galerkin-based algorithm for reinforcement learning. Recognizing that reinforcement learning can be viewed as a collection of methods and techniques to approximately solve the dynamic programming equations, we frame our problem as a continuous-time stochastic optimal feedback control problem in which the reward (cost function) is known, but the environment model (state dynamics) is not; instead, sample trajectories are available for learning the environment model. Using a Gaussian mixture model to represent the dynamics, model parameters are learned from the sample trajectories. The associated

Hamilton-Jacobi-Bellman (HJB) equations are then solved using the deep Galerkin method; here the optimal policy (feedback control law) and value function (cost-to-go) are modelled as independent deep learning networks. Monte Carlo sampling is then used to solve the associated HJB equations and obtain an optimal policy. Experiments are undertaken to assess the performance of our approach vis-à-vis existing reinforcement learning algorithms. As a by-product of our work, a taxonomy of existing reinforcement learning algorithms from a stochastic optimal control perspective is also provided.

Keywords: Stochastic Optimal Control, Dynamic Programming, Bellman equation, HJB Equation, Reinforcement Learning, Deep Galerkin Method

Student Number: 2018-25735

Contents

List of Tables

List of Figures

1

Introduction

Reinforcement learning (RL) is a computational approach to automating goal-directed learning and decision making. There are two main strategies for RL problems. The first is to search in the space of behaviors to find one that performs well. Genetic algorithms and genetic programming are well-known examples of the first strategy. The second is to use dynamic programming methods to estimate the cumulative reward [2]. The focus of thesis is on the second strategy.

A reinforcement learning system is defined by five main elements: an agent, an environment, a policy, a reward, and a value function. The agent behaves following a policy, while the environment determines the agent's next state and provides an instant reward. The goal of reinforcement learning is to find the policy that maximizes the cumulative reward for running an agent [3]. Such interaction between the agent and the environment in reinforcement learning systems is modeled as a Markov decision problem, i.e., sequential decision-making in which the control is determined by the current state only.

RL	Optimal control
Agent	Controller
Action	Control
Environment	System
Learning a model	System identification
Reward	Cost
Deep RL	Approximate dynamic programming
Prediction	Policy evaluation
Sweep	Apply the DP operator at all states
Greedy policy	Minimizing policy

Table 1.1: A list of commonly used terms in reinforcement learning and their optimal control counterparts [1].

Early reinforcement learning algorithms were designed to solve primarily small-scale, discrete-time, discrete-state problems in which all state-action pairs could be represented in table form (for example, navigating in a finite-size grid world with obstacles [4]). Recent advances in deep learning, however, have led to significant enhancement of the capabilities of reinforcement learning as a practical and viable tool. Deep reinforcement learning methods, in which deep learning networks have been used to approximate one or more elements of reinforcement learning, have been successfully applied to not only large-scale complex problems such as the game of go [5, 6], but also to planning and control of complex nonlinear continuous-time systems such as robot grasping [7] and autonomous driving [8, 9]

Recently, [1] has pointed out in a comprehensive way the close parallels between reinforcement learning and stochastic optimal feedback control. Table 1.1

provides a list of selected terms commonly used in reinforcement learning and their optimal control counterparts. The connections between reinforcement learning and stochastic optimal control have in fact been recognized in limited contexts by various researchers for some time. Markov decision problems arise frequently in stochastic control theory via discrete dynamic programming [10], while [11] proposed a policy iteration method for Markov decision problems.

As first pointed out by Bellman and now well-known, dynamic programming suffers from the curse of dimensionality, meaning that its computational and memory requirements grow exponentially with the number of state and control variables. [1] makes the legitimate claim that reinforcement learning can be exactly characterized as approximate dynamic programming, i.e., as a collection of methods for obtaining approximate solutions to the dynamic programming problem that mitigate the curse of the dimensionality.

1.1 A Taxonomy of Reinforcement Learning Algorithms

Reinforcement learning algorithms are classified according to whether the update occurs online or offline, and whether it is model-based (i.e., involves an explicit system identification procedure) or model-free (i.e., the objective is evaluated without a dynamic model). Offline methods compute the entire value function before the first control is applied, while online methods compute the values just after the current state becomes known. Model-based methods assume that the transition probability of the system for any state is available, while in the model-free case the values are obtained via Monte Carlo simulation.

[1] provides a more informative classification of reinforcement learning techniques according to what they approximate. This classification contains more information about the technique used in the algorithm than simply classifying the method according to whether or not a model is used. Later in this thesis, we propose a refined taxonomy from a control-theoretic perspective. We attempt to show, for each reinforcement learning algorithm, the underlying control-theoretic assumptions and solution techniques. For discrete-time formulations of reinforcement learning, the Bellman equation is the governing equation for obtaining an optimal policy. For continuous-time optimal feedback control problems, two possible approaches exist: (1) Solving the Bellman equation using discrete control methods via function approximation, and (2) Solving the continuous-time Hamilton-Jacobi-Bellman partial differential equations.

Most existing reinforcement learning algorithms follow the former approach [12, 13, 14]. These methods are typically subdivided into two distinct discrete dynamic programming problems: (1) Discounted dynamic programming (DDP) and (2) Average reward dynamic programming. Discounted dynamic programming attempts to find a policy that minimizes the sum of the discounted rewards over an infinite-time horizon. DDP assumes that a transition probability will be fixed at any given time; in practice however, many problems do not satisfy this property [15]. Average reward dynamic programming optimizes the expected average reward per step over time t as $t \to \infty$ (the definition of average reward is presented later in Chapter 2.2.2). These two dynamic programming problems can be solved by value iteration (VI) and the policy iteration (PI). In general, reinforcement algorithms based on solving the Bellman equation via approximation do not necessarily guarantee convergence to the correct solution. Moreover, this approach involves an additional approximation step involving discretization of space and time, leading

to further accumulation of approximation errors over multiple steps.

The second class of methods attempt to solve the continuous-time HJB equation. While in principle numerical methods for the solution of continuous-time equations invariably involve discretization at some level, the algorithms are designed within the continuous-time framework and thus differ from methods based on the discrete Bellman equations. There is extensive control-theoretic literature on attempts to numerically solve continuous-time optimal control problems via approximation techniques. Some model-based reinforcement learning methods in this category have tried to leverage existing control-theoretic methods for continuous systems: [16, 17] solve the LQG problem by applying data-driven robust control theory. Guided policy search (GPS) has been used to address problems in robot manipulation with iterative linear quadratic Gaussian (iLQG) methods [18, 19, 20]. We note that GPS discretizes the system, yet also solves the linear quadratic Gaussian (LQG) problem by approximating the derivatives of the Q-function and the dynamics model.

1.2 Reinforcement Learning and the Deep Galerkin Method

In this section we describe the main contribution of this thesis: a reinforcement algorithm that attempts to solve the continuous-time HJB equations using a pair of deep learning networks to parameterize the policy and value functions. Our method is inspired by recent work that uses deep learning networks to solve general partial differential equations.

Including the Hamilton-Jacobi Bellman (HJB) equation, high-dimensional partial differential equations (PDEs) are ubiquitous in modeling problems. Since closed-form solutions for PDEs are rare, often one must resort to numerical methods.

There are four major approaches to obtaining numerical solution to PDEs: (1) finite-difference method; (2) Galerkin method; (3) finite-element method; and (4) Monte-Carlo method [21, 22]. Finite-difference methods approximate the differential operators by discretizing space and time, and using the difference in values at the grid points. The fourth-order Runge-Kutta method is widely used among finite difference methods. A major shortcoming of finite difference methods is that the number of required grids increases exponentially with the dimension of the domain.

The Galerkin method approximates the solution function by using a linear combination of finite basis functions. One difficulty of the Galerkin method is the choice of basis function, because of the non-trivial integrals involved in evaluating the inner products between functions. The finite element method can be seen as a localized version of the Galerkin method; First, this method discretizes the domain over small regions, and then locally approximates the solution using the basis function. The Monte-Carlo method is based on simulation of a stochastic process; from the Feynman-Kac theorem, the PDE can be viewed as a partial differential equation governing the probability density function evolution of a some stochastic differential equation. This method also requires the calculation of the gradient, whose computational burden increases exponentially with the problem dimension.

Recently, neural networks have been used as function approximators in an attempt to address the dimensionality issues described above. The idea of solving differential equations using a neural network was first proposed in [23]. A loss function of the form

$$L = \int_{\Omega} (F[u] - f)^2(x)\, dx + \int_{\partial\Omega} (B[u] - f)^2(x)\, dx \tag{1.2.1}$$

is proposed, in which F is a differential equation operator on the domain X, and B

is a boundary condition operator on the domain boundary ∂X. Details of training for these networks are provided later in Section 4.1. [24, 25, 26, 27] also propose deep learning-based PDE solvers using a similar loss function. These algorithms use an a priori defined mesh as inputs, and minimize the loss function at the mesh points. When domain geometries are known, [25] and [26] modify the last layer of the network to satisfy the boundary conditions without using the B term in the loss function. [27] propose a method for learning the domain geometries for complex-shaped boundary problems. In our problem, we are primarily interested in methods for solving the HJB equation without any a priori specified boundary (a free-boundary problem); instead, a terminal state condition is typically given.

Another complicating issue for us is that because the dynamics are typically assumed unknown, the PDEs themselves are also unknown. Recently, data-driven methods for PDE identification were proposed in [28, 29, 30, 31]. A physics-informed neural network (PINN) uses prior knowledge about the order of the PDE, and identifies the parameters from collected data [28, 29]. Hidden physics models (HPM) identify the models using a Gaussian process model (GP) without assuming any prior knowledge about the model [30]. Deep hidden physics models (DHPM) are similar but use a deep neural network in place of a Gaussian process. These methods are for the most part ineffective for identifying the HJB equation since no value, state-value, or state-action value data is available. State-action pairs and rewards can, however, be obtained from simulations. Our algorithm identifies the HJB equation using dynamic learning methods from model-based reinforcement learning rather than PDE identification techniques.

PINN and DHPM also train networks with a similar loss function. However, the previous loss function does not guarantee convergence of a neural network to a real solution of the PDE. The most impactful result for deep PDE solvers is likely

the neural network approximation theorem proved by [32], in which they prove the convergence of the neural network to a PDE solution using a specific update rule, and a deep Galerkin method that samples the input points randomly from the domain instead of at fixed mesh points. In this thesis we propose a modified deep Galerkin method to solve the HJB equation driven by policy iteration. The value function and the control input are iteratively updated.

