清 华 大 学

综 合 论 文 训 练

题目：HAAR: 基于优势函数辅助奖励的分层强化学习算法

系　　别：机械工程系

专　　业：机械工程

姓　　名：王芮

指导教师：雷丽萍副教授

2019 年 5 月 26 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____ 导师签名：_____ 日　期：_____

中文摘要

强化学习问题当中的一个重要挑战就是一些环境中的稀疏奖励问题。环境反馈给智能体的信号过于稀疏会导致智能体学习很慢。为此，人们提出分层强化学习的方法，来帮助强化学习算法提高采样效率，加速学习。在分层强化学习范式当中，特别地，我们需要为下层策略设计一个辅助奖励函数，从而促进下层策略的学习。在这篇论文中，我们提出了一个分层强化学习的框架，通过一个基于优势函数的下层奖励函数设计，它可以单调地同时优化上下层策略。截止目前，绝大多数最前沿的分层强化学习算法都对状态空间的表达非常敏感，或者依赖于手动设计一个只在特定环境下可用的下层奖励函数。得益于我们的辅助奖励函数设计只与优势函数的值相关这一特点，我们的算法不依赖于状态空间的表示方式，因此和其他方法相比则更为通用和实用，尤其是在机器人连续动作空间这类应用场景中。我们从理论上证明了我们的方法可以确保上下层整体策略的单调优化特性。实验方面，在 Mujoco 仿真环境的多个任务中，我们的算法表现大大优于现有的最佳算法。最后，我们还搭建了一个自动驾驶仿真环境来验证我们的算法在更现实和复杂的环境下的表现。我们把算法成功从原来的实验环境迁移到了自动驾驶环岛路径规划问题中，并汇报了我们的结果。同时我们也指出了在这个环境中的下一步研究方向。

关键词：强化学习；分层强化学习；机器人学；规划算法；自动驾驶；路径规划；优化；策略单调性

ABSTRACT

One of the key challenges in reinforcement learning is related to sparse reward signals from the environment. Hierarchical reinforcement learning (HRL) can be utilized to solve the credit assignment problem in a sparse reward task. In the HRL paradigms, we need to design a low-level auxiliary reward function to facilitate low-level learning. In this paper, we propose an HRL framework which monotonically optimizes the joint policy of the upper-level and lower-level through the design of an advantage function-based auxiliary reward. Most state-of-the-art HRL algorithms are sensitive to state representations, or require careful design of the auxiliary reward functions for low-level training. Our method is more generic and applicable in continuous control tasks since our auxiliary reward for low-level policy update is dependent only on the advantage function of the high-level policy, therefore invariant of the state representations. We theoretically prove that monotonic improvement of the joint-policy is guaranteed in our method. Experimental results show that our algorithm performs dramatically better than other state-of-the-art methods across multiple tasks in the Mujoco learning environment. Finally, we design a complex autonomous-driving simulation environment as a testbed for our algorithm in more realistic settings. We transfer our algorithm to this new environment and report the experiment results of our framework on the self-driving car's planning task at a roundabout, as well as point out the direction for future research.

Keywords: Reinforcement Learning; Hierarchical Reinforcement Learning; Robotics; Planning Algorithms; Self-driving; Path Planning; Optimization; Monotonic Improvement

目 录

主要符号对照表

s	状态（任意状态）
a	行动（任意行动）
π	策略
$\pi(s)$	策略函数
$\pi(a\|s)$	策略在状态 s 下采取行动 a 的概率
r	奖励
$r(s,a)$	在状态 s 下采取行动 a 获得的奖励
G_t	时间 t 之后的总奖励
$v(s)$	状态 s 的价值函数
$q_\pi(s,a)$	在策略 π 下，在状态 s 下采取行动 a 的价值
$V(s)$	状态 s 的价值函数的估计值
$Q(s,a)$	状态 s，动作 a 的价值的估计值
$p(s'\|s,a)$	在状态 s 采取动作 a，状态转移到 s' 的概率
$p(s',r\|s,a)$	在状态 s 采取动作 a，状态转移到 s' 并且获得奖励 r 的概率
$\pi(a\|s,\theta)$	策略函数的参数是 θ，在状态 s 下采取行动 a 的概率
π_θ	策略函数，其参数为 θ
$\nabla\pi(a\|s,\theta)$	策略函数的参数是 θ，在状态 s 下采取行动 a 的概率对 θ 的导数
γ	总奖励 G_t 计算时，对于后续 R_{t+t_i} 的折扣率
ρ	重要性采样比例（importance-sampling ratio）
δ_t	时序差异误差（TD error）
RL	强化学习（reinforcement learning）
MPC	模型预测控制（model-predictive control）
TD	时序差异（temporal-difference）
MDP	马尔可夫过程 (Markov Decision Process)
POMDP	部分可观测的马尔可夫过程 (Partially Observable Markov Decision Process)
DQN	深度 Q-学习神经网络（Deep Q Network）
DRQN	深度循环 Q-学习神经网络（Deep Recurrent Q Network）

SNN	随机神经网络（Stochastic Neural Network）

第 1 章　绪论

近年来，在人工智能领域，强化学习得到了极为广泛的重视和研究。强化学习是一种常见的人工智能范式，旨在解决智能体和环境交互中的决策问题。在这篇论文中，我们为解决一类特殊的问题（‘‘稀疏奖励问题’’）而提出了一种分层强化学习的架构。我们提出了一个新的分层强化学习算法 ‘‘基于优势函数辅助奖励的分层强化学习”(Hierarchical reinforcement learning with Advantage-based Auxilliary Reward, HAAR)，在强化学习社区常用的基准实验环境（Benchmark[1]）中进行了实验测试，在相同的实验环境下与其他算法对比，取得了当前学术界的最佳成绩。在此基础上，我们把这套算法应用于自动驾驶仿真环境，解决了一个自动驾驶路径和速度规划的问题。

接下来，我们将会首先介绍强化学习的一般问题，其中将重点介绍分层强化学习这一重要分支，并由此引出强化学习中有待研究的问题，指出我们具体研究的问题及其意义。

1.1　强化学习预备知识

在强化学习的著作 *Reinforcement Learning An Introduction*[2] 中，Sutton 对强化学习的理论进行了详尽的讨论。在这里，我们对强化学习的发展做一个精炼的综述，并从契合课题的框架进行一些讨论。另外，我们还将详细讨论强化学习的一些最新进展和它们在本课题中的应用。

1.1.1　问题描述

Sutton 在 1988 年首次提出强化学习问题的一般描述[3]。诸如学习走路、学习下棋的一般性问题都可以归为强化学习的问题。强化学习问题的四个基本元素是：一个策略（policy），一个奖励函数（reward），一个价值函数（value function）和一个环境模型（model）。例如，自动驾驶问题中，智能体面对它观察到的一个周边车流的**环境**，会给出一个输出，例如以某一加速度加速。在采取这一行动之后，智能体会得到一个**奖励**（可能有延迟），例如它到达了目的地，我们给它价值 100 的奖励。而如何决定加速度，就是我们想要智能体去学习的**策略**。在制定策略的过程中，智能体会去试图衡量目前所处状态的**价值**（价值函数），例

如周围车辆非常拥堵，这个状态的价值就比较低，是我们希望避免进入的状态；而车前非常空旷则可能是我们赋予价值较高，希望进入的状态。

在这里，我们将用一些研究者所默认的记号对我们的问题进行数学抽象。假设下标 t 表示任一时刻，而 S_t 代表某一时刻的状态，A_t 表示智能体采取的行动，π 表示智能体的策略，R_t 表示智能体某一步获得的奖励。我们的目标是让智能体通过学习，获得尽可能大的总奖励，记为 G。有时我们可以定义

$$G = \sum_t R_t$$

但是多数时候我们会考虑加入一个折扣 $\gamma(0 < \gamma < 1)$，认为一个行为直接获得的奖励对它更有意义，而很久以后获得的奖励与这个行为的关联越来越小，记为

$$G_{t_0} = \sum_{t=t0}^{t_n} \gamma^{t-t_0} R_t \tag{1-1}$$

其中 t_n 要么是一个过程因为某种原因终止的时刻，要么是 ∞

强化学习解决的是一个马尔可夫决策过程（MDP）[4]。在 MDP 中，S_{t+1} 只取决于 S_t 和 A_t，而不受到之前历史过程的影响。也就是说，S_t 状态本身就包含了它所有的历史对后续过程的影响。在这个假设下，我们可以这样表达强化学习的目标[5]：记 $v^\pi(s)$ 为状态 s 的价值函数。我们有

$$v^\pi(s) = E[G_{t_0}|S_{t_0} = s, \pi(s)]$$

$$J(\pi) = \max_\pi E(v^\pi(s))$$

$$\pi^*(s) = \operatorname{argmax}_\pi J(\pi)$$

强化学习的终极目标就是找到这个 $\pi^*(s)$。Bellman 在 *Dynamic Programming* 一书中提出了著名的 Bellman 最优性等式[6]

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{1-2}$$

同时，我们用 $q(s, a)$ 表示在状态 s 下采取行动 a 可以获得的价值，那么 Bellman 最优性等式又可以表示为

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{1-3}$$

这两个表达式在后续的推导中会非常有用。

1.1.2 强化学习典型算法综述

强化学习包含许多的算法，也有很多分类方法。其中，有一大类是列表方法，可以理解为是把 $v(s)$ 保存起来的方法；另一大类是近似方法，也就是用一个函数去近似 $v(s)$[2]。其中，第二类方法在近年来使用最多，神经网络使得函数可以近似非常复杂的映射关系，也让许多复杂问题通过强化学习得到了解决。

列表式的强化学习方法包括动态规划[6]，蒙特卡罗方法，时序差分学习（Temporal-difference learning，简称 TD 方法，包括 Sarsa[2], Q-learning[7])[3]，多步自助法[8]（n-step bootstrapping），Dyna 规划方法[2] 等。

TD 方法是列表法强化学习的核心。这一族的算法有许多衍生版本，但其核心是利用经验不断更新价值函数 $V(s)$，直至其收敛。更新的基本模式为

$$V(s) \leftarrow V(s) + \alpha[r + V(s') - V(s)]$$

其中，α 表示函数更新的步幅。而当我们用 $Q(s,a)$ 去替代 $V(s)$ 的时候，算法就成为了 Sarsa（state, action, reward, state, action[9]）:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

注意这里，a' 是依据当前策略 π 采取的行动。而当我们把 $Q(s',a')$ 替换成 $\max_{a'} Q(s',a')$ 的时候，算法就成为了 Q-学习法（Q-learning）:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1-4}$$

在这里，我们发现 Sarsa 和 Q-学习的重要区别：Sarsa 是同策略学习的（on-policy），估算 Q 值时采用的策略就是当前策略，而 Q-学习是采用异策略（off-policy）学习的，估算 Q 值时采用的策略不是当前策略，而是最优策略。同策略与异策略也作为强化学习中的两种并行方法，本身并不改变算法架构，但是会影响收敛性等特性。许多算法都可以在这个维度上进行变异。

注意到在以上的算法中，我们每一步都对 Q 或者 V 进行更新。在多步骤自助法（n-step bootstrapping）中，我们则将更新进行延迟。例如，某一行动结束，n 步过后，我们才观察这一行为造成的影响，并且对它的值进行更新，其更新方式如下

$$V(S_t) \leftarrow V(S_t) + \alpha(G_{t:t+n} - V(S_t))$$

其中，$G_{t:t+n} = \sum_{i=1}^{n} \gamma^{i-1} R_{t+i} + \gamma^n V(S_{t+n})$

Sutton 等人指出，多步骤自助法强化学习在训练时的表现往往比蒙特卡罗和 TD 方法都要更优[2]。

归功于深度强化学习的提出[10][9]，近似函数的方法在近年来得到了极大的发展。近似函数有两种基本思路。第一种，是用某一类型的函数去近似状态的价值函数 $v(s)$，从而帮助智能体根据不同的 v 采取不同的策略。第二种思路是跳过 v 的计算，而直接使用函数去近似一个策略 π。在第二种思路中，$\pi(s)$ 的输出将直接是智能体应该采取的行为（或者行为的一个概率分布）。这种方法被称为 ‘‘策略优化’’，围绕策略优化问题衍生出一系列的策略梯度求解方法，成为近年来强化学习的热点研究方向之一。进一步的，在策略优化问题中，如果我们同时对价值函数 $v(s)$ 进行近似和更新，那么可以得到一类框架，我们称之为 actor-critic 架构[2]。在后续的篇目中，我们会对此进行具体介绍。

在深度强化学习中，比较著名的算法有 DDPG[11]，A2C[12]，TRPO[13]，ACKTR[14]，ACER[15]，PPO[16] 等。这些算法全部忽略了环境的模型，或者环境模型是学习得到的（TRPO）。还有一类算法，其中的环境模型是提前设计建模得到的而非习得，其中有代表性的就是 AlphaZero[17]。这类算法也是我们想探究的方向——是否通过提前建模，能够让自动驾驶车辆在环岛这类复杂环境中有更好的表现? 在接下来的章节中，我们会详细介绍其中有代表性，并且在我们的场景下易于使用的算法。

另外，在对价值函数进行优化时（梯度下降），不同的算法可能带来不同的结果。A3C 采用了 RMSProp 算法[18]，我们往往还可以选择随机梯度下降法（SGD），带动量的随机梯度下降法[19] 等方法，在算法设计时需要进行尝试和选择。Sutton 在书中对此进行了讨论[2]。此外，Ruder 在 2016 的一篇综述文中对各种梯度下降算法进行了详尽的探讨[20]。

1.1.3 无模型的深度强化学习

无模型的深度强化学习是我们在本课题中的重要算法基础之一。其中，2017 年被提出的 TRPO 算法[13] 是我们在本课题中的重点。基于 TRPO，我们构建了分层结构，并且基于 TRPO 中策略单调提升的结论，我们证明了 HAR 算法的单调性。因此接下来我们将介绍无模型的深度强化学习问题及其典型算法。

Mnih 等人在 2013 年首度提出深度 Q-网络（deep Q-network，简称 DQN）[10]，2015 年，他们的另一篇论文同样描述了 DQN[21]，激发了一轮深度神经网络与

强化学习结合的热潮[9]。DQN 的基本思路是创造一个很一般的端到端的网络 $Q(s,a|\theta)$，其中 θ 是这个网络的参数。例如对于一个游戏，网络的输入直接就是未处理的图像，输出直接就是在某个阶段，游戏中的动作对应的价值。在传统 Q-learning 的框架下，DQN 在每一步更新 $q(s,a)$ 值的时候，不是直接使用 (1-4)，而是利用梯度下降法去更新网络 Q 的参数 θ，使得 Q 估值总损失最小

$$\nabla_{\theta_i} Loss(\theta_i) = E_{s,a,s'}[(r + \gamma \max_{a'} Q(s', a'|\theta_{i-1}) - Q(s,a|\theta_i)) \nabla_{\theta_i} Q(s,a|\theta_i)]$$

这里 DQN 的网络结构参考了 Hinton 等人提出的 AlexNet[22]，也就是典型的卷积神经网络（CNN）结构。DQN 的最大贡献在于，由于使用了经验回放的方法，DQN 可以收敛，从而使得深度神经网络在强化学习中的应用成为了可能。由于神经网络的高度非线性，DQN 得以模拟足够复杂的场景。DQN 在 Atari 系列游戏中取得了突破性的进展，甚至在一些游戏中超越了人类专家的水平。

不过由于我们想探究的问题更多地涉及到低维度的传感器信息输入，而不是高维度的图像直接输入，因此 DQN 一类的方法对我们的帮助并不是很大。而策略梯度方法则对我们有很大的借鉴意义，因此我们也将集中讨论这一类的方法。

前面我们提到，在策略优化问题中，有一类架构被称为 actor-critic 架构。Mnih 等人在 2016 年提出的 A3C（asynchronous advantage actor-critic）算法[12] 成为应用 actor-critic 架构的经典案例。A3C 算法同时维护一个策略函数 $\pi(a|s;\theta)$ 和一个价值函数的估计 $V(s;\theta_v)$，这两个函数均为卷积神经网络 CNN，并且两个网络除了输出层以外的层均是共享的。其中，$\pi(a|s;\theta)$ 称为 actor，而 $V(s;\theta_v)$ 称为 critic。actor-critic 会计算一个 ‘‘优势’’，定义为 $\delta = R + \gamma V(S';\theta_v) - V(S;\theta_v)$[2]。A3C 借鉴了多步骤自助法的思路（设步数为 k），在计算优势时，变异为 $\delta_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(S_{t+k};\theta_v) - V(S;\theta_v)$。每一步，我们依然对 θ 和 θ_v 进行更新，梯度的计算方法如下（注意计算梯度以后，还需要选择一个梯度下降方法对 θ 和 θ_v 进行更新）

$$d\theta = \delta_t \nabla_\theta log_\pi(A_t|S_t;\theta) \tag{1-5}$$

$$d\theta_v = \delta_t \nabla_{\theta_v} \delta_t^2 \tag{1-6}$$

如果我们采用多个智能体并行训练，把它们各自的梯度叠加在一起（也就是累加每个智能体各自求得的 (1-5)(1-6) 式），最后统一更新 θ 和 θ_v，就称之为异步训练方法，这种算法也因此被称为 asynchronous advantage actor-critic。如果条件

受限没有异步，而只是用一个智能体进行训练，这个算法就称为 A2C（advantage actor-critic）。actor-critic 架构被认为是能够有效加速训练的，因此越来越多的算法都基于这一架构进行设计。

2015 年，Schulamn 等人提出了经典的（Trust Region Policy Optimization, TRPO）算法[13]。在文章中，他们首先从理论上证明了一种策略迭代更新的 MM（minorization-maximization）算法，可以确保策略对应的总价值（$\eta(\pi) = E_{s_0}[\sum \gamma^t r(s_t); \pi_\theta]$）单调不下降。依据这一理论基础，他们进行了多步近似，将 MM 算法中的优化（求 argmax）的问题转化为了一个更简单的加限制的优化问题（信任空间优化，trust-region optimization）。在进行进一步近似后，优化式中的期望形式可以用蒙特卡罗方法进行估算。最后，优化式本身则利用一种近似的方法进行优化。在实验中，他们采用了神经网络作为策略 π 的表达。TRPO 的更新方法是：

$$\begin{aligned} \theta &= \underset{\theta}{\operatorname{argmax}}\, E\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{\theta_{old}}(s,a)\right] \\ &\text{subject to } E[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))] \le \delta \end{aligned} \tag{1-7}$$

