

**毕 业 设 计（论 文）**



**设计(论文)题目：** 面向无人驾驶的深度强化学习算法

应用与研究

学生姓名： 王家骅 指导教师： 龚如宾

二级学院： 计算机工程学院 专　　业： 计算机科学与技术

班　　级：16计算机科学与技术(J) 学　　号： 1613902007

提交日期： 2019年4月27日 答辩日期： 2019年5月16日

目录

[摘 要 4](#_Toc40459362)

[Abstract 5](#_Toc40459363)

[第1章 绪论 7](#_Toc40459364)

[1.1 研究背景 7](#_Toc40459365)

[1.1.1无人驾驶的发展历史 7](#_Toc40459366)

[1.1.2无人驾驶的分级 8](#_Toc40459367)

[1.2 国内外研究现状 9](#_Toc40459368)

[1.3 研究目的与内容 10](#_Toc40459369)

[1.3.1 研究目的 10](#_Toc40459370)

[1.3.2 研究内容 10](#_Toc40459371)

[1.4 章节内容安排 11](#_Toc40459372)

[第2章 理论研究 12](#_Toc40459373)

[2.1 深度学习 12](#_Toc40459374)

[2.1.1 多层感知机 12](#_Toc40459375)

[2.1.2 反向传播法 13](#_Toc40459376)

[2.1.3 神经网络中的优化器 14](#_Toc40459377)

[2.1.4 卷积神经网络 15](#_Toc40459378)

[2.1.5 池化层 17](#_Toc40459379)

[2.2 强化学习 18](#_Toc40459380)

[2.2.1 马尔科夫决策过程 18](#_Toc40459381)

[2.2.2 强化学习基本原理 19](#_Toc40459382)

[2.3 深度强化学习 21](#_Toc40459383)

[2.3.1 深度强化学习介绍 21](#_Toc40459384)

[2.3.2 深度强化学习的分类 21](#_Toc40459385)

[2.4 深度强化学习训练技巧 22](#_Toc40459386)

[2.4.1 经验回放（Experience Replay） 22](#_Toc40459387)

[2.4.2 Separate Target Network 22](#_Toc40459388)

[2.4.3 Multi-Agent 22](#_Toc40459389)

[2.5 DDPG算法 23](#_Toc40459390)

[2.6 A3C算法 25](#_Toc40459391)

[2.7 SAC算法 27](#_Toc40459392)

[2.8 奖励函数的设计 31](#_Toc40459393)

[2.9 本章小结 32](#_Toc40459394)

[第3章 无人驾驶系统开发技术与平台 33](#_Toc40459395)

[3.1 Tensorflow框架 33](#_Toc40459396)

[3.2 PyTorch框架 33](#_Toc40459397)

[3.3 Keras框架 33](#_Toc40459398)

[3.4 Gym强化学习环境 34](#_Toc40459399)

[3.5 无人驾驶模拟器 34](#_Toc40459400)

[3.5.1 CARLA 34](#_Toc40459401)

[3.5.2 TORCS 34](#_Toc40459402)

[3.5.3 DonkeyCar Simulator 35](#_Toc40459403)

[3.6本章小结 35](#_Toc40459404)

[第4章 需求分析与架构设计 36](#_Toc40459405)

[4.1需求分析 36](#_Toc40459406)

[4.2运行环境需求分析 36](#_Toc40459407)

[4.3稳定性及安全性需求分析 36](#_Toc40459408)

[4.4系统架构设计 37](#_Toc40459409)

[4.5系统功能设计 38](#_Toc40459410)

[4.6系统交互流程分析 39](#_Toc40459411)

[4.7本章小结 41](#_Toc40459412)

[第5章 详细设计 42](#_Toc40459413)

[5.1训练模块设计 42](#_Toc40459414)

[5.2验证模块设计 43](#_Toc40459415)

[5.3手动控制模块设计 44](#_Toc40459416)

[5.4环境感知模块设计 44](#_Toc40459417)

[5.5 本章小结 45](#_Toc40459418)

[第6章 系统实现 46](#_Toc40459419)

[6.1 Gym环境的实现 46](#_Toc40459420)

[6.2 A3C算法实现 47](#_Toc40459421)

[6.3 DDPG算法实现 49](#_Toc40459422)

[6.4 SAC算法实现 52](#_Toc40459423)

[6.5 奖励函数实现 53](#_Toc40459424)

[6.6 本章小结 54](#_Toc40459425)

[第7章 分析与测试 55](#_Toc40459426)

[7.1 各算法训练结果分析 55](#_Toc40459427)

[7.1.1 A3C性能分析 55](#_Toc40459428)

[7.1.2 DDPG性能分析 55](#_Toc40459429)

[7.1.3 SAC性能分析 56](#_Toc40459430)

[7.2 奖励函数对训练的影响 58](#_Toc40459431)

[7.3 真实环境下的驾驶测试 60](#_Toc40459432)

[7.4 本章小结 62](#_Toc40459433)

[第8章 总结与展望 63](#_Toc40459434)

[8.1 总结 63](#_Toc40459435)

[8.2 展望 63](#_Toc40459436)

[参考文献 65](#_Toc40459437)

[致 谢 68](#_Toc40459438)

[附 录 69](#_Toc40459439)

面向无人驾驶的深度强化学习算法应用与研究

# 摘 要

随着人工智能的高速发展，无人驾驶成为了现在产业界和学术界的热点话题，它能够对交通事故的减少产生积极作用，并且能够减少交通拥堵，实现更有效地城市规划。无人驾驶的实现能为人们的生活带来非常大的便利，因此一个可靠的无人驾驶系统的实现也就有了非常大的必要性。

跟随人工智能的浪潮，机器学习的发展也非常迅速，深度强化学习（Deep Reinforcement Learning，DRL）算法已经成为此领域当中最受关注的算法之一。DRL是将深度学习与强化学习结合起来的一种机器学习方法，能够解决许多真实环境中仅靠深度学习无法解决的问题，其在目前的许多无人驾驶研究中也被广泛使用。

本文主要以无人驾驶为目标，对几种深度强化学习算法进行研究。具体研究成果如下：

1. 对A3C（Asynchronous Advantage Actor-Critic）,DDPG（Deep Deterministic Policy Gradient），SAC（Soft Actor-Critic）三种算法在无人驾驶问题中的优缺点进行了分析，并且针对无人驾驶问题对算法进行修改和应用。将三种算法应用到模拟器上进行实验，分析不同参数下的表现与结果。重点研究了SAC算法中熵权重因子α和奖励函数的变化对于驾驶行为的影响。

2. 实现了无人驾驶系统，在对驾驶模拟器环境中的智能体进行训练和验证的基础上，探讨将算法模型移植到真实小车无人驾驶环境的方案，通过提供通用的gym应用接口，同时支持模拟环境和真实驾驶环境的算法研究和软件部署。

3. 提出了通过CGAN (Conditional Generative Adversarial Network)来将模拟器中得到的算法模型迁移到真实环境的方案，探讨了通过VAE (Variational Autoencoder)技术来对场景图像进行降维，以提升训练和验证效率的方案。

本文在算法设计和无人驾驶系统设计的基础上进行了研究与测试，收集了实验成果并进行了分析和改进，最终取得了良好的运行效果。

关键词：无人驾驶；深度强化学习；机器视觉；A2C；A3C；DDPG；SAC

**Application and Research of Deep Reinforcement Learning Algorithms for Self-Driving**

# Abstract

With the rapid development of artificial intelligence, self-driving has become a hot topic in industry and academia. It can have a positive effect on reducing traffic accidents and traffic congestion, and make the urban planning become more efficient. The implementation of self-driving can bring huge convenience to human life, so it’s very necessary to develop a reliable self-driving system.

In recent years, with the continuous development of machine learning, Deep Reinforcement Learning (DRL) algorithms have become one of the most concerned algorithms in this field. DRL combined deep learning and reinforcement learning, which can be used to solve the perceptual decision-making problems of complex systems. It is also widely used in many current self-driving researches.

This paper focuses on the research of several deep reinforcement learning algorithms for the purpose of self-driving. The main contributions of this paper are as follows:

1. We analyzed the advantages and disadvantages of three algorithms, A3C (Asynchronous Advantage Actor-Critic), DDPG (Deep Deterministic Policy Gradient)，and SAC (Soft Actor-Critic), and we modified them and applied them to the problem of self-driving. The three algorithms are applied to the driving simulator for experiments, and we analyzed the performance and results of every algorithm with different parameters. We focused on the influence of entropy weight factor α and reward function on driving behavior in SAC algorithm.

2. We implemented the self-driving system. Based on the training and verification of the agents in the driving simulator environment, we discussed the scheme of transplanting the algorithm models to the real car environment. We made the algorithm research and software deployment can be supported in both simulator environment and real environment by providing the general application interface of Gym.

3. We proposed a schema to transfer the algorithm models from the simulator to the real environment by using CGAN (Conditional Generative Adversarial Network). And we discussed the schema of reducing the dimension of scene image by using VAE (Variational Autoencoder), this technology can improve the efficiency of training and verification.

Based on the algorithm design and self-driving system design, relevant experiments are completed. We obtained the experimental results and analyzed them, .

**Key words:** Self-Driving; Deep Reinforcement Learning; Computer Vision; A2C; A3C; DDPG; SAC

# 第1章 绪论

## 1.1 研究背景

### 1.1.1无人驾驶的发展历史

无人驾驶[1-2]的发展在很久以前就已经开始，在上世纪20年代，以无线电设备为主要产品的Houdina Radio Control公司就已经发布过一辆名为American Wonder 的“无人驾驶”的汽车，但是这里所谓的“无人驾驶”仅仅停留在字面意义。因为这辆车其实只是通过无线电设备进行远程的人为控制以实现对车辆的无人驾驶。这辆American Wonder在展示后还在人来人往的纽约市中成功进行了驾驶。虽然它的原理本质上还是人为的控制，但它就是最初的无人驾驶了，由此也开启了无人驾驶在全世界范围内的研究。

19世纪30年代末，一名美国工业设计师提出了对未来交通的构思：未来的交通道路下可以铺设电子设备，车辆可以通过它们来进行导航与能量供给。1940年Bel Geddes在他自己的书籍中重新陈述了他的观点——人类应该从驾驶中被解放出来。他提出了在高速公路上驾驶时使用自动驾驶模式，而在离开高速公路后再更换成手动驾驶模式的想法。1953年时，美国的RCA公司将这个想法变为了现实，他们将导线铺设到路面下方，控制携带了通信设备的汽车并为其进行导航。到了60年代，无人驾驶的热潮不断扩散，吸引了很多企业和研究机构也陆续对其进行研究，但它们中有相当大的一部分还是通过事先铺设在道路下的电子设备来实现无人驾驶。这种方法在当时被多次实验并且证明了其效果，但这种方法的硬件需求太高，需要对道路进行很大的改造，而随着经济的发展，道路越来越多是必然趋势，如果使用这样的无人驾驶方案，修建和改造道路资金成本将会变得异常巨大，所以在这种方法的无人驾驶随着时代的推进也渐渐被淘汰。

70年代中期，针对无人驾驶的逻辑算法研究逐渐掀起了浪潮，而伊力诺依大学正是其中的主力军。随着无人驾驶逻辑算法的发展，世界上第一辆通过摄像头来捕捉视觉信息并进行决策的自动驾驶汽车于1977年在日本诞生。筑波工程实验室的研究员在这辆车上安装了两个摄像头，且成功使车辆能够在安全驾驶的前提下达到30km/h的时速，但是需要高架轨道来辅助。这也是无人驾驶领域的一次重大突破，首次证实了通过视觉信息来实现无人驾驶的可行性。

到了80年代初，这个时候基于视觉的无人驾驶也逐渐发展起来，以德国的Ernst Dickmanns为首的团队对基于视觉信息的驾驶决策进行研究并成功研制了一辆无人驾驶汽车，它能够以63km/h的速度安全地进行驾驶。随后EUREKA公布了普罗米修斯计划，投入7.5亿欧元用于无人驾驶汽车的研究，在庞大的资金资助下，多辆无人驾驶汽车被成功研发。同一时期的美国也不甘在无人驾驶领域落后，DARPA公布了ALV计划，并与斯坦福等知名机构进行了合作，他们首次将便携式计算设备带上了无人驾驶汽车，并且利用计算机视觉、激光雷达等技术成功实现了通过机器人对车辆进行自动驾驶的决策，并且这辆汽车能够以31km/h的速度安全地进行驾驶。

而正是在这个时候，中国也开始在无人驾驶领域与其他国家进行角逐，哈尔滨工业大学、沈阳自动化研究所和国防科技大学作为国内无人驾驶的先驱，一同参与了“遥控驾驶的防核化侦察车”这个项目。后来，著名的ATB-1无人车由国内多家单位合作研发，这是中国第一辆能够在真正意义上进行自动驾驶的测试样车，它的能够以21km/h的速度进行安全的自动驾驶。而这辆ATB-1的成功研发也是中国正式踏入无人驾驶领域的证明。

到了现在，无人驾驶技术日趋成熟，但它依然是学术界的研究热点。而且无人驾驶的实现不仅能够对城市交通起到积极作用，在我国经济方面也会产生很大的影响。根据相关数据的统计，我国每年需要支出全国总GDP的2%来为路拥堵买单，而在2018年，交通拥堵问题更是为我国带来了占到城市人口可支配收入的20%的巨大损失。而随着国民车辆拥有率的提高，这样的损失还会持续地增加，而无人驾驶正是能够为国家解决这个问题的一个重要技术。

### 1.1.2无人驾驶的分级

随着无人驾驶技术的日趋成熟，必然需要一些规范与标准来供大家参考，目前在世界范围也已经有了几种无人驾驶分级标准，但相互之间也有着一定的差异，美国高速公路安全管理局（NHTSA）将整个无人驾驶目标分为五个层次，而美国机动车工程师协会（SAE）则将其分成了L0到L5共六个层次，主要区别在于SAE的标准认为NHTSA中最高级的L4级别的目标应该被细化为L4与L5两个级别。SAE的无人驾驶标准具体如下：