We note that there exist deep PDE solvers for quasilinear PDEs based on the Monte-Carlo method [33, 34, 35]. These approximate the gradient of the solution and simulate a backward stochastic differential equation to obtain an analytical solution. The algorithms developed in [33, 34, 35] obtain the solution of PDEs at a single point, while [32] obtains a complete solution across time and space. We choose the deep Galerkin method over BSDE methods in order to generalize the controller throughout the entire domain.

1.3 Thesis Organization

In Chapter 2, we review the basics of stochastic optimal control problems. Four types of stochastic optimal control problems, discrete/continuous system for finite/infinite-time horizon, are presented with assumptions that guarantee the existence and uniqueness of the solution. We review Bellman's optimality principle and the Hamilton-Jacobi-Bellman partial differential equations. Early reinforcement learning algorithms treat the infinite-time horizon discrete system problems based on discrete dynamic programming methods. Two dynamic programming methods (policy iteration and value iteration) will be presented here.

In chapter 3, we summarize early RL algorithms based on the dynamic programming methods with Robbins-Monro stochastic approximation. RL research

fields are extended to continuous system problems by integrating function approximation schemes such as deep neural network. However, we note that such an approach amplifies approximation errors by adding another layer (the approximation of the dynamic programming equation) on top of the discretization. Another approach to solving continuous problems is approximating continuous control methods. Since optimal control theory for continuous problems requires a dynamics model, all algorithms using this approach are model-based. We briefly review the model-based algorithms solving continuous problems, and then we propose a new taxonomy of RL algorithms from the control-theoretic perspective. This taxonomy emphasizes the underlying control methods for each RL algorithm, and is useful in identifying algorithms appropriate for a given problem.

Chapter 4 is devoted to describing deep PDE solver-based RL algorithms for solving the HJB equation directly, without discretization of the system. Numerical methods for PDEs also suffer from the curse of dimensionality, and although deep learning methods have been addressed as one possible remedy, theoretical convergence guarantees for deep neural network approximation of PDEs' solution has only recently been proven. We review deep PDE solvers and numerical PDE theory, and also deep Galerkin methods for their solution. For reinforcement learning, the HJB equation must be identified before solving the PDE using deep Galerkin methods. Unlike usual PDE identification, in RL dynamics learning methods must be employed. Our algorithm uses a Gaussian mixture model (GMM) prior dynamics learning method to identify the system dynamics from exploration data, and solves the identified HJB equation using deep Galerkin methods. To obtain the optimal policy, we propose a policy iteration-like scheme with a policy network that allows for the integration of various policy gradient methods.

In Chapter 5, we first describe a one-dimensional LQG problem and obtain an

analytical solution. Then, we compare our new algorithm and other discrete dynamic programming-based RL algorithms with respect to the value function approximation. We then apply our proposed algorithm to solve a twelve-dimensional LQG problem.

The thesis concludes in Chapter 6 with a summary of the main results and findings, and a discussion of both the potential practical applications and limitations of the proposed method.

2

Stochastic Optimal Control

In this chapter we review the stochastic optimal control problem for both continuous-time and discrete-time systems. We begin with the stochastic optimal control problem in continuous-time in Section 2.1, followed by discrete-time systems in Section 2.2. The goal of this chapter is to show more explicitly the connections and differences between stochastic optimal control and reinforcement learning.

2.1 Continuous-Time Systems

2.1.1 Optimal Control Problem

In what follows we consider a finite-time horizon problem $t \in [0, T]$. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $\left\{\mathcal{F}_{\sqcup t \geq 0}\right\}$ a filtration on $\mathcal{F}$ and $W(\cdot)$ a $\left\{\mathcal{F}_{\sqcup t \geq 0}\right\}$ Brownian motion in $\mathbb{R}^m$. Let $\mathcal{U}$ be a subset of all progressively measurable stochastic processes $u : [0, T] \times \Omega \to U$; we call this set the control set, and element u of this set a control. Let $f : \mathbb{R}^n \times [0, T] \times U \to \mathbb{R}^n$ and $\sigma : \mathbb{R}^n \times [0, T] \times U \to \mathbb{R}^{n \times m}$ be

continuous functions in $C^1(\mathbb{R}^n \times [0,T])$ satisfying the following conditions:

$$\begin{aligned} |f(x,t,u) - f(y,t,u)| &\leq C(|x-y| + |t-s|), \\ |\sigma(x,t,u) - \sigma(y,t,u)| &\leq C(|x-y| + |t-s|) \end{aligned} \tag{2.1.1}$$

and

$$\begin{aligned} |f(x,t,u)| &\leq C(1+|x|), \\ |\sigma(x,t,u)| &\leq C(1+|x|) \end{aligned} \tag{2.1.2}$$

where $|\sigma|^2 = \sum_{i,j} |\sigma_{ij}|^2$.

Let the state $x(t)$ be an Itô process, and consider the following stochastic control system:

$$\begin{aligned} dx(s) = f(x(s), s, u(s))ds + \sigma(x(s), s, u(s))\, dW(s), \\ x(t) = x. \end{aligned} \tag{2.1.3}$$

The previous Lipschitz continuity assumptions on $f$and σ guarantee the uniqueness and existence of strong solutions with continuous paths for any choice of $u(\cdot)$. Furthermore, we assume that $u(s) = u(s, x(s))$ *i.e. the control depends only on the present state (a Markov control).* With the Markov control assumption, the state dynamics defined by the Itô process becomes a Markov decision process.

Let $r : \mathbb{R}^n \times [0,T] \times U \to \mathbb{R}$ and $q : \mathbb{R}^n \to \mathbb{R}$ be continuous functions such that

$$\begin{aligned} |r(x,t,u) - r(y,t,u)| &\leq C(|x-y| + |t-s|), \\ |r(x,t,u)| &\leq C(1+|x|), \\ |q(x)| &\leq C(1+|x|). \end{aligned} \tag{2.1.4}$$

The function r is the running cost and the function q is the terminal cost.

Definition 2.1.1. The cost functional $J : \mathbb{R}^n \times [0,T] \times U \to \mathbb{R}$ is defined by

$$J(x,t,u(\cdot)) := E_x \left[\int_t^T r(x(s), s, u(S))\, ds + q(x(T)) \right]$$

where E_x denotes the expectation with respect to the measure induced by $x(s)$ starting at x.

We now define the value function V and optimal control u^*.

Definition 2.1.2. The value function $V : \mathbb{R}^n \times [0,T] \to \mathbb{R}$ is defined by

$$V(x,t) := \inf_{u(\cdot)\in\mathcal{U}} J(x,t,u(\cdot))$$

and the optimal control $u^* \in \mathcal{U}$ is defined by

$$u^* := \arg\min_{u(\cdot)\in\mathcal{U}} J(x,t,u(\cdot)).$$

THe finite-time horizon stochastic optimal control problem can now be characterized as follows:

$$\begin{cases} V(x,t) = \inf_{u(\cdot)\in\mathcal{U}} J(x,t,u(\cdot)), \\ V(x,T) = q(x(T)), \\ \text{s.t.} \quad dx(s) = f(x(s),s,u(s,x(s)))ds + \sigma(x(s),s,u(s,x(s)))\, dW(s), \\ s \in [t,T]\,, x(t) = x \end{cases} \tag{2.1.5}$$

For the infinite-time horizon problem, a discount factor $\gamma > 0$ is introduced in the definition of the cost functional J.

Definition 2.1.3. The cost functional $J_\infty : \mathbb{R}^n \times [0,T] \times U \to \mathbb{R}$ is defined by

$$J_\infty(x,t,u(\cdot)) := E_x\left[\int_t^\infty e^{\gamma s} r(x(s),s,u(S))\, ds\right]$$

.

The infinite-time horizon stochastic optimal control problem can now be characterized as follows:

$$\begin{cases} V(x,t) = \inf_{u(\cdot)\in\mathcal{U}} J_\infty(x,t,u(\cdot)), \\ \text{s.t.} \quad dx(s) = f(x(s),s,u(s,x(s)))ds + \sigma(x(s),s,u(s,x(s)))\, dW(s), \\ s \in [t,\infty)\,, x(t) = x. \end{cases} \tag{2.1.6}$$

2.1.2 Dynamic Programming

In this section we review Bellman's principle of optimality and the stochastic Hamilton-Jacobi-Bellman (HJB) equations; we refer the reader to [36] for further details.

Theorem 2.1 (Bellman's principle of optimality)**.** *Suppose that the dynamical system satisfies $x(t) = x$. The value function then satisfies the following recursive equation:*

$$V(x,t) = \inf_{u(\cdot)\in\mathcal{U}} E_x\left[\int_t^{t+h} r(x(s),s,u(s))\,ds + V(x(t+h),t+h)\right]$$

.