其中，δ 是一个认为选定的小值。注意，在论文的附录中，作者提出采用一阶近似对上式进行求解，优化问题就可以就变为一个典型的拉格朗日共轭问题[23]。不过，由于 TRPO 只是保证了每一步更新，策略都变得更好，却没有考虑探索性，因此，TRPO 的结果很容易困在局部最优值。在 Schulman 之后，Wu 等人于 2017 年又提出了一个改进算法 ACKTR[14]（Actor Critic using Kronecker-Factored Trust Region），采用 actor-critic 架构，并利用 Kronecker 因子去近似梯度下降。同年，Schulman 等人又提出了 PPO 算法[16]，在实现难度和算法表现方面取得了平衡。PPO 算法的基本思路有三点：

- 去除 TRPO 的优化限制条件，在 TRPO 的基础上，采用一个 clip 函数，从而阻止 θ 每一次更新太大，确保收敛
- 在优化的函数里，通过引入系数的形式，同时优化 actor 和 critic，整个优化过程只有一个优化目标函数
- 通过在优化目标中加入一个熵奖励，增加算法的探索性

PPO 把强化学习问题转化为一个形式简单的单一函数优化问题，在许多方面取得了比以往算法（A2C, ACER, TRPO, etc.）更优的表现。

在 2015 年，Silver 首度提出了针对连续动作空间的确定性算法 DPG（deterministic policy gradient）[24]。与确定性算法对应的是随机算法，也就是策略是行动

的概率分布。在高维的动作空间内（例如动作是连续的），DPG 比与之对应的随机算法获得了更优的结果。DPG 采用 actor-critic 架构，并且是异策略（off-policy）训练的。注意这里，DPG 的 actor-critic 架构与 Sutton 书中提到的 actor-critic 架构有所区别。它是不以价值函数作为基准的，而正常的 actor-critic 架构应该以价值函数作为基准，计算优势，再进行求导[2]，参见前面提到的 A3C 算法[12]。同时，DPG 算法采用的是 Q-learning 结构，不是用 $V(s)$ 对价值进行估计，而是用 $Q(s,\pi(s))$ 对价值进行估计。由于算法本身是确定性的，DPG 与随机异策略 actor-critic 架构[25] 不同，不需要针对策略进行重要性采样（importance sampling，在[2] 中有详细讨论）。DPG 本身没有选择神经网络，而是可以采用一般的函数，尤其是线性函数。

DPG 算法对 $\pi(s)$ 和 $\mu(s)$ 的更新方法如下：

$$\begin{aligned}
&\delta_t = r_t + \gamma Q(S_{t+1}, A_{t+1}; \theta_Q) - Q(S_t, A_t; \theta_Q) \\
&\theta_Q \leftarrow \theta_Q + \alpha_{\theta_Q} \delta_t \bigtriangledown_{\theta_Q} Q(s, a; \theta_Q) \\
&\theta_\mu \leftarrow \theta_\mu + \alpha_{\theta_\mu} \bigtriangledown_{\theta_\mu} \mu(s; \theta_\mu) \bigtriangledown_a Q(s, a; \theta_Q)|_{a=\mu(s;\theta_\mu)}
\end{aligned} \tag{1-8}$$

基于 DPG，Lillicrap 等人提出的改进算法，DDPG 算法[11]，则全面采用了深度神经网络。为了能够让训练收敛，DDPG 使用了一个目标网络（target network）的概念。目标网络的定义是缓慢地跟踪依据梯度下降优化得到的网络参数 θ_Q 和 θ_μ，θ_Q 是状态行动对的价值估计网络 Q 的参数，而 θ_μ 是 actor-critic 架构中 actor 采用的策略的参数。目标网络参数 θ'_Q 和 θ'_μ 的更新方法如下（在每一步迭代过后进行这个更新）：

$$\theta' = \tau\theta + (1-\tau)\theta', \tau \ll 1$$

这样，相当于强迫目标网络缓慢地变化，从而增强了训练的收敛性。目标网络主要用于带入 (1-8) 中 δ_t 的计算。同时，DDPG 采用了经验回放的方法，在每一步迭代中都采用之前历史中一批随机挑选的过往经验进行批量计算，而非一步的结果，这样可以确保网络训练时不带有时序偏差。另外，DDPG 没有像 DPG 一样采用异策略学习，而是在策略基础上加上一个噪声分布以确保 ‘‘探索’’ 性。

1.1.4 基于模型的强化学习

基于模型的强化学习并非本课题的重点，但是是强化学习问题当中重要的一部分，和无模型的强化学习算法并驾齐驱。目前，分层强化学习问题多数基于

无模型的强化学习算法，很少考虑模型问题。但是实际上，模型可以大大提高算法的成功率、采样效率和稳定性。为完整性考量，在这里我们也对基于模型的强化学习算法进行简要的介绍，也作为给未来研究方向的一个灵感。

基于模型的深度强化学习问题可以分为模型是 ‘‘习得’’ 的和模型是给定的两种。在这里，我们首先关注一下给定模型的强化学习的应用。

在模型给定的算法中，最著名的例子就是 2016 年提出的 AlphaGo[26]。Deepmind 在 2017 年进一步提出了 AlphaZero[17]，其算法和 AlphaGo 相比有几大特点：

1. AlphaZero 完全采用逐步迭代的方式，而 AlphaGo 在训练中会挑选之前出现过的最好策略
2. AlphaZero 拥有的先验知识只有棋盘的格局和下棋的规则，并且这些知识只用来进行模拟
3. AlphaZero 对参数的变动更加鲁棒，不同棋类游戏用的都是同一套参数
4. AlphaZero 在训练中没有对输入进行任何数据加强的操作，而 AlphaGo 则对棋盘进行了镜像操作

AlphaZero 采用蒙特卡罗树状搜索（Monte Carlo Tree Search, MCTS[2]）的方法，每次优先访问训练历史中访问次数低，但访问概率 $\pi(a|s)$ 应该较高，且 $Q(s,a)$ 的状态行为对 (s,a)。同时，在此基础上加上一个噪声（在 AlphaZero 的例子中，一个狄利赫里分布的噪声），确保探索性。AlphaZero 采用深度卷积神经网络作为策略 $\pi(s)$ 的拟合函数，其输出为 $\pi(s) = (\mathbf{p}, v)$，其中，$\mathbf{p}$ 为在某个状态下选择各个行为的概率，v 为根据 s 推测的状态估值,$v = \mathbb{E}[z]$（胜负平局，z 的值分别为 $+1, -1, 0$）。针对该网络定义的损失函数为

$$L = (z - v)^2 - \pi^T log(p) + \lambda||\theta||^2 \tag{1-9}$$

可以看出，这个损失函数试图使 v 的预测和棋局结果 z 更加接近；π 代表了训练中各个动作的搜索概率，损失函数试图让神经网络的输出 (p) 与之尽可能接近。

这里所谓“给定模型”，其实就是环境是已知的，不需要真的和环境交互，而是在仿真里直接进行下棋这个活动即可。然而，这也意味着这个问题本身与直接和未知环境交互并没有区别。尽管模型给定，算法并没有从模型本身得出什么规律，因此这个模型本质上就是一个环境。

在强化学习问题中，从与环境的交互历史中逐渐学习到模型，则是另一个

有趣的思路。在[2] 中，Dyna 架构就是这样一种思路。在强化学习中，利用模型去近似求解 $v(s)$ 是一个 ‘‘规划’’（planning）问题。传统的列表式方法可以根据训练中得到的环境反馈不断更新模型。2015 年，Oh 等人在[27] 一文中提出了一种利用神经网络学习高维度环境的方法，使得复杂环境的模拟也成为了可能。

基于[27] 的结果，Racanière 等人在论文[28] 中提出了一种利用模型去增强无模型强化学习的方案，简称为 I2C 算法。这种方法本质上是细化了的 A3C 算法。注意到 A3C 本身是一个无模型的方法，它仅仅依赖于 $\pi(a|s)$ 和 $v(s)$ 的估计（回顾一下 (1-5)(1-6) 两个更新式）。但是，I2C 算法修改了 θ 的组织形式，构建了一个较为复杂的网络结构，因而引入了环境模型。其基本架构如图1.1所示。

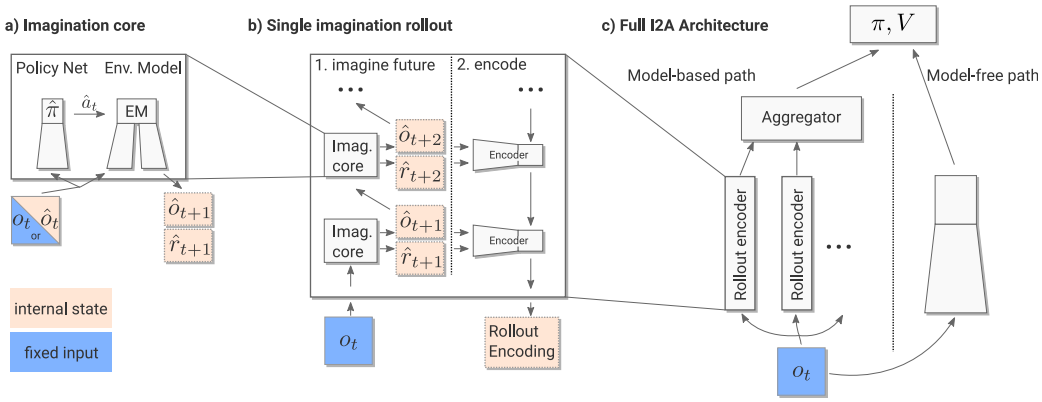


图 1.1　I2A 的基本结构：a) 一个预先训练好的神经网络，可以预测在策略 $\hat{\pi}$ 下，状态 s 的一系列时序变化；b) 一个子网络，用于 ‘‘理解”a) 的输出结果；c) 整体架构

图1.1c) 展示了网络整体架构，可以看到整体网络的输入为 o_t，也就是对状态 S_t 的观测，输出为策略 π 和价值函数估值 V（用于 actor-critic 架构）。因此，这个结构满足 A3C 的基本框架。在框架内部，网络结构采用了含有预测模型网络处理的输入，也采用了无模型网络处理的输入，分别作为两个分支，最后通过一个子网络汇合。具体的结构参数可以参见论文[28] 的附录。通过引入这个特殊的网络结构，I2A 算法将习得的模型结合进了原有的无模型学习框架中。同时，实验结果表明，这里的模型可以有错误和不精确的地方，学习结果依然是鲁棒的。

1.2 分层强化学习问题构建

分层强化学习算法（HRL）是一类特殊的强化学习问题。它主要想要解决两类问题。问题一是策略迁移能力。问题二是稀疏奖励问题。与人走路类比，我们可以自然地把行为分为不同层级。高层级的策略控制大的行为，例如决定朝哪里走，低层级的行为控制小的行为，例如具体怎么走。这样就达到了两个目的：对于问题一，由于具体怎么走与人所处的环境无关，在房间里学会的走路方法，在迷宫里一样可以用，因此 HRL 有助于实现底层策略的迁移。对于问题二，当我们把策略分层时，高层次的策略可以更多考虑 ‘‘远见’’ 的问题。在走路问题当中，假如我们希望达到一个目标，高层策略可以更多地考虑远处的目标，并给底层策略提供一个短距离的指导，一步步引导底层策略把智能体带向目标。

在本篇论文中，分层强化学习要解决两个主要问题：

1. 如何利用分层结构，让环境的稀疏奖励尽可能密集地扩散给处于早期探索阶段的智能体。
2. 如何利用上层策略，合理地整合多种不同的底层策略，成为一个新的复合策略。

分层强化学习依然是建立在强化学习的一般问题框架之下的。分层强化学习解决的仍是一个马尔可夫决策过程（MDP）[4]。Sutton 等人早年提出的选择（options[29]）框架下，MDP 问题会变成 SMDP（Semi-Markov Decision Process，半马尔可夫问题），在此框架下，原来推导时适用于 MDP 的一些结论也需要进行细微的调整。

在 MDP 中，S_{t+1} 只取决于 S_t 和 A_t，而不受到之前历史过程的影响。也就是说，S_t 状态本身就包含了它所有的历史对后续过程的影响。在这个假设下，我们可以这样表达强化学习的目标[5]：记 $v^{\pi}(s)$ 为状态 s 的价值函数。我们有

$$v^{\pi}(s) = E[G_{t_0}|S_{t_0} = s, \pi(s)]$$

$$J(\pi) = \max_{\pi} E(v^{\pi}(s))$$

$$\pi^{*}(s) = \operatorname*{argmax}_{\pi} J(\pi)$$

强化学习的终极目标就是找到这个 $\pi^{*}(s)$。Bellman 在 *Dynamic Programming*

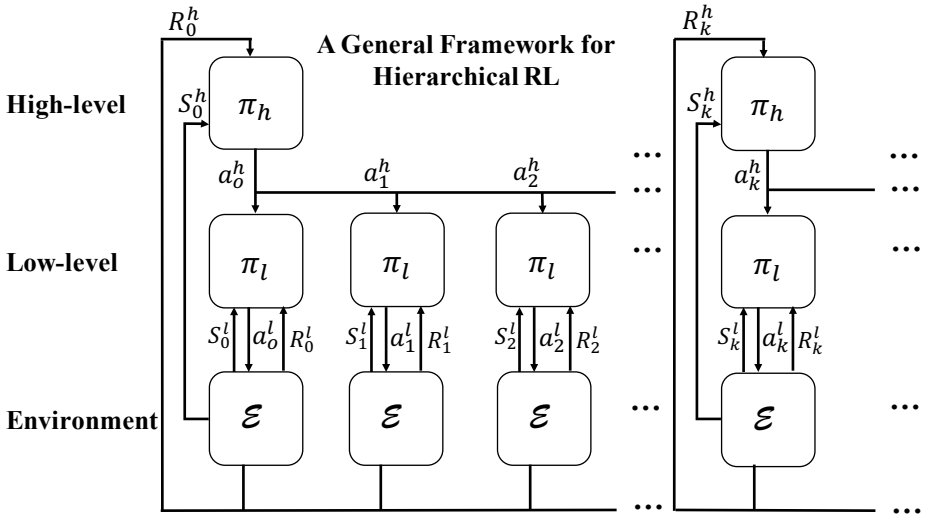


图 1.2　分层强化学习的一般架构。可以分为上层策略，下层策略（或者多层策略，此处仅展示两层），和环境三部分。上层策略的输入是环境观测 S^h，输出的行动 a^h 是下层策略的输入，下层策略的另一输入为环境观测 S^l，输出为智能体的实际行为，和环境进行交互。环境返回给智能体的奖励 R 作为对上层策略的奖励，而下层策略的奖励则是算法设计者自定义的部分，往往与环境、智能体状态、上层行为、价值函数等等有关。HRL 的关键问题其实就在于如何定义上层动作，和如何定义下层获得的奖励。

一书中提出了著名的 Bellman 最优性等式[6]

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{1-10}$$

同时，我们用 $q(s, a)$ 表示在状态 s 下采取行动 a 可以获得的价值，那么 Bellman 最优性等式又可以表示为

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{1-11}$$

1.3 分层强化学习研究现状

1.3.1 常见算法思路对比

分层强化学习是用于解决复杂强化学习问题和稀疏奖励问题的常见手段[30][31][32][33]。选择框架[29][30] 是分层强化学习中的一个常见框架，而基于选择框架的 Option-Critic 结构[34] 则为早期的 HRL 的端到端训练提供了重要的理论基础。然而 Option-critic 当中，上层策略是一个简单的 ϵ-贪心策略，也就是基于价值的迭代算法，不属于策略梯度算法，这也限制了这个算法在复杂场景的应用前景。

近两年来分层强化学习的研究聚焦于连续动作空间。应用于离散空间问题的 HRL 由于对 ‘‘目标’’ 的定义更为简单，因此算法无法拓展到连续空间。在本篇论文中，连续动作空间将是我们解决的主要问题。连续空间的分层强化学习研究可以主要分为两个不同的思路，一个是下层基于上层目标的强化学习，另一个是下层基于上层选择的强化学习。特别地，Option-critic 虽然不是针对连续动作空间的算法，但它属于第二种方案。它同时训练上层（option）和下层（actor）策略。接下来我们将分别介绍这两类思路。

1.3.1.1 子目标设计

其中一个思路是，训练一个上下层的策略结构，上层策略需要给下层策略提供一个子目标（subgoal），下层策略朝向这个子目标去走，并且依据与自身状态和子目标相关的某个函数，而获得一个内部反馈（intrinsic reward，或者说辅助反馈，auxiliary reward），而上层策略则获得环境给出的真实反馈（extrinsic reward）。许多算法都采用了这个思路[30][35][36][37][38]。

这个产生内部反馈的函数，有不同的选择。例如在高效分层强化学习（HIRO[36]）算法中，这个函数是下层状态与智能体目标状态的欧氏距离。在 HAC[37] 算法中，训练场景根据智能体状态是否达到子目标确定，而 ‘‘达到子目标’’ 的判据则也是欧氏距离。而在层级制网络强化学习算法（FeUdal Network[38]，其主要受到[32] 这篇论文的启发）当中，这个函数是下层行动方向与上层给出目标方向的余弦距离。同时在 FeUdal Networks 这篇论文当中，下层策略也获得了外界环境的奖励，这与内部反馈一起加权处理后成为底层策略获得的总的奖励，亦即 $R_{low} = R_{extrinsic} + \alpha R_{intrinsic}$。注意到在分层强化学习的这种思路当中，上下层策略都是一起从零开始训练的。