L0（有人驾驶）：完全由人类来进行驾驶。

L1（驾驶者辅助）：这个级别的汽车能对自身进行一些辅助性的功能，比如危险警示、偏离车道提示以及对车体自身的稳定控制系统。现在的许多车辆都具备了这样的一些功能。

L2（部分自动化）：达到这个级别的汽车至少需要让两个或以上的主要驾驶功能实现完全的自动化。其实这个级别的车辆也已经慢慢地融入了我们的生活，比如一些汽车公司会为旗下的一些高端车辆提供主动的自动导航和自动车道保持等功能。

L3（条件自动化）：在L2级别的基础之上，这个级别车辆可以在某些限定的情况下无需驾驶员的操作就能实现自动的转向、加减速控制，但是驾驶员也可以根据实际情况随时取得汽车的完全控制权并对汽车进行控制。L3级的自动驾驶汽车还不具备完全进行自动驾驶的能力，所以对于这个级别的车辆来说驾驶员还是必要的。

L4（高度自动化）：这个级别车辆已经达到了普遍意义上的无人驾驶，完全不需要驾驶员的辅助。L4级的无人驾驶汽车能够监视道路的具体状况且安全可靠的进行全程自动驾驶，但可能无法适应所有的驾驶场景。

L5（完全自动化）：L5级别的无人驾驶汽车能够在真正意义上达到人类驾驶的水平，拥有完全的自主驾驶系统。且在任何复杂场景、难以驾驶的地区都能够进行自动驾驶，对于可能发生的情况也都具备应对能力。

## 1.2 国内外研究现状

目前，全世界范围内已有许多公司和机构在对无人驾驶进行研究，谷歌作为最先发展无人驾驶技术的公司，他们在无人驾驶领域已经做过了许多研究，对无人驾驶技术的发展做出了巨大的贡献。谷歌Waymo在独立一年之后进行了一次真正意义上的无人驾驶测试，即不配备人类驾驶员的测试。当然，除了上述的一些传统的汽车公司和研究机构之外，苹果、Uber等公司也已经开始将业务范围向无人驾驶汽车方面拓展，并取得了一些成果。2017年，NVIDIA发布了PilotNet并且用它成功实现了无人驾驶，让通过卷积神经网络[3]来实现无人驾驶成为了可能，为整个无人驾驶领域的发展取得了重大突破。这个项目的构思是利用摄像头来获取当前道路的图片，通过卷积神经网络来提取道路的特征，再对特征进行计算来得到合适的驾驶决策。因此只需要在人类正常驾驶时记录当前环境和控制数据，然后利用这些数据来训练PilotNet，系统就能逐渐掌握驾驶的方法，在进行了正确的训练后甚至能够在一些较为苛刻的环境下进行驾驶，如没有车道标志线的道路、山路等。

虽然起步比较晚，但国内的无人驾驶领域也已经有了一些成果，诸如百度、长安等企业和一些研究机构都在无人驾驶领域崭露头角，长安和百度的无人驾驶汽车也都已经进行了真实环境下的驾驶测试。

但是，目前的无人驾驶技术并没有完全成熟，许多方面还有待优化，例如提高自动化驾驶程度，提物体高识别精度，传感器失效时的出错处理等。2016年就曾经发生过一场悲剧，一位车主在驾驶中开启了他的特斯拉Model S的自动驾驶辅助功能，并因此发生车祸且在这场车祸中丧命。驾驶的安全性不足是会为乘客带来生命危险的，解决这些问题不仅是当前无人驾驶技术的重要需求，对于整个无人驾驶领域而言也有着十分重大的意义。由此，面向无人驾驶的深度强化学习[4-7]算法的研究就有很大的必要性。

## 1.3 研究目的与内容

### 1.3.1 研究目的

本项目旨在通过在无人驾驶模拟器上实现并运行多种深度强化学习算法，最终得到一个能够平稳驾驶的可靠模型，本项目的主要目标有以下几点：

1. 为了实现无人驾驶的目标，本项目会在模拟器环境中使用基于视觉的DRL算法来达到完全自动控制的驾驶能力，也就是L4级别的无人驾驶。
2. 对深度强化学习算法进行研究时应当保证严谨性与全面性，因此本项目将使用多种DRL算法并对它们逐个进行测试。
3. 通过训练过程中得到数据和最后的测试结果，深入分析各算法的特点，整理分析结果。
4. 选取合适的DRL算法并应用到树莓派小车上观察运行状态并记录结果。
5. 根据在模拟器和树莓派小车上得到的分析结果，对已有的算法模型进行合理的改进，并使小车拥有更好的驾驶能力。

### 1.3.2 研究内容

本文的研究内容主要是针对无人驾驶这个目标对几种DRL算法进行研究。通过对比多种深度强化学习模型，选出几种可行性较大的方法并加以实现和调整，然后在模拟器上进行训练，对比结果，分析各种模型在不同情况下的优劣。最后训练出一个能够平滑地在模拟赛道中驾驶，且具有一定避障能力的模型，然后将其投入到无人小车上实现真实环境中的驾驶。

主要研究内容如下：

1. 根据模拟器的要求搭建相应的环境并安装驾驶模拟器，在Python中安装gym和所需的机器学习开发包。
2. 调用驾驶模拟器开放的API，并通过API构造Gym环境，使DRL算法能够按照一般Gym环境的流程进行应用。
3. 选取几种不同的DRL算法并对其进行搭建、训练及测试，收集实验数据，并且根据实际需求和遇到的问题进行修改和调整。
4. 比较几种模型在模拟器上的表现，分析其原因并加以改进。
5. 安装无人小车并将算法模型投入使用，观察实际表现并进行进一步改进和调优。

## 1.4 章节内容安排

本论文共有八个章节，主要内容如下：

第1章 绪论。对无人驾驶的研究背景、研究意义进行介绍，阐述无人驾驶的发展史和现在的国内外研究现状，介绍本课题的主要目标与内容。

第2章 理论研究。首先需要介绍深度学习和深度强化学习的基本理论知识，然后对一些深度强化学习在训练过程中能够使用的技巧进行介绍，最后对本课题主要使用的DDPG、A3C、SAC三种算法的原理进行详细的论述。

第3章 无人驾驶系统开发技术与平台。对当今流行的几种机器学习框架如Tensorflow、PyTorch等进行介绍，对强化学习工具包Gym进行介绍，最后介绍目前在无人驾驶中比较常用的几种驾驶模拟器。

第4章 需求分析与架构设计。针对无人驾驶的应用场景，针对无人驾驶系统的功能和性能进行需求分析，并对无人驾驶系统进行总体架构的设计，设计各个功能模块，分析用户与系统的交互流程。

第5章 详细设计。参考第四章中给出的需求和架构设计方案，对整个系统的各个部分与功能进行详细设计。

第6章 系统实现。将选取的几种DRL算法在无人驾驶系统中进行具体的实现，给出各算法的关键代码。

第7章 分析与测试。针对各个DRL算法指定其测试方案，对各个算法的测试结果进行分析。将算法应用于无人小车并对其运行结果进行分析。

第8章 总结与展望。对研究内容进行概括性的总结，对无人驾驶系统进行客观评价，说明其优缺点，并讨论能够对其进行改进的方向。

# 第2章 理论研究

本章主要对深度学习和深度强化学习的一些基本理论进行介绍，给出一些深度强化学习在训练过程中能够使用的技巧，以及对本课题主要使用的DDPG、A3C、SAC三种算法的基本理论进行论述。

## 2.1 深度学习

本节阐述深度学习中一些重要的基本概念和技术，如多层感知机、反向传播法、优化器、卷积神经网络等。

### 2.1.1 多层感知机

多层感知机(Multilayer Perceptron)由多个网络层组成，而每个网络层由许多带有权重和偏置的神经元构成，其结构如图1所示。

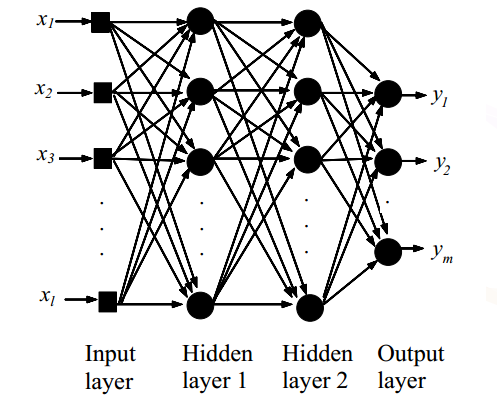


图2.1 多层感知机结构

多层感知机是一个用于拟合非线性函数的模型，它能够将输入数据和输出数据的函数关系用一个非线性映射表示出来。多层感知机中的每一层都有很多个神经元，前后两层中的每个神经元都依靠权重和偏置互相连接，这样的一个连接可以看作是一次简单的非线性传递，由于多层感知机中有大量的神经元，也就叠加了大量的简单非线性传递，最终整个多层感知机能够拟合非常复杂的函数。

在每一个神经元中，输入值都乘上了一个权重并累加一个偏置，然后作为下一层网络的输入参数。也就是说多层感知机中的数据流向总是向前传递的，这也是多层感知机也被称为前馈神经网络的原因。多层感知机的输入层不设置权重和偏置，直接把输入的数值传递给后面的隐藏层网络，经过中间的隐藏层网络的计算和处理后传递到输出层。

只要多层感知机中的每个神经元的权重和偏置以及激活函数设置得足够合理，它就能拟合任何复杂的函数。但是人为地去设置这么多参数并不现实，因此需要通过大量的输入输出数据来对多层感知机进行训练，进行一次需要一组训练数据，也就是一个输入向量X和它的正确输出向量Y，即标签。训练时，只要提供大量的训练数据，不断地让感知机计算预测值与真实值的误差并对参数进行优化，直到多层感知机的输出数值趋于正确。而这个优化参数的过程则需要用到基于链式求导法则的反向传播法（Back Propagation），要描述实际输出和真实输出之间的差异则可以根据应用场景用不同的损失函数（Loss Function）来表达。

而在后续的研究中，随着深度学习的出现，多层感知机中的感知机层也被人们称为全连接层（Fully Connected Layer），其意义为：每层中的每一个节点与前一层的所有节点都通过权重和偏置互相连接的网络层。

### 2.1.2 反向传播法

反向传播算法（Back Propagation，BP）是现在的神经网络中最广为使用的一种算法，它通过两个过程反复循环迭代(激励传播和权重更新)，直到网络能够在得到输入向量后计算得出接近实际标签的输出向量，这两个过程也被称为正向传播和反向传播。在正向传播阶段中，数据从输入方向流向输出方向，输入的向量从输入层流向隐藏层，经过隐藏层的计算后传递到输出层，最后用损失函数来计算预测值与实际值的差异，即loss值。接下来就到了反向传播阶段，在这个阶段中，数据从输出方向流向输入方向，途中计算目标函数相对于网络中的每个神经元权值的偏导数，得到所有权值的梯度，然后以一定的比例对权值进行修改，当误差达到可以接受的范围时学习结束。

每次迭代中的激励传播包含两步：

1. 将训练输入到网络中，然后按照往前的方向进行计算和处理，最后在输出层得到结果。(前向传播)
2. 用输出值与训练输入对应的标签以一定的损失函数来计算loss值，按照往后的方向进行逐层计算，获得每个权值误差。(反向传播)

对于神经网络中每个神经元的权重，则按照以下步骤来进行更新：

1. 计算输入值与误差的乘积，得到每个权重的梯度值。
2. 把梯度值乘以一个给定的比例值，加上负号后加到原本的权重上。

在这个过程中，梯度需要乘的比例值会对训练过程的速度与效果造成影响，因此我们也可以称它为“学习率”（Learning Rate）。梯度的方向指明了目标函数高点的方向，因此在更新神经网络中的权重时应该对其进行取反，来引导权重向误差更小的方向更新。

### 2.1.3 神经网络中的优化器

神经网络的训练需要使用一个优化器（优化方法）来让网络中的权重以合适的学习率来更新，下面将介绍几种比较重要的优化器。

1. **SGD**（Stochastic Gradient Descent）是一种非常简单的优化方法，其思想就是每次随机地选取一个样本计算神经网络的误差，然后沿着梯度的方向前进一定距离。用数学公式来描述可以表示为下式：

(2.1)

上式中，为神经网络的权值，而在第i笔数据时损失函数对于当前的梯度则用来表示，表示学习率。

1. **Momentum**是在SGD基础上进行了改进的优化器，它加入了动量，通过引入可以加速 SGD。其数学公式可表示为如下式子：

(2.2)

(2.3)

式中可以理解为动量，表示之前累积的更新方向，是一个衰弱因子，表示前一次的梯度方向对本次更新起到多大程度的影响，可以根据场景进行调整，一般取0.9左右。在式中加入动量之后，与先前梯度方向不变的维度上的更新速度会变得更快，而与先前梯度方向不同的维度上的更新速度则会变慢，这样对于整个训练过程就能够起到加快收敛速度和减小震荡的作用。

1. **AdaGrad**（Adaptive Gradient）同样是一种考虑了先前梯度的优化方法，但与Momentum不同，它能够根据先前的梯度来对每个参数的学习率进行调整，更新稀疏特征时学习率会更高，而更新非稀疏特征时学习率就会变小，所以Adagrad在特征数据比较稀疏的情况下可以具有良好的效果。其数学公式可表示为如下式子：

(2.4)

(2.5)

(2.6)

上式中是一个极小值，以避免分母为0，其值一般可以被设定为。AdaGrad的优点是可以动态地对学习率进行调整，缺点则是它会记录过去所有的梯度平方和，以至于分母越来越大，更新速度会越来越缓慢，经过一定步数后可能会变得难以优化。另外有一种优化方法Adadelta对AdaGrad进行了改进，将分母部分的历史梯度求和根改为历史梯度的均方根，可以避免因为分母膨胀导致的学习率过低。