Theorem 2.2 (Hamilton-Bellman-Jacobi)**.** *Suppose that the value function $V \in C^2(O)\cap C(\bar{O})$ where $(x,t)\in O$ and $\bar{O}$ is the closure of O. Then the value function satisfies the following form of the HJB equations:*

$$\begin{gathered}\frac{\partial V}{\partial t}(x,t) + \min_{u\in\mathcal{U}}\left\{\frac{\partial V}{\partial x}(x,t)\cdot f(x,t,u) + \frac{1}{2}tr(\sigma\sigma^T)\,\mathrm{Hess}_x\, V + r(x,t,u)\right\} = 0,\\ V(x,T) = q(x(T))\end{gathered} \tag{2.1.7}$$

This equation can be derived straightforwardly in the one-dimensional case. By Theorem 2.1,

$$V(x,t) = \inf_{u(\cdot)\in\mathcal{U}} E_x\left[r(x(t),t,u(t))dt + V(x(t+dt),t+dt)\right]. \tag{2.1.8}$$

Taylor expanding $V(x(t+dt),t+dt)$, we have

$$\begin{aligned}V(x(t+dt),t+dt) = V(x,t) &+ \frac{\partial V}{\partial t}(x,t)dt + \frac{\partial V}{\partial x}(x,t)dx\\ &+ \frac{1}{2}\frac{\partial^2 V}{\partial t^2}(x,t)(dt)^2 + \frac{\partial^2 V}{\partial t\partial x}(x,t)dtdx + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(x,t)(dx)^2 + h.o.t..\end{aligned} \tag{2.1.9}$$

Apply the usual stochastic multiplication rules

$$(dt)(dW) = (dt)^2 = 0, \quad (dW)^2 = dt, \tag{2.1.10}$$

we obtain

$$\begin{aligned} (dx)^2 &= f^2(Dt)^2 + \sigma^2(dW)^2 + f\sigma(dt)(dW) = \sigma^2\, dt, \\ dt\, dx &= f(dt)^2 + \sigma(dt)(dW) = 0. \end{aligned} \tag{2.1.11}$$

Equation (2.1.9) therefore becomes

$$\begin{aligned} V(x(t+dt), t+dt) = V(x,t) + \frac{\partial V}{\partial t}(x,t)\, dt + \frac{\partial V}{\partial x}(x,t) \cdot f(x,t,u)\, dt \\ + \frac{\partial V}{\partial x}(x,t) \cdot \sigma(x,t,u)\, dW + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(x,t)(dx)^2 + h.o.t.. \end{aligned} \tag{2.1.12}$$

Substituting (2.1.12) into (2.1.8) we get

$$\begin{aligned} V(x,t) = \inf_{u(\cdot)\in\mathcal{U}} E_x[r(x(t),t,u(t))\, dt + V(x,t) + \frac{\partial V}{\partial t}(x,t)\, dt + \frac{\partial V}{\partial x}(x,t) \cdot f(x,t,u)\, dt \\ + \frac{\partial V}{\partial x}(x,t) \cdot \sigma(x,t,u)\, dW + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(x,t)(dx)^2]. \end{aligned} \tag{2.1.13}$$

Since $E_x\left[\frac{\partial V}{\partial x}(x,t) \cdot \sigma(x,t,u) dW\right] = 0$, dividing by dt we get the HJB equation (2.1.7). We note that the continuity and differentiabiity assumptions on the value function can be further relaxed; we refer the reader to [37] for further details, including an exact proof of Theorem 2.1 and 2.2, which require further results from stochastic analysis.

2.2 Discrete-Time Systems

2.2.1 Optimal Control Problem

For discrete-time systems, the state and control at time k are denoted by $x_k \in X_k$ and $u_k \in U_k$, where $k \in 1, 2, \ldots, N$ for the finite-time horizon case and $k \in \mathbb{N}$.

The system evolves N times, and the system dynamics includes a disturbance w_k, which is characterized by a conditional probability distribution $P_k(w_k \mid x_k, u_k)$. The dynamics is of the form

$$x_{k+1} = f_k(x_k, u_k, w_k) \tag{2.2.14}$$

Note that we only consider Markov decision processes throughout, *i.e., the transition probability depends only on current state and current action, and the control depends current state only.* In the MDP problem, rather than optimizing over the control sequences $\{u_0, , \ldots, u_{N-1}\}$, we seek the optimal polices $\pi \in \Pi$ that consist of function sequences $\{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ where $\mu_k : X_k \to U(x_k)$ satisfies $\mu_k(x_k) = u_k \in U(x_k)$ for all $x_k \in X_k$. A well-known result from optimal control states that the optimal controller can be expressed in feedback form. Stochastic dynamics is also included in the controller through feedback (this is the reason why we consider policy functions rather than deterministic control sequences). In what follows let $r_k : X_k \times U(x_k) \to \mathbb{R}$ and $q : X_k \to \mathbb{R}$ be continuous functions.

Definition 2.2.1. The cost functional $J(x_k)$ is defined by

$$J(x_k; u_k, \ldots, u_{N-1}) = E_{w_k}\left[\sum_{m=k}^{N-1} r_m(x_m, u_m, w_m) + q(x_N)\right].$$

Given an initial state x_0 and policy $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$, the state-cost functional J_π is defined by

$$J_\pi(x_0) = E_{w_k}\left[\sum_{k=0}^{N-1} r_k(x_k, \mu_k(x_k), w_k) + q(x_N)\right].$$

The optimal policy $\pi^* \in \Pi$ is defined by

$$\pi^*(\cdot) = \arg\min_{\pi \in \Pi} J_\pi(x_0)$$

The finite-time horizon stochastic optimal control problem can now be characterized as follows:

$$\begin{cases} V(x_0) = \min_{\pi\in\Pi} J_\pi(x_0), \\ V(x_N) = q(x_N), \\ \text{s.t.} \quad x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots, N-1 \end{cases} \tag{2.2.15}$$

For the infinite-time horizon case, a discount factor γ is added to cost functional J_k, similarly to the case for continuous-time systems, but with the range of discounting factor set to $0 < \gamma < 1$ since the running cost is directly multiplied by the discount factor without an exponential function.

Definition 2.2.2. The cost functional $J_k(x_k)$ is defined by

$$J(x_k; \{u_i\}_{i=k}^{\infty}) = E_{w_m}\left[\sum_{m=k}^{\infty} \gamma^{m-k} r_m(x_m, u_m, w_m)\right]$$

with given initial state x_0 and policy π. The state-cost functional J_π^∞ is defined by

$$J_\pi^\infty(x_0) = E_{w_k}\left[\sum_{k=0}^{\infty} \gamma^k r_k(x_k, \mu_k(x_k), w_k) + q(x_N)\right].$$

The optimal policy $\pi^* \in \Pi$ is defined by

$$\pi^*(\cdot) = \arg\min_{\pi\in\Pi} J_\pi^\infty(x_0)$$

Infinite-time horizon problems are of the following form:

$$\begin{cases} V(x_0) = \min_{\pi\in\Pi} J_\pi^\infty(x_0), \\ \text{s.t.} \quad x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots \end{cases} \tag{2.2.16}$$

2.2.2 Dynamic Programming Principle

Unlike the continuous-time case, the optimality conditions for discrete-time problems are formulated in terms of the Bellman equation (BE), since time derivatives

are not well-defined for the discrete-time case. The value function for finite-time horizon problems can be obtained more easily than for the infinite-horizon case, simply by iterating the dynamic programming equations backward from the terminal condition to the initial state. In this section, we consider only the infinite-horizon case, which is the case of most interest in reinforcement learning.

There are two approaches to solving infinite-horizon problems: (1) Solve the discounted dynamic programming equation (see Theorem 2.3); (2) Solve the average reward dynamic programming equation. Both approaches share common techniques, but the average reward equation is a modified equation in which the discount factor is dropped via the average reward function. Theorem 2.3 shows the discounted DP problems.

Theorem 2.3 (Bellman's principle of optimality)**.** *Suppose that the dynamical system satisfies $x(k) = x_k$. The value function then satisfies following recursive equation:*

$$V(x_k) = \inf_{u(x_k) \in \mathcal{U}(x_k)} E_x \left[r(x_k, u_k, w_k) + V(x_{k+1}) \right]$$

with $V(x_N) = q(x_N)$ for finite-time horizon problems, and

$$V(x_k) = \inf_{u(x_k) \in \mathcal{U}(x_k)} E_x \left[r(x_k, u_k, w_k) + \gamma V(x_{k+1}) \right]$$

for infinite-time horizon problems.

We refer the reader to [38] for a proof of Bellman's principle of optimality. Discounted DP assumes that a transition probability will be fixed at any given time, however, many domains do not have this property [15]. Furthermore, discounting tends to sacrifice bigger long-term rewards but contain smaller short-term rewards which is undesirable. To avoid the problems of discounted DP, average reward dynamic programming optimizes the expected average reward per step over time t as $t \to \infty$.

Definition 2.2.3. Given policy π, the expected average-reward is defined by

$$\rho^{\pi}(x_k) = \lim_{t\to\infty} \frac{E\left[\sum_{k=0}^{t-1} r(x_k)\right]}{t} \tag{2.2.17}$$

where $r(x_k)$ is the reward received at time k starting from state x_k. In average reward MDP, ssince the undisconunted sum of rewards can be unbounded, we use diffrent value function,

$$V^{\pi}(x_k) = \lim_{t\to\infty} \frac{E\left[\sum_{k=0}^{t-1} r(x_k) - \rho^{\pi}\right]}{t} \tag{2.2.18}$$

Theorem **??** shows the average reward Bellman equation.

Theorem 2.4. *Suppose that the dynamical system satisfies $x(k) = x_k$. The value function then satisfies following recursive equation:*

$$V(x_k) + \rho = \inf_{u(x_k)\in\mathcal{U}(x_k)} E_x\left[r(x_k, u_k, w_k) + V(x_{k+1})\right]$$

where ρ is the optimal average reward.

Those two dynamic programming problems can be solved by the value iteration (VI) and the policy iteration (PI). In this thesis, we will review the control methods for discounted DP only. We refer the reader to mahadevan1996average for details.

2.2.3 Dynamic Programming Methods

The Bellman equation can be solved by two methods: (1) value iteration; (2) policy iteration. Before explaining the two methods, we review the Bellman Operator (dynamic programming operator) and its properties.

Let $(\mathbb{B}, ||\cdot||_{\infty}, d_{\infty})$ be a metric space where $\mathbb{B} = \{\psi : \Omega \to \mathbb{R} | \text{continuous and bounded}\}$, $||\psi||_{\infty} := \sup_{x\in X} |\psi(x)|$, $d_{\infty}(\psi, \psi') = \sup_{x\in X} |\psi(x) - \psi'(x)|$.

Definition 2.2.4. Given a policy π, the state-value function V^π is defined by

$$V^\pi(x_0) := J_\pi^\infty(x_0),$$

and the Bellman operator $T^\pi : \mathbb{B} \to \mathbb{B}$ is defined by

$$T^\pi\psi(x_k) = r(x_k, \pi(x_k)) + \gamma E_x[\psi(x_{k+1})].$$

The Bellman optimal operator $T : \mathbb{B} \to \mathbb{B}$ is defined by

$$T\psi(x_k) = \min_{u_k \in U(x_k)} \{r(x_k, u_k) + \gamma E_x[\psi(x_{k+1})]\}$$

We now list the following two important properties of the Bellman operator and the main theorem that forms the cornerstone of the value iteration and policy iteration methods.