这些算法共同的问题是需要利用函数来衡量特定的状态空间差距。常用的

欧氏距离和余弦距离本质上都是把状态空间当作欧几里得空间，这要求，（1）状态空间的各个维度的数量级相似，具有可比拟性，（2）欧氏距离可以表征智能体状态和预期状态的差距。其中第（2）点也可以用数学语言翻译成，如果对于状态 S_1, S_2 和 S_3，$d(S_1, S_2) > d(S_1, S_3)$，则状态 S_3 比 S_2 和 S_1 相似。这一特征在状态表征为观测值时，是无法保证的。2019 年的最新论文[39] 中也论证了这个问题。一些算法试图解决这个问题。它们的思路是不再使用状态的欧氏距离作为状态差异的表征，而采用一种特殊的 ‘‘表征’’（representation），将状态空间投影到一个新的空间里。例如 Ghosh 等人在 2019 年的论文[39] 采用了状态对应的行动（actionable representation）的欧氏距离；Nachum 等人在 2018 年[40] 提出了对目标的表示进行学习的方法，学习一个投影 $f(S)$，下层策略获得上层给定的特定目标 g，并结合自身观察值而决定采取的行为。在下层训练时，奖励值由 $f(S)$ 与 g 的欧氏距离决定。

1.3.1.2 下层预训练

另一种思路是上层策略输出对下层策略的一个选择，因而上下层策略充分解离，常用对上下层策略分开训练[41] [42] [43]。下层策略的训练往往依赖于特定的应用域知识，有的算法会加上奖励多样性（熵）的方式。这种算法的优势在于，由于底层策略是共享的，一方面同一套底层策略可以用于多种不同的具体任务，另一方面在特定任务的训练中，由于底层策略已经比较成熟，因而可以大大加速学习过程。

在基于随机神经网络的分层强化学习算法（SNN for HRL[42]）论文中，作者们提出了一种典型的分层架构：利用随机神经网络构建上层策略，输出 1-hot 向量表示的隐式编码（latent code）（一个一维向量 $[x_1, x_2, ..., x_n]^T$，其中只有一个元素为 1，其余均为 0），来表示选择哪一个下层策略。下层策略的输入为环境的观察值 o_t 和 1-hot 向量 v，输出值是智能体动作空间的行动。为了获得可用的下层策略，SNN for HRL 要求进行底层策略预训练。预训练法是分层强化学习当中常用的一种算法。它的基本思路是，在一个比较简单的、环境反馈（即奖励值）比较密集的环境中训练下层策略。由于实际策略在操作的时候不需要获得 ‘‘奖励’’，因此只要观测空间和动作空间的维度相同，下层策略就可以直接在不同环境中迁移。SNN 的一大贡献在于，为了能够训练出需要 1-hot 向量输入的下层策略，引入了一个共享信息熵的奖励（Mutual-Information, MI bonus）。2017 年提出的 MLSH 算法[44] 也需要预训练，不过它在预训练阶段同时训练了上下层策

略。为了解决上层策略动态的问题，这个算法限制上层策略只参与若干次训练，之后都集中训练下层策略。底层策略集在新场景应用的时候，则完全不训练底层策略。

正如 SNN 一样，为了能够获得复用性强的下层策略，许多研究聚焦于提升预训练策略的多样性（diversity）[41] [43]。例如 DIYAN（diversity is all you need）算法甚至只考虑多样性而不接受环境的奖励。有的算法[43] 为了获取较好的底层策略，需要手动设计一些简单的预训练场景来 ‘‘引导’’ 底层策略学习，这也是我们所不希望的。这类现有预训练算法的共同问题在于，下层策略一旦训练出来，在 HRL 的框架下就不可以再改变（参数被 ‘‘冻结’’）。这是有较大问题的。例如，SNN 文章中提出，在蚂蚁行走的环境中，如果下层策略无法改变，那么在上层策略切换下层策略的时候，蚂蚁会很容易摔倒。由此我们联想到，能否在训练上层策略的同时修正下层策略，来提高整体策略的表现？SNN 文章中作者指出，有一种潜在的方法可以联合训练上下层策略，而这种方法是基于对整体网络的梯度下降[45]。然而，正如 FeUdal Network[38] 一文指出的，这样一类直接对整体网络结构进行上下层一起优化的算法可能会丢失上层输出的语义信息，另外，这类算法最终容易退化成（i）上层策略始终只选择一个底层策略（ii）上层策略不断重新选择底层策略。这两种退化模式都等效于丢失了分层结构。因此，我们的方法决定采用上下层同时训练，但分成两次优化。这样，我们可以确保上下层策略具有足够的解离度，分层结构得以保留，上下层输出动作的含义都是可解释的。

1.3.2 马尔可夫过程的可观测性

尽管上述许多算法能够很好地完成基准实验环境下强化学习的任务，但是这些算法在机器人的实际操作中存在很大的问题。其中，马尔可夫过程中环境的观测性是一个很大的问题。

上述 HAC[37]，HIRO[36]，MLSH[44] 等算法在设计下层奖励值的时候，都用到了欧氏距离。欧氏距离有一个很大的问题在于我们需要获取环境的真实表现（ground truth）而非仅仅观察值。例如，一个机器人在迷宫中行走，如果想要采取上述方法，那么就需要机器人知道自身的 (x, y) 坐标。实际上，这个坐标时很难获取的。也就是说，实际当中，智能体所处的马尔可夫过程并非完全可观测。考虑到如果智能体完全看不到目标，就永远无法知道应该往哪里走，因此在实际操作中，我们减小难度，让智能体能够穿透墙面看到目标。理论上，对于一个

不是非常复杂的迷宫，智能体和目标的相对位置与智能体的观察值是一一对应的。但实际当中，这个对应关系非常难以恢复。换言之，我们可以说智能体面临的仍然是一个较难观测的马尔可夫问题。

事实上，在 2019 年的一篇文章[46] 中，作者通过实验指出，这些基于上层给下层目标的分层强化学习都面临着 ‘‘目标空间’’ 的定义难题。如图1.3所示，文中作者对这些论文给出的实验环境修改了状态的表示（也就对应着 ‘‘目标空间’’ 的定义），经过一些细微的变异，这些算法就无法学出来。经过试验，我们也论证了 HAC[37]，HIRO[36] 都不能在这样的观测下成功。这一点是可以解释的。当观测目标由欧式空间的 (x, y) 坐标变为其他表示时，欧氏距离的远近不再能够表征智能体实际距离目标的远近，因此这些算法在非欧式观察空间下是不能成功。

最新的一些研究试图从修改状态空间表示的方法上去纠正状态空间中欧式距离的表示能力问题[39] [40]。这些算法解决了针对原始状态空间直接进行距离计算的问题，但是我们注意到，在他们的实验中，依然给智能体的观察空间注入了与位置坐标有关的信息。例如，ICLR2019 的这篇修正目标空间表示的论文中[40]，几个不同的实验都采用了或强或弱的宏观空间位置表示方法，如输入 (x, y) 或者是输入一个简化的鸟瞰视图。这样直接告诉机器人 (x, y) 位置必然会导致智能体直接将坐标与价值联系起来，而智能体的训练和测试都在同样的环境下进行，这就导致智能体的训练结果其实是过拟合的，智能体只是记住了不同位置具有不同的价值，而没有真正学会归纳这些位置的特点。

然而在实际中，机器人应用场景多数都无法直接观察自身的位置（除非通过 SLAM 一类的位置估计算法进行地图构建，不在我们的讨论范围内），即使可以观察，机器人的应用场景也远远比单个环境要复杂，不可能有足够大的训练集让机器人去拟合，或者说记住所有可能的场景。因此这些对空间可观测性要求非常明确的算法在实际中往往是不实用的。

1.3.3 观察空间的分离

我们前面提到，分层强化学习的上下层状态观测 S^h，S^l 未必相同。事实上，我们应该鼓励上下层观测不同。例如，针对走迷宫的例子。一个自然的想法是，假如我们把策略分解成了上下两层，那么下层策略就不应该理会自己在大环境中的相对位置，而应该集中精力把具体的每一小步走好。这时对于下层策略，与环境有关的观察，例如坐标、距离终点的距离等，没有实质作用。

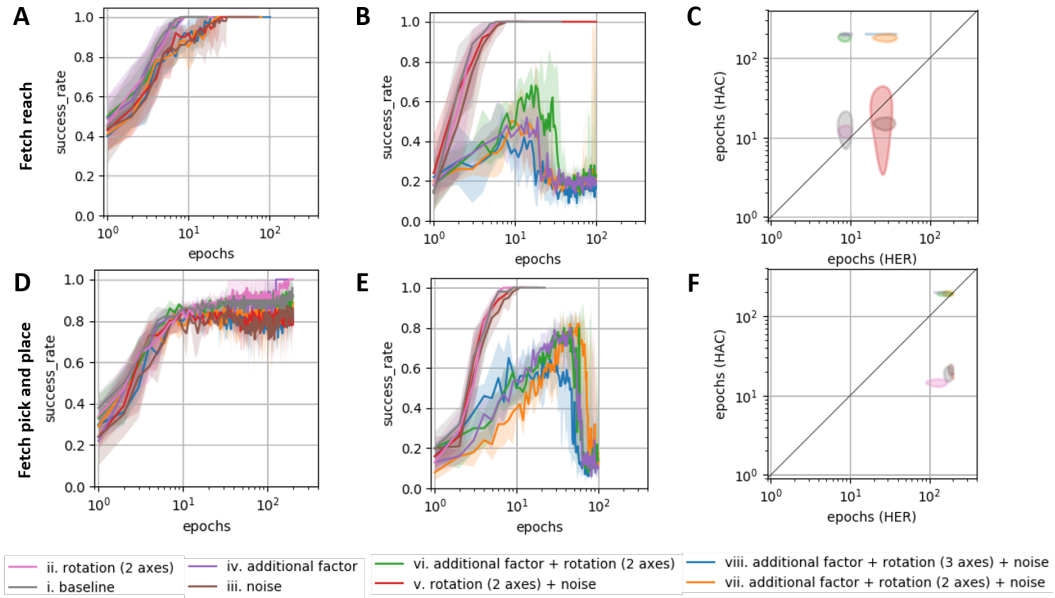


图 1.3　几个典型 HRL 算法在状态空间进行一定变化以后的训练结果变化图[46]。这个示意图表明，环境的状态表示发生变化时，HRL 算法如 HAC 和 HER 会受到极大地影响。具体的实验描述可以参见原文。

1.3.4　与 HAAR 相关的工作

在本部分，我们将介绍一些启发了我们的 HAAR 算法的，或是与我们研究的问题相关的工作。在后续章节我们详细介绍算法后，读者可以得知，HAAR 的特点在于：

1. 同时训练上下层策略，提高采样效率的同时使技能可扩展
2. 对状态空间的具体表征方式不敏感
3. 仿照现实，将智能体观察到的信息量限制在基于它自身位置可观测的环境

因此，我们对同样具有这些特点的，但采取了不同解决方案的算法感兴趣。

2017 年提出的 MLSH 算法[44] 和我们的算法类似，都采用了同一批经验同步训练上下层策略。MLSH 与我们的区别在于我们对于奖励函数定义的区别。MLSH 的底层策略依赖环境的奖励值来进行训练，因而只能解决底层策略复用的问题，但不适合应用于稀疏奖励的问题。

有两份工作具有潜在的，和我们的算法解决能力相似的功能。上文中提到过的 Ghosh 等人在 2019 年的论文[39]。他们试图去解决子目标范式下，目标表示不够一般化的问题。为此，他们采用了状态对应的行动（actionable representation）的欧氏距离。第二份工作是 Nachum 等人在 2018 年[40] 提出了对目标的表示进行

学习的方法，学习一个投影 $f(S)$，下层策略获得上层给定的特定目标 g，并结合自身观察值而决定采取的行为。在下层训练时，奖励值由 $f(S)$ 与 g 的欧氏距离决定。这些工作的核心都在于使得。

HAAR 大量地受到上文中提到过的 TRPO 算法的影响，因为 TRPO 是我们进行优化的基本算法。另外，HAAR 也受到了 SNN4HRL 的影响，在实验设计上，我们采用和 SNN4HRL 一文相近的实验环境，并把它作为基线算法进行比较，另一方面，我们采用 SNN4HRL 的方法来预训练可以直接使用的底层策略。

1.4 论文结构

本篇论文分为五章。在第一章绪论当中，我们构建了分层强化学习解决的一般问题，回顾了大量近四年的最前沿研究，并指出我们的工作面对的较为具体的限制和意义。第二章我们着重介绍了我们提出的算法，基于优势函数奖励的分层强化学习算法（HAAR），并且证明了这种方法在优化联合策略时具备的单调递增特性。在第三章当中，我们介绍了设计的大量验证实验，通过在 Mujoco 仿真环境中的可移植性，证明了和以往的算法相比，HAAR 算法在机器人这类连续动作空间的应用场景下具备巨大的优势。第四章我们着重介绍了一个复杂的自动驾驶仿真实验环境，将我们的算法应用在这个实验环境中。最后一章我们总结了我们的工作的内容以及 HAAR 算法的优势。

第 2 章　辅助奖励强化学习算法

2.1　本章简介

我们的算法如示意图2.1所示。我们采用 HRL 结构，因此可以把整体框架分解为上层策略，下层策略和环境三层。每一大步是上层策略走一步，上层策略产生一步动作，在我们的例子中也就是隐编码 z。这同一个个隐式编码（latent code）被输入给了 k 小步的下层策略，同时下层策略还会观察环境获得 S_i^l，这两个向量合并后成为下层策略的‘‘观察量’’。由此下层策略输出一系列下层动作 $a_1, ..., a_k$，直接作用于环境。上层策略获取环境反馈直接作为它的奖励函数，而下层策略的奖励函数则根据上层策略来获取，并且同一大步内的每一小步获得相同的奖励值。

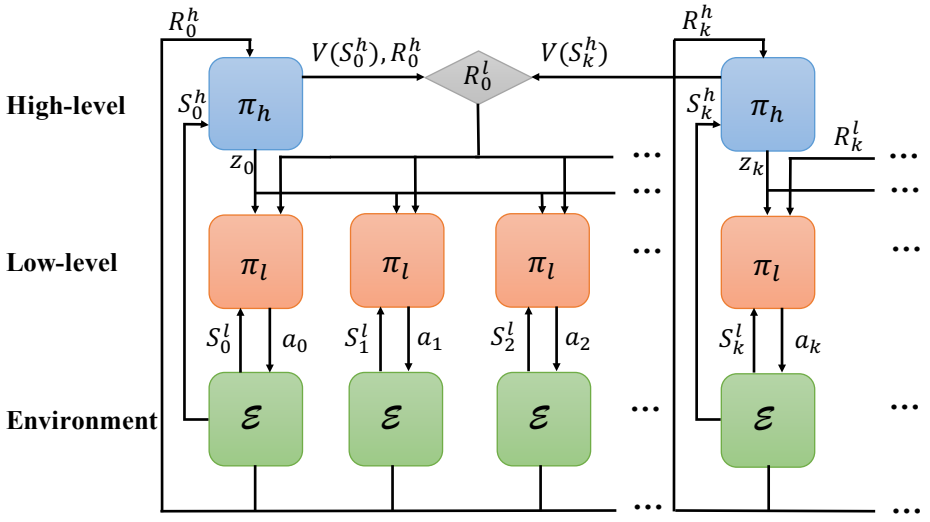


图 2.1　HAAR 算法的基本架构示意图。每一大步，上层策略产生一步动作 a^h，在我们的例子中也就是隐编码 z。这个隐式编码被输入给了 k 小步的下层策略，同时下层策略还会观察环境获得 S_i^l，由此输出一系列下层动作 $a_1, ..., a_k$。上层策略获取环境反馈直接作为它的奖励函数，而下层策略的奖励函数则根据上层策略来获取，并且同一大步内的每一小步获得相同的奖励值。

2.2 底层策略预训练

和原来的专利文件类似。具体地，我们是采用 SNN4hrl 这篇论文的方法预训练，底层策略共享神经网络结构。训练的时候每次会额外输入一个 1-hot vector（latent code, 一堆 0 中间只有一个 1，表示挑一个底层策略出来），训好之后上层策略就会把这个 1-hot vector 输入下层网络，表示使用哪个下层策略。

2.3 辅助奖励函数定义

2.3.1 定义方法一：基于优势函数

算法的关键在于下层策略奖励函数的设计。我们通过一个简单的定义将奖励函数和优势函数联系在一起。上层策略优势函数的定义是

$$A_{\pi_h}(s_t^h, a_t^h) = \mathbb{E}_{s_{t+k}^h}[Q_{\pi_h}(s_t^h, a_t^h) - V_{\pi_h}(s_t^h)] \tag{2-1}$$

$$= \mathbb{E}_{s_{t+k}^h}[r_t^h + \gamma V_{\pi_h}(s_{t+k}^h) - V_{\pi_h}(s_t^h)]. \tag{2-2}$$

我们将上层策略当前的优势函数估计值平均分配到这一步上层策略下的每一步下层步，作为下层策略的辅助奖励函数。

$$R_l(s_t^l..s_{t+k-1}^l) = A_{\pi_h}(s_t^h, a_t^h), \tag{2-3}$$

其中，$R_l(s_t^l..s_{t+k-1}^l)$ 表示 k 步辅助奖励值之和。由于上层策略产生的同一隐编码连续输入给下层策略了 k 次，因此我们认为这个奖励值应该平均分配到每一步的下层策略。因此我们定义

$$\underset{t\le i<t+k}{R_l(s_i^l)} = \frac{1}{k} A_{\pi_h}(s_t^h, a_t^h) \tag{2-4}$$

根据经验估计奖励函数 $A_{\pi_h}(s_t^h, a_t^h)$ 的真实值有许多不同的方法[47]。在这里，我们采用简单的单步估算方法

$$A_{\pi_h}(s_t^h, a_t^h) = r_t^h + \gamma V_{\pi_h}(s_{t+k}^h) - V_{\pi_h}(s_t^h) \tag{2-5}$$

$$\underset{t\le i<t+k}{R_l(s_i^l)} = \frac{r_t^h + \gamma V_{\pi_h}(s_{t+k}^h) - V_{\pi_h}(s_t^h)}{k} \tag{2-6}$$

在实际实现当中，由于 γ 的值非常接近 1，我们可以忽略式 (2-5)(2-6) 当中的 γ 这个系数。经验表明，略去这个系数可以使得训练的方差减小，但对于训

练结果没有显著影响。这在直观上也是易于理解的：如果去除 γ，又因为我们是一个稀疏奖励问题，$r_t^h = 0$，优势函数将会退化为上层策略的价值函数估计值 V_h 之差。注意到上层策略得到的奖励完全来自于环境，因此它的价值函数表达的意思就等效于整体策略的价值函数。假设上层策略的 V 估计是接近真实值的，那么对于下层策略来说，它使得 V_h 之差越大，就越说明它在鼓励智能体朝向整体价值高的地方走，而价值高就对应着策略越好。上层策略越优，这个价值之差与真实价值之差就越接近，这个奖励函数就越合理。