1. **RMSprop**（Root Mean Square Prop）。为了进一步优化损失函数在更新中的震荡问题，并进一步加快函数的收敛速度，RMSProp算法对先前的历史梯度进行微分平方再取其加权均值。 其数学公式可以表示为如下式子：

(2.7)

(2.8)

(2.9)

1. **Adam**[8]（Adaptive Moment Estimation）也是一种可以为神经网络中的每个参数给出合适学习率的优化方法，它把先前的历史梯度的平方保存下来并计算其指数衰减平均值，这一点与Adadelta和 RMSprop等方法相同，另外它还会把先前历史梯度也保存下来并计算其指数衰减平均值，这一点比较类似于Momentum，和的公式如下：

(2.10)

(2.11)

但以上公式还存在一些问题，当和被初始化为0向量时，它们就会很容易向0偏置，尤其是当衰减因子，接近1时这个问题会更加严重。但如果对与进行偏差修正（bias-corrected）的话就能够避免这个问题：

(2.12)

(2.13)

最终，Adam的梯度更新公式为：

(2.14)

Adam由于融合了多种优化方法的特性，在实际使用中的表现很好，在目前的许多研究和实际应用中Adam都是一种很常用优化方法。

### 2.1.4 卷积神经网络

在机器视觉的研究中，卷积神经网络从发展初期开始就备受青睐，它能够有效地提取图片中的特征信息。卷积运算可以简单地理解为在输入矩阵中寻找代表某个特征的子矩阵并构造成新的输出矩阵作为下一个网络层的输入参数，如图2.2所示，第一个矩阵是一个输入矩阵，这里称其为M，第二个矩阵为过滤器F（也叫卷积核），第三个矩阵为F，它是输入矩阵与卷积核进行了一次点积运算后得到的矩阵。卷积核F以一个固定的定步（stride）长在输入矩阵M上逐次进行运算，全部运算完成后得到最右侧的输出矩阵O。而这样的一个流程就是卷积神经网络中最核心的部分，在实际使用卷积神经网络时，我们还可以添加一些额外的参数。

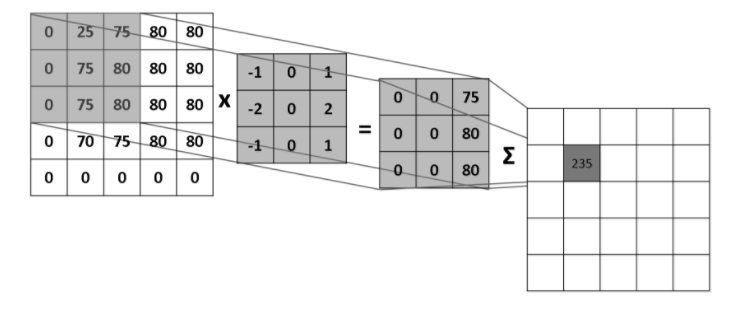


图2.2 卷积操作过程

当遇到图片数据时，我们可以认为是在对多个色彩通道的相同大小的矩阵进行处理，假设一张图片的宽为w，它的高为h，它的深度为d，那么它可以被表示为h\*w\*d。这里的深度表示的就是图像的有多少色彩通道，比如RGB图像的深度为3，每一层分别代表红、绿、蓝三色的颜色值。在计算时，一张h\*w\*d的图片可以看作是d个h\*w的矩阵，例如6x16x3的图片就可以被视为三个6x16的矩阵。

进行卷积运算时会给定一个大小为F\*F的矩阵，称为卷积核，又叫做过滤器，卷积核的大小被称为感受野。卷积核的深度d和输入数据的深度需要相同，因此卷积核的形状就是F\*F\*d，可以把它看作是d个F\*F的矩阵。在实际的卷积神经网络中，根据应用场景的不同可以设置不同数量的过滤器，其数量记作K，其中每一个过滤器都包含d个F\*F的矩阵，并且通过与输入矩阵进行卷积来得到一个输出矩阵。

大小确定的输入矩阵和大小确定的卷积核，再加上一些额外参数，能够生成大小确定的输出矩阵，接下来介绍这些参数。

1. Padding。当神经网络使用卷积运算时，输入矩阵的边缘像素参与的计算次数会比非边缘部分的像素少，所以我们可以在输入矩阵的周围补零，这样的操作称为padding，在边缘补零的数量称为P。比如当边缘补零数量P为1时，原本的5\*5输入矩阵会在处理时转换成图2.3的样子，方框里面的部分为原来的输入矩阵，其周围用0包围。

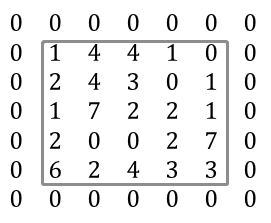


图2.3 经过Padding的矩阵

1. Stride。卷积核在输入矩阵上进行点积运算时移动的步长被称为stride，记为S。当S=2时，filter每次移动2个单元。如图2.4所示，两个方框是在S=2时的第一步操作位置和第二步操作位置：

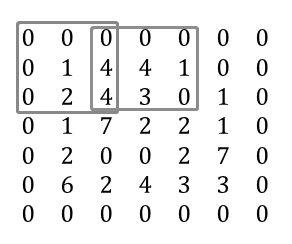


图2.4 stride为2时的卷积运算过程

有了以上两个参数P和S，再加上参数W（输入矩阵的大小），过滤器的大小F，输出矩阵的大小为：

*O= (* (2.15)

在对卷积神经网络进行结构设计时，应该先确定每一层输入矩阵的大小、输出矩阵的大小和它们的深度，这样才能在后续的层中进行确定性的结构设计，也能够得出一个确定的输出矩阵。此外，在实际使用卷积神经网络时，使用多个连续的卷积层，然后适当地在每个卷积层后加入池化层也是很常用的一个技巧，这样能够提取越来越高维的特征。

### 2.1.5 池化层

除了卷积层之外，池化层（Pooling Layer）也是卷积神经网络中一种非常重要的网络层。在大多数情况下我们可以把池化层加在一个卷积层的后面，这样可以使卷积层提取出的特征的维度变得更低，并且可以减少神经网络参数的总量，不仅避免了神经网络的过拟合情况，还可以对整个模型的容错能力进行提升。图片数据是静态的，也就是说在图像的某个位置中有用的特征在其他位置也可能会适用。因此，为了对一张图片的特征进行更高维的概括，Pooling的核心思想就是对不同区域的特征进行聚合统计，比如对一个区域的特征取均值或最大值来压缩这个区域的特征。

在所有的池化层中，Max Pooling（最大值池化）和Mean Pooling（均值池化）是最常用也最简单有效的两种方式。Max Pooling就是对一个特征区域中的所有特征值取最大值，具体如图2.5所示。

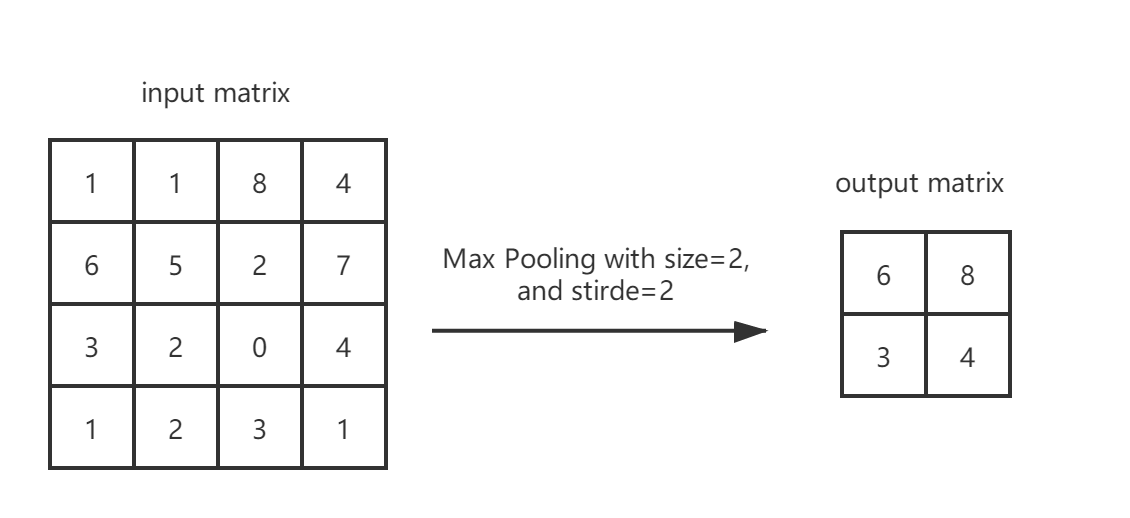


图2.5 Max Pooling运算过程

Pooling层的过滤器与卷积层过滤器一样，也可以设置size和stride，图2.5中就是以size为2，stride为2的情况下进行计算的。而Mean Pooling也与之相似，只要把求最大值改为求均值即可。

## 2.2 强化学习

本节主要阐述深度强化学习中的一些重要概念，如马尔科夫决策过程[9]（MDP）和深度学习强化学习的分类。

### 2.2.1 马尔科夫决策过程

1. **马尔科夫性**

马尔科夫性就是指一个系统接下来的一个状态只会与现在的状态有关，并且它和当前状态之前的所有状态都无关。

定义：如果是具有马尔科夫性的一个状态，那么这个状态必须满足P[|]=P[|……]，其意义为：本身包含了之前所有状态的信息，它与到之间所有的状态的集合是等价的。

1. **马尔科夫决策过程的定义**

MDP的核心部分就是一个像<S,A,T,R>这样的四元组。其中S是包含所有状态的有限集合，A是包含所有动作的有限集合，T定义为S×A×S→[0,1]，它被用来描述在某个特定状态、特定动作下，某个特定状态的转移概率，R代表奖励函数，根据环境的不同它可以有多中不同的定义方式，如R:S→R，R:S×A→R和R:S×A×S→R。

马尔科夫决策过程的核心思想就是首先环境要具有马尔科夫性，也就是包含了足够的信息，之前的状态都不必要，并且即使是通过不同的状态转移到当前状态，只要到达了这个状态，那么这个状态下的动作概率分布就应该是相同的。在这样一个符合MDP过程的前提下，我们就可以通过不断的探索来为每个找到正确的动作概率分布。

<S,A,T,R>四元组是MDP中最为重要的部分，下面来为它们分别进行介绍：

* 状态S，有限集合{,,...,}。对于建模的问题来说，状态是所有信息中唯一的特征。
* 动作A，有限集合{,,...,}。对于一个特定状态s∈S而言，它的可用动作集合可以表示为A(s)，其中A(s)⊆A。先决条件函数(precondition funcion):S×A→{true,false}的作用是描述动作a∈A是否能够在状态s∈S中使用。
* 转换函数T可以被定义为：S×A×S→[0,1]，它被用于描述(s,a)到下一个s′的概率，其概率表示为T(s,a,s′)，其值需要满足0≤T(s,a,s′)≤1且T(s,a,s′)=1，也就是概率必须要满足实际，否则无意义。
* 奖励函数R可以被定义为三种形式：R:S→R，R:S×A→R或R:S×A×S→R，分别对应了三种不同环境的奖励计算方式：根据特定状态给出固定奖励值、在特定状态作出特定动作时给出固定奖励，以及在执行某个状态转换时给出固定奖励。最后一种定义方式可以非常方便的应用于Model Free的算法，因此是广泛使用的定义方式。

奖励函数可以说是MDP中最重要的一个部分，因为奖励其实在无形中定义了这个系统的学习目标。在许多情况下，奖励函数也会给非目标的中间状态分配一定的奖励值，这可以理解为是为了学习全局策略而为每个状态设置相应的奖励或惩罚，使整个学习任务被分为许多子任务。

### 2.2.2 强化学习基本原理

强化学习[10]的基本流程如下图：

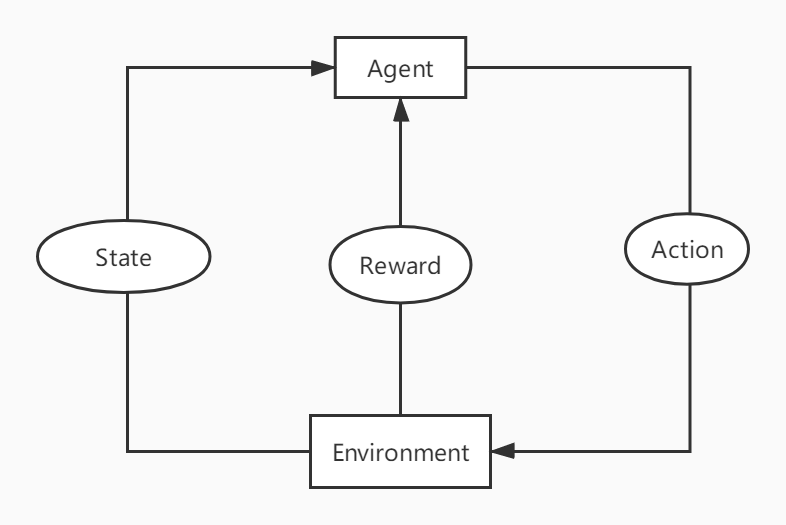


图2.6强化学习中Agent与环境的交互示意图

如果Agent按照某个策略来行动后导致环境得到了积极反馈(正的奖励值)，那么Agent在后续的行为中就会更加趋向于使用这个策略。Agent的目标是在每个离散状态发现对于当前状态而言最优的策略来使期望的折扣奖励和最大。

强化学习其实是一个Agent在环境中不断试错的过程，Agent根据自己的策略选择一个动作去与环境交互，环境根据输入的动作进行相应的变化并返回下一个状态、奖励值以及一些额外信息，Agent通过得到的信息来评价自己行为的好坏并对自己进行优化，然后再根据策略选择可能赢得最大奖励的动作去与环境继续交互。在这个过程中，每次执行的动作不仅会影响着紧接着的下一个状态与奖励值，还会对后续的状态和奖励产生影响，甚至对于最终的合计奖励值也可能会产生决定性的影响。