Proposition. The Bellman operators T^π, T are both monotone, *i.e.,* if

$$\psi(x) \leq \psi'(x) \quad \forall x \in X,$$

then

$$\begin{aligned} T^\pi\psi(x) &\leq T^\pi\psi'(x) \quad \forall x \in X, \\ T\psi(x) &\leq T\psi'(x) \quad \forall x \in X. \end{aligned}$$

Proposition. The Bellman operators T^π, T are a contraction with modulus γ with respect to the sup norm $||\cdot||_\infty$, *i.e.,*

$$\begin{aligned} \left|\left|T^\pi\psi - T^\pi\psi'\right|\right|_\infty &\leq \gamma \left|\left|\psi - \psi'\right|\right|_\infty \quad \psi, \psi' \in \mathbb{B}, \\ \left|\left|T^\psi - T\psi'\right|\right|_\infty &\leq \gamma \left|\left|\psi - \psi'\right|\right|_\infty \quad \psi, \psi' \in \mathbb{B}. \end{aligned}$$

Theorem 2.5 (Contraction Mapping Theorem)**.** *Let* $(\mathbb{B}, ||\cdot||_\infty, d_\infty)$ *be a metric space and* $T : \mathbb{B} \to \mathbb{B}$ *a contraction mapping with modulus* γ*. Then*

(i) *T has an unique fixed point in $\mathbb{B}$, i.e., there exists a unique $f^* \in \mathbb{B}$ s.t. $Tf^* = f^*$.*

(ii) *The sequence $f_{n+1} = Tf_n$, $f_0 \in \mathbb{B}$, satisfies $\lim_{n\to\infty} T^n f_0 \to f^*$.*

Theorem 2.5 implies that a value function can be obtained by applying the Bellman operators T or T^π iteratively to an arbitrary initial function $V_0 \in \mathbb{B}$. Value iteration involves using the Bellman optimal operator to solve the dynamic programming equations, while policy iteration corresponds to using the Bellman operator.

The value iteration method calculates the value function by applying the Bellman optimal operator to an initial function with respect to $x \in X$ until $||V_{k+1} - V_k||_\infty < \epsilon$; the optimal policy is then obtained as

$$\pi^*(x) \in \arg\min_{\pi\in\Pi} \left\{ r(x, \pi(x)) + \gamma E_x \left[V_k(x') \right] \right\} \quad \forall x \in X. \tag{2.2.19}$$

Policy iteration consists of two steps: (1) policy evaluation; (2) policy improvement. Given a policy, the policy evaluation step calculates the state-value function by iteratively applying the Bellman operator. In the policy improvement step, a new policy (so-called greedy policy) is obtained so as to minimize the state-value function obtained from the policy evaluation step.

In the next chapter we focus on the relation between RL and stochastic optimal control. RL was originally developed to solve infinite-horizon discrete-time system stochastic dynamic programming equations in which only a black-box environment model can be accessed, unlike the situation in classical optimal control problem where the system state equations are assumed known and given.

3

Reinforcement Learning

3.1 Introduction

In classical dynamic programming methods for policy iteration and value iteration, the value function for each state is updated at each time-step via the Bellman operator. This update process requires the calculation of the expectation for every state-action pair. However, it is impractical to simulate trajectories over every state-action pair. Instead, reinforcement learning approximates the Bellman equation in three ways: (1) expectation approximation, (2) value function approximation, and (3) policy approximation:

$$V(x_k) = \underbrace{\inf_{\pi \in \Pi}}_{\substack{\text{Approximation} \\ \text{in policy space}}} \overbrace{E_\pi}^{\substack{\text{Approximation} \\ \text{of } E[\cdot]}} \left[r(x_k, \pi(x_k)) + \gamma E_{x \sim p}[\underbrace{V(x_{k+1})}_{\substack{\text{Approximation} \\ \text{in value space}}}] \right] \tag{3.1.1}$$

Calculation of the expectation can be approximated using Monte-Carlo search or the certainty equivalence method. For a comprehensive description of these methods we refer the reader to [1]. Our focus in this thesis will instead be on approximation of the value and policy functions.

Stochastic optimal control requires the calculation of the expectation for every state-action pair, and since for typical reinforcement learning problems the dynamics and reward functions are usually unknown, one cannot directly calculate the expectation. Value functions are therefore often estimated from sample trajectories obtained experimentally or from simulations. A rather obvious downside of this approach is that an excessively large number of simulations are required to obtain trajectories that fill the entire state-action space. Robbins-Monro stochastic approximation is one method developed to circumvent this sampling issue. Before we introduce Robbins-Monro stochastic approximation, we introduce some alternative definitions of the value function that also appear in the literature.

Definition 3.1.1. Given a policy π, the state-action value function is defined by

$$Q^{\pi}(x_k, a_k) = r(x_k, a_k) + \gamma E_x \left[\sum_{t=1}^{\infty} r(x_{k+t}, \pi(x_{k+t})) \right].$$

The optimal Q-function Q^* is defined by

$$Q^*(x_k, a_k) = r(x_k, a_k) + \min_{\pi \in \Pi} \gamma E_x \left[\sum_{t=1}^{\infty} r(x_{k+t}, \pi(x_{k+t})) \right].$$

The Bellman operator and the Bellman optimal operator over the Q-function can be defined in the same way. We now illustrate the Robbins-Monro stochastic approximation method. Consider the fixed point equation

$$x = L(x), \tag{3.1.2}$$

where $L : \mathbb{R}^n \to \mathbb{R}^n$ is an operator. Assuming the mapping is a contraction and satisfies conditions for the fixed point theorem, the above equation can be solved via the deterministic iteration

$$x_{k+1} = L(x_k). \tag{3.1.3}$$

It is also possible to add a step size α_k as follows:

$$x_{k+1} = (1 - \alpha_k)x_k + \alpha_k L(x_k). \tag{3.1.4}$$

Now suppose $L(x)$ is an expectation of the form $E[f(x,\xi)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i, \xi_i)$ where ξ is a random variable. To evaluate this expectation, N simulations are required. Robbins-Monro stochastic approximation algorithm uses a single noisy sample of the form

$$x_{k+1} = (1 - \alpha_k)x_k + \alpha_k L(x_k + w_k), \tag{3.1.5}$$

where $w_k = f(x,\xi) - E[f(x,\xi)]$. The stochastic approximation algorithm guarantees convergence of this algorithm under contraction or monotone assumptions of the mapping L *i.e., x_n converges to x^* with probability 1* subject to the assumptions

$$\sum_{k=0}^{\infty} \alpha_k = +\infty \text{ and } \sum_{k=0}^{\infty} \alpha_k^2 < +\infty. \tag{3.1.6}$$

Early reinforcement learning algorithms incorporate stochastic approximation into value iteration or policy iteration. Policy iteration with the value function is called temporal difference (TD), PI with the Q-function is called SARSA, while value iteration with the Q-function is called Q-learning. If we update value $V(x_t)$ the using costs at all states $x_t, x_{t+1}, \cdot$ generated by a given fixed policy with discount factor λ in the TD and SARSA methods, we denote these methods TD(λ) and SARSA(λ), respectively.

Because Bellman operator-based methods solve infinite-horizon discrete-time problems, however, applying these algorithms using the Bellman operator to continuous-time problems means solving the discretization approximated problems. Furthermore, we cannot solve finite-horizon problems exactly, because the convergence of TD, SARSA and Q-learning algorithms are guaranteed under the same conditions as Equation (3.1.6).

3.2 Approximation in Value and Policy Space

For small-scale discrete problems, the value function can be updated following the Robbins-Monro stochastic approximation for every state in tabular form. Although stochastic approximation circumvents calculation of the expectation, policy iteration and value iteration require the update of the value function for all possible states at each timestep. Because many practical engineering problems have a large-scale state space, the issue of scalability is important.

An appealing approach to resolve this drawback is an approximation in value space, which is often called adaptive dynamic programming. There are three types of approximation in value space methods: Problem approximation, On-line approximate optimization, and Parametric approximation [1]. The first method, including aggregation, obtains the optimal value functions of the simplified optimization problem. The second method uses a sub-optimal policy or heuristic obtained by other methods such as roll-out or model predictive control and approximate optimal values on-line. The last method obtains a sub-optimal solution in a given parametric class of functions. In this thesis, we only discuss parametric approximation methods to construct new RL taxonomy because the first method is problem dependent and second method is supplemental.

Parametric approximation methods are divided into direct and indirect methods. The direct method aims to solve the following least-squares problem:

$$\min_{\theta} \sum_{x \in X} (V(x;\theta) - V^{\pi}(x)) \tag{3.2.7}$$

where $V^{\pi}(x)$ is approximated as a cumulative sum of simulation cost data generated by given policy π. Deep RL algorithms such as deep Q network (DQN) come under the direct methods. Indirect methods approximate value function with a linear architecture of the form

$$V(x;\theta) := \phi(x)^{T}\theta, \tag{3.2.8}$$

where ϕ is a feature vector and θ is a weight vector. Direct methods solve the Bellman equation and find the sub-optimal solution near the optimal solution inside parametric function space, while indirect methods project the Bellman equation onto linear parametric function space and obtain the solution of the projected Bellman equation. Least-squares temporal difference (LSTD), least-squares policy evaluation (LSPE) and TD are indirect methods.

Another issue of the solving equation 3.1.1 is obtaining the cost minimizing policy. Similar to the calculation of value function, in discrete action space, optimal action can be determined by comparing values at the next state with respect to every action. This approach is called policy search, and applying policy search to the policy iteration is called ϵ greedy policy search. However, as we have noticed before, comparing every policy is impractical in large scale problems. Policy approximation by using a parametric function approximator was proposed to avoid this intense calculation. [39] introduced a useful theorem called stochastic policy gradient which states the relation between the gradient of value function and the gradient of parameterized policy function.