在后文中，我们将从数学上证明 (2-6) 这个奖励函数的设计能够确保整体策略的单调提升。

2.3.2 定义方法二：基于速度方向

方法一是我们最终确定采用和研究的主要方法。在方法二当中，我们则采用了另一种辅助奖励的方法，‘‘引导速度’’。在有监督的强化学习当中，人们会手动给智能体设计一个速度流，但这种方法显然是不够泛化的。我们希望能够设计出一种根据稀疏的奖励，自动生成速度流的方法。这种思想在 FeUdal 论文[38]当中也有所体现，不过与我们采用的引导速度有所差异。

和方法一去除 γ 以后的直观解释一样，我们先假设上层策略的价值函数估计 V_h 已经比较优。接下来，我们希望底层策略能鼓励智能体朝向上层策略的价值函数 V_h 增大的地方运动。价值函数 V_h 是状态变量 S 的函数。由于 $V_h(S)$ 在空间每个点都有定义，因此也可以看作是一个关于 (x, y) 的函数 $V(S(x, y))$。在实际训练中，取决于状态空间的不同表达方式，$V_h(S)$ 可能可以，也可能不可以关于 (x, y) 求导（梯度）。为了一般试验下能够表示，我们不直接求导，而是通过‘‘虚拟步’’的方法来估算这个导数。假设智能体此刻位于 (x_0, y_0)。我们让智能体朝八个预定义的方向 $(dx_i, dy_i), i = 1, 2, ..., 8$ 各进行一小步的探索，采集采取这一步以后的 V_h 值，其中最大的 $V_h(S(x_0 + dx_m, y_0 + dy_m))$ 就对应了梯度方向，简记为

$$\nabla V_h(x, y) = [dx_m, dy_m]^T \tag{2-7}$$

我们由此定义下层策略获得的奖励函数为梯度和速度的点积

$$\underset{t \le i < t+k}{R_l(s_i^l)} = \nabla V_h(x, y) \cdot [dx^l, dy^l]^T \tag{2-8}$$

其中，(dx^l, dy^l) 表示实际下层步骤得到的位移，由于单步时间差是固定的，也

可以认为就是速度。

方法二的缺陷在于需要根据应用场景分析什么是‘‘速度’’，什么是‘‘方向’’，也意味着它的泛化能力不如方法一。另外，假如通过智能体走一小步‘‘虚拟步’’的方法来估算 V_h 关于位移的导数，我们在仿真环境训练的时候需要花费 9 倍的时间，效率上是不划算的。同时，在现实世界中，条件可能不允许进行这样的‘‘虚拟步’’。尽管可以通过对环境模型的拟合来替代这个‘‘虚拟步’’的效果，但这样也增加了这个问题的复杂性，可以作为后续研究的方向，在这里不作讨论。因此在最终的实验论证中，我们采用了方法一。

最终的方法详细展开也可以画成图2.2展示的形式。

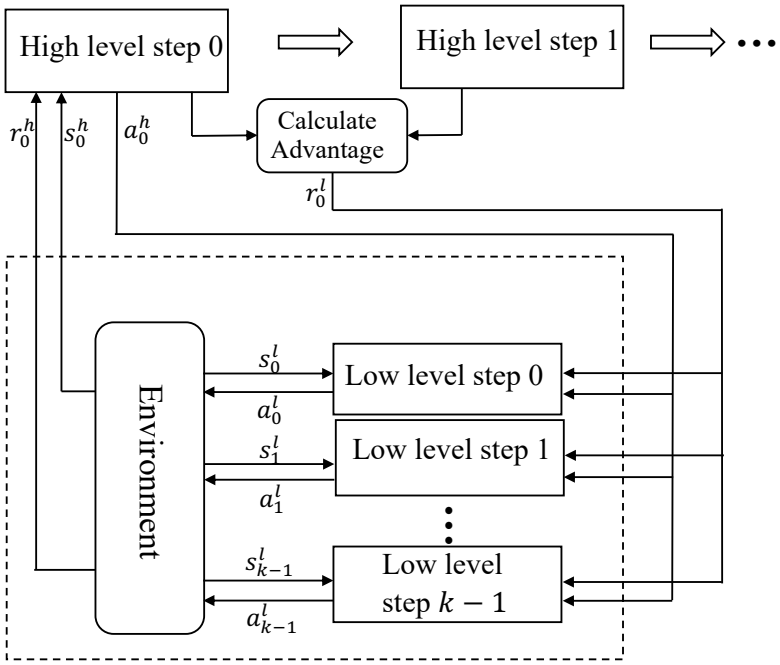


图 2.2　和图2.1类似，不过这里具体展现了上层采取一大步时，下层具体展开走 k 小步的一个周期。

2.4　算法细节讨论

2.4.1　底层技能长度退火算法

在以往的‘‘选择者’’模式分层强化学习工作当中，底层策略终止有两个方式。第一个方式是基于 option 框架的工作，是否终止是学习出来的一个函数。如

文献综述中提到的，在其他一些方法当中[37,42,44]，底层技能长度 k 是固定的。当 k 太小的时候，在最开始训练的探索阶段，底层技能的应用并不能有效缩短上层策略面临问题的时间域长度（horizon）。当 k 过大时，上层策略变得不够灵活，因为在 S_t^h 状态下采取的行动 s_t^h 会一直延续到 S_{t+k-1}，也就是上层策略不会进行及时的调整。这意味着，k 很大时，层次策略将不是最优的。为了在训练初期的探索性和训练末期的最优性之间找到一个平衡，我们开发了一种模拟退火应用于 k 的方法。k_i 将随着训练当前迭代次数 i 发生这样的改变：

$$k_i = k_1 e^{-\tau i}$$

其中，k_1 是最一开始训练时的底层技能执行长度，τ 是退火温度，反映了底层技能长度的衰减速度快慢。我们定义一个最短技能执行长度 k_s，当 $k_i < k_s$ 是，我们令 k_i 等于 k_s。这样可以防止层次策略退化为一个单步策略。

在图2.3中，我们直观地展示了底层技能步长退火算法。步长一开始很大，之后逐渐变小，并且变小的速度减慢。当 k 减小到 k_s 时，退火过程结束，分层算法将固定在这一结构进行不断地优化。我们认为，这样一种模式可以加速智能体初阶段的探索学习能力，从而提高采样效率。在后续的实验当中，我们也会通过控制变量的方法来探索这个问题，也就是

- 退火算法是否依然能够保证获得奖励的增长，并且在最后依然收敛到最优?
- 和非退火算法相比，退火算法能否真的加快学习速度?

这将在后文有进一步的讨论。

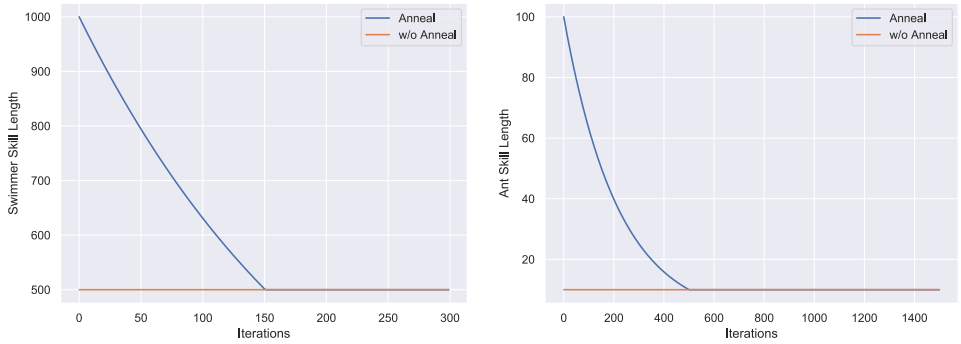


图 2.3　左右两图分别是在后续实验 Ant 环境和 Swimmer 环境当中采取的底层步长退火算法。步长一开始很大，之后逐渐变小，并且变小的速度减慢。当 k 减小到 k_s 时，退火过程结束，分层算法将固定在这一结构进行不断地优化。我们认为，这样一种模式可以加速智能体初阶段的探索学习能力，从而提高采样效率。

2.4.2 底层技能预训练的影响

我们认为，底层技能的预训练对我们算法的表现是有非常显著的影响的。这是因为，直观上讲，假如上层策略的优势函数能够给下层策略提供正确的指导，那么需要上层策略的优势函数学习得相对较优。假如底层策略不好，导致智能体一直无法成功探索到达目的地，就意味着上层策略很难获得非 0 反馈，那么优势函数的值将一直为 0，无法给下层训练提供指导。

为了解决这个问题，我们提出将底层策略进行预训练的方法。我们采用任意一种现有的技能发现（skill discovery）算法来进行预训练，获取一个具备*多样性*和*优秀性*的底层技能集。底层策略相对较好的时候，上层策略能够有效地获取环境反馈，从而为下层提供指导。

2.5 HAAR 算法流程

为表达清晰，下面的框图展示了我们完整的实际算法流程。

Algorithm 1 HAAR algorithm

Pre-train low-level policy $\pi_l(a_t^l|s_t^l)$.
Randomly initialize high-level policy $\pi_h(a_t^h|s_t^h)$.
for iteration $\in \{1, ..., N\}$ **do**
  Anneal low-level step length k.
  Collect experiences with M paths under current policy, put in set $\mathcal{S}$.
  for each path $\in \mathcal{S}$ **do**
    for each high-step $(S_{t_i}^h, S_{t_i+k-1}^h; S_t^l, S_{t+1}^l, ..., S_{t_i+k-1}^l) \in$ each path **do**
      Process sampled experience: $R_t^l|_{t=t_i,...,t_i+k-1} \leftarrow \frac{\gamma V_{\pi_h}(s_{t_i+k}^h) - V_{\pi_h}(s_{t_i}^h)}{k}$.
    end for
  end for
  Concatenate all high-level experiences.
  Optimize π_h with TRPO, update V_h
  Concatenate all processed low-level experiences.
  Optimize π_l with TRPO
end for
return π_h, π_l.

2.6 单调递增性结论

我们的算法是分别对上下层策略进行更新，接下来我们证明，单独更新上层策略和单独更新下层策略都可以使得整体策略变优。

在接下来的推导过程中，我们用上标或下标 h 和 l 来表示强化学习中的一系列参数，是针对上层策略（high）还是下层策略（low）。

2.6.1 整体策略的度量方法

首先我们定义对整体策略好坏的度量方法。注意到，对于上层策略来说，底层策略相当于环境动力学（dynamics）。当底层策略发生了变化时，可以看做是环境动力学发生了变化。我们用 $\mathcal{E}(\pi_l)$ 表示环境，并且环境与 π_l 有关。定义 π_h 带来的折扣奖励期望

$$\begin{aligned}\eta(\pi_h) = V_h(s_0^h) = \mathbb{E}_{s_0^h, a_0^h, \ldots}\left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} r_h(s_t^h, a_t^h)\right], \text{where} \\ s_0^h \sim \rho_0^h(s_0^h), a_t^h \sim \pi_h(s_t^h), s_{t+1} \sim P(s_{t+1}^h | s_t^h, a_t^h, \mathcal{E}(\pi_l))\end{aligned} \tag{2-9}$$

这个期望值同时和 π_h 和 π_l 有关，也就是我们对整体策略的度量。

2.6.2 上层策略单调递增性

首先，我们说明，在下层策略不变的情况下，利用 TRPO 更新上层策略，可以保证整体策略的单调递增，即 $\eta(\pi_h)$ 单调递增。这一步很简单，既然底层策略不变，那么可以认为底层策略是包含在环境中的，因此 TRPO 去优化上层策略，就相当于 TRPO 直接作用于一个单层的强化学习结构，对应的优化目标将是近似单调递增的。注意到对于上层策略来说，它得到的 R 就是上下层策略作为一个整体得到的真实 R。

2.6.3 下次策略单调递增性：连续情况

接下来，我们推导，采用 $\frac{\gamma_h V_h(s_{t+k}^h) - V_h(s_t^h)}{k}$ 作为上层给下层的输入，从而优化 π_l，也可以使得整体策略单调变优。

我们优化策略的算法采用的是 TRPO[13]。在 TRPO 当中，我们最终希望去优化的东西是 $V_\pi(s_0)$。TRPO 算法等价于在基于策略梯度定理给出的策略梯度优化中，增加了对步长的限制，从而确保策略更新的单调性。在策略梯度方法的论文[48] 中，Sutton 指出，对一个策略优化的目标函数 J 既可以采用 $V_\pi(s_0)$ 的定义，也可以采用 R 的平均值的定义，它们可以分别推导得出策略梯度定理，而

这个定理中的策略梯度，与 TRPO 试图优化的替代函数的梯度是相等的。因此，TRPO 也可以理解成是在最大化 $\mathbb{E}_{s,a\sim\pi}[R(s,a)]$。要注意，这里要求马尔可夫过程必须是满足稳态分布的，也就是 $P(s_t = s)$ 存在。在2.6.6的引理**??**里，我们证明了满足这些条件的情况下，最大化 $\mathbb{E}_{s,a\sim\pi}[R(s,a)]$ 与最大化 $V_{s_0\sim\rho(s_0)}(s_0)$ 等价。

如果把 R 的均值作为优化目标，我们有

$$\nabla J(\theta) = \nabla\mathbb{E}_{s\sim\pi}[\sum_a \pi(s,a)R(s,a)] = \nabla\mathbb{E}_{s,a\sim\pi}[R(s,a)]. \tag{2-10}$$

我们定义了

$$R_l(s^l_{t+i}, a^l_{t+i})|_{i=0,1,\ldots,k-1} = \frac{V_h(s^h_{t+k}) - V_h(s^h_t)}{k}. \tag{2-11}$$

前面我们提到，TRPO 完成的任务是最大化 $\mathbb{E}_{s,a\sim\pi}[R(s,a)]$，因此我们在利用 TRPO 针对底层策略进行优化的时候，相当于最大化 $\mathbb{E}_{s,a\sim\pi_l,\pi_h}[R_l(s_l,a_l)]$。注意这里，我们把 π_h 看做是一个固定的概率分布函数，而不对它进行优化。π_h 可以看做是环境的一部分，而它输出的隐式编码（latent code）则是我们观察（observation）的一部分。

根据 (2-11) 中的结果，我们两边取期望，可以得到

$$\mathbb{E}_{s^l,a^l\sim\pi_l,\pi_h}[R_l(s^l,a^l)] = \frac{1}{k}\mathbb{E}_{s^h,a^h\sim\pi_l,\pi_h}[\gamma_h V_h(s^h_{t+k}) - V_h(s^h_t)]. \tag{2-12}$$

回忆在强化学习中，我们关于优势函数 $A(s,a)$ 的定义，为

$$A(S_t,A_t) = Q(S_t,A_t) - V(S_t) = R(S_t,A_t) + \gamma V(S_{t+1}) - V(s). \tag{2-13}$$

由于在稀疏强化学习问题中有稀疏奖励条件：

$$R(S_t,A_t) = 0, \forall t \neq t_{end}. \tag{2-14}$$

因此，(2-13) 变为

$$A(S_t,A_t) = \gamma V(S_{t+1}) - V(s). \tag{2-15}$$

由 (2-12) 和 (2-15) 可知，TRPO 优化下层策略的结果，等效于优化了上层的优势函数，也就是

$$\mathbb{E}_{s^l,a^l\sim\pi_l,\pi_h}[R_l(s^l,a^l)] = \frac{1}{k}\mathbb{E}_{s^h,a^h\sim\pi_l,\pi_h}[A^h(s^h_t,a^h_t)]. \tag{2-16}$$

接下来我们证明，通过最大化这个期望值，就近似等效于优化了整体策略的表现。

我们用 $\tilde{\eta}(\pi_h)$ 来表示 π_l 发生变化（变为 $\tilde{\pi}_l$）而 π_h 没有发生变化以后的折扣奖励值期望。将 TRPO 根据[49] 得到的引理 1（**Lemma 1**）稍加变形，我们可以得到类似的结果:（具体推导参见2.6.6中的结论2.1）

$$\tilde{\eta}(\pi_h) = \eta(\pi_h) + \mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_{\pi_h}(s_t^h, a_t^h) \right]. \tag{2-17}$$

等式 (2-17) 右边第二项的含义就是，新的整体策略相对于旧的整体策略的优势。我们想要最大化的就是这个优势。

我们定义折扣访问频率 ρ 为

$$\rho_{\pi_h}(s^h) = \sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} P(s_t^h = s | \mathcal{E}(\pi_l)) \tag{2-18}$$

当底层策略发生改变而上层策略不变的时候，ρ_{π_h} 相应地变为 $\tilde{\rho}_{\pi_h}$。

表达式 (2-17) 中，等式右侧第二项可以变形为（参见 TRPO 论文中的等式 (2) 推导，写在2.6.6中引理2.2）

$$\sum_{s^h} \tilde{\rho}_{\pi_h}(s^h) \sum_{a^h} \pi_h(a^h | s^h) A_{\pi_h}(s^h, a^h). \tag{2-19}$$

对于一个起点随机产生的，非周期、不可约的稳态马尔可夫问题，我们有

$$P(s_0^h = s^h) = P(s_1^h = s^h) = \ldots = P(s_i^h = s^h) \tag{2-20}$$

注意到这里 $P(s_i^h = s^h)$ 也就是采样中状态的分布。

将 (2-20) 带入 (2-18)，可以得到

$$\tilde{\rho}_{\pi_h}(s^h) = \frac{1}{1-\gamma_h} P(s_i^h = s^h) \tag{2-21}$$

从而，我们可以把 (2-17) 写成期望的形式:

$$\tilde{\eta}(\pi_h) = \eta(\pi_h) + \frac{1}{1-\gamma_h} \mathbb{E}_{s^h, a^h \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[A_{\pi_h}(s^h, a^h) \right]. \tag{2-22}$$