监督学习与强化学习具有很大的不同，它们的差别主要体现在奖励值，虽然其作用与监督学习中的标签有一定的相似程度，但是环境向Agent反馈的奖励值是在对策略产生的动作的优劣进行评价，而监督学习则是直接提供一个正确标签给Agent。因为环境提供的信息仅仅只有当前动作（可能也包含了先前动作）的好坏，所以强化学习中的Agent需要与环境进行不断的交互，把交互过程中得到的数据保存下来并对其进行学习。通过这种方式，Agent能够从环境的反馈中逐渐掌握一些规律并改进自己的策略，以在环境中获取更高的奖励。

## 2.3 深度强化学习

本节对深度强化学习进行介绍，并对深度强化学习的分类进行阐述。

### 2.3.1 深度强化学习介绍

传统的强化学习方法的动作空间和状态空间会受内存大小的限制，且难以支持连续的状态和动作空间。但是在实际情况或者比较复杂的模拟环境中，往往需要使用连续的状态空间和动作空间。而且当输入信息为视频、音频等信息时，不仅空间为连续，输入数据的维度也会变得很大，传统的强化学习很难处理。而深度强化学习就是把深度学习应用到强化学习中，让强化学习能够对连续的空间进行处理。

### 2.3.2 深度强化学习的分类

首先根据是否在系统中根据问题来构造模型，强化学习有Model-Free和Model-Based[12]两种。其区别就是Model-Based方法需要计算环境的模型，而Model-Free方法则不需要。Model-Based的优势是agent可以根据模型观察到更明确的规律或规则，更有针对性地进行学习；而它的劣势则是真实的环境与计算的环境模型之间的误差，这样的误差可能会成为Model-Based算法在真实环境中学习的瓶颈。Model-Free方法不需要针对环境去构造模型，所以比较容易实现，目前的许多研究也是会更偏向于采用Model-Free方法。但是在模型的规则比较明确且比较易于实现时，Model-Based算法也能有非常好的效果，著名的AlphaGo Zero[13-16]就是一个Model-Based的深度强化学习算法。

根据算法本身是否基于策略，深度强化学习能被分成三类：

1. Value-Based方法，如：Q-Learning[17]，Deep Q-Learning(DQN)[18]。
2. Policy-based方法，如：Policy Gradient[19]，Deterministic Policy Gradient (DPG)等。
3. 结合了Value-Based和Policy-Based二者优点的Actor-Critic方法，如：Proximal Policy Optimization[20](PPO)，Deep Deterministic Policy Gradient (DDPG)[21]，Asynchronous Advantage Actor-Critic (A3C)[22]，Soft Actor-Critic (SAC)[23,24]。

另外，根据行为策略（Behavior Policy）和目标策略（Target Policy）是否是同一个策略，深度强化学习还能被分为On-Policy和Off-Policy。

## 2.4 深度强化学习训练技巧

本节主要阐述深度强化学习训练时使用到的一些技巧，如经验回放、Target Network、Multi-Agent等方法。

### 2.4.1 经验回放（Experience Replay）

许多强化学习算法中都会使用到深度神经网络，实际上深度强化学习之深度，也是源自于此。而深度神经网络本质上是一种监督学习，数据的独立同分布对它来说非常重要，但是在许多的环境中，状态与状态之间是具有很强的关联性的。为了把数据之间的关联性降低，我们可以使用Experience Replay方法，通过将经验数据存储到经验池中再随机取出的方式来打破这种关联性。

具体实现方式就是定义一个确定最大长度的队列，在每一个step中存储当前时刻的s、a、r和s′（针对不同的环境也可以存储当前的其他信息）。然后在每次需要对agent进行训练时，从队列中取出一组或多组数据进行训练，这样就打破了环境信息的前后关联性，增大参数更新时的方差（variance）。

此外，Experience Replay还有一个被优化的版本——Prioritized experience replay[11](PER)，具体则是对经验回放池中的数据进行优先级排序，让更有用的经验被反复学习。

### 2.4.2 Separate Target Network

Separate Target Network最早是在DQN中使用的，因为当我们使用一个非线性函数去拟合Q值函数的时候，损失函数产生的梯度容易发生震荡，呈现出不稳定的学习行为。为此引入目标网络。也可以简单地理解为是DQN把预测q值当做回归问题。而回归问题需要监督信号，如果用一个网络同时承担预测和监督的任务，会造成moving target，造成难以逼近target。而Target Network能够在训练过程中暂时固定Q值函数，从而使学习过程更加稳定。

### 2.4.3 Multi-Agent

Multi-Agent，顾名思义，其实就是在训练时采用多个agent，每个agent都与一个独立的环境交互。其中有一个是Main-Agent，其它的Worker-Agent每隔一段时间需要从Main-Agent那里复制神经网络中的weight，然后各自在自己的环境中学习，在经过一定的时间后将自己的weight按一个比例更新到Main-Agent上。

Multi-Agent这样的方法不仅能够与Experience Replay一样降低数据相关性对训练神经网络带来的影响，同时因为多个Agent同时运作，能够使Agent以成倍的速度进行学习。

## 2.5 DDPG算法

Deep Deterministic Policy Gradient（DDPG）是一种确定性的策略梯度强化学习方法。确定性策略是它的一大特点，它与随机策略（Stochastic Policy）的区别就在于随机策略需要去学习每个状态下所有可执行动作的概率，而确定性策略则是直接给出预期回报最高，也就是概率最高的那个动作。在离散动作空间的维度特别高的情况，或者是连续动作空间的情况下，确定性策略能够比随机策略更快地收敛，所需的样本也会更少，因为它只需要找出每个状态下最优的那个动作就可以了。

DDPG的主体部分还是一对Actor-Critic网络，其中Actor是一个确定性的策略网络，为了避免DQN中难以收敛的问题，我们需要为Actor和Critic分别加上一个target网络，所以它一共有两个Actor网络和两个Critic网络。

DDPG不需要像DDQN那样通过选取最大Q值的动作来作出决策，取而代之的是用Actor现实网络来作出确定性的决策，而Critic现实网络的作用就相当于DDQN中的Q现实网络，用于计算并更新Q值。但在DDPG中Critic的作用并不是作出决策，而是用它的目标网络去给Actor作更新的参考标准，同样Critic的更新也需要Actor目标网络的辅助来得到参考标准。

DDPG四个网络的负责的具体任务如下：

1.  Actor当前网络：负责策略网络参数θ的迭代更新，根据当前状态S来进行决策并得到动作A，用来与环境交互并生成下一个状态S′和奖励R。

2.  Actor目标网络：根据经验回放池中采样的经验数据中的下一状态S′来进行决策并得到动作A′，它的网络参数θ′需要定期地去复制Actor当前网络的参数θ。

3.  Critic当前网络：负责价值网络参数w的迭代更新，先计算当前Q值再计算与目标Q值=R+γQ′(S′,A′,w′)的误差。

4.  Critic目标网络：计算目标Q值中下一个状态的目标Q值，即Q′(S′,A′,w′)的部分。它的参数w′需要定期地去复制Critic当前网络的参数w。

另外，在DDPG中，两个目标网络的更新会采用soft-update的更新方式而不是DQN中的hard-update，即每次都按照比例进行小幅度更新：

(2.16)

(2.17)

上式中是更新系数，决定了每次更新的幅度，一般取值为0.1或0.01等。

在训练过程中，为了保证探索性，DDPG为action的决策引入了随机噪声*N*，所以最终与环境交互的动作A的表达式为：

(2.18)

最后，DDPG的更新也分为两部分，对于Critic当前网络，其损失函数是和DQN类似的MSE，即：

(2.19)

(2.20)

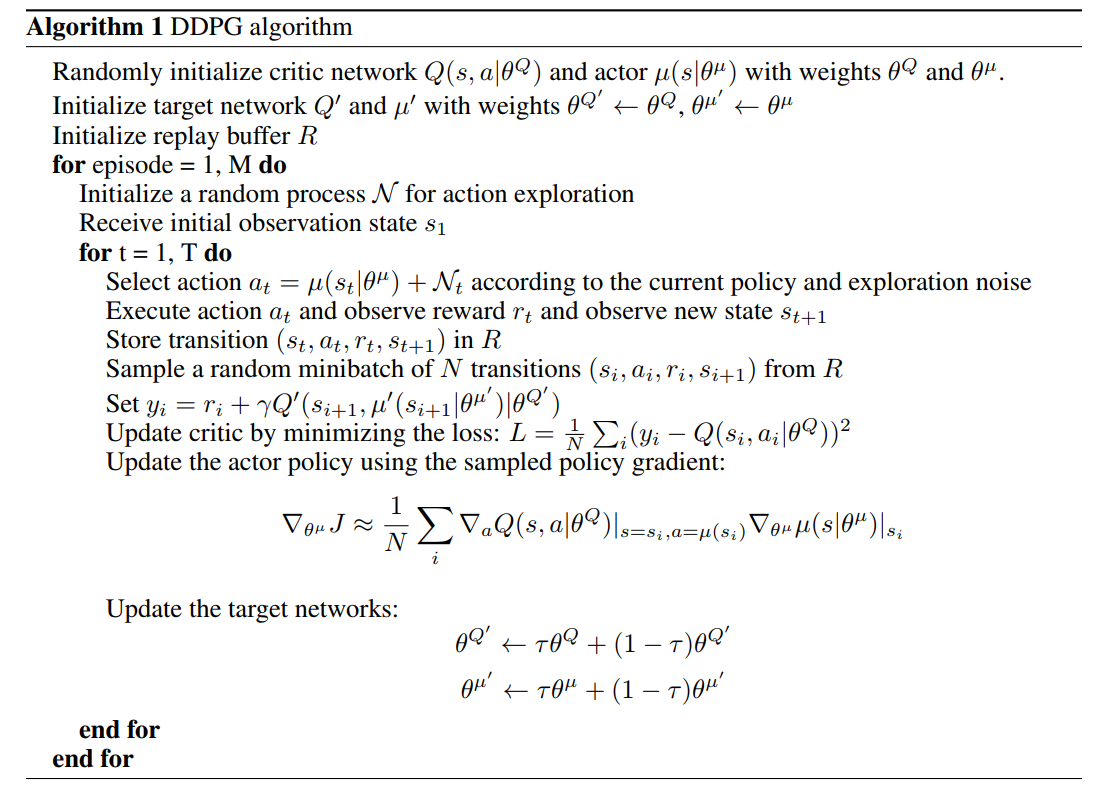
而对于Actor当前网络，论文原文定义的损失梯度是：

(2.21)

上述的公式的意义是：假如对于同一个状态，我们有动作a1和a2可以选择，那么从Critic当前网络就可以得它们对应的Q值即Q1和Q2，而它们之中更高的一个就代表着其对应的动作能够取得更好的成绩。而策略梯度的思想就是在这个时候增加这个更优动作被选中的概率，所以在DDPG中，Actor的损失可以认为是与Q值成反比，因此Actor的更新只要用负的Q值作为参考即可，即：

(2.22)

根据上述理论，我们可以得出DDPG的伪代码：



## 2.6 A3C算法

A3C算法（Asynchronous Advantage Actor-Critic）与DDPG一样是一种Actor-Critic算法。A3C被设计出来的初衷其实同样也是为了解决Actor-Critic类算法难以收敛的问题。我们知道经验回放能够降低数据的相关性，但实际上还是无法完全打破这个相关性的（因为经验池中的数据还是由同一个Agent产生的）。而A3C最主要的特点就是异步（Asynchronous），它在多个线程中运行独立的Agent，每个线程都定期把通过学习得到的梯度更新到一个公共的主要Agent上，然后再统一升级所有的Agent。这样做能让A3C避免数据相关性过强而导致的难以收敛，因为它使输入数据的分布变得更加均匀，从而让V(s)更接近真实值，另外异步并发的学习模型能够以成倍的速度进行样本采集与学习。

A3C对于传统Actor-Critic的优化体现三个地方，分别是实现了异步学习，对网络结构进行了优化，以及对Critic的评估点进行了优化。而在这三者中，实现了异步学习是最重要的一个优化点，A3C的异步框架如下图所示：

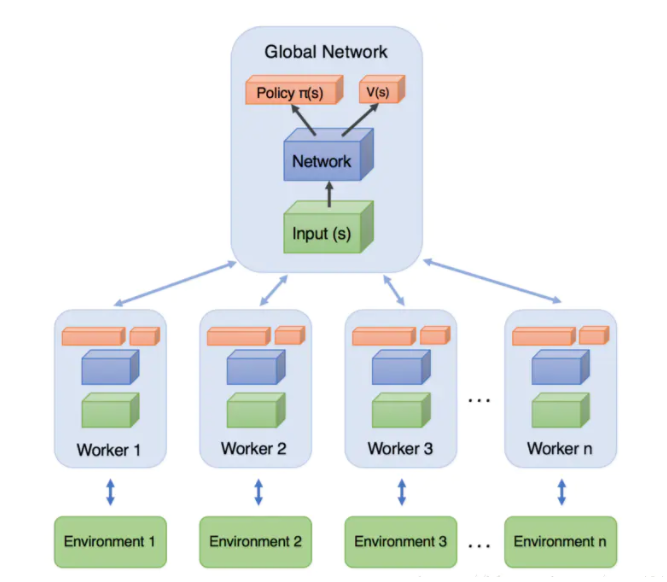


图2.7 A3C算法结构图

上图就是A3C的大体框架，图中的全局网络（Global Network）就是前文中描述的公共的主要Agent，其下有多个线程，每个线程中Agent的网络结构都与全局网络的结构一样，且每个线程会不断地与自己的环境进行交互并得到经验数据。

经过了一定的时间或者探索了一定的步数后，线程中的Agent就会用自己的经验数据来计算自己网络的loss并得到相对于权重的梯度，但它们不会用这些梯度进行自我更新，而是去更新公共的主要Agent的网络参数。当经过一定的步数后，线程会主动去获取主要Agent的参数并复制到自己的网络中让自己得到优化。