Theorem 3.1 (Policy Gradient Theorem)**.** *Under parameterized policy π_{theta}, define the cost function $\eta(\theta) := V^{\pi_\theta}(x_0)$. Let $d^{\pi_\theta}(x) := \sum_{t=0}^{\infty} \gamma^t P^{\pi}(x_t = x|x_0)$, where $P^{\pi}(x_t = x|x_0)$ is the transition probability from x_0 to x in t stages under policy π. Then,*

$$\nabla\eta(\theta) = \sum_{x\in X} d^{\pi_\theta}(x) \sum_{u\in U(x)} Q^{\pi_\theta}(x,u)\nabla_\theta \pi_\theta(u|x;\theta) \tag{3.2.9}$$

which can be rewritten as

$$\nabla\eta(\theta) = E_\pi \left[\gamma^t \sum_{u\in U(x)} Q^{\pi_\theta}(x,u)\nabla_\theta \pi_{theta}(u|x;\theta)|x_0 \right]. \tag{3.2.10}$$

This theorem allows us to update policy function parameters according to the gradient of expected value. [40] proved the deterministic version of policy gradient theorem and proposed deterministic policy gradient (DPG) method [40]. One critical drawback of policy gradient methods is that the policy is rapidly changed by updating according to the cumulative cost gradient. To address this issue, [41] proposed a conservative policy iteration scheme for mixed policy and [42] developed monotone improvement using KL divergence constraints between previous policy and new policy functions, which is called trust region policy optimization (TRPO). Proximal Policy Optimization achieved an efficient version of TRPO by only taking advantages of TRPO by clipping policy [43].

However, naive use of value obtained by the Monte-Carlo method causes high variance in the training process. It seems to be natural to integrate value approximation and policy approximation, and this approach is called approximation in policy space on top of approximation in value space. A more famous name is actor-critic methods, where actor means policy function and critic means policy evaluator, value function. Off-policy deterministic actor-critic (OPDAC) is based on the policy iteration methods with an approximation of Q-function by using nonlinear

approximator, and compatible off-policy deterministic actor-critic Q-learning/gradient Q-learning critic (COPDAC-Q/GQ) are using linear approximators [40]. A deep deterministic policy gradient (DDPG) is based on the value iteration methods with an approximation of Q-function by using deep neural networks [44].

It is natural to consider the other actor-critic methods that approximate value function, not Q-function. Generalized Advantage Estimation approximated another value-like function, advantage function, but it is proved that temporal difference residual becomes unbiased estimation of advantage function [45]. Soft actor-critic approximates the value function by using the policy iteration method with maximum entropy objective [46]. Importance Weighted Actor-Learner Architecture (IMPALA) proposed a V-trace method to address mismatch policy problems of the offline update by using importance sampling. This method is based on the policy iteration and approximate value function.

We have seen model-free RL algorithms and we remark that those methods can be classified according to what control methods are used (PI or VI), what kind of value-like function is approximated, which function approximator is used to approximate value-like function, and how to obtain optimal policy for each update step (policy search or policy gradient). In the next section, we propose a new taxonomy of RL algorithms according to those criteria. Before classifying RL algorithms, we briefly review some model-based algorithms with other control methods.

3.3 Taxonomy of Reinforcement Learning

In this section, we summarize other control methods used in RL and briefly review model-based RL algorithms. Then, we propose a new taxonomy summarizing RL

algorithms according to the choice of control method, value-like function, policy approximation, function approximator, and model learning method. As we have seen in chapter 2.2.2, continuous system stochastic optimal control problems can be formulated as the PDE form, HJB equation. Solving the Bellman equation effectively by employ learning schemes, RL also can integrate learning methods with continuous control methods solving the HJB equation. Owing to the dynamics requirement of the HJB equation, however, one more step is required, dynamics-learning. Therefore, only model-based RL can try to solve the HJB equation. We note that the model-based RL can be used in the Bellman equation approach. For example, probabilistic inference for learning control (PILCO) is a representative model-based RL algorithm based on the policy iteration method [47].

The dynamics model can be estimated from exploration data by fitting parametric function approximator in a supervised-learning fashion. Gaussian mixture model (GMM) and deep neural network are frequently used, and the Gaussian process (GP) also can be used. The dynamics-learning step of the PILCO approximates the dynamics model assumed to be a Gaussian distribution by propagating through GP models [47]. GPS uses GMM prior dynamics-learning methods, and stochastic optimal control with latent representations (SOLAR) uses a deep Bayesian linear-quadratic system model [48]. We will divide algorithms into four types according to the choice of model learning: free, GMM, GP, and deep neural network (DNN).

Although we can approximate the dynamics model from simulation data, solving the HJB equation is difficult. Therefore, conventional RL circumvents to solve HJB equation directly, but approximates [18, 19, 20] or transforms [49, 50, 51] the HJB equation as linear PDE form. We denote that [17, 16] proposed coarse model identification methods and integrated continuous control methods for only

the LQG problem which is the easiest version of the HJB equation.

Guided policy search algorithms [18, 19, 20] are based on iLQG methods. Assuming that dynamics are locally linear, iLQG, expends LQG to nonlinear models by reusing LQG iterative to refine the trajectory. The first step approximates the dynamics model to the first order and cost function to the second order. Then, we find one step optimal control input and next state by solving LQR and iteratively repeat LQG. For a more detailed explanation, we refer the reader to [52]. Path integral methods [49, 50, 51] circumvent the difficulty of solving the HJB-equation by transforming the HJB into a linear PDE. Exponential transformation of the value function allows applying the Feynman-Kac formula which connects the solution of PDE and solution of SDE. Path integral based RL algorithms can solve the HJB equation by simulating the stochastic differential equation and those methods are known as the most powerful methods for continuous problems.

Now, we provide a taxonomy of RL algorithms. Table 3.1 and table 3.2 summarize the choice of control problem and control method, value-like function, policy approximation, function approximator, and model learning method. First, control problems RL solved are divided into three types: (1) Discounted dynamic programming, (2) Average reward dynamic programming, (3) Continuous dynamic programming. We categorize control methods as policy iteration, value iteration, iLQG, robust control, and path integral with following notation T^{π}, T, iLQG, robust control, and PI, respectively. We indicate value function, state-value function, state-action value function as V, V^{π}, Q (or Q^{π} if we use policy iteration method). In the policy approximation column, PS and PG denote policy search and policy gradient, respectively. We classify the value approximation as the Monte-Carlo search, linear and nonlinear parametric approximation, and model category is classified as free, GMM, GP or DNN. We also provide an acronym description of a

taxonomy of reinforcement learning algorithm in table 3.3 and table 3.4.

Every reinforcement learning algorithm in table 3.1 is based on the DP methods solving discounted Bellman equation. There are also model-based algorithms including PILCO. Average reward DP is a reformulated problem of discounted DP with ergodic assumption. According to new taxonomy, most of the reinforcement learning algorithms have improved computation efficiency of DP methods to achieve scalability. However, using the Bellman equation involves one more approximation step which can explode approximation error through over multi approximation steps. Here, we remark that there is no converges guarantee of an approximate policy iteration. The approximate policy iteration method make rapid progress in the beginning, but it begins to oscillate. Furthermore, the generic error bound of the approximate policy iteration method and note that this error bound also diverge as a discounting factor goes to 1. In chapter 5, we apply reinforcement learning algorithms based on the control methods solving discounted DP to 1 dimensional LQG problems. We can obtain an analytical solution of this toy example by applying Itô calculus and we compare the real solution and the approximated value function of those algorithms.

To avoid these drawbacks, developing an efficient method to solve the HJB equation is better than improving discounted DP based reinforcement learning. From table 3.2, we can notice that there are only a few algorithms that attempt to solve the HJB equation. iLQG approximates nonlinear system optimal control problems pretty well, but we can identify the HJB equation with the same effort of dynamics learning. The path integral method transforms the HJB equation to a tractable linear PDE form and obtains the solution by applying the Feynman-Kac formula. However, this method involves stochastic process simulation to obtain a solution. We remark that the HJB approach for continuous system problems is

the blue ocean in reinforcement learning. In the next chapter, we propose a new reinforcement learning algorithm that solves not a modified HJB equation but the HJB equation directly. We include our new algorithm to a taxonomy at the bottom of the table 3.2.

Problem	Method	Value function	Policy approx.	Value approx.	Model	Algorithm
Discounted DP	T	Q	PS	MC	Free	Q-learning
				Nonlinear	Free	DQN
			PG	Nonlinear	Free	DDPG
	T^π	V^π	PS	MC	Free	TD, TD(λ), PEGASUS, REPS
				Linear	Free	TD, LSTD
				Nonlinear	Free	TD
			PG	MC	Free	TRPO, PPO, PGPE, IW-PGPE
				MC	GP	GPREPS
				MC	LSCDE	M-PGPE
				Linear	GP	PILCO
				Nonlinear	Free	GAE, SAC, IMPALA
		Q^π	PS	MC	Free	SARSA
				Linear	Free	TD, LSTD
				Nonlinear	Free	SARSA
			PG	MC	Free	DPG
				Linear	Free	COPDAC-Q/GQ
				Nonlinear	Free	OPDAC

Table 3.1: A taxonomy of reinforcement learning algorithms solving discounted DP

Problem	Method	Value function	Policy approx.	Value approx.	Model	Algorithm
Average Reward DP	T^π	V^π	PS	MC	Free	R / Modified-R learning
		Q^π	PS	MC	Free	Q-P learning
	T	V	PS	MC	Free	B learning, H learning
		Q	PS	MC	Free	G learning
Continuous DP	iLQR	V	PS		GP	AGP-iLQR
			PG		GMM	GPS
					DNN	SOLAR
	Robust control	V			DNN	Coarese-ID method
	PI	V	PG	MC	Kernel	PI^2 (-CMA/ CMAES)
	Deep PDE sover	V	PG	Nonlinear	GMM	New method

Table 3.2: A taxonomy of reinforcement learning algorithms solving other problems

T	Value iteration	PEGASUS	Policy Evaluation-of-Goodness And Search Using Scenarios
T^{π}	Policy iteration	TRPO	Trust region policy optimization
iLQG	Iterative linear quadratic Gaussian	PPO	Proximal policy optimization
PI	Path integral	PGPE	Policy gradient with parameter -based exploration
V	Value function	IW-PGPE	Importance-weighted PGPE
V^{π}	State-value function	GPREPS	Gaussian process relative entropy policy search
Q^{π}	State-action -value function	M-PGPE	Model-based PGPE
PS	Policy search	PILCO	Probabilistic inference for learning control
PG	Policy gradient	GAE	Generative adversarial
MC	Monte-Carlo	SAC	Soft actor critic
GMM	Gaussian mixture model	IMPALA	Importance Weighted Actor-Learner Architecture