我们发现最大化 (2-16) 恰好也就最大化了 (2-22)。由此我们证明了，只要利用 TRPO 对下层策略进行了更新，$\eta(\pi_h)$，也就是整体策略的表现度量，也将是近似单调递增的（因为 TRPO 是近似单调递增的）。

2.6.4 下层策略单调递增性：按集划分情况

实际上对于多数强化学习问题，我们应该采用按集划分的方法（episodic case）。每一集就是采集经验样本时的从头到尾的过程。也就是说，对于一个MDP，初始状态 s_0 是服从一个固定的分布 ρ_0 的，并且每次采样会有一个固定的终止条件。事实上，这样的问题建模更为常见。例如，智能体走迷宫问题，一集就是从起点出发，走到出口采样即终止，而不会无限制地走下去。和连续的情况相比，按集划分的情况要得到同样的结论，用到了一个近似，γ_h 和 γ_l 都应该接近 1 并且 k 不能太大。接下来，我们进行证明。

对于一个更新后的底层策略，我们可以把新的联合策略 $\tilde{\pi}$ 的优化目标函数写成是相对于 π 目标函数的优势如下（在 2.6.6节的引理2.1中有详细证明过程）：

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{(s_t^h, a_t^h)\sim\tilde{\pi}}\left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_h(s_t^h, a_t^h)\right]. \tag{2-23}$$

由于 $\eta(\pi)$ 与 $\eta(\tilde{\pi})$ 无关，我们可以把整体策略的优化写成是

$$\max_{\tilde{\pi}} \eta(\tilde{\pi}) = \max_{\tilde{\pi}} \mathbb{E}_{(s_t^h, a_t^h)\sim\tilde{\pi}}\left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_h(s_t^h, a_t^h)\right]. \tag{2-24}$$

我们把新的底层策略记为 $\tilde{\pi}_l$。对于按集分的情况，底层策略 $\tilde{\pi}_l$ 的优化函数的直接优化目标是

$$\eta_l(\tilde{\pi}_l) = \mathbb{E}_{s_0^l\sim\rho_0^l}[V_l(s_0^l)] = \mathbb{E}_{(s_t^l, a_t^l)\sim(\tilde{\pi}_l, \pi_h)}\left[\sum_{t=0,1,2,\ldots} \gamma_l^t r_l(s_t^l, a_t^l)\right]. \tag{2-25}$$

回顾我们对底层策略奖励函数的定义 (2-6) 并把它带入 (2-25)，我们有（详细推导过程在2.6.6节引理2.4当中）

$$\eta_l(\tilde{\pi}_l) = \mathbb{E}_{s_0^l}[V_l(s_0^l)] \approx \frac{1-\gamma_l^k}{1-\gamma_l} \mathbb{E}_{(s_t^h, a_t^h)\sim(\tilde{\pi}_l, \pi_h)}\left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_h(s_t^h, a_t^h)\right]. \tag{2-26}$$

注意到在公式 (2-26) 当中，我们做了一步假设。这个假设在 γ_h 和 γ_l 都足够接近于 1 并且 k 不是非常大的情况下成立。事实上这在多数情况下也是成立的。现在我们比较一下 (2-26) 当中的目标函数和 (2-24) 当中的目标函数. 由于 $\frac{1-\gamma_l^k}{1-\gamma_l}$ 是正常数，很显然的，提升 (2-26) 当中的数值，亦即底层策略的目标函数，就也会增加 π 的目标函数 (2-24).

2.6.5 上下层策略联合优化方法

综上所述，如果我们单次更新，分别固定 π_h 更新 π_l 或者是固定 π_l 更新 π_h，都将使得整体策略的价值 $\eta(\pi_h)$ 单调递增。

在实际操作中，为了提高采样效率，我们每次采样之后，都既优化 π_h 又优化 π_l。实验结果表明，同时优化上下层策略虽然在理论上无法保证整体策略 $\eta(\pi_h)$ 的单调增特性，却能够将采样效率提高为交替优化单层策略的两倍。也就是说，在每次更新 π_l 和 π_h 变化都较小的情况下，同时优化两者并不会导致整体策略效果变差。具体内容在后续实验章节会有详细表述。在这里，我们可以先参照图2.4来理解这两者的训练表现差异。

图 2.4 对比同时进行上下层训练和交替进行上下层训练的训练表现差异。图中，蓝色曲线是每采集一组经验都同时训练上下层的训练区县，橙色曲线是每采集一组经验都只训练上层或者只训练下次的曲线。可以明显地看出，同时训练上下层的表现大约是交替训练上下层采样效率的两倍。

2.6.6 辅助结论的证明

结论2.1源自 TRPO 论文的引理 1[13]，结论2.2也源自该论文，在表达式上有区别，但是思路相同。

引理 2.1：

$$\tilde{\eta}(\pi_h) = \eta(\pi_h) + \mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_{\pi_h}(s_t^h, a_t^h) \right]. \tag{2-27}$$

证明

$$\mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_{\pi_h}(s_t^h, a_t^h) \right] \tag{2-28}$$

$$= \mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} (r_h(s_t^h) + \gamma_h V_{\pi_h}(s_{t+k}^h) - V_{\pi_h}(s_t^h)) \right] \tag{2-29}$$

$$= \mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[-V_{\pi_h}(s_0^h) + \sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} r_h(s_t^h) \right] \tag{2-30}$$

$$= -\eta(\pi_h) + \tilde{\eta}(\pi_h) \tag{2-31}$$

□

引理 2.2：

$$\mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_{\pi_h}(s_t^h, a_t^h) \right] = \sum_{s^h} \tilde{\rho}_{\pi_h}(s^h) \sum_{a^h} \pi_h(a^h|s^h) A_{\pi_h}(s^h, a^h). \tag{2-32}$$

证明

$$\mathbb{E}_{s_0^h, a_0^h, \ldots \sim \pi_h, \mathcal{E}(\tilde{\pi}_l)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_{\pi_h}(s_t^h, a_t^h) \right] \tag{2-33}$$

$$= \sum_{t=0,k,2k,\ldots} \sum_{s^h} P(s_t^h = s^h | \pi_h, \mathcal{E}(\pi_l)) \sum_{a^h} \pi_h(a^h|s^h) \gamma_h^{t/k} A_{\pi_h}(s^h, a^h) \tag{2-34}$$

$$= \sum_{s^h} \sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} P(s_t^h = s^h | \pi_h, \mathcal{E}(\pi_l)) \sum_{a^h} \pi_h(a^h|s^h) A_{\pi_h}(s^h, a^h) \tag{2-35}$$

$$= \sum_{s^h} \tilde{\rho}_{\pi_h}(s^h) \sum_{a^h} \pi_h(a^h|s^h) A_{\pi_h}(s^h, a^h). \tag{2-36}$$

□

引理 2.3： lemma:ER=EV

$$\mathbb{E}_{s \sim \rho_\pi, a \sim \pi}[R(s, a)] = \mathbb{E}[V_{s_0 \sim \rho(s_0)}(s_0)] \tag{2-37}$$

证明

$$\mathbb{E}[V_{s_0 \sim \rho(s_0)}(s_0)] = \mathbb{E}_{s_t, a_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \tag{2-38}$$

$$= \sum_{t=0}^{\infty} \sum_{s} P(s_t = s|\pi) \sum_{a} \pi(a|s)\gamma^t R(s,a) \tag{2-39}$$

$$= \sum_{s} \sum_{t=0}^{\infty} P(s_t = s|\pi)\gamma^t \sum_{a} \pi(a|s)R(s,a) \tag{2-40}$$

$$= \sum_{s} \rho(s|\pi) \sum_{a} \pi(a|s)R(s,a) \tag{2-41}$$

$$= \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[R(s,a)]. \tag{2-42}$$

□

引理 2.4：当我们定义辅助奖励函数 $r_i^l = \frac{1}{k} A_h(s_t^h, a_t^h)$ 的时候，底层策略 $\tilde{\pi}_l$ 的期望起始位置价值函数为

$$\mathbb{E}_{s_0^l}[V_l(s_0^l)] \approx \frac{1-\gamma_l^k}{1-\gamma_l} \mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)} \left[\sum_{t=0,k,2k,\ldots} \gamma_h^{t/k} A_h(s_t^h, a_t^h) \right] \tag{2-43}$$

其中，τ_h 是上层步伐的轨迹 $= s_0^h, a_0^h, s_k^h, a_k^h, \ldots$。这一步近似在一定条件下成立。要求底层折扣 γ_l 和上层折扣值 γ_h 都接近于 1，并且底层步长 k 不是非常大。

证明 我们定义 τ 是联合策略 $(\tilde{\pi}_l, \pi_h)$ 采集的轨迹。

$$\tau = s_0^h, a_0^h, s_1^l, a_1^l, \ldots, s_{nk}^h, a_{nk}^h, s_{nk+1}^l, a_{nk+1}^l \ldots$$

上层轨迹定义为

$$\tau_h = s_0^h, a_0^h, s_k^h, a_k^h, \ldots, s_{nk}^h, a_{nk}^h, \ldots$$

上层的一步下，下层的轨迹 (s_t^h, a_t^h) 为

$$\tau_l(t) = s_t^l, a_t^l, \ldots s_{t+k-1}^l, a_{t+k-1}^l$$

现在我们可以把初始状态的期望价值函数值写为

$$\mathbb{E}_{s_0^l}[V_l(s_0^l)] = \mathbb{E}_{\tau\sim(\tilde{\pi}_l,\pi_h)} \left[\sum_{t=0,1,2,\ldots} \gamma_l^t r_l(s_t^l, a_t^l) \right] \tag{2-44}$$

$$= \mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)} \left[\sum_{t=0,k,2k,\ldots} \mathbb{E}_{\tau_l(t)\sim(\tilde{\pi}_l,\pi_h)} \left[\sum_{i=0}^{k-1} \gamma_l^{t+i} A_h(s_t^h, a_t^h) \right] \right] \tag{2-45}$$

$$= \mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)} \left[\sum_{t=0,k,2k,\ldots} \sum_{i=0}^{k-1} \gamma_l^{t+i} A_h(s_t^h, a_t^h) \right] \tag{2-46}$$

$$=\mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)}\left[\sum_{t=0,k,2k,\ldots}\gamma_l^t\frac{1-\gamma_l^k}{1-\gamma_l}A_h(s_t^h,a_t^h)\right] \tag{2-47}$$

$$=\frac{1-\gamma_l^k}{1-\gamma_l}\mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)}\left[\sum_{t=0,k,2k,\ldots}\gamma_l^t A_h(s_t^h,a_t^h)\right] \tag{2-48}$$

当 γ_l 和 γ_h 都接近于 1 并且 k 不是非常大的时候，我们可以将 γ_l 近似写成是 $\gamma_h^{1/k}$，重新带入上式，得到

$$\begin{aligned}&\frac{1-\gamma_l^k}{1-\gamma_l}\mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)}\left[\sum_{t=0,k,2k,\ldots}\gamma_l^t A_h(s_t^h,a_t^h)\right]\\ \approx&\frac{1-\gamma_l^k}{1-\gamma_l}\mathbb{E}_{\tau_h\sim(\tilde{\pi}_l,\pi_h)}\left[\sum_{t=0,k,2k,\ldots}\gamma_h^{t/k}A_h(s_t^h,a_t^h)\right]\end{aligned} \tag{2-49}$$

□

第 3 章　基准环境实验

3.1　基准环境介绍

3.2　实验环境搭建

3.3　实验结果及对比分析

3.4　模型简化测试——控制变量

3.5　实验附录：参数和细节

第 4 章　自动驾驶环境算法迁移

4.1　自动驾驶实验环境架构

4.2　实验参数

4.3　实验结果及分析

4.4　自动驾驶实验总结

第 5 章　总结及结论

5.1　理论成果总结

5.2　工作内容总结

插图索引

表格索引

公式索引

参考文献

[1] Duan Y, Chen X, Houthooft R, et al. Benchmarking deep reinforcement learning for continuous control[C]//Proceedings of The 33rd International Conference on Machine Learning. [S.l.: s.n.], 2016: 1329-1338.

[2] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. Second ed. [S.l.]: The MIT Press

[3] Sutton R S. Learning to predict by the methods of temporal differences[J]. Machine Learning, 1988, 3(1): 9-44.

[4] Bellman R. A markovian decision process[J]. Indiana Univ. Math. J., 1957, 6: 679-684.

[5] Deep reinforcement learning framework for autonomous driving[J]. Electronic Imaging.

[6] Bellman R E. Dynamic programming[M]. [S.l.]: Dover Publications, Inc., 2003

[7] Watkins C J C H, Dayan P. Q-learning[J]. Machine Learning, 1992, 8(3): 279-292.

[8] van Seijen H. Effective Multi-step Temporal-Difference Learning for Non-Linear Function Approximation[J]. arXiv e-prints, 2016: arXiv:1608.05151.

[9] Li Y. Deep Reinforcement Learning: An Overview[J]. arXiv e-prints, 2017: arXiv:1701.07274.

[10] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[M]// NIPS Deep Learning Workshop. [S.l.: s.n.], 2013

[11] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning [J]. arXiv e-prints, 2015: arXiv:1509.02971.

[12] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning [C]//Balcan M F, Weinberger K Q. Proceedings of Machine Learning Research: volume 48 Proceedings of The 33rd International Conference on Machine Learning. New York, New York, USA: PMLR, 2016: 1928-1937.

[13] Schulman J, Levine S, Moritz P, et al. Trust region policy optimization[C]//ICML. [S.l.: s.n.], 2015.

[14] Wu Y, Mansimov E, Liao S, et al. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation[J]. arXiv e-prints, 2017: arXiv:1708.05144.

[15] Wang Z, Bapst V, Heess N, et al. Sample efficient actor-critic with experience replay[C]// ICLR. [S.l.: s.n.], 2017.

[16] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[Z]. [S.l.: s.n.], 2017.

[17] Silver D, Hubert T, Schrittwieser J, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play: volume 362[Z]. American Association for the Advancement of Science, 2018: 1140-1144.

[18] Tieleman T, Hinton G. Lecture 6.5 rmsprop: Divide the gradient by a running average of its recent magnitude.[J]. COURSERA: Neural Networks for Machine Learning, 2012, 4.

[19] Qian N. On the momentum term in gradient descent learning algorithms[J]. Neural Netw., 1999, 12(1): 145-151.

[20] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv e-prints, 2016: arXiv:1609.04747.

[21] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518: 529 EP -.

[22] Krizhevsky A, Sutskever I, E. Hinton G. Imagenet classification with deep convolutional neural networks[J]. Neural Information Processing Systems, 2012, 25.

[23] Boyd S, Vandenberghe L. Convex optimization[M]. [S.l.]: Cambridge University Press, 2004

[24] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. [S.l.: s.n.], 2014.

[25] Degris T, White M, Sutton R S. Linear off-policy actor-critic[C]//ICML. [S.l.: s.n.], 2012.

[26] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.

[27] Oh J, Guo X, Lee H, et al. Action-conditional video prediction using deep networks in atari games[M]//Advances in Neural Information Processing Systems 28. [S.l.: s.n.]

[28] Racanière S, Weber T, Reichert D, et al. Imagination-augmented agents for deep reinforcement learning[M]//Advances in Neural Information Processing Systems 30. [S.l.: s.n.]

[29] Precup D, Sutton R S, Singh S. Theoretical results on reinforcement learning with temporally abstract options[C]//European Conference on Machine Learning (ECML). [S.l.]: Springer, 1998.

[30] Sutton R S, Precup D, Singh S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning[J]. Artif. Intell., 1999, 112(1-2): 181-211.

[31] Dietterich T G. Hierarchical reinforcement learning with the maxq value function decomposition[J]. J. Artif. Int. Res., 2000, 13(1): 227-303.

[32] Dayan P, Hinton G E. Feudal reinforcement learning[M]//Hanson S J, Cowan J D, Giles C L. Advances in Neural Information Processing Systems 5. [S.l.]: Morgan-Kaufmann, 1993: 271-278

[33] Chentanez N, Barto A G, Singh S P. Intrinsically motivated reinforcement learning[M]//Saul L K, Weiss Y, Bottou L. Advances in Neural Information Processing Systems 17. [S.l.]: MIT Press, 2005: 1281-1288

[34] Bacon P L, Harb J, Precup D. The option-critic architecture[C]//AAAI. [S.l.: s.n.], 2017: 1726–1734.

[35] Kulkarni T D, Narasimhan K, Saeedi A, et al. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation[M]//Advances in Neural Information Processing Systems 29. [S.l.]: Curran Associates, Inc., 2016: 3675-3683

[36] Nachum O, Gu S S, Lee H, et al. Data-efficient hierarchical reinforcement learning[M]//Bengio S, Wallach H, Larochelle H, et al. Advances in Neural Information Processing Systems 31. [S.l.: s.n.], 2018: 3303-3313

[37] Levy A, Jr. R P, Saenko K. Learning multi-level hierarchies with hindsight[C]//Proceedings of The 33rd International Conference on Machine Learning. [S.l.: s.n.], 2018.

[38] Vezhnevets A S, Osindero S, Schaul T, et al. Feudal networks for hierarchical reinforcement learning[C]//International Conference on Machine Learning. [S.l.: s.n.], 2017: 3540-3549.

[39] Ghosh D, Gupta A, Levine S. Learning Actionable Representations with Goal-Conditioned Policies[J]. arXiv e-prints, 2018.

[40] Nachum O, Gu S, Lee H, et al. Near-Optimal Representation Learning for Hierarchical Reinforcement Learning[J]. arXiv e-prints, 2018.

[41] Eysenbach B, Gupta A, Ibarz J, et al. Diversity is All You Need: Learning Skills without a Reward Function[J]. arXiv e-prints, 2018.

[42] Florensa C, Duan Y, Abbeel P. Stochastic neural networks for hierarchical reinforcement learning[C]//Proceedings of The 34th International Conference on Machine Learning. [S.l.: s.n.], 2017.

[43] Heess N, Wayne G, Tassa Y, et al. Learning and Transfer of Modulated Locomotor Controllers [J]. arXiv e-prints, 2016.

[44] Frans K, Ho J, Chen X, et al. META LEARNING SHARED HIERARCHIES[C]//International Conference on Learning Representations. [S.l.: s.n.], 2018.