从上面的过程就能够看出，公共的主要Agent的网络就是我们的目标模型，而其他线程中的Agent的职责就是在各自的环境中获取数据并计算梯度，这样做可以让整个训练过程中拥有更少的数据相关性，从而使全局的学习不容易收敛到局部最优并且在整体上加速了收敛。

而A3C的第二个优化则是网络结构的优化。A3C的原论文提出了将策略网络和评价网络整合到一起的方法，也就是构造一个输入为状态s，输出为相应策略和价值的网络。其结构如下图所示：

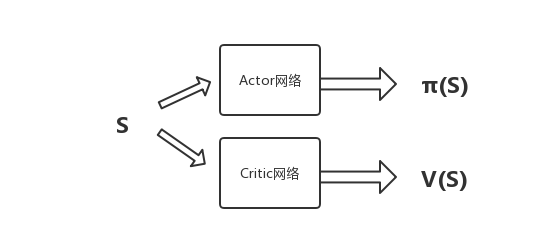


图2.8 A3C中的网络结构优化

A3C的第三个优化是对Critic的优化，也就是A3C中的Advantage（优势）部分，也就是优势函数A代替传统Actor-Critic中的Q函数来供Actor参考，A的表达式为：

(2.23)

Q(S,A)的值一般可以通过单步采样近似估计，即：

(2.24)

这样优势函数去掉动作可以表达为：

(2.25)

式中的V(s)由Critic网络得到。

A3C的采样方式也不相同，它使用了N步采样法，这样就可以比完成整个episode再进行采样要更快，所以新的A函数可表示为：

(2.26)

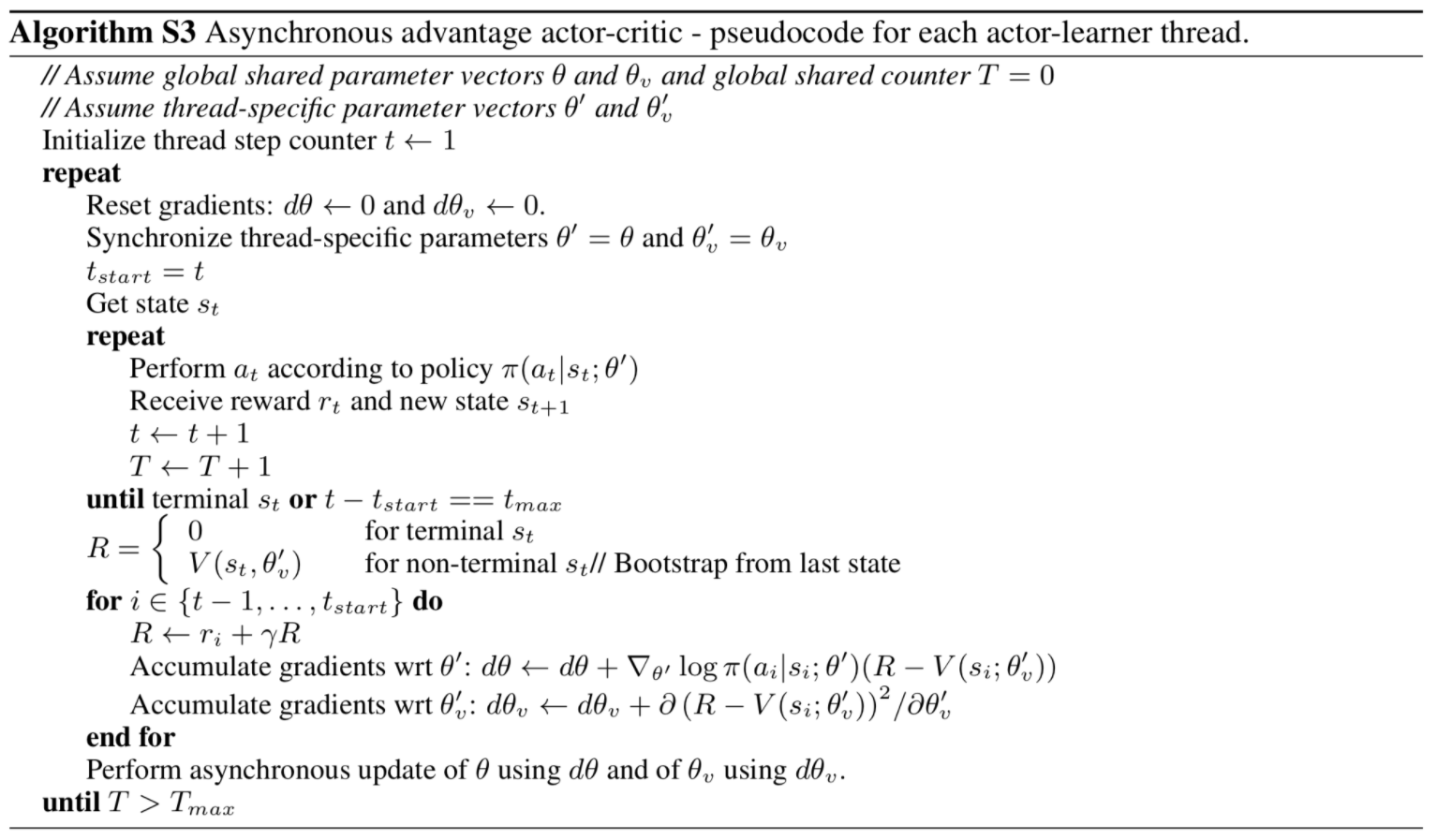
对于上面的公式，我们可以将其理解为：在t时刻，状态s的Advantage值为往后N个s的reward之和（代入衰弱因子）加上t+N时刻的V值再减去当前的V值。

此外，A3C对于策略函数的损失函数也进行了优化，即在策略函数的损失函数中加入了熵项，所以A3C策略函数优化后的梯度更新公式如下：

(2.27)

加上这个熵项的意义就在于，agent与environment进行互动的过程中最好能够平衡地进行探索，最好是在每个状态都能对各种动作进行尝试，从而不会使采样得到的样本太过于集中。因而需要加入这个熵项来使策略输出的动作概率分布能够更加的趋于平衡，也就是说策略会变得更加随机。在后续的研究中，这种方法也被证实了可行性，并且还衍生出了Soft Actor Critic这样的以最大化熵为主要目标之一的强化学习方法。

根据上述理论，我们能够得到A3C的伪代码如下：



## 2.7 SAC算法

为了设计稳定且高效的Model-Free强化学习方法，加州伯克利大学提出了一种Off-Policy的Actor-Critic算法——Soft Actor-Critic（SAC），SAC是一种基于最大熵的强化学习框架。在这个框架中，Actor的目标是在最大化期望奖励的同时，也使熵最大化。不同于DDPG将Off-Policy Actor-Critic与确定性策略结合，SAC是将Off-Policy Actor-Critic与随机策略进行了结合。DDPG虽然实现了在Off-policy learning中重复利用样本，但是它对超参数比较敏感，并且收敛性脆弱。

SAC最大的特点就是把在学习中熵的最大化作为一个重要目标，这样做可以让输出的策略中的每个动作的概率都更加平均，也就使整个策略变得更加随机。而这个特点也正是Soft的含义——拥有更加平滑的策略概率分布，也更接近真实的概率分布，如图2.8所示。

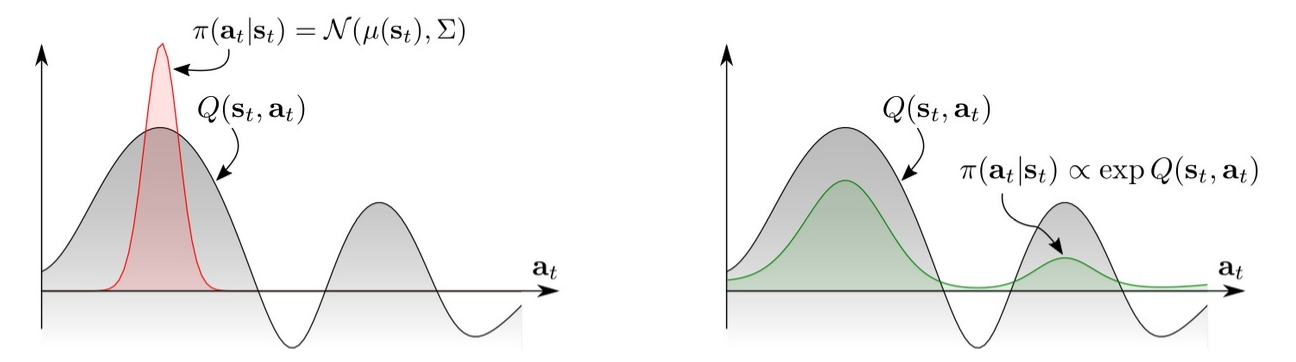


图2.9 SAC与其他随机策略算法的概率分布对比图

在前文2.5章节中所提到的DDPG是一种确定性策略，即对于每个状态只考虑一个最优的动作，它的优势就是能够用少量的经验数据学习到策略。显然，SAC作为与之相对的随机策略，那么它会有数据需求量相对更大的缺点，但它也有非常明显优势：随机策略对每一个状态的每个动作都会拟合出相应的概率，这就使SAC的策略能够拥有很强的鲁棒性，因为它不会像确定性策略那样永远只考虑一个情况的最优操作，它永远都在依照动作的概率进行随机行动。而SAC中的最大熵的作用则是保证每个状态下所有动作都会被平均地探索到，也就是能够让随机策略的概率分布更加趋于真实情况。综上所述，SAC有着以下优势：

1. 学习出的策略能够在后续研究更复杂的新任务时，作为初始化的模型，因为通过最大化熵学习到的模型对所有可能的情况都被概率分布覆盖，更具适应性。
2. 具有更强的探索能力，这让SAC能够在多个奖励目标的环境下找到更优的策略。
3. 更强的鲁棒性，因为要在环境中尽可能多得去探索各种动作和情况，所以在与环境交互中遇到意外情况或者一些随机性的干扰时SAC也能够有不错的表现。

通常的强化学习方法目标是最大化累积奖励，而SAC的目标是加入了最大化熵的累积奖励：

(2.28)

上式中的参数*α*用于调整动作的熵在整个目标中的比重，即越大的*α*会让Agent的策略越平均地覆盖到每个动作，代表服从策略的概率分布。

在对策略的好坏进行评判时，SAC把最大化熵（式2.28）作为目标来计算其对应的值函数。而策略对应的软Q值的Bellman方程为：

(2.29)

上式中V()如下：

(2.30)

代表软状态值函数（Soft state value function）。软策略评估可以通过迭代，理论上只要不断地进行迭代，Q就会越来越靠近策略π的软Q值函数。此时定义的带熵的奖赏公式则是：

(2.31)

软Q函数的更新规则为：

(2.32)

而在更新策略时，SAC原文中有更新公式：

(2.33)

上式中，我们把Q函数转换成一个表示策略的概率分布，然后求策略和Q值策略的KL散度(Kullback-Leibler divergence，又称相对熵)值为最小的时候的策略。公式中的，而是最优化情况下上式的策略，也同样在策略空间中，满足，这样就能够保证每次更新策略都优于旧的策略。

上述迭代过程是基于表格式环境（具有离散状态空间的环境）来推导的，对于状态空间和动作空间都连续情况就需要引入函数逼近。首先定义软状态值函数，软Q值函数以及策略函数，对应的参数分别是。

这时软状态值函数目标函数就是：

(2.34)

软状态值函数梯度：

(2.35)

软Q值的目标函数是：

(2.36)

其中为代入参数后的软Q值Bellman方程（2.29所示），加入参数后表达如下：

(2.37)

软Q值函数梯度：

(2.38)

策略更新的目标函数：

(2.39)

这里策略网络输出的是高斯分布的均值和协方差，但是如果直接用输出的均值和协方差去采样的话就会遇到一个问题——动作a将会是不可导的。所以这里用到了reparameterization trick。简单来说就是先从一个高斯分布采样，然后再把得到的采样值乘以协方差再加上均值，这样网络就变成可导的了。因此上式需要变形如下：

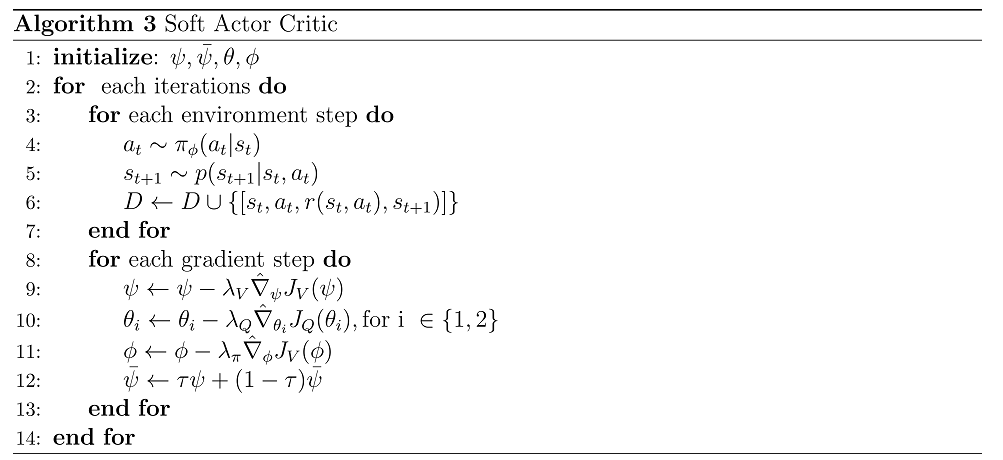
(2.40)

(2.41)

而策略函数的梯度如下：

(2.42)