Table 3.3: Acronym description in a taxonomy of reinforcement learning algorithms 1

GP	Gaussian process	SARSA	State-action-reward-state-action
DNN	Deep neural network	COPDAC-Q/GQ	Compatible Off-Policy Deterministic Actor-Critic Q-learning critic
DQN	Deep Q network	OPDAC	Off-Policy Deterministic Actor-Critic
DPG	Deterministic policy gradient	AGP-iLQR	Approximate iterative LQR with Gaussian Processes
DDPG	Deep deterministic policy gradient	GPS	Guided policy search
TD	Temporal difference	SOLAR	Stochastic optimal control with latent representations
LSTD	Least square TD	PI^2	Policy improvement with path integrals

Table 3.4: Acronym description in a taxonomy of reinforcement learning algorithms 2

4

Deep PDE Solver and Reinforcement Learning

4.1 Deep PDE solver

Consider the nonlinear PDE

$$\begin{cases} \frac{\partial u}{\partial t}(t,x) + \mathcal{N}[u](t,x) = 0, \quad (t,x) \in [0,T] \times \Omega \\ u(T,x) = q(x), \quad x \in \Omega \\ u(t,x) = b(t,x), \quad (t,x) \in [0,T] \times \partial\omega \end{cases} \tag{4.1.1}$$

Let $u_\theta(t,x)$ be a parametric function approximator of the solution $u(t,x)$, and F be an operator defined by $F[\psi](t,x) = \frac{\partial \psi}{\partial t}(t,x) + \mathcal{N}[\psi](t,x)$. Then $u_\theta(t,x)$ can be trained by minimizing the L^2 error

$$J(u_\theta) = ||F[u_\theta] - f||^2_{2,[0,T]\times\Omega} + ||u_\theta(T,\cdot) - g||^2_{2,\Omega} + ||u_\theta - b||^2_{2,[0,T]\times\partial\Omega}\,. \tag{4.1.2}$$

Approximation of the PDE solution by using deep learning have been doubted in two convergence senses: (1) $J(u_\theta) \to 0$ as $n \to \infty$ and (2) $u_\theta \to u$ as $n \to \infty$. Most

important question is whether $u_\theta \to u$ as $n \to \infty$ or not. Theorem 4.1 shows the convergence of the error [32]. Now, we introduce the theory of the PDE approximation with deep neural network, deep Galerkin method. Let n be the number of hidden units of neural network, and u_θ^n be a network with n hidden units. Define $\mathcal{C}^n$ be the class of neural networks with a single layer and n hidden neuron. For given activation function ψ, $\mathcal{C}^n$ can be defined as

$$\mathcal{C}^n(\psi) = \left\{ \phi(t,x) : \mathbb{R}^{1+d} \mapsto \mathbb{R} : \phi(t,x) = \sum_{i=1}^{n} \beta_i \psi \left(\alpha_{1,i} t + \sum_{j=1}^{d} \alpha_{j,i} x_j + c_j \right) \right\} \tag{4.1.3}$$

Theorem 4.1. *Let ψ be in $\mathcal{C}^2(\mathbb{R}^d)$, bounded and non-constant. Define $\mathcal{C}(\psi) = \bigcup_{n\geq 1} \mathcal{C}^n(\psi)$. Assume that Ω_T is compact and consider the measures μ_1, μ_2, μ_3 whose support is contained in Ω_T, Ω and $\partial\Omega_T$. Assume that the PDE 4.1.1 is quasi-linear parabolic and has a unique classical solution, and the nonlinear terms are locally Lipschitz in $(u, \nabla u)$ with Lipschitz constant that can have at most polynomial growth on u and p, uniformly with respect to t, x. Then, for every $\epsilon > 0$, there exists a positive constant $K > 0$ that may depend on $\sup_{\Omega_T} |u|, \sup_{\Omega_T} |\nabla_x u|$ and $\sup_{\Omega_T} \left|\nabla_x^2 u\right|$ such that there exists a function $f \in \mathcal{C}(\psi)$ that satisfies*

$$J(f) \leq K\epsilon$$

The later convergence is also proved under additional conditions, and we refer to theorem 7.3. in [32]. We note that those additional conditions are required to guarantee the unique bounded classical solution of the equation 4.1.1. Theorem 7.3 in [32] states that f^n converges to the unique solution strongly in $L^\rho(\Omega_T)$ for every $\rho < 2$. Furthermore, if the sequence $\{f^n\}_n$ is uniformly bounded in n and equicontinuous, then the convergence to the solution is uniform in Ω_T. We remark that

theoretical convergence guarantees are proved in the class of quasi-linear parabolic PDEs.

4.2 Reinforcement Learning based on Deep Galerkin Method

Our objective is to identify the stochastic HJB-equation from simulation data and approximate the solution of the stochastic HJB-equation using DGM. Unlike discrete system problems, the continuous system usually assumes the given reward function and terminal condition such as minimum energy or distance from the goal both on the optimal control and the reinforcement learning. We also assume that the reward function and terminal condition. In section 2.1.2., we have seen the stochastic HJB equation of the form

$$\begin{cases} \frac{\partial V}{\partial t}(t,x) + \min_{u\in U}\left\{\frac{\partial V}{\partial x}\cdot f(x,t,u) + \frac{1}{2}tr(\sigma\sigma^T)\,\mathrm{Hess}_x\, V + r(x,t,u)\right\} = 0 \\ u(T,x) = q(x(T)) \end{cases} \tag{4.2.4}$$

Unknown properties of our problem is the dynamics of the system f and σ so that our problem requires PDE identification from data. In addition, we must calculate minimum operations to solve the HJB equation. To obtain the value function, we need to calculate the minimizing control input before, but minimizing control input also depends on the value function. Similar to policy iteration methods, we resolve this issue by updating the value function and policy function iteratively. We can see our iterative method as a continuous system version of value iteration. Our algorithm involves three steps: (1) HJB identification, (2) Policy evaluation, and (3) Policy improvement.

Conventional PDE identification methods require data associated with the variable of PDEs. For example, velocity data of fluids is needed to identify Navier-Stokes equations. However, we cannot directly observe the value, the state-value or the state-action-value from simulation data. In RL communities, identification of the system dynamics is called dynamics learning. We employ the GMM prior dynamics learning methods [19] to identify the underlying HJB-equation. Our algorithm uses two networks to approximate the value function and policy function independently. We defined three operators as the following form

Definition 4.2.1. Given function ψ the HJB operator F is defined by

$$F[V_\theta](x,t) := \frac{\partial V_\theta}{\partial t}(t,x) + \min_{u\in U}\left\{\frac{\partial V_\theta}{\partial x}\cdot f(x,t,u) + \frac{1}{2}tr(\sigma\sigma^T)\operatorname{Hess}_x V_\theta + r(x,t,u)\right\}$$

and boundary condition operator B is defined by

$$B[V_\theta](x,T) = V_\theta(x,T) - q(x)$$

for given terminal condition q. Given policy π, we also define the PDE operator F^π as following form

$$F_\pi[V_\theta](x,t) := \frac{\partial V_\theta}{\partial t}(t,x) + \min_{\pi\in\Pi}\left\{\frac{\partial V_\theta}{\partial x}\cdot f(x,t,\pi(x)) + \frac{1}{2}tr(\sigma\sigma^T)\operatorname{Hess}_x V_\theta + r(x,t,\pi(x))\right\}.$$

We also denote the Hamiltonian operator H

$$H\left(u, V_\theta, \frac{\partial V_\theta}{\partial x}, \frac{\partial^2 V_\theta}{\partial x^2}\right) = \frac{\partial V_\theta}{\partial x}\cdot f(x,t,u) + \frac{1}{2}tr(\sigma\sigma^T)\operatorname{Hess}_x V_\theta + r(x,t,u).$$

A value network is trained by minimizing the loss function of the form

$$L_V = \frac{1}{N}\sum_{i=1}^{N} ||F[V_\theta](x_i,t_i)||_2^2 + \frac{1}{N_b}\sum_{j=1}^{N_b} ||B[V_\theta](x_j,t_i)||_2^2 . \tag{4.2.5}$$

It is difficult to find $\hat{u}^* \in U$ such that

$$\hat{u}^* = \min_{u\in U}\left\{H\left(u, V_\theta, \frac{\partial V_\theta}{\partial x}, \frac{\partial^2 V_\theta}{\partial x^2}\right)\right\}. \tag{4.2.6}$$

Therefore, we use an approximated policy, $\pi_\phi(x,t)$ as the minimizing control input in policy evaluation step. Policy network can be trained by minimizing following loss function

$$L_\pi = \frac{1}{N_\pi}\sum_{l=1}^{N_\pi} H\left(\pi_\phi, V_\theta, \frac{\partial V_\theta}{\partial x}, \frac{\partial^2 V_\theta}{\partial x^2}\right)(x_l.t_l). \tag{4.2.7}$$

With this policy network, we actually minimize following loss function in value function learning

$$L_V = \frac{1}{N}\sum_{i=1}^{N} \left|\left|F_{\pi_\phi}[V_\theta](x_i,t_i)\right|\right|_2^2 + \frac{1}{N_b}\sum_{j=1}^{N_b} ||B[V_\theta](x_j,t_i)||_2^2 . \tag{4.2.8}$$

Algorithm RL based on DGM

Input: # iterations K, # trajectories N_{exp}, Time steps N_T

Output: final value model V_θ^K, policy π_ϕ^K

$V_\theta(x.t), \pi_\phi(x,t) \rightarrow$ Initialize θ, ϕ

for `iteration` $k \in \{1, \cdots, K\}$ **do**

  (1) Dynamics learning

  for `iteration` $j \in \{1, \cdots, N_{exp}\}$ **do**

    Simulate π_ϕ and store $\mathcal{D} = \{x_i, t_i\}_{i=1}^{N_T}$

    Fit linear Gaussian dynamics $dx = f(x,u)dt + \sigma(x,u)dW$ using samples in $\mathcal{D}$

  end for

  (2) Policy Evaluation

  for $(x_i, t_i) \in \mathcal{D}$ **do**

    Calculate L_V (equation 4.2.8)

    Update $\theta_{k+1} = \theta_k - \eta \nabla_\theta L_V$

  end for

  (3) Policy Improvement

  for $(x_i, t_i) \in \mathcal{D}$ **do**

    Calculate L_π (equation 4.2.7)

    Update $\phi_{k+1} = \phi_k - \eta \nabla_\phi L_\pi$

  end for

end for

return V_θ^K, π_ϕ^K

5

Experiments

5.1 Experiment Environment

In this chapter, we apply RL algorithms based on control theories to solve DP and our algorithm to approximately calculate the solution of the finite-horizon continuous-time stochastic optimal control problem. We chose the minimum energy problem with simple dynamics because we can compare the approximated solution with an analytical solution. We consider a simple stochastic LQG control problem with system dynamics as the following form

$$\begin{aligned} dx &= 2udt + \sqrt{2}dW \\ r(x,t,u) &= u^2 \\ q(x) &= \ln\left(\frac{1+||x||^2}{2}\right) \end{aligned} \tag{5.1.1}$$

where $x \in X \subset \mathbb{R}^n$, $u \in \mathcal{U} \subset \mathbb{R}^n$ and $W \in \mathbb{R}^n$. Then, the value function follows the HJB-equation of the form

$$\frac{\partial V}{\partial t}(x,t) + \min_{u \in \mathcal{U}} \left\{ \frac{\partial V}{\partial x} \cdot u + \nabla \cdot \nabla V + u^2 \right\} = 0. \tag{5.1.2}$$

Applying Itô's formula, we an obtain the analytical solution of this HJB-equation.