[45] Jang E, Gu S, Poole B. Categorical reparameterization with gumbel-softmax[C]//International Conference on Learning Representations. [S.l.: s.n.], 2017.

[46] Dwiel Z, Candadai M, Phielipp M J, et al. Hierarchical policy learning is sensitive to goal space design[Z]. [S.l.: s.n.], 2019.

[47] Schulman J, Moritz P, Levine S, et al. High-dimensional continuous control using generalized advantage estimation[C]//Proceedings of the International Conference on Learning Representations (ICLR). [S.l.: s.n.], 2016.

[48] Sutton R S, McAllester D A, Singh S P, et al. Policy gradient methods for reinforcement learning with function approximation[M]//Solla S A, Leen T K, Müller K. Advances in Neural Information Processing Systems 12. [S.l.]: MIT Press, 2000: 1057-1063

[49] Kakade S, Langford J. Approximately optimal approximate reinforcement learning[C]//ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning. [S.l.: s.n.], 2002: 267-274.

致　谢

衷心感谢导师 xxx 教授和物理系 xxx 副教授对本人的精心指导。他们的言传身教将使我终生受益。

在美国麻省理工学院化学系进行九个月的合作研究期间，承蒙 xxx 教授热心指导与帮助，不胜感激。感谢 xx 实验室主任 xx 教授，以及实验室全体老师和同学们的热情帮助和支持！本课题承蒙国家自然科学基金资助，特此致谢。

感谢 LaTeX 和 ThuThesis[?]，帮我节省了不少时间。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

清 华 大 学

综 合 论 文 训 练
开 题 报 告

选 题 范 围: 基 于 有 模 型 强 化 学 习 的 自 动 驾 驶 环 岛 路 径 规 划

系　　别: 机 械 工 程 系

专　　业: 机 械 工 程

姓　　名: 王 芮

指导教师: 雷 丽 萍 副 教 授

2019 年 4 月 29 日

目 录

主要符号对照表

s	状态（任意状态）
a	行动（任意行动）
π	策略
$\pi(s)$	策略函数
$\pi(a\|s)$	策略在状态 s 下采取行动 a 的概率
r	奖励
$r(s,a)$	在状态 s 下采取行动 a 获得的奖励
G_t	时间 t 之后的总奖励
$v(s)$	状态 s 的价值函数
$q_\pi(s,a)$	在策略 π 下，在状态 s 下采取行动 a 的价值
$V(s)$	状态 s 的价值函数的估计值
$Q(s,a)$	状态 s，动作 a 的价值的估计值
$p(s'\|s,a)$	在状态 s 采取动作 a，状态转移到 s' 的概率
$p(s',r\|s,a)$	在状态 s 采取动作 a，状态转移到 s' 并且获得奖励 r 的概率
$\pi(a\|s,\theta)$	策略函数的参数是 θ，在状态 s 下采取行动 a 的概率
π_θ	策略函数，其参数为 θ
$\nabla\pi(a\|s,\theta)$	策略函数的参数是 θ，在状态 s 下采取行动 a 的概率对 θ 的导数
γ	总奖励 G_t 计算时，对于后续 R_{t+t_i} 的折扣率
ρ	重要性采样比例（importance-sampling ratio）
δ_t	时序差异误差（TD error）
RL	强化学习（reinforcement learning）
MPC	模型预测控制（model-predictive control）
TD	时序差异（temporal-difference）
MDP	马尔可夫过程 (Markov Decision Process)
POMDP	部分可观测的马尔可夫过程 (Partially Observable Markov Decision Process)
DQN	深度 Q-学习神经网络（Deep Q Network）
DRQN	深度循环 Q-学习神经网络（Deep Recurrent Q Network）

1 研究课题介绍

1.1 选题背景及意义

1.1.1 选题背景

自动驾驶成为近年来人工智能领域发展最迅猛的技术之一，Alphabet 旗下的子公司 Waymo 为代表的一批自动驾驶企业已经让车辆上路[1]。不过在复杂场景下，这些车辆的行为仍然不够智能，需要人类驾驶员的干预。

自 Deepmind 的科学家在 *Nature* 发文提出深度强化学习在游戏中的应用后[2]，强化学习作为人工智能的下一个发展大方向，也在 2015-2018 年迎来了爆发点。然而，尽管强化学习在棋牌等游戏中获得了巨大的成功，目前的强化学习在自动驾驶上仍没有成熟的应用。

把强化学习应用于自动驾驶是一个极其自然的想法。自动驾驶的应用场景本质上是移动机器人的问题，而强化学习应用于机器人更是由来已久[3][4][5]。因此，在这个项目中我们考虑了一个较为复杂的自动驾驶场景——环岛场景，希望应用强化学习框架，设计出适合于自动驾驶在此类场景中决策的算法。

1.1.2 自动驾驶的问题

把强化学习应用于自动驾驶是一个自然的想法，我们可以利用强化学习进行决策，由此构建一个自动驾驶的框架[5]。

在图1中，我们绘制了一个框架的示意图。可以看出，决策（decision-making, planning）是自动驾驶的‘‘大脑’’，是在不同场景中处理问题的核心算法。

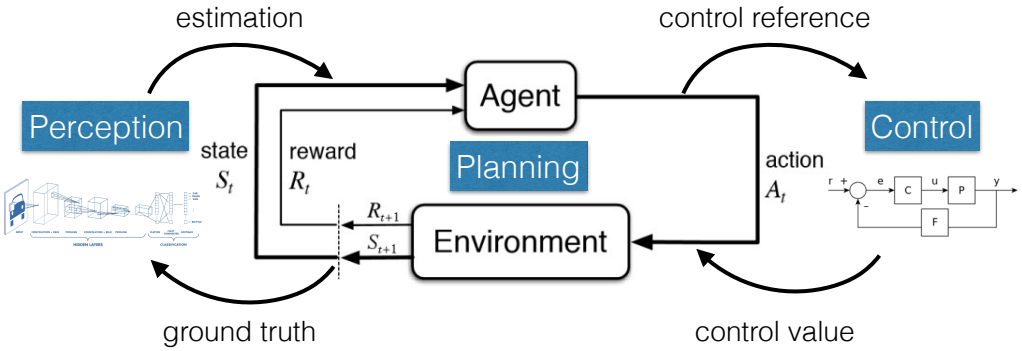


图 1　自动驾驶的基本框架：三个回路，从左到右依次是感知，决策，和控制

事实上，感知算法和控制算法目前都已经做的相应较为成熟，并且由于其一般性更强，因此解决得比较好。相比之下，自动驾驶的决策目前仍然依赖于人

工基于规则编码，缺乏泛化的能力。环岛路径规划涉及到与其他智能体的高频交互，例如超车换道，路权退让，环岛路径汇入汇出等，是自动驾驶的一个复杂场景。

1.1.3 强化学习的问题

强化学习是一种强人工智能范式。它目前的应用场景集中于虚拟游戏和机器人领域，其中，能够在产品中得到成熟使用的只有游戏环境。当前的强化学习算法存在以下问题：

- 样本利用率非常低，需要进行大量训练。
- 对于环境的过拟合，导致已习得策略的迁移困难。
- 真实环境的不稳定性，导致任务无法识别，习得策略应用困难。

因此，我们希望通过针对环岛路径规划这一具体问题的研究，对强化学习的应用和如何解决这些缺点进行一些探索。具体的，我们将从结合有模型与无模型强化学习的角度，和分层、多任务强化学习的角度，探究提升强化学习样本利用率和解决复杂任务能力的方法。在后续部分我们会详细阐释我们采用的方法。

1.2 场景描述

在本节，我们会对环岛路径规划问题的具体实验设计进行介绍。

我们在仿真环境中建设双车道环岛。分为三进口、四进口两类；每一类场景有根据车流密度 1 辆/5s 和 1 辆/20s（新车进入环岛的频率）分为两种情况，这样一种形成四种实验场景，如示意图2所示。

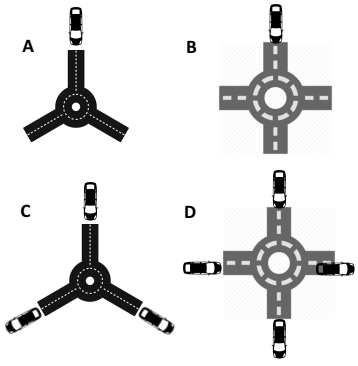


图 2 四个实验场景，(A) 三进口，1 辆/20s (B) 四进口，1 辆/20s (C) 三进口，1 辆/5s (D) 四进口，1 辆/5s

所有其他车辆都遵循基于规则的驾驶方案进行驾驶。

之所以选择四个实验场景，是基于我们对算法的设计。我们将有三套不同的算法在实验环境中进行对比，具体对比内容将在后面介绍。

具体任务类似于一个最优控制问题。我们对车辆的控制量 u 是一个二维向量，第一维代表刹车/油门强度，第二维表示是否变道。

$$u = [u_1\ u_2]^T,\ u_1 \in [-1, 1],\ u_2 \in 0, 1$$

控制目标是让车辆尽快、稳定地无事故地通过环岛。具体如何量化这些指标，取决于我们如何设计强化学习的奖励函数。初始条件是车辆以一个定速度到达环岛入口。和最优控制的区别在于，我们无法用一个简单的微分方程表达车辆的状态方程，因而这个问题也不再是简单的最优控制问题。在理论基础一节，我们会介绍强化学习如何去表达一个复杂环境的动态特性。

1.3 研究方案

实验平台采用 SUMO[6]+Webots[7] 进行搭建。SUMO 负责生成交通流，而 Webots 则用于构建车辆模型。

我们采用强化学习范式来解决环岛路径规划问题。我们把算法研究分为三个阶段。

1. 第 I 阶段，利用现有算法，解决部分可观测的马尔可夫问题的强化学习。
2. 第 II 阶段，结合无模型和基于模型的强化学习算法，用于提升采样效率。
3. 第 III 阶段，多任务强化学习框架，具体包括复杂任务分解、相似子任务识别、知识共享、子任务策略重组的方法。

这三个阶段的算法将会和之前介绍的四个场景进行结合，从而用于检验算法效果，如表1所示。

表 1　实验设计

实验场景	**I 和 II 对比**	**I 和 III 对比**	**II 和 III 对比**
A	训练时间	训练时间	×
B	获得奖励值	获得奖励值	训练时间
C	获得奖励值	获得奖励值	训练时间
D	×	×	获得奖励值

2 理论基础

由于我们的问题描述是一个比较典型化的强化学习问题，输入是由仿真环境提供的低维度抽象信息，而输出是在 $[-1, 1]$ 之间的一个值，因此模型本身比较简单。仿真环境虽然提供了较为具体和复杂的场景，但是在强化学习理论中，这都作为 ‘‘环境’’ 被理解，而暂不考虑 ‘‘环境’’ 内部的动态特性问题。我们在下一阶段的工作中会考虑将模型进行一定程度的拓展，更多地囊括进 ‘‘环境’’ 本身的特性，但也不会超脱出典型强化学习模型的范畴。因此，我们在理论部分将会首先对现有的强化学习理论进行一个总结，再拓展到强化学习在自动驾驶问题中的运用。

2.1 强化学习文献综述

在强化学习的著作 *Reinforcement Learning An Introduction*[8] 中，Sutton 对强化学习的理论进行了详尽的讨论。在这里，我们对强化学习的发展做一个精炼的综述，并从契合课题的框架进行一些讨论。另外，我们还将详细讨论强化学习的一些最新进展和它们在本课题中的应用。

2.1.1 问题描述

Sutton 在 1988 年首次提出强化学习问题的一般描述[9]。诸如学习走路、学习下棋的一般性问题都可以归为强化学习的问题。强化学习问题的四个基本元素是：一个策略（policy），一个奖励函数（reward），一个价值函数（value function）和一个环境模型（model）。例如，自动驾驶问题中，智能体面对它观察到的一个周边车流的**环境**，会给出一个输出，例如以某一加速度加速。在采取这一行动之后，智能体会得到一个**奖励**（可能有延迟），例如它到达了目的地，我们给它价值 100 的奖励。而如何决定加速度，就是我们想要智能体去学习的**策略**。在制定策略的过程中，智能体会去试图衡量目前所处状态的**价值**（价值函数），例如周围车辆非常拥堵，这个状态的价值就比较低，是我们希望避免进入的状态；而车前非常空旷则可能是我们赋予价值较高，希望进入的状态。

在这里，我们将用一些研究者所默认的记号对我们的问题进行数学抽象。假设下标 t 表示任一时刻，而 S_t 代表某一时刻的状态，A_t 表示智能体采取的行动，π 表示智能体的策略，R_t 表示智能体某一步获得的奖励。我们的目标是让智能

体通过学习，获得尽可能大的总奖励，记为 G。有时我们可以定义

$$G = \sum_t R_t$$

但是多数时候我们会考虑加入一个折扣 $\gamma(0 < \gamma < 1)$，认为一个行为直接获得的奖励对它更有意义，而很久以后获得的奖励与这个行为的关联越来越小，记为

$$G_{t_0} = \sum_{t=t0}^{t_n} \gamma^{t-t_0} R_t \tag{1}$$

其中 t_n 要么是一个过程因为某种原因终止的时刻，要么是 ∞。

强化学习解决的是一个马尔可夫决策过程（MDP）[10]。在 MDP 中，S_{t+1} 只取决于 S_t 和 A_t，而不受到之前历史过程的影响。也就是说，S_t 状态本身就包含了它所有的历史对后续过程的影响。在这个假设下，我们可以这样表达强化学习的目标[5]：记 $v^\pi(s)$ 为状态 s 的价值函数。我们有

$$v^\pi(s) = E[G_{t_0}|S_{t_0} = s, \pi(s)]$$

$$J(\pi) = \max_\pi E(v^\pi(s))$$

$$\pi^*(s) = \operatorname*{argmax}_\pi J(\pi)$$

强化学习的终极目标就是找到这个 $\pi^*(s)$。Bellman 在 *Dynamic Programming* 一书中提出了著名的 Bellman 最优性等式[11]

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{2}$$

同时，我们用 $q(s, a)$ 表示在状态 s 下采取行动 a 可以获得的价值，那么 Bellman 最优性等式又可以表示为

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{3}$$

这两个表达式在后续的推导中会非常有用。

2.1.2 典型算法综述

强化学习包含许多的算法，也有很多分类方法。其中，有一大类是列表方法，可以理解为是把 $v(s)$ 保存起来的方法；另一大类是近似方法，也就是用一

个函数去近似 $v(s)$[8]。其中，第二类方法在近年来使用最多，神经网络使得函数可以近似非常复杂的映射关系，也让许多复杂问题通过强化学习得到了解决。

列表式的强化学习方法包括动态规划[11]，蒙特卡罗方法，时序差分学习（Temporal-difference learning，简称 TD 方法，包括 Sarsa[8], Q-learning[12])[9]，多步自助法[13]（n-step bootstrapping），Dyna 规划方法[8] 等。

TD 方法是列表法强化学习的核心。这一族的算法有许多衍生版本，但其核心是利用经验不断更新价值函数 $V(s)$，直至其收敛。更新的基本模式为

$$V(s) \leftarrow V(s) + \alpha[r + V(s') - V(s)]$$

其中，α 表示函数更新的步幅。而当我们用 $Q(s,a)$ 去替代 $V(s)$ 的时候，算法就成为了 Sarsa（state, action, reward, state, action[14]）：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

注意这里，a' 是依据当前策略 π 采取的行动。而当我们把 $Q(s',a')$ 替换成 $\max_{a'} Q(s',a')$ 的时候，算法就成为了 Q-学习法（Q-learning）：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{4}$$

在这里，我们发现 Sarsa 和 Q-学习的重要区别：Sarsa 是同策略学习的（on-policy），估算 Q 值时采用的策略就是当前策略，而 Q-学习是采用异策略（off-policy）学习的，估算 Q 值时采用的策略不是当前策略，而是最优策略。同策略与异策略也作为强化学习中的两种并行方法，本身并不改变算法架构，但是会影响收敛性等特性。许多算法都可以在这个维度上进行变异。

注意到在以上的算法中，我们每一步都对 Q 或者 V 进行更新。在多步骤自助法（n-step bootstrapping）中，我们则将更新进行延迟。例如，某一行动结束，n 步过后，我们才观察这一行为造成的影响，并且对它的值进行更新，其更新方式如下

$$V(S_t) \leftarrow V(S_t) + \alpha(G_{t:t+n} - V(S_t))$$

其中，$G_{t:t+n} = \sum_{i=1}^{n} \gamma^{i-1} R_{t+i} + \gamma^n V(S_{t+n})$

Sutton 等人指出，多步骤自助法强化学习在训练时的表现往往比蒙特卡罗和 TD 方法都要更优[8]。

归功于深度强化学习的提出[15][14]，近似函数的方法在近年来得到了极大的发展。近似函数有两种基本思路。第一种，是用某一类型的函数去近似状态的价值函数 $v(s)$，从而帮助智能体根据不同的 v 采取不同的策略。第二种思路是跳过 v 的计算，而直接使用函数去近似一个策略 π。在第二种思路中，$\pi(s)$ 的输出将直接是智能体应该采取的行为（或者行为的一个概率分布）。这种方法被称为 ‘‘策略优化’’，围绕策略优化问题衍生出一系列的策略梯度求解方法，成为近年来强化学习的热点研究方向之一。进一步的，在策略优化问题中，如果我们同时对价值函数 $v(s)$ 进行近似和更新，那么可以得到一类框架，我们称之为 actor-critic 架构[8]。在后续的篇目中，我们会对此进行具体介绍。

在深度强化学习中，比较著名的算法有 DDPG[16]，A2C[17]，TRPO[18]，ACKTR[19]，ACER[20]，PPO[21] 等。这些算法全部忽略了环境的模型，或者环境模型是学习得到的（TRPO）。还有一类算法，其中的环境模型是提前设计建模得到的而非习得，其中有代表性的就是 AlphaZero[22]。这类算法也是我们想探究的方向——是否通过提前建模，能够让自动驾驶车辆在环岛这类复杂环境中有更好的表现? 在接下来的章节中，我们会详细介绍其中有代表性，并且在我们的场景下易于使用的算法。