综上所述，SAC的伪代码如下：



Soft Actor Critic通过在普通的Actor-Critic中加入最大化熵项，让先前研究中的样本效率低下、收敛性差、超参数敏感等问题得到了很大的改善。

## 2.8 奖励函数的设计

在进行强化学习时，奖励函数是必不可少的一部分，很多gym环境都自带了奖励函数，但是当需要处理一些自定义的问题时，就需要人工设计一个合理的奖励函数。作为强化学习的目标，奖励函数的好坏会对Agent的训练成果产生决定性作用。那么对于本项目中的无人驾驶问题，我们来分析一下我们的核心驾驶需求：

1. 车辆应该安全地驾驶在路面上，偏离车道或者发生碰撞将会得到最大的惩罚。
2. 车辆的速度应该与奖励成正比，但是应该设置一个max speed来防止车辆行驶过于快速，当超过max speed时也需要得到惩罚。

按照上述基本需求，我们可以得到一个最简单的无人驾驶奖励函数：

(2.43)

式中的，为可变参数，可以根据重要性对其进行调整。在这样的奖励函数之下，Agent为了得到更高的奖励，必须要以一个近似expect speed的速度行驶，并且尽力不去碰撞和越过车道。但是经过实验我们会发现：车辆会在驾驶中不停地左右摇摆。其实这很容易理解，因为我们的奖励函数并没有约束它不让其产生这样的行为，而为了能让车辆平稳地驾驶，我们可以有如下的一些改进方案：

1. 策略函数输出相对转向而不是绝对转向：可以降低摇摆的频率。
2. 添加一个连续转向的惩罚：agent无法正确地被优化，并且可能出现无法保持在车道驾驶。而如果降低这个惩罚值则可能会使agent忽略它。
3. 限制最大的转向值：agent在直线上驾驶地更加平稳一些，但是当车道转角过大时会无法很好地转弯。
4. 加入前几个连续状态的速度信息：略微降低摇摆频率。
5. 加入当前CTE的惩罚并设置权重：在连续动作空间中具有较为平稳的表现。

可以发现以上尝试的方法都不能起到明显的作用或有一定的限制，但是实际上我们也能够结合它们的优势来让车辆驾驶得更加平稳。另外在具有路径规划的无人驾驶中，我们还可以加入车辆与目标路径的偏离程度作为惩罚加入到奖励函数中。

## 2.9 本章小结

本章首先在前两节介绍了本项目涉及到的核心技术——深度学习与强化学习的基本原理，并对其进行了详细的介绍与分析，多层感知机、反向传播法、卷积神经网络和几种具有代表性的优化方法都包含其中。第三节介绍了一些在DRL算法中能让模型训练更加有效和高效的训练技巧。之后介绍了深度强化学习的分类，并对本项目中需要使用到的DDPG、A3C、SAC三种算法进行了详细的原理分析和公式推导。最后，笔者就无人驾驶的目标给出了奖励函数的雏形。

# 第3章 无人驾驶系统开发技术与平台

上一章已经讲述了本项目中一些核心技术的原理，本章主要介绍当前比较常用的机器学习框架和一些无人驾驶模拟器。

## 3.1 Tensorflow框架

TensorFlow作为当今机器学习领域最为人熟知的框架，其本质是一个基于数据流编程的符号数学系统，现在许多的机器学习算法都能够通过它轻松实现，它最早是由Google的神经网络算法库DistBelief演变而来的。TensorFlow由谷歌大脑（Google Brain）进行开发和维护，从2015年11月开始，TensorFlow依据阿帕奇授权协议正式对其源码进行开源。

Tensorflow拥有着多层级结构，新手用户可以调用使用方法简单的高级API，而更加熟练的用户则可以通过更底层的API来实现一些自定义的特殊操作以及高效率的执行。也正因为这个特点，Tensorflow可以在各类服务器、PC终端和网页上被部署，并且能够利用GPU和TPU来进行高性能的数值计算，这也是它能够在许多科研领域大放异彩的原因之一。

## 3.2 PyTorch框架

PyTorch是Torch的Python版本，是一个由Facebook开发并在社区开源的神经网络框架，用户能够用它来编写DNN的程序。Torch 本质上是一个能够计算多维矩阵的张量（tensor）库，但是不同于使用静态计算图的Tensorflow，PyTorch拥有着动态的计算图，它可以根据当前计算的需求实时地对计算图进行改变。但由于原版的Torch采用了Lua语言，导致它在国内一直都没有被广泛使用，并且由于Tensorflow的崛起，也一度失去了许多用户。但现在的PyTorch为Python开发者提供了API，而Torch本身的易用性和简洁的特点也让它在机器学习领域拥有了一席之地。

## 3.3 Keras框架

Keras 是一套用Python编写的用于编写神经网络程序的高级API，它能够以 [TensorFlow](https://github.com/tensorflow/tensorflow)，[CNTK](https://github.com/Microsoft/cntk)，或者[Theano](https://github.com/Theano/Theano)作为底层驱动来运行。Keras的核心目标是能够让用户快速地进行实验，让用户在最短的时间内对自己的想法进行实验并得到结果。它让用户能够快速而简单地对DNN进行设计、调试、评估、应用和可视化，因此在实验和测试场景中，Keras是一个非常好的选择。

Keras严格按照面向对象的方法进行编写，它具有非常高的[模块化](https://baike.baidu.com/item/%E6%A8%A1%E5%9D%97%E5%8C%96/3295536)程度并且具有[可扩展性](https://baike.baidu.com/item/%E5%8F%AF%E6%89%A9%E5%B1%95%E6%80%A7/8669999)，在用户使用角度它也考虑到了初学者的感受，对复杂算法的实现进行了简化，降低了开发难度。Keras能够对当前[人工智能](https://baike.baidu.com/item/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD/9180)领域的许多主流算法进行支持，诸如前馈结构、递归结构的神经网络也包含其中，用户也可以使用它来构建统计学习模型。在GPU、TPU计算在机器学习领域被广泛使用的现在，Keras也能在不同的OS下支持多GPU的[并行计算](https://baike.baidu.com/item/%E5%B9%B6%E8%A1%8C%E8%AE%A1%E7%AE%97/113443)，并且用户可以自行选择使用Theano、Tensorflow或CNTK作为后台。

## 3.4 Gym强化学习环境

OpenAI Gym[25]是OpenAI于2016年5月4日发布的一个用于开发强化学习算法并对其进行测试的工具包，它能够持训练智能体（agent）做任何事——从行走到玩Pong或围棋之类的游戏都在范围中。它不会对Agent的结构进行假设，用户能够对Agent进行完全自定义，并且它与许多的机器学习框架（如Tensorflow、PyTorch、Keras等）兼容，使用户能够简单地将自己编写的算法或处理逻辑应用到环境中。Gym中内置了许多已有的强化学习环境，所以用户也可用它来开发RL算法库，并对各种算法进行多种情况下的比较。另外，Gym的一个很大的特点是可以可视化，也就是将环境的实际图像渲染出来，把强化学习算法与环境的交互用动画的形式呈现在用户眼前。

## 3.5 无人驾驶模拟器

本节对笔者用到的一些驾驶模拟器进行说明，具体有CARLA、TORCS和DonkeyCar。

### 3.5.1 CARLA

CARLA[26]是一个开源的模拟器，能够真实地模拟出来现实中的行人行为、交通道路环境、车用传感器信号以及很多生活中存在的事物。它能够支持自主驾驶系统的开发、训练和验证，除了对代码和协议进行了开源之外，Carla还对用户开放了许多已经构建好的环境模型（城市布局、建筑物、交通工具等模型），用户能够任意选择并进行布置。CARLA仿真平台支持多种传感器，天气环境条件，对所有动态和静态对象也能够进行完全的控制。模拟器由C++和虚幻 (Unreal) 引擎组成，用户可以通过模拟器提供的Python API来对模拟器的环境和对象进行操作和控制。

### 3.5.2 TORCS

TORCS[27]（The Open Racing Car Simulator）是一种高度便携的多平台赛车模拟器，是用来研究普通赛车游戏、人工智能赛车游戏的一个平台。它能够在多种操作系统上编译并运行，其代码完全使用C++和C编写。它的3D车模能够满足研究人员、工程师和老师的研究需要；因为有着复杂的物理模型和许多库和附加软件，使之更加符合现实世界，并且支持大多数的输入设备例如方向盘、操纵杆、游戏手柄等等。而模块化的架构也让它易于修改、添加和创建内容，并且具备优异的性能和稳定性。

### 3.5.3 DonkeyCar Simulator

Donkey Car[28,29]是一个开源的小型自动驾驶汽车，下图3.1是donkey样车，在它上面有一个摄像头、树莓派、单片机、电机、舵机和电源。但是制作一辆实体的Donkey Car需要一定的成本，并且也需要一些时间，且第一次组装Donkey Car的人可能会遇到许多的问题，实际测试中也很容易发生碰撞导致硬件损坏。为了解决这些问题，我们可以使用开源社区提供的Donkey Car Simulator[30]。用户能够通过Python API来操作模拟器中的小车进行驾驶，在本项目中，我们在代码中封装了模拟器的控制指令使其符合gym框架，并在此之上使用深度强化学习算法进行学习。



图3.1 基于树莓派的Donkey Car实体小车

## 3.6本章小结

本章主要介绍了本项目可能用到的一些技术框架和驾驶模拟器，主要有Tensorflow、PyTorch和Keras三个机器学习框架，在实际使用他们能够让我们方便地进行网络构建和训练等。而模拟器方面则介绍了CARLA、TORCS和Donkey Car Simulator三种，就模拟的真实程度和复杂程度而言，CARLA位居第一，因此本项目也主要围绕CARLA来进行。

# 第4章 需求分析与架构设计

上一章讲完了无人驾驶系统所需的一些技术框架和模拟器，本章将对无人驾驶系统进行需求分析，然后对它整体的框架结构进行设计。

## 4.1需求分析

本课题需要设计并实现基于DRL算法的无人驾驶系统，通过训练让agent能够自主做出驾驶决策，代替人的手动驾驶。在本系统中，用户具有如下的几个主要功能需求：

1. 可用且可靠的无人驾驶算法模型。

作为一个无人驾驶系统，最重要的就是能够可靠地进行无人驾驶，而实现这一点也是本系统的核心——算法设计和模型训练。

1. 能够进行自定义的训练，并保存模型。

除了能够使用已经训练好的算法模型进行无人驾驶之外，用户也需要能够自己来训练新的模型，并且训练时的超参数是能够自定义设置的。而在训练完成之后，用户也能将训练中得到的网络模型保存到本地。

1. 能够自定义新的算法模型。

用户除了能够直接使用系统已经事先给出的三个深度强化学习算法外，用户也能够自己编写自定义的算法模型，系统中会提供相应的API给用户。

1. 手动驾驶模式。

为了让用户熟悉环境，以及应对一些意外情况，用户需要能够使用手动驾驶模式来控制车辆。

1. 视觉选项。

用户能够在使用期间打开深度图像和语义分割图像，以供参考。

## 4.2运行环境需求分析

无人驾驶系统进行训练时的硬件环境需求如表4.1所示：

表4.1 无人驾驶系统训练环境的硬件需求

|  |  |
| --- | --- |
| 硬件类别 | 硬件配置 |
| CPU | 四核Intel或AMD处理器，2.5GHz或以上 |
| 显卡 | NVIDIA Geforce GTX1080Ti以上 |
| 显存 | 8GB以上 |
| 内存 | 16GB以上 |

无人驾驶系统进行测试驾驶时的硬件环境需求如表4.2所示：

表4.2 无人驾驶系统测试环境硬件需求

|  |  |
| --- | --- |
| 硬件类别 | 硬件配置 |
| CPU | 四核Intel或AMD处理器，2.5GHz或以上 |
| 显卡 | NVIDIA Geforce GTX1050Ti以上 |
| 显存 | 4GB以上 |
| 内存 | 8GB以上 |

无人驾驶系统的软件环境需求如表4.3所示：

表4.3 无人驾驶系统的软件环境

|  |  |
| --- | --- |
| 软件类别 | 软件配置 |
| 操作系统 | Microsoft Windows10或Ubuntu16.04 |
| Python解释器 | Python3.5版本，并按照Requirements.txt预装所有所需的包 |
| 驾驶模拟器 | CARLA0.9.5以上、Donkey Car Simulator v18.9以上 |
| 显卡驱动 | 为支持CARLA模拟器和python机器学习框架，需要410以上版本 |
| CUDA | 10.0以上版本 |