Proposition. The minimizing control input of the HJB-equation 5.1.2 is given by

$$u = -\nabla V$$

and the exact solution admits the explicit formula

$$V(x,t) = -\ln\left(E\left[-q(x+\sqrt{2}W_{T-t})\right]\right)$$

We can compare the accuracy of the value function approximated by our algorithm with above exact solution by error of the form

$$E_V = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} ||V_\theta(x_i,t_i) - V(x_i,t_i)||_2^2 \,. \tag{5.1.3}$$

Value function and policy was represented by deep neural networks with 7 layers, with 50 nodes per layer, with tangent hyperbolic nonlinearity. Before the policy iteration step, the boundary condition fitting step ran for 100 iterations, with 100 numbers of uniformly sampled boundary points per each iteration. The policy iteration ran for 20 iterations, with 15 times explorations. We modified a network architecture used in [32] by adding residual terms in every LSTM like layers.

5.2 Experimental Results

First, as we have mentioned in chapter 3.3, we examine the value approximation performance of conventional RL algorithms based on the control methods solving

discounted DP. We compare value functions approximated by DDPG, Twin delayed deep deterministic policy gradients (TD3), SAC, TRPO and PPO with the analytical solution in figure 5.1. Because those RL algorithms solve infinite-horizon problems which have a time-invariant solution, it cannot approximate the finite-horizon solution. Even in a simple 1-dimensional LQG problem, figure 5.1 shows that DDPG, TD3 and SAC cannot approximate value function. TRPO and PPO are better than others, but those algorithms fail to find an optimal control solution because of local optimum. The goal position of this example is the origin, but the value function approximated by TRPO and PPO indicate other positions as an optimal point. We note that the analytical solution is not smooth, because we calculate the analytical solution by simulating the stochastic differential equation.

In contrast to conventional RL algorithms, our new scheme can approximate the real solution of optimal control problems. We exhibit the results of 1-dimensional problem where $x \in X = [0, 10]$ and $t \in [0, 10]$ in figure 5.2. As we have seen in section 5.1, the optimal policy is proportionate to the minus gradient of the value function. We note that other stochastic optimal control examples also have the optimal feedback controller in proportion to the gradient of value function. Solutions obtained by TRPO and PPO have a declining trend as going to the origin, but the slope of solutions is steeper than a real solution which means those algorithms cannot obtain the optimal feedback controller.

On the other hand, figure 5.2 shows that our algorithm can approximate the solution exactly. Because DGM can use sample points of terminal condition as much as we want (we uniformly sampled 100 points per iteration in $[0, 10]$), the network exactly fits the terminal condition. Furthermore, the new algorithm can obtain the solution of the finite-horizon problems, that is time-varying. The terminal condition stabilizes the solution so that approximation is more accurate near

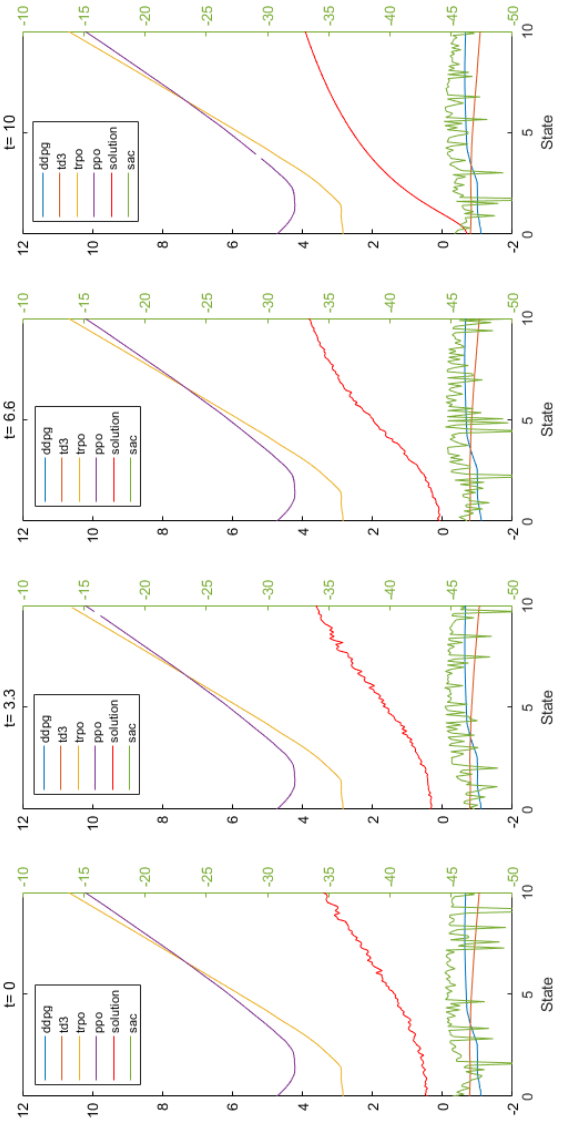


Figure 5.1: 1-dimensional LQG experiment results of conventional RL algorithms. DDPG, TD3, TRPO, PPO and analytical solution correspond to the left y-axis, and SAC corresponds to the right y-axis.

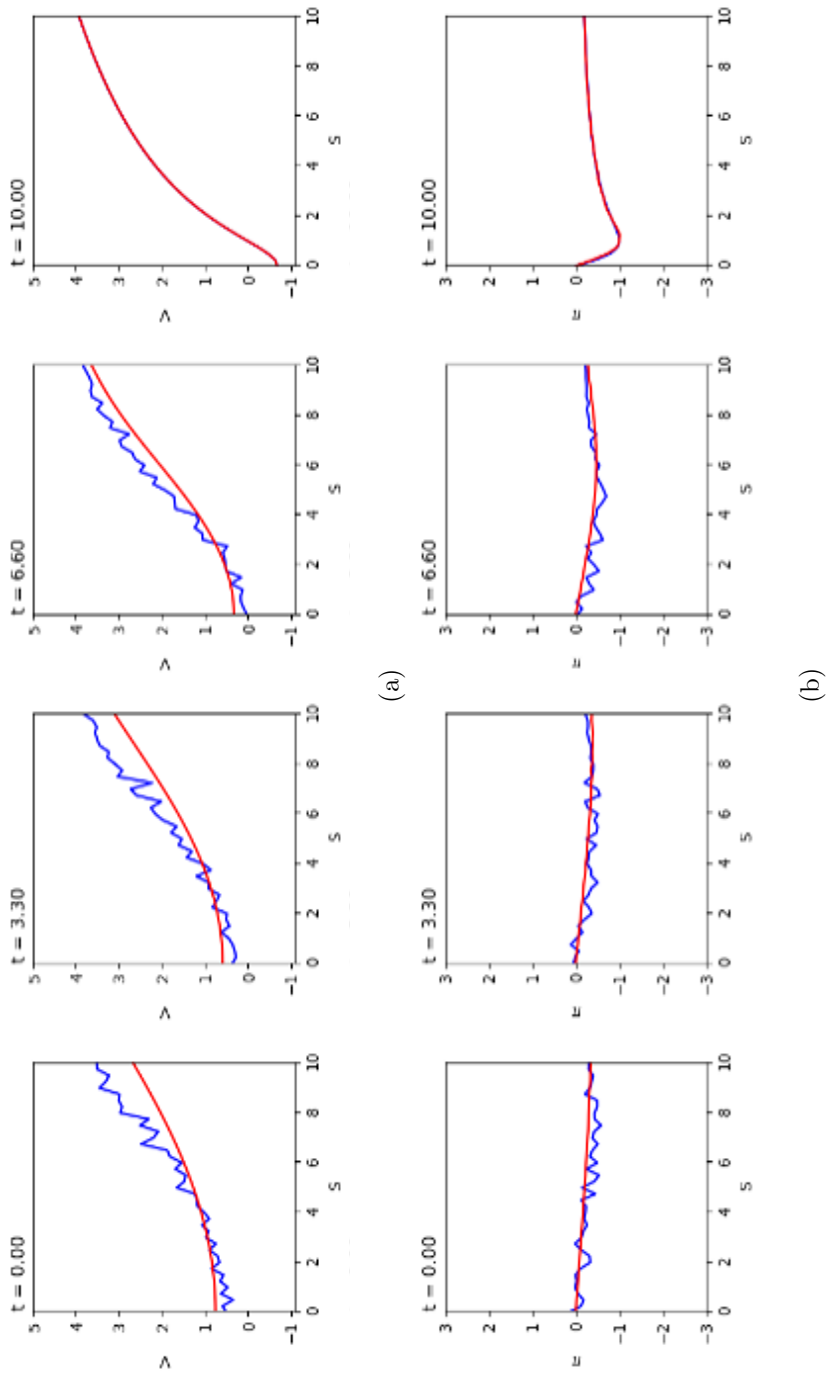
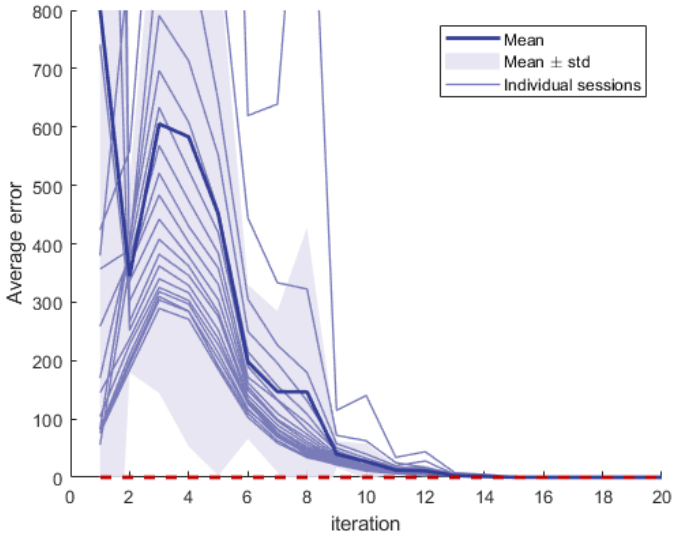