另外，在对价值函数进行优化时（梯度下降），不同的算法可能带来不同的结果。A3C 采用了 RMSProp 算法[23]，我们往往还可以选择随机梯度下降法（SGD），带动量的随机梯度下降法[24] 等方法，在算法设计时需要进行尝试和选择。Sutton 在书中对此进行了讨论[8]。此外，Ruder 在 2016 的一篇综述文中对各种梯度下降算法进行了详尽的探讨[25]。

2.1.3 无模型的深度强化学习

Mnih 等人在 2013 年首度提出深度 Q-网络（deep Q-network，简称 DQN)[15]，2015 年，他们的另一篇论文同样描述了 DQN[2]，激发了一轮深度神经网络与强化学习结合的热潮[14]。DQN 的基本思路是创造一个很一般的端到端的网络 $Q(s,a|\theta)$，其中 θ 是这个网络的参数。例如对于一个游戏，网络的输入直接就是未处理的图像，输出直接就是在某个阶段，游戏中的动作对应的价值。在传统 Q-learning 的框架下，DQN 在每一步更新 $q(s,a)$ 值的时候，不是直接使用 (4)，而是利用梯度下降法去更新网络 Q 的参数 θ，使得 Q 估值总损失最小

$$\nabla_{\theta_i} Loss(\theta_i) = E_{s,a,s'}[(r + \gamma \max_{a'} Q(s', a'|\theta_{i-1}) - Q(s,a|\theta_i)) \nabla_{\theta_i} Q(s,a|\theta_i)]$$

这里 DQN 的网络结构参考了 Hinton 等人提出的 AlexNet[26]，也就是典型的卷积神经网络（CNN）结构。DQN 的最大贡献在于，由于使用了经验回放的方法，DQN 可以收敛，从而使得深度神经网络在强化学习中的应用成为了可能。由于神经网络的高度非线性，DQN 得以模拟足够复杂的场景。DQN 在 Atari 系列游戏中取得了突破性的进展，甚至在一些游戏中超越了人类专家的水平。

不过由于我们想探究的问题更多地涉及到低维度的传感器信息输入，而不是高维度的图像直接输入，因此 DQN 一类的方法对我们的帮助并不是很大。而策略梯度方法则对我们有很大的借鉴意义，因此我们也将集中讨论这一类的方法。

前面我们提到，在策略优化问题中，有一类架构被称为 actor-critic 架构。Mnih 等人在 2016 年提出的 A3C（asynchronous advantage actor-critic）算法[17] 成为应用 actor-critic 架构的经典案例。A3C 算法同时维护一个策略函数 $\pi(a|s;\theta)$ 和一个价值函数的估计 $V(s;\theta_v)$，这两个函数均为卷积神经网络 CNN，并且两个网络除了输出层以外的层均是共享的。其中，$\pi(a|s;\theta)$ 称为 actor，而 $V(s;\theta_v)$ 称为 critic。actor-critic 会计算一个 ‘‘优势’’，定义为 $\delta = R + \gamma V(S';\theta_v) - V(S;\theta_v)$[8]。A3C 借鉴了多步骤自助法的思路（设步数为 k），在计算优势时，变异为 $\delta_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(S_{t+k};\theta_v) - V(S;\theta_v)$。每一步，我们依然对 θ 和 θ_v 进行更新，梯度的计算方法如下（注意计算梯度以后，还需要选择一个梯度下降方法对 θ 和 θ_v 进行更新）

$$d\theta = \delta_t \bigtriangledown_\theta log_\pi(A_t|S_t;\theta) \tag{5}$$

$$d\theta_v = \delta_t \bigtriangledown_{\theta_v} \delta_t^2 \tag{6}$$

如果我们采用多个智能体并行训练，把它们各自的梯度叠加在一起（也就是累加每个智能体各自求得的 (5)(6) 式），最后统一更新 θ 和 θ_v，就称之为异步训练方法，这种算法也因此被称为 asynchronous advantage actor-critic。如果条件受限没有异步，而只是用一个智能体进行训练，这个算法就称为 A2C（advantage actor-critic）。actor-critic 架构被认为是能够有效加速训练的，因此越来越多的算法都基于这一架构进行设计。

2015 年，Schulamn 等人提出了经典的（Trust Region Policy Optimization, TRPO）算法[18]。在文章中，他们首先从理论上证明了一种策略迭代更新的 MM（minorization-maximization）算法，可以确保策略对应的总价值（$\eta(\pi) = E_{s_0}[\sum \gamma^t r(s_t);\pi_\theta]$）单调不下降。依据这一理论基础，他们进行了多步近似，将

MM 算法中的优化（求 argmax）的问题转化为了一个更简单的加限制的优化问题（信任空间优化，trust-region optimization）。在进行进一步近似后，优化式中的期望形式可以用蒙特卡罗方法进行估算。最后，优化式本身则利用一种近似的方法进行优化。在实验中，他们采用了神经网络作为策略 π 的表达。TRPO 的更新方法是：

$$\begin{aligned} \theta &= \underset{\theta}{\operatorname{argmax}}\, E\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{\theta_{old}}(s,a)\right] \\ &\text{subject to } E[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))] \le \delta \end{aligned} \tag{7}$$

其中，δ 是一个认为选定的小值。注意，在论文的附录中，作者提出采用一阶近似对上式进行求解，优化问题就可以就变为一个典型的拉格朗日共轭问题[27]。不过，由于 TRPO 只是保证了每一步更新，策略都变得更好，却没有考虑探索性，因此，TRPO 的结果很容易困在局部最优值。在 Schulman 之后，Wu 等人于 2017 年又提出了一个改进算法 ACKTR[19]（Actor Critic using Kronecker-Factored Trust Region），采用 actor-critic 架构，并利用 Kronecker 因子去近似梯度下降。同年，Schulman 等人又提出了 PPO 算法[21]，在实现难度和算法表现方面取得了平衡。PPO 算法的基本思路有三点：

- 去除 TRPO 的优化限制条件，在 TRPO 的基础上，采用一个 clip 函数，从而阻止 θ 每一次更新太大，确保收敛
- 在优化的函数里，通过引入系数的形式，同时优化 actor 和 critic，整个优化过程只有一个优化目标函数
- 通过在优化目标中加入一个熵奖励，增加算法的探索性

PPO 把强化学习问题转化为一个形式简单的单一函数优化问题，在许多方面取得了比以往算法（A2C, ACER, TRPO, etc.）更优的表现。

在 2015 年，Silver 首度提出了针对连续动作空间的确定性算法 DPG（deterministic policy gradient)[28]。与确定性算法对应的是随机算法，也就是策略是行动的概率分布。在高维的动作空间内（例如动作是连续的），DPG 比与之对应的随机算法获得了更优的结果。DPG 采用 actor-critic 架构，并且是异策略（off-policy）训练的。注意这里，DPG 的 actor-critic 架构与 Sutton 书中提到的 actor-critic 架构有所区别。它是不以价值函数作为基准的，而正常的 actor-critic 架构应该以价值函数作为基准，计算优势，再进行求导[8]，参见前面提到的 A3C 算法[17]。同时，DPG 算法采用的是 Q-learning 结构，不是用 $V(s)$ 对价值进行估计，而是用 $Q(s,\pi(s))$ 对价值进行估计。由于算法本身是确定性的，DPG 与随机异策略

actor-critic 架构[29] 不同，不需要针对策略进行重要性采样（importance sampling，在[8] 中有详细讨论）。DPG 本身没有选择神经网络，而是可以采用一般的函数，尤其是线性函数。

DPG 算法对 $\pi(s)$ 和 $\mu(s)$ 的更新方法如下：

$$\begin{aligned}
&\delta_t = r_t + \gamma Q(S_{t+1}, A_{t+1}; \theta_Q) - Q(S_t, A_t; \theta_Q) \\
&\theta_Q \leftarrow \theta_Q + \alpha_{\theta_Q} \delta_t \bigtriangledown_{\theta_Q} Q(s, a; \theta_Q) \\
&\theta_\mu \leftarrow \theta_\mu + \alpha_{\theta_\mu} \bigtriangledown_{\theta_\mu} \mu(s; \theta_\mu) \bigtriangledown_a Q(s, a; \theta_Q)|_{a=\mu(s;\theta_\mu)}
\end{aligned} \tag{8}$$

基于 DPG，Lillicrap 等人提出的改进算法，DDPG 算法[16]，则全面采用了深度神经网络。为了能够让训练收敛，DDPG 使用了一个目标网络（target network）的概念。目标网络的定义是缓慢地跟踪依据梯度下降优化得到的网络参数 θ_Q 和 θ_μ，θ_Q 是状态行动对的价值估计网络 Q 的参数，而 θ_μ 是 actor-critic 架构中 actor 采用的策略的参数。目标网络参数 θ'_Q 和 θ'_μ 的更新方法如下（在每一步迭代过后进行这个更新）：

$$\theta' = \tau\theta + (1 - \tau)\theta', \tau \ll 1$$

这样，相当于强迫目标网络缓慢地变化，从而增强了训练的收敛性。目标网络主要用于带入 (8) 中 δ_t 的计算。同时，DDPG 采用了经验回放的方法，在每一步迭代中都采用之前历史中一批随机挑选的过往经验进行批量计算，而非一步的结果，这样可以确保网络训练时不带有时序偏差。另外，DDPG 没有像 DPG 一样采用异策略学习，而是在策略基础上加上一个噪声分布以确保 ‘‘探索’’ 性。

2.1.4 有模型的深度强化学习

有模型的深度强化学习问题可以分为模型是 ‘‘习得’’ 的和模型是给定的两种。在这里，我们首先关注一下给定模型的强化学习的应用。

在模型给定的算法中，最著名的例子就是 2016 年提出的 AlphaGo[30]。Deepmind 在 2017 年进一步提出了 AlphaZero[22]，其算法和 AlphaGo 相比有几大特点：

1. AlphaZero 完全采用逐步迭代的方式，而 AlphaGo 在训练中会挑选之前出现过的最好策略
2. AlphaZero 拥有的先验知识只有棋盘的格局和下棋的规则，并且这些知识只用来进行模拟

3. AlphaZero 对参数的变动更加鲁棒，不同棋类游戏用的都是同一套参数
4. AlphaZero 在训练中没有对输入进行任何数据加强的操作，而 AlphaGo 则对棋盘进行了镜像操作

AlphaZero 采用蒙特卡罗树状搜索（Monte Carlo Tree Search, MCTS[8]）的方法，每次优先访问训练历史中访问次数低，但访问概率 $\pi(a|s)$ 应该较高，且 $Q(s,a)$ 的状态行为对 (s,a)。同时，在此基础上加上一个噪声（在 AlphaZero 的例子中，一个狄利赫里分布的噪声），确保探索性。AlphaZero 采用深度卷积神经网络作为策略 $\pi(s)$ 的拟合函数，其输出为 $\pi(s) = (\mathbf{p}, v)$，其中，$\mathbf{p}$ 为在某个状态下选择各个行为的概率，v 为根据 s 推测的状态估值,$v = \mathbb{E}[z]$（胜负平局，z 的值分别为 $+1, -1, 0$）。针对该网络定义的损失函数为

$$L = (z - v)^2 - \pi^T log(p) + \lambda||\theta||^2 \tag{9}$$

可以看出，这个损失函数试图使 v 的预测和棋局结果 z 更加接近；π 代表了训练中各个动作的搜索概率，损失函数试图让神经网络的输出 (p) 与之尽可能接近。

这里所谓“给定模型”，其实就是环境是已知的，不需要真的和环境交互，而是在仿真里直接进行下棋这个活动即可。然而，这也意味着这个问题本身与直接和未知环境交互并没有区别。尽管模型给定，算法并没有从模型本身得出什么规律，因此这个模型本质上就是一个环境。

在强化学习问题中，从与环境的交互历史中逐渐学习到模型，则是另一个有趣的思路。在[8] 中，Dyna 架构就是这样一种思路。在强化学习中，利用模型去近似求解 $v(s)$ 是一个 ‘‘规划’’（planning）问题。传统的列表式方法可以根据训练中得到的环境反馈不断更新模型。2015 年，Oh 等人在[31] 一文中提出了一种利用神经网络学习高维度环境的方法，使得复杂环境的模拟也成为了可能。

基于[31] 的结果，Racanière 等人在论文[32] 中提出了一种利用模型去增强无模型强化学习的方案，简称为 I2C 算法。这种方法本质上是细化了的 A3C 算法。注意到 A3C 本身是一个无模型的方法，它仅仅依赖于 $\pi(a|s)$ 和 $v(s)$ 的估计（回顾一下 (5)(6) 两个更新式）。但是，I2C 算法修改了 θ 的组织形式，构建了一个较为复杂的网络结构，因而引入了环境模型。其基本架构如图3所示。

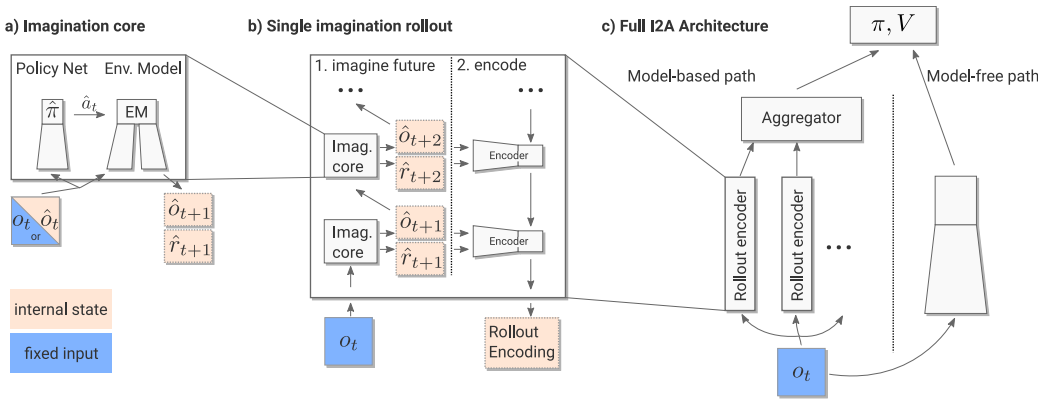


图 3　I2A 的基本结构：a) 一个预先训练好的神经网络，可以预测在策略 $\hat{\pi}$ 下，状态 s 的一系列时序变化；b) 一个子网络，用于 ‘‘理解”a) 的输出结果；c) 整体架构

图3c) 展示了网络整体架构，可以看到整体网络的输入为 o_t，也就是对状态 S_t 的观测，输出为策略 π 和价值函数估值 V（用于 actor-critic 架构）。因此，这个结构满足 A3C 的基本框架。在框架内部，网络结构采用了含有预测模型网络处理的输入，也采用了无模型网络处理的输入，分别作为两个分支，最后通过一个子网络汇合。具体的结构参数可以参见论文[32] 的附录。通过引入这个特殊的网络结构，I2A 算法将习得的模型结合进了原有的无模型学习框架中。同时，实验结果表明，这里的模型可以有错误和不精确的地方，学习结果依然是鲁棒的。

2.2　部分可观测的马尔可夫决策过程

我们要解决的问题是典型的部分可观测马尔科夫决策过程。当在马尔可夫过程中，状态 s 不能被完全观测时（例如汽车只能看到自己视野范围内的几辆车），我们称这个问题为部分可观测马尔科夫决策过程（partially observable Markov decision process, POMDP）。POMDP 在强化学习社区中被当做一种特殊的问题看待，也围绕这类问题衍生出许多解决方案。2015 年，Hausknecht 等人提出了 DRQN[33]（deep recurrent Q network），把它应用于经过修改的部分可观测的 Atari 游戏上并取得了比 DQN 更好的结果。

DRQN 利用了循环神经网络（RNN，recurrent neural network）和 LSTM[34]（long short term memory）的概念，用这样一层 LSTM 神经元替代 DQN 网络全联通层的第一层。在我们的设计中，LSTM 也被用来解决状态部分可观测的问题。

2.3 有模型和无模型结合的高效强化学习

有模型强化学习和无模型强化学习有各自的优缺点和适用场合。例如，有模型强化学习可以大大提高样本利用率，但是如果模型有偏差，会造成习得错误的策略，对于一些特别复杂的问题，很难得到精准的模型。无模型的强化学习可以应用在复杂场景，但是样本利用率很低，在一些容易规范化的问题中不宜采用。因此我们考虑在环岛路径规划中结合两者的优势。

2017 年，Bansal 等人提出 MBMF 算法（Model-Based Priors for Model-Free RL）[35]。记策略为 π_θ，代价函数为 $J = -G_{t:t+n}$。MBMF 最终想要得到的是代价函数 J 和策略参数 θ 的关系 $J(\theta)$，从而优化策略的参数。MBMF 的算法如下：首先根据观察，训练出系统的动力学特性，记为一个函数 $s_{t+1} = f(s_t, a)$（模型）。接下来，利用 f，可以预测在某个策略 π_θ 下，对应的奖励值期望（原文称之为代价函数 J），记为 $J_{MB}(\theta)$。这是基于模型得到的代价函数估计值。考虑到这个模型存在偏差，MBMF 的 MF 部分把 $J_{MB}(\theta)$ 作为初值，利用观察到的数据对 $(\theta, J(\theta))$，进行拟合函数优化，优化得到最终的函数 $J_{MF}(\theta)$。特别地，在上述拟合函数 f, J_{MB}, J_{MF} 当中，他们没有使用神经网络，而是使用了高斯过程[36]（gaussian process, GP）和贝叶斯优化[37]（Bayesian optimization, BO）的方法。MBMF 的论文认为，通过引入模型，提高了采样效率，加速了学习过程，通过无模型的训练部分，避免了模型的偏差问题。从而两者的优势被结合起来。因此，MBMF 算法的关键在于，把 J_{MB} 作为贝叶斯优化的一个 ‘‘初值’’。与 Dyna 架构[8] 不同，MBMF 的 MF 部分只采用真实环境里采集的数据集 $(\theta, J(\theta))$ 对拟合函数 J_{MF} 进行优化，而不像 Dyna 架构利用学习到的模型去生成了大量的合成数据。