## 4.3稳定性及安全性需求分析

对于一个无人驾驶系统来说，最重要也最基本的需求就是安全性和稳定性，而要实现它们则需要从两个部分考虑：

1. 软件安全性。

在本课题中，实验应秉持严谨认真的态度进行多次，且多方位多角度的验证与测试，以确保模型做出的决策不会出现重大失误。

1. 硬件安全性。

无人驾驶系统需要agent持续地进行计算和做出决策，因此硬件资源必须要稳定运行，如果出现运行卡顿或卡死都会使无人驾驶系统出现严重失误。

## 4.4系统架构设计

本节主要对无人驾驶系统进行架构设计，其架构图如下：

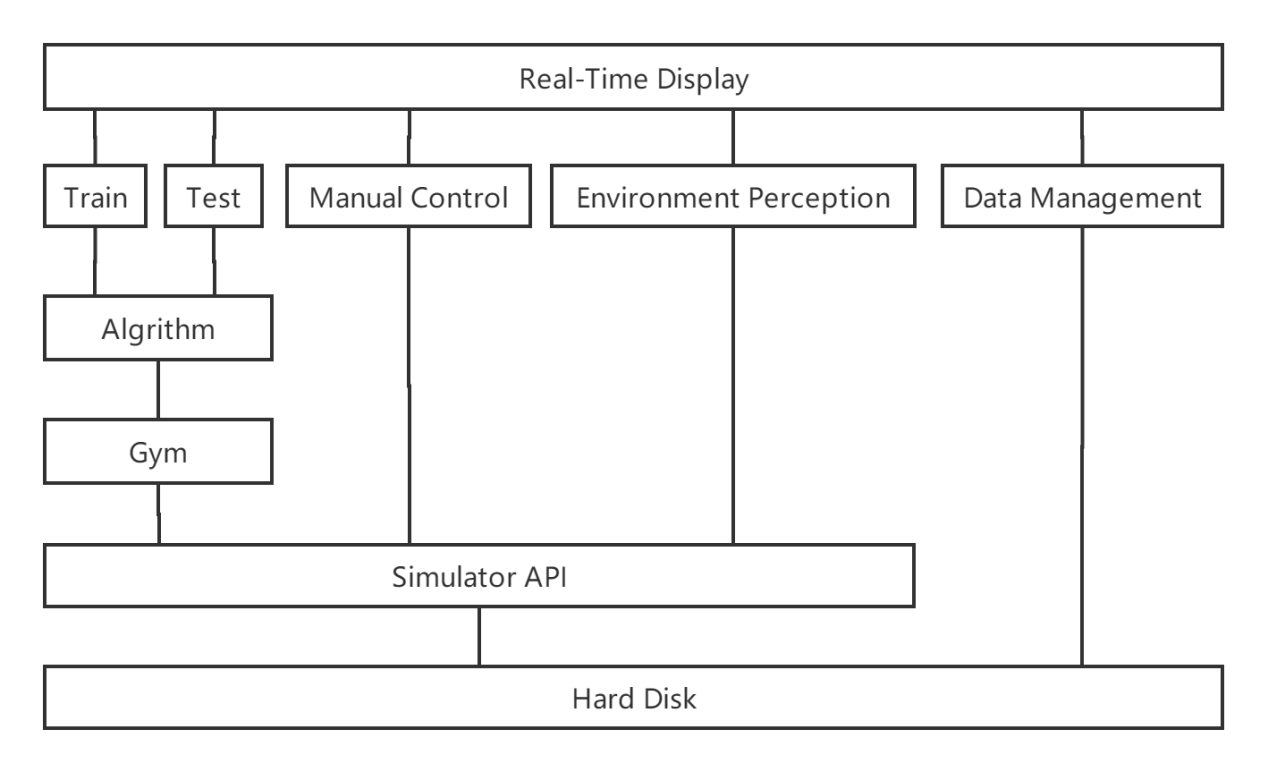


图4.1 无人驾驶系统框架图

本系统的整体架构一共分为6层，具体功能如下：

第一层为存储层，主要作用是存储系统产生的权重文件、日志文件、配置文件等数据，同时也负责运行模拟器的程序本体。

第二层为模拟器交互层，主要是通过Python调用模拟器的开放API，使我们能够通过程序来操作模拟器。

第三层为Gym框架层，主要是对模拟器的API进行封装，实现一个Gym强化学习环境，使我们的算法模型能够调用它并进行学习和验证。

第四层为算法模型层，主要实现了DDPG、A3C、SAC三种算法模型，另外，用户能够参照第三层的gym环境进行自定义模型的编写。

第五层为应用层，主要实现了训练、测试、手动控制、环境感知、存储数据等功能。

第六层为展示层，主要用于实时显示当前车辆的运行视角画面，根据选项设置也能够额外显示深度图像和语义分割图像。

## 4.5系统功能设计

本课题中的无人驾驶系统的功能如下图：

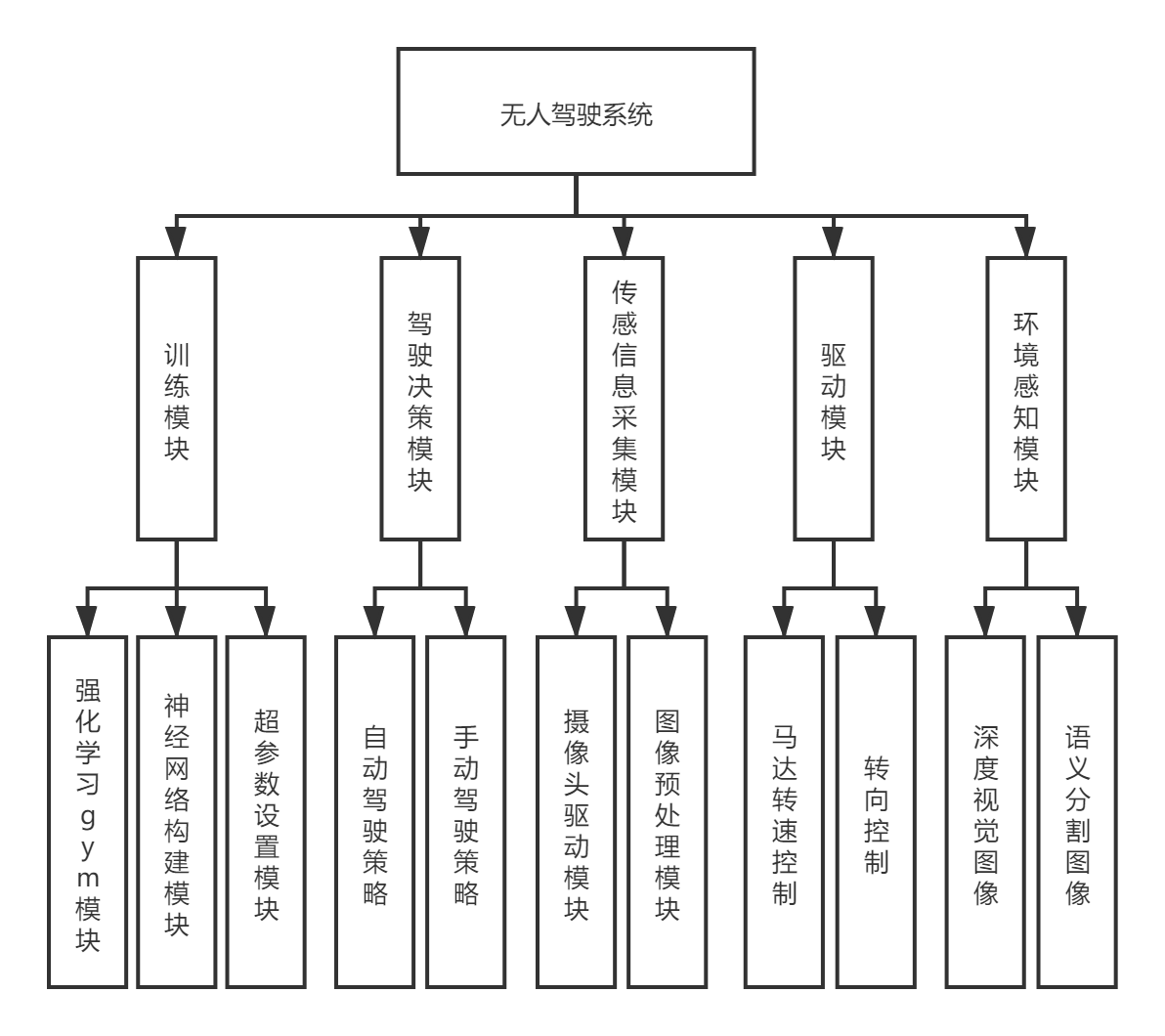


图4.2 无人驾驶系统功能模块图

如上图所示，无人驾驶系统一共具有五大模块，分别是训练模块、驾驶决策模块、传感信息采集模块、驱动模块和环境感知模块。其中，训练模块是本项目最核心的模块，它包括强化学习gym模块、神经网络构建模块、超参数设置模块。而驾驶决策模块则是对已有的模型进行验证和测试，另外也支持用户直接手动控制车辆的驾驶。传感器信息采集模块包括了摄像头的驱动模块和图像的预处理模块，但是要注意的是，在对模拟器使用无人驾驶系统时，并不存在摄像头驱动模块，因为图片能够直接从模拟器中获取，而这个模块是用于Donkey Car实体小车的。同理，驱动模块部分也是用于Donkey Car实体小车控制的。最后的环境感知模块则是用于打开深度视觉图像和语义分割图像。

## 4.6系统交互流程分析

无人驾驶系统的训练需要遵循下图的流程，先由用户下达训练指令，无人驾驶系统与模拟器进行连接并进行初始化，如果需要继续学习则需要再读取本地的模型文件，然后开始与环境交互并优化模型。结束后将新的权重文件保存到本地并输出训练信息。

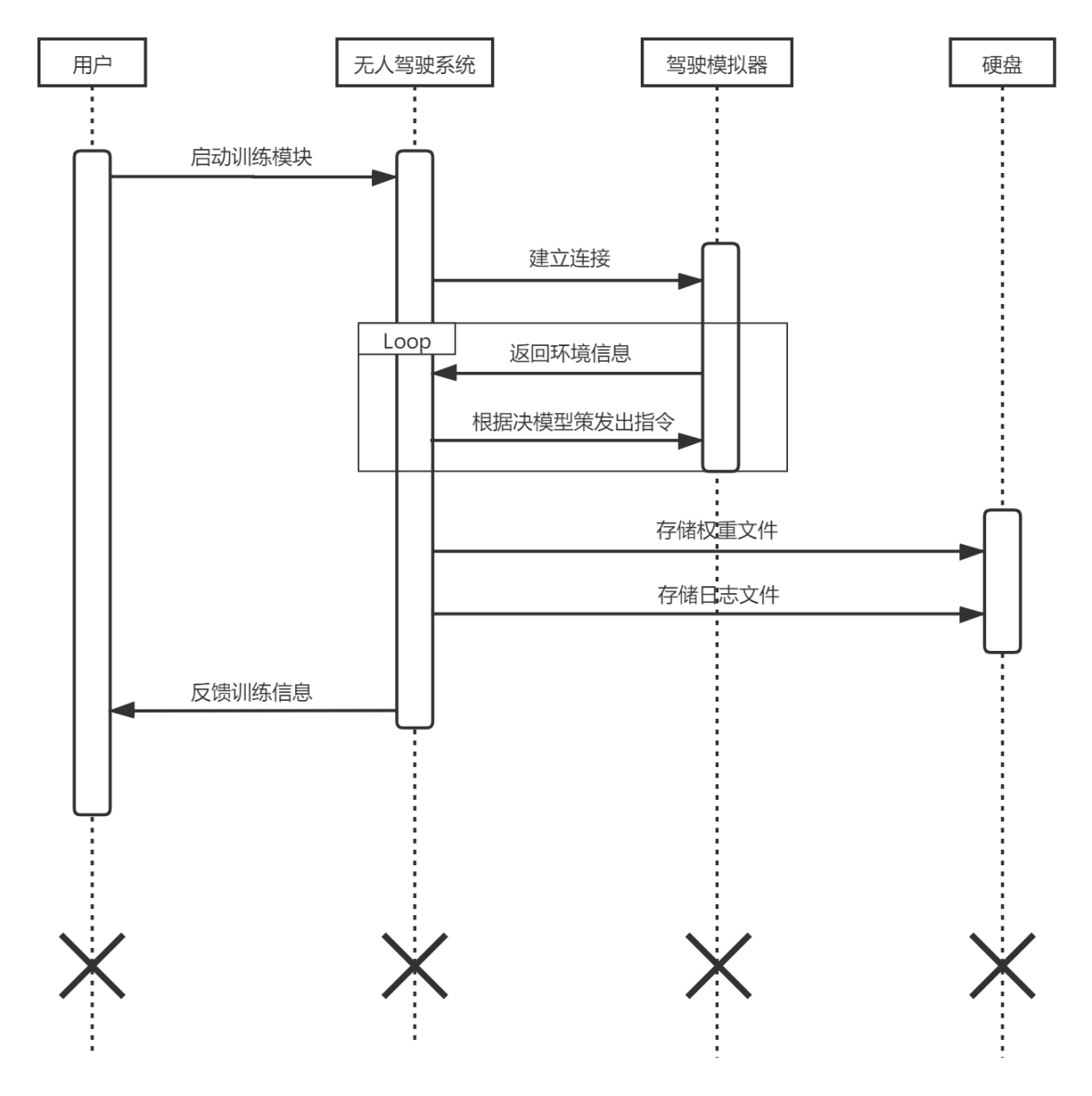


图4.3 无人驾驶系统训练模式时序图

无人驾驶的验证需要遵循下图的流程，先由用户下达驾驶指令，无人驾驶系统与模拟器进行连接并进行初始化并读取本地的权重文件，然后开始用读取到的模型与环境交互，在经过一定步数后结束驾驶。