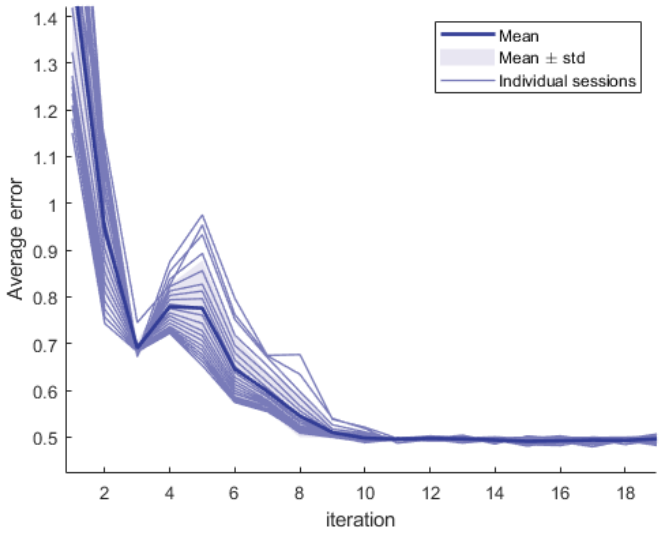
Figure 5.2: 1-dimensional LQG experiment results of DGM based RL algorithm. Comparison between analytical solution and network output of (a) the value function and (b) the policy function; red line: network approximation, blue line: analytical solution.

the terminal time.

We also execute experiments on high dimensional problems. Figure 5.3 shows the value loss and the relative error with respect to the iteration numbers for the 12-dimensional case. We calculate mean square error between approximated value function and analytical solution at 1000 uniformly random sampled points in domain $[0, 10]^{12}$. Our method achieves relative error of 0.51%. Figure 5.3 (a) shows that value loss has high variance but it converges within 15 iterations. Figure 5.3 (b) exhibits the error of the value function, E_V. We can see that the value error converges within 12 iterations.



(a) Loss of the value function per iteration(L_V)



(b) Mean square error between analytical solution and approximated value function

Figure 5.3: 12-dimensional LQG experiment results of DGM based RL algorithm

6

Conclusion

RL algorithms are classified according to whether the update occurs on-line or off-line, and whether it is model-based or model-free . Off-line methods computes entire value function before the first control is applied, while on-line methods compute the values just after the current state becomes known. Model-based methods assumes that the transition probability of system for any state is available, while model-free case is done with Monte Carlo simulation.

[1] provided the more informative classification of reinforcement learning techniques according to what they approximate. This classification contains more information about the technique used in the algorithm than simply dividing it into using a model or not, making it easier to select an algorithm that fits the situation. This thesis proposed a more refined taxonomy that shows which control problems are solved and which control theory is based by each RL algorithm.

For discrete time and space problems, RL should solve the Bellman equation using discrete dynamic programming methods. However, there are two approaches

to solving continuous-time optimal feedback control problems: (1) Solving the Bellman equation using discrete control methods via function approximation, and (2) Solving the Hamilton-Jacobi-Bellman equation.

Most RL algorithms follow the former approach. However, solving the Bellman equation via approximation does not guarantee convergence to the solution. Moreover, this approach involves an additional approximation step involving discretization of space and time, leading to accumulation and explosion of approximation errors over multiple steps.

While some model-based RL algorithms attempt to solve the Hamilton-Bellman-Jacobi (HJB) equation in modified form, as of yet there is no RL algorithm that attempts to directly solve the HJB equation. Nonlinear PDEs can be solved by numerical methods, but heavy computation is required. In this thesis, we proposed a new RL algorithm that solves the HJB equation directly by using deep PDEs solver, DGM.

However, DGM is not a perfect method to solve every PDE. DGM can be applied to hyperbolic, elliptic, and partial-integral differential equations but other types are not guaranteed yet. We expect that future improvements of deep-PDE solver will bring success in RL problems, too. Stability analysis of the learning process and efficient exploration are also important problems, but we leave this room for improvement for future work.

Bibliography

[1] Dimitri P Bertsekas. *Reinforcement learning and optimal control*. 2019.

[2] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[4] Peter Dayan. Navigating through temporal difference. In *Advances in neural information processing systems*, pages 464–470, 1991.

[5] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[6] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[7] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[8] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 593–600, 2005.

[9] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[10] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[11] Ronald A Howard. Dynamic programming and markov processes. 1960.

[12] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[13] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[14] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[15] Prasad Tadepalli and DoKyeong Ok. Model-based average reward reinforcement learning. *Artificial intelligence*, 100(1-2):177–224, 1998.

[16] Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. On the sample complexity of the linear quadratic regulator. *arXiv preprint arXiv:1710.01688*, 2017.

[17] Stephen Tu, Ross Boczar, Andrew Packard, and Benjamin Recht. Non-asymptotic analysis of robust control from coarse-grained identification. *arXiv preprint arXiv:1707.04791*, 2017.

[18] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[19] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[20] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[21] Alfio Quarteroni and Alberto Valli. *Numerical approximation of partial differential equations*, volume 23. Springer Science & Business Media, 2008.

[22] Aslak Tveito and Ragnar Winther. *Introduction to partial differential equations: a computational approach*, volume 29. Springer Science & Business Media, 2004.

[23] MWMG Dissanayake and N Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[24] B Ph van Milligen, V Tribaldos, and JA Jiménez. Neural network differential equation and plasma equilibrium solver. *Physical review letters*, 75(20):3594, 1995.

[25] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[26] Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

[27] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[28] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[29] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.

[30] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[31] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[32] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[33] Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[34] E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[35] Christian Beck, E Weinan, and Arnulf Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, pages 1–57, 2017.

[36] Shige Peng. Stochastic hamilton–jacobi–bellman equations. *SIAM Journal on Control and Optimization*, 30(2):284–304, 1992.

[37] Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer, 2003.

[38] Dimitir P Bertsekas and Steven Shreve. *Stochastic optimal control: the discrete-time case*. 2004.

[39] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[40] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[41] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

[42] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[45] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[46] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[47] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[48] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine. Solar: deep structured representations for model-based reinforcement learning. *arXiv preprint arXiv:1808.09105*, 2018.

[49] Hilbert J Kappen. An introduction to stochastic control theory, path integrals and reinforcement learning. In *AIP conference proceedings*, volume 887, pages 149–181. American Institute of Physics, 2007.

[50] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning policy improvements with path integrals. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 828–835, 2010.

[51] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *journal of machine learning research*, 11(Nov):3137–3181, 2010.

[52] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

국문초록

본 논문에서는 제어 이론적 관점에서 강화 학습 알고리즘의 분류법을 제안하고, 연속적인 시스템에서의 최적 제어 문제를 풀기위한 새로운 강화 학습 방법을 제안한다. 강화 학습 알고리즘을 모델 기반 알고리즘과 모델을 사용하지 않는 알고리즘으로 나누는 대신, 우리는 강화 학습 알고리즘이 어떤 방정식을 해결하느냐에 초점을 맞춘다. 후자의 관점에 따르면 연속적인 시스템 문제를 해결하기 위한 두 가지 접근방식이 있는데, (1) 이산적인 시스템의 동작 계획 방법으로 벨만 방정식을 푸는 방식에 딥러닝 같은 함수 근사 기법을 더하여 연속적인 시스템의 근사해를 구하는 방법, (2) 편미분 방정식 형태의 벨만 방정시인 해밀턴-자코비-벨만 방정식을 푸는 방법이다. 대부분의 강화 학습 알고리즘은 첫 번째 접근법을 따르고 있지만, 근사치로 벨만 방정식을 풀 때 해의 수렴성이 보장되지 않는다. 또한, 이 접근방식은 연속적인 시스템을 이산적으로 쪼개는 하나의 근사 단계를 더 수반하여, 근사 오차가 쌓일 수 있다. 몇몇 모델 기반 강화 학습 알고리즘 중에는 해밀턴-자코비-벨만 방정식을 선형 편미분방정식 꼴로 수정하여 풀거나 비선형시스템을 부분적 선형시스템으로 가정해서 푸는 방법이 있지만, 해밀턴-자코비-벨만 방정식을 직접 푸는 강화 학습 알고리즘은 없다. 본 논문에서는 딥러닝을 이용해 편미분방정식을 푸는 방법인 심층 갤러킨 방법에 기초하여 새로운 강화 알고리즘을 제안한다. 우리의 새로운 알고리즘은 해밀턴-자코비-벨만 방정식을 재조정이나 단순화 없이 해결한다. 실험부분에서는 첫 번째 접근법에 따른 기존의 강화 학습 알고리즘과 새로운 알고리즘을 단순한 1차원 최적 제어 문제에서의 근사 성능을 비교하고, 또한 고차원 문제에서 새로운 알고리즘을 테스트한다.

주요어: 확률적 최적 제어, 동작 계획 방법, 벨만 방정식, 해밀턴-자코비-벨만 방정식, 강화학습, 심층 갤러킨 방법

학번: 2018-25735