2018 年，Pong 等人提出了结合有模型和无模型的一种训练方法时许差异模型（temporal-difference model, TDM）[38]。之前的一些论文通过目标条件价值函数（goal-conditioned value function）$Q(s, a, s_g)$ 的定义，将无模型强化学习（仅学习 Q 值）和有模型强化学习（需要学习状态转移概率分布 $p(s_{t+1}|s_t, a)$）被联系在了一起。其中，s_g 表示某一目标状态。如果定义奖励函数 $r(s_t, a, s_{t+1}, s_g) = -D(s_{t+1}, s_g)$，其中，$D$ 为一个表示状态间距离的函数，那么 $Q(s, a, s_g)$ 最终将收敛于 $-D(s_{t+1}, s_g)$。$Q(s, a, s_g)$ 表示的含义是就是从 s 转移到 s_g 的能力，因此包含了模型的信息，但同时，这个方法没有显式地去学习状态转移函数，无模型强化学习方法能够用的地方它也都能用。TDM 的贡献在于引入了时序差异的目标条件价值函数，$Q(s, a, s_g, \tau)$，表示在 τ 步之内，状态 s 采取行动 a 后，再通过某

种方式转移到 s_g 的能力。因此，Q 的定义（和更新方法）为

$$Q(s_t, a_t, s_g, \tau) = \mathbb{E}_{s_{t+1}\sim s_t, a}[-D(s_{t+1}, s_g)1[\tau = 0] + \max_a Q(s_{t+1}, a, s_g, \tau - 1)1[\tau \neq 0]] \tag{10}$$

和传统强化学习对 G 的定义 (1) 不同，TDM 算法不采用折扣 γ。τ 的引入也让 TDM 算法能够进行稀疏采样，每隔 K 个时间才对奖励 r 进行一次累积，从而加快训练速度。

$$\begin{aligned} a_t = & \operatorname*{argmax}_{a_{t:K:t+T}, s_{t:K:t+T}} \sum_{i=t,t+K,\ldots,t+T} r(s_i, a_i) \\ & \text{subject to } Q(s_i, a_i, s_{i+K}, K - 1) = 0, \forall i \in \{t, t + K, \ldots, t + T - K\} \end{aligned} \tag{11}$$

K 最终可以扩大到 T，也就是，不再考虑过程中其余时刻的奖励函数值，直接用 T 时刻的奖励函数值作为总价值。这在一些路径规划问题中是非常有用的（例如，奖励函数非常稀疏，只有在最终到达目的地时才出现一个大的奖励）。

注意到 (11) 是一个有限制的优化问题，为了避免解这样一个问题，TDM 的论文在实际操作中采用了一些构造 Q 函数形式的技巧，具体内容参加论文[38]，不在此详细讨论。不过，TDM 的一个限制是它只适用于结果导向的问题，例如一个任务的目标是到达一个地点，而收获的奖励和到达这个点过程中的路径细节无关。因为只有这样，才可以采用稀疏采样获取的奖励值作为总的奖励，来优化 a_t 的选择。

2.4 多任务分层强化学习

多任务和分层强化学习 Automated vehicle overtaking based on a multiple-goal reinforcement learning framework

2.5 强化学习和模型预测控制的对比

模型预测控制[39]（MPC, model-predictive control）是一种最优控制的算法，主要利用模型对确定性离散时间系统进行控制。MPC 和 RL 有许多相似之处。MPC 和 RL 的主要区别在于，RL 会 ‘‘学习’’ 一个模型或者不显式地利用模型，而 MPC 则利用模型进行预测，从而进行最优化[40]。

MPC 的方法如下：假设环境的模型为 $s_{t+1} = f(s_t, a_t)$，每次规划后面 n 步的行动序列 $a_t, a_{t+1}, \ldots, a_{t+n-1}$，选取最优策略序列的头一个行动 a_t^* 作为最优行动。

$$a_t^* = \operatorname*{argmax}_{a_t, s_{t+1}, a_{t+1}, s_{t+2}, \ldots, a_{t+n-1}, s_{t+n}} G_{t:t+n-1} \text{ subject to } s_{t+i+1} = f(s_{t+i}, a_{t+i}) \tag{12}$$

在 RL 的研究中，有不少利用/结合 MPC 的算法（在这里找一些!！我的论文中本来就有:TDM，pieter abbeel 也有一些无人机的论文!!）。

2.6 强化学习在自动驾驶领域的应用

目前对于自动驾驶在环岛环境下的路径规划这一特定问题，尚没有采用强化学习解决的方案。2018 年，Zyner 等人提出了利用 RNN 估计环岛车辆目的地的方法[41]。他们的工作采用了深度学习的算法，但是只是涉及预估，并没有涉及具体策略的制定。

2017 年，Li 等人提出了一种基于搜索树的自动驾驶决策算法[42]。他们的算法和强化学习类似，不过是通过依次设定初始状态，之后采用搜索 + 剪枝的方法来完成对 $Q(s,a)$ 的估计，并记录在表格里面。这种方法比较平凡，并且只能应对观测空间较小，并且动作空间离散的问题，因此实际应用能力比较有限。

2018 年，Ran 等人使用博弈论的方法对这个问题进行了探索[43]。他们采用的算法类似于强化学习，但也有所不同。在这篇文章中，决策过程利用了博弈论的方案。作者将车辆的驾驶人员分为保守和激进两种类型，在驾驶过程中，智能体自主车辆（ego vehicle）会不断依据对方司机的行为更新对对方车辆（opponent vehicle）分类的估计（通过神经网络完成）。完成这个估计以后，智能体会采取博弈的方法，每次采用某一行动，让对方采取它策略下的行动，由此预测对方在此后 n 步的行为。通过暴力搜索，可以得到最优的一个行动序列 $A_t, A_{t+1}, ..., A_{t+n-1}$。通过大量的训练，作者拟合了一个神经网络去存储状态 S_t 到最优动作 A_t 的映射，从而在实际操作中避免了搜索（离线训练）。

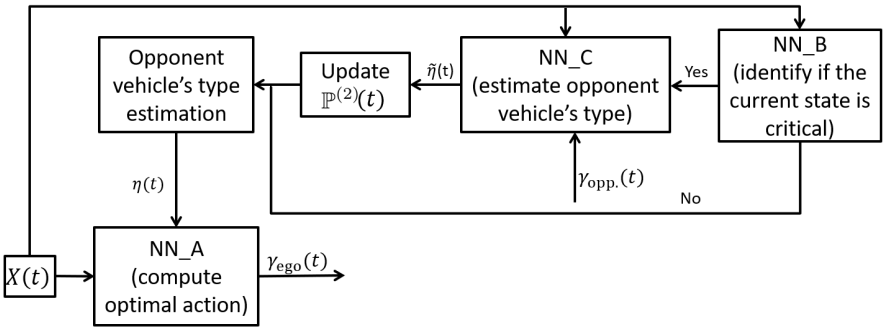


图 4　论文[43] 提出的基于博弈论的决策方案整体框架。

图片4展示了基于博弈论的算法的整体框架。其整体流程如下：首先，某一状态输入神经网络 B，判断其是否属于‘‘关键状态’’（义为，这个状态下两辆车

是否已经足够接近，开始相互影响），如果是，则进入下一步，通过神经网络 C 依据对方车辆的行为来判断对方车辆的类别是保守还是激进；判断结束后，将判断结果和当前状态一起输入神经网络 A，从而获取当前状态下应采取的最优行动。这其中的神经网络都是依据上面提到的博弈论算法来进行预训练的。

这个算法本质上是一个有监督学习算法，通过大量的训练 ‘‘记忆’’ 每个场景下应该采取的策略。和强化学习的区别在于，此基于博弈论的算法没有对状态和行动的价值进行估计，同时，由于没有获取环境的实际反馈（例如在仿真环境中，训练实际是否成功，并没有作为神经网络训练的输入），而是先验地认为通过对对方车辆类型的分类 + 预测可以获取最优解，因此并不够准确。这个方法的另一个缺陷在于对方的策略未必是简单的极限保守和极限激进两种，因此这个预测是不准确的。这也是我们在今后实验设计中需要考虑的问题：我们的对弈方驾驶车辆是怎样进行驾驶的？应该采取怎样的策略？

3 课题内容及开展情况

3.1 第一阶段

3.1.1 综述

目前已经完成如前所述的算法 I 的实现和实验。我们利用 openAI 的 baseline 算法库[44]，采用 DDPG[16] 算法，结合 LSTM[33] 搭建网络，在我们的实验平台（SUMO+Webots）上实现了基线方案。

3.1.2 建模方法

简要对我们的我们对观察空间进行这样的建模：我们关注离智能体车辆最近的两辆其他车辆，C_1 和 C_2。观察空间定义为

$$[d^{C_1}, v^{C_1}, cos\theta^{C_1}, sin\theta^{C_1}, d^{C_2}, v^{C_2}, cos\theta^{C_2}, sin\theta^{C_2}, x^{ego}, y^{ego}, v^{ego}, a^{ego}]^T$$

其中 d 表示距离，v 表示速度，a 表示加速度。

为了避免高维度的动作空间，车辆只决定每一时刻的加速和减速值，介于 $[-1, 1]$。车辆的方向由预先规定的路径点决定，由事先设计好的 PID 控制器直接进行调控，而不受强化学习算法的决策控制。

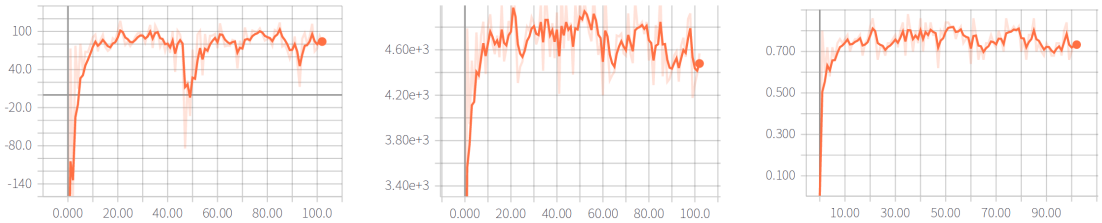
3.1.3 奖励函数设计

我们的奖励函数有如下机制：

- 0.95 的奖励，当智能体驶出环岛
- 0.2 的奖励，每次足够接近一个预先确定的路径点时
- $0.05(\dot{v} - 19)/30$，表示速度越快得到的奖励越多
- −0.4 的奖励（惩罚），当智能体偏离道路
- −100 的奖励，当智能体发生碰撞时

3.1.4 实验结果

实验结果如图5。我们可以看出，汽车在 2000 次训练后可以以 75% 以上的成功率通过环岛。在测试中，我们采用经过 500 次训练后得到的智能体，在双车道四进口的环岛进行五次测试，环岛车流量分别为 1 辆/5s 和 1 辆/20s。具体成功率统计数据呈现在表格2中。



(a) Average reward (b) Average speed (c) Success rate

图 5 DDPG 算法训练的实验结果。(a) 为平均获得的奖励值 (b) 为汽车在仿真环境中的平均速度 (c) 为平均成功率。每个数据点均为五十次训练的平均值，横坐标为百次训练，每次训练以发生事故或通过环岛为终止条件。

表 2 实验成功率统计（百分比）

平均车流量 n（1 辆/ns）	**1**	**2**	**3**	**4**	**5**	**平均成功率**
5	75.4	73.8	75.0	77.0	74.8	75.2 ± 1.0
20	94.0	94.4	94.0	94.6	97.0	94.8 ± 1.1

3.2 第二阶段

第二阶段我们将着重开展结合无模型和有模型强化学习的算法研究。目前已经进行了文献调研，具体工作将在后续完成。

4 进度计划

按时间划分，我们的进度计划如表3所示。

表 3　进度计划

周次	**任务安排**
寒假 +1	进一步的文献调研，熟悉现有系统架构 SUMO+Webots
2，3	在实验平台复现前人工作 (I2A[32])
4，5	阶段 II: 结合无模型和有模型的强化学习算法设计
6，7	实验验证 + 算法迭代
8	多任务强化学习框架和分层强化学习调研
9	中期报告
10，11	阶段 III: 多任务强化学习框架和分层强化学习算法设计
12，13	实验验证 + 算法迭代
14，15	论文撰写（会议论文 + 毕设论文）
16	结题 + 论文答辩

参考文献

[1] Waymo[EB/OL]. https://waymo.com.

[2] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518: 529 EP -.

[3] Kober J, Bagnell J A, Peters J. Reinforcement learning in robotics: A survey[J]. Int. J. Rob. Res., 2013, 32(11): 1238-1274.

[4] Polydoros A S, Nalpantidis L. Survey of model-based reinforcement learning: Applications on robotics[J]. Journal of Intelligent & Robotic Systems, 2017, 86(2): 153-173.

[5] Deep reinforcement learning framework for autonomous driving[J]. Electronic Imaging.

[6] Lopez P A, Behrisch M, Bieker-Walz L, et al. Microscopic traffic simulation using sumo[C]// The 21st IEEE International Conference on Intelligent Transportation Systems. [S.l.]: IEEE, 2018.

[7] Michel O. Webots: Symbiosis between virtual and real mobile robots[C]//VW '98: Proceedings of the First International Conference on Virtual Worlds. [S.l.: s.n.], 1998: 254-263.

[8] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. Second ed. [S.l.]: The MIT Press

[9] Sutton R S. Learning to predict by the methods of temporal differences[J]. Machine Learning, 1988, 3(1): 9-44.

[10] Bellman R. A markovian decision process[J]. Indiana Univ. Math. J., 1957, 6: 679-684.

[11] Bellman R E. Dynamic programming[M]. [S.l.]: Dover Publications, Inc., 2003

[12] Watkins C J C H, Dayan P. Q-learning[J]. Machine Learning, 1992, 8(3): 279-292.

[13] van Seijen H. Effective Multi-step Temporal-Difference Learning for Non-Linear Function Approximation[J]. arXiv e-prints, 2016: arXiv:1608.05151.

[14] Li Y. Deep Reinforcement Learning: An Overview[J]. arXiv e-prints, 2017: arXiv:1701.07274.

[15] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[M]// NIPS Deep Learning Workshop. [S.l.: s.n.], 2013

[16] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning [J]. arXiv e-prints, 2015: arXiv:1509.02971.

[17] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning [C]//Balcan M F, Weinberger K Q. Proceedings of Machine Learning Research: volume 48 Proceedings of The 33rd International Conference on Machine Learning. New York, New York, USA: PMLR, 2016: 1928-1937.

[18] Schulman J, Levine S, Moritz P, et al. Trust region policy optimization[C]//ICML. [S.l.: s.n.], 2015.

[19] Wu Y, Mansimov E, Liao S, et al. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation[J]. arXiv e-prints, 2017: arXiv:1708.05144.

[20] Wang Z, Bapst V, Heess N, et al. Sample efficient actor-critic with experience replay[C]// ICLR. [S.l.: s.n.], 2017.

[21] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[Z]. [S.l.: s.n.], 2017.

[22] Silver D, Hubert T, Schrittwieser J, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play: volume 362[Z]. American Association for the Advancement of Science, 2018: 1140-1144.

[23] Tieleman T, Hinton G. Lecture 6.5 rmsprop: Divide the gradient by a running average of its recent magnitude.[J]. COURSERA: Neural Networks for Machine Learning, 2012, 4.

[24] Qian N. On the momentum term in gradient descent learning algorithms[J]. Neural Netw., 1999, 12(1): 145-151.

[25] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv e-prints, 2016: arXiv:1609.04747.

[26] Krizhevsky A, Sutskever I, E. Hinton G. Imagenet classification with deep convolutional neural networks[J]. Neural Information Processing Systems, 2012, 25.

[27] Boyd S, Vandenberghe L. Convex optimization[M]. [S.l.]: Cambridge University Press, 2004

[28] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. [S.l.: s.n.], 2014.

[29] Degris T, White M, Sutton R S. Linear off-policy actor-critic[C]//ICML. [S.l.: s.n.], 2012.

[30] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.

[31] Oh J, Guo X, Lee H, et al. Action-conditional video prediction using deep networks in atari games[M]//Advances in Neural Information Processing Systems 28. [S.l.: s.n.]

[32] Racanière S, Weber T, Reichert D, et al. Imagination-augmented agents for deep reinforcement learning[M]//Advances in Neural Information Processing Systems 30. [S.l.: s.n.]

[33] Hausknecht M J, Stone P. Deep recurrent q-learning for partially observable mdps[C]//AAAI Fall Symposia. [S.l.: s.n.], 2015.

[34] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[35] Bansal S, Calandra R, Chua K, et al. Mbmf: Model-based priors for model-free reinforcement learning[Z]. [S.l.: s.n.], 2017.

[36] Deisenroth M P, Fox D, Rasmussen C E. Gaussian processes for data-efficient learning in robotics and control[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(2): 408-423.

[37] Snoek J, Larochelle H, Adams R P. Practical bayesian optimization of machine learning algorithms[M]//Advances in Neural Information Processing Systems 25. [S.l.: s.n.]

[38] Pong V, Gu S, Dalal M, et al. Temporal difference models: Model-free deep rl for model-based control[M]//ICLR. [S.l.: s.n.], 2018.

[39] Morari M, Lee J H. Model predictive control: past, present and future[J]. Computers & Chemical Engineering, 1999, 23(4): 667 - 682.

[40] Ernst D, Glavic M, Capitanescu F, et al. Reinforcement learning versus model predictive control: A comparison on a power system problem[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2009, 39(2): 517-529.

[41] Zyner A, Worrall S, Nebot E. A recurrent neural network solution for predicting driver intention at unsignalized intersections[J]. IEEE Robotics and Automation Letters, 2018, PP: 1-1.

[42] N L, H C, I K, et al. An explicit decision tree approach for automated driving[J]. ASME. Dynamic Systems and Control Conference, 2017, 1.

[43] Tian R, Li S, Li N, et al. Adaptive game-theoretic decision making for autonomous vehicle control at roundabouts[Z]. [S.l.: s.n.], 2018.

[44] Dhariwal P, Hesse C, Klimov O, et al. Openai baselines[J]. GitHub repository, 2017.

综合论文训练记录表

学生姓名		学号		班级	
论文题目					
主要内容以及进度安排	指导教师签字： 考核组组长签字： 年　　月　　日				
中期考核意见	考核组组长签字： 年　　月　　日				

指导教师评语	指导教师签字：_____ 年　　月　　日
评阅教师评语	评阅教师签字：_____ 年　　月　　日
答辩小组评语	答辩小组组长签字：_____ 年　　月　　日

总成绩：_____

教学负责人签字：_____

年　　月　　日