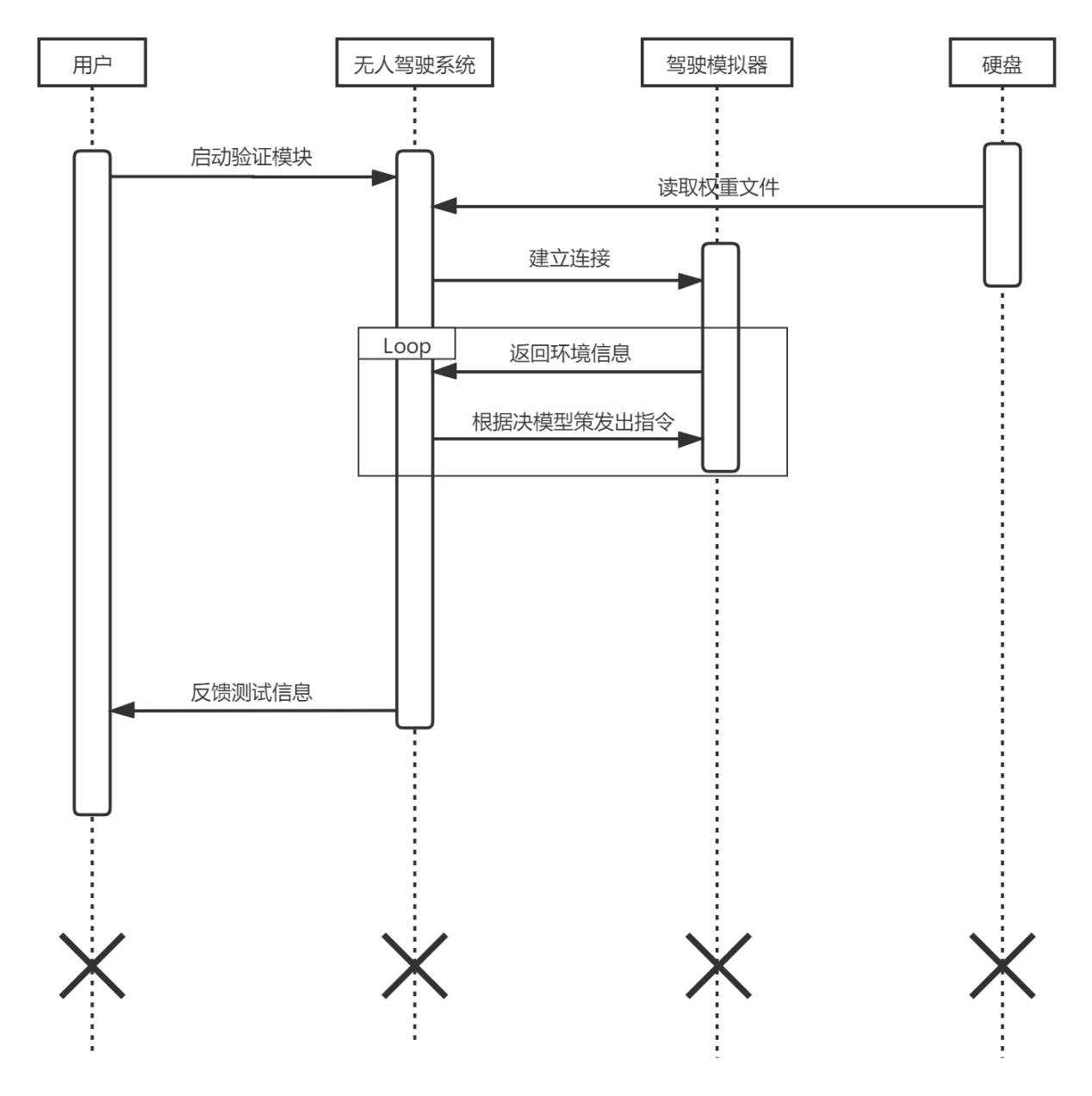


图4.4 无人驾驶系统验证模式时序图

无人驾驶的手动控制需要遵循下图的流程，先由用户下达驾驶指令，无人驾驶系统与模拟器进行连接并进行初始化，然后由用户发出实时的控制指令。

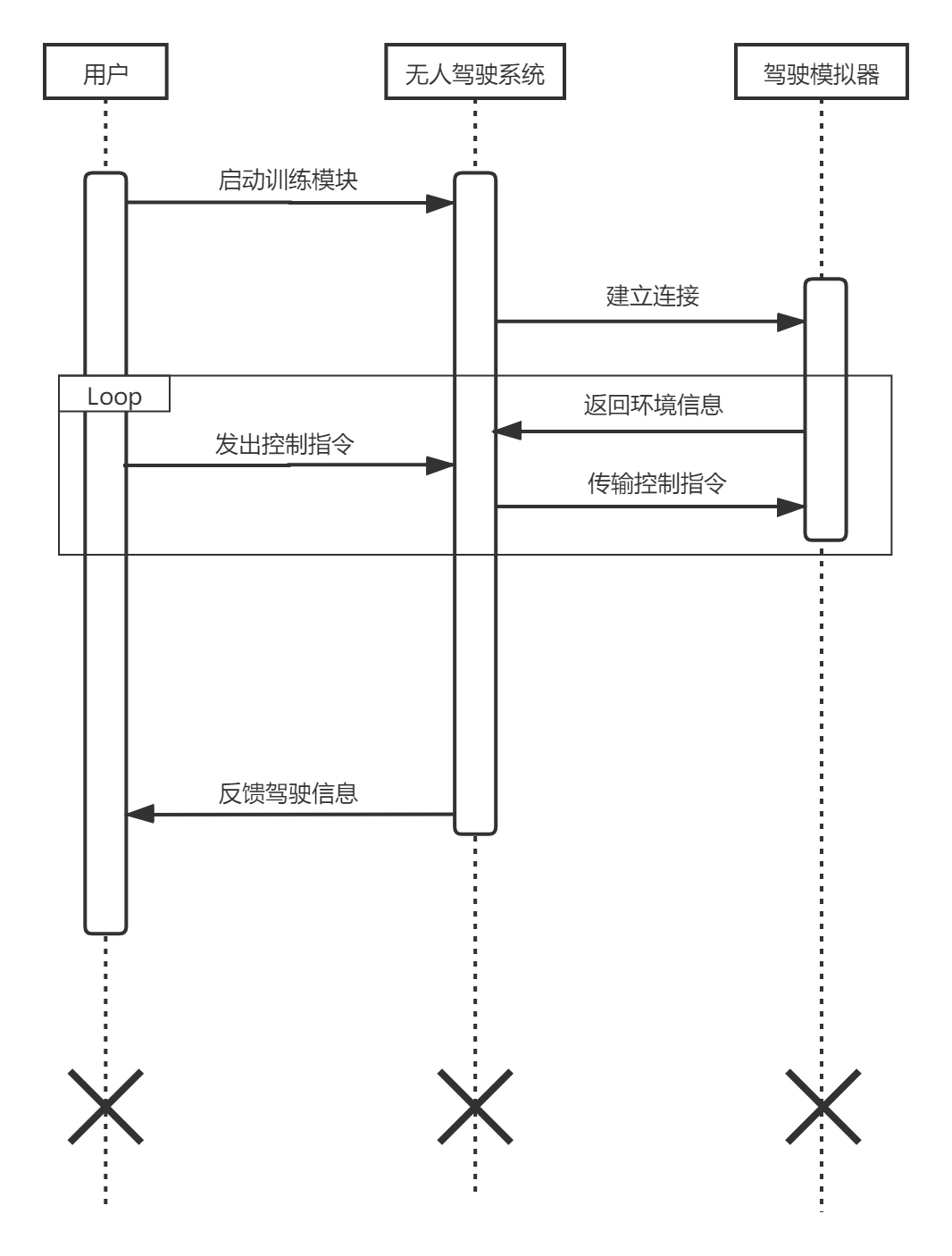


图4.5 无人驾驶系统手动驾驶模式时序图

## 4.7本章小结

本章主要对无人驾驶系统进行了需求分析和架构设计，明确了系统需要达到的目标和对软硬件环境的要求，同时也对系统中的各部分功能模块进行了概述。并对用户的训练、验证和手动控制的交互流程做了分析。

# 第5章 详细设计

上一章对无人驾驶系统进行了需求分析和架构设计，本章将对无人驾驶系统进行详细的设计与实现。

## 5.1训练模块设计

训练模块是本系统中最核心的部分，它起到训练模型的作用，而模型训练的好坏将直接影响到无人驾驶的效果。在这个模块中，首先需要连接到模拟器，然后获取环境数据，计算reward后存储到经验回放池，用于每个回合的学习。具体过程如图6.3所示。

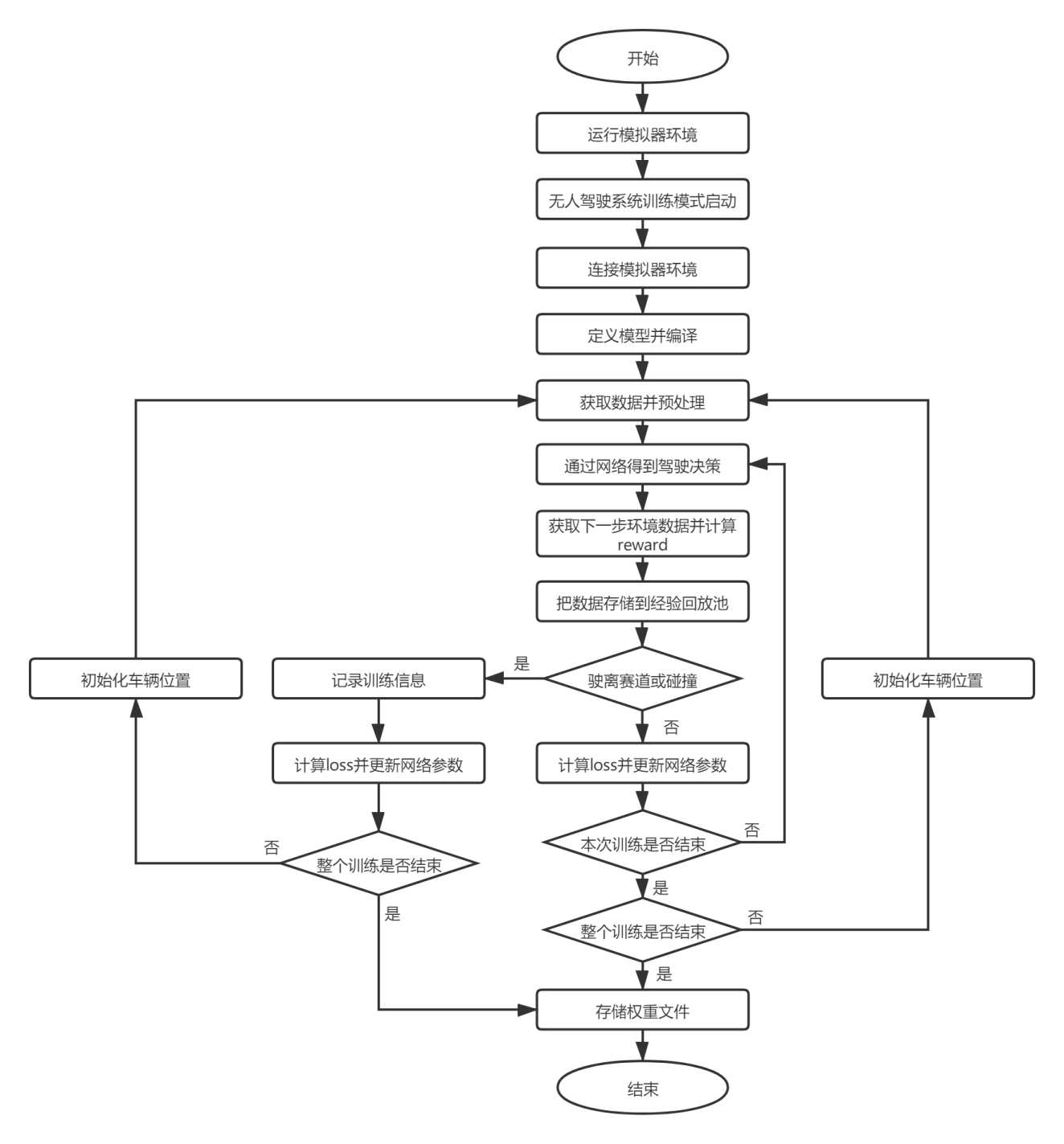


图5.1 无人驾驶系统训练模块流程图

## 5.2验证模块设计

验证模块是无人驾驶系统中用于展示模型效果的模块，同时也是应用过程中最重要的部分。而模型训练的好坏将直接影响到无人驾驶的效果。在这个模块中，首先需要连接到模拟器，读取已经训练好的权重文件并加载模型，然后用它与模拟器环境进行交互，验证模块的具体流程图如图5.2所示。

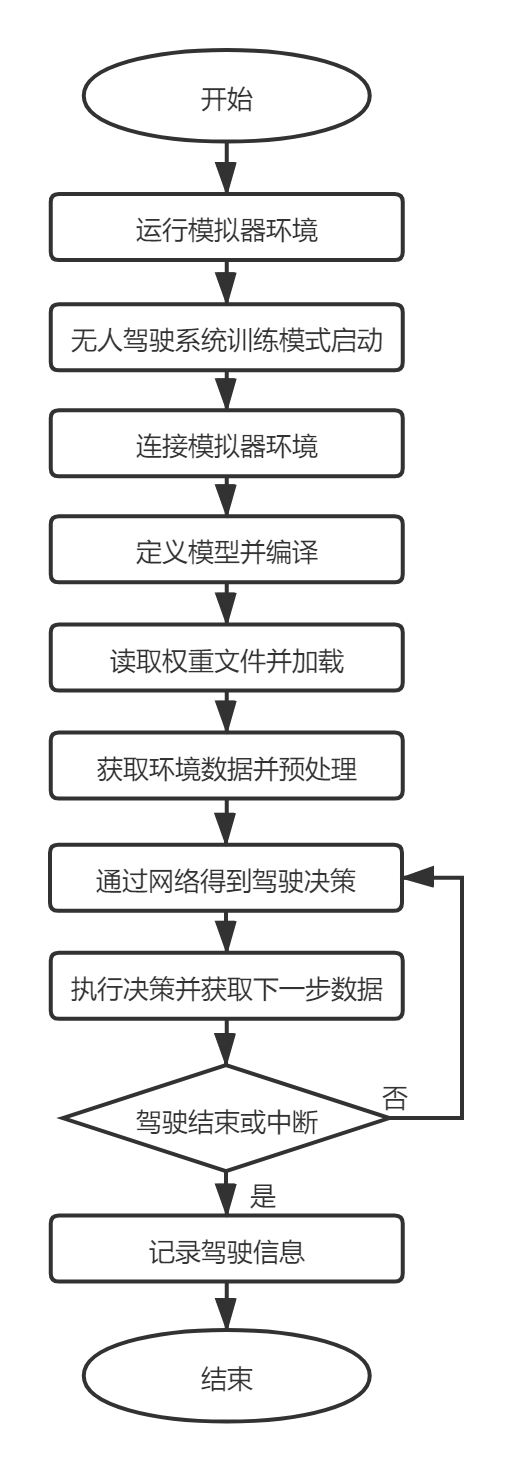


图5.2 无人驾驶系统验证模块流程图

## 5.3手动控制模块设计

手动控制模块用于支持用户直接控制车辆。在这个模块中，首先需要连接到模拟器，然后通过模拟器提供的API与模拟器进行交互，手动控制模块的具体流程图如图5.3所示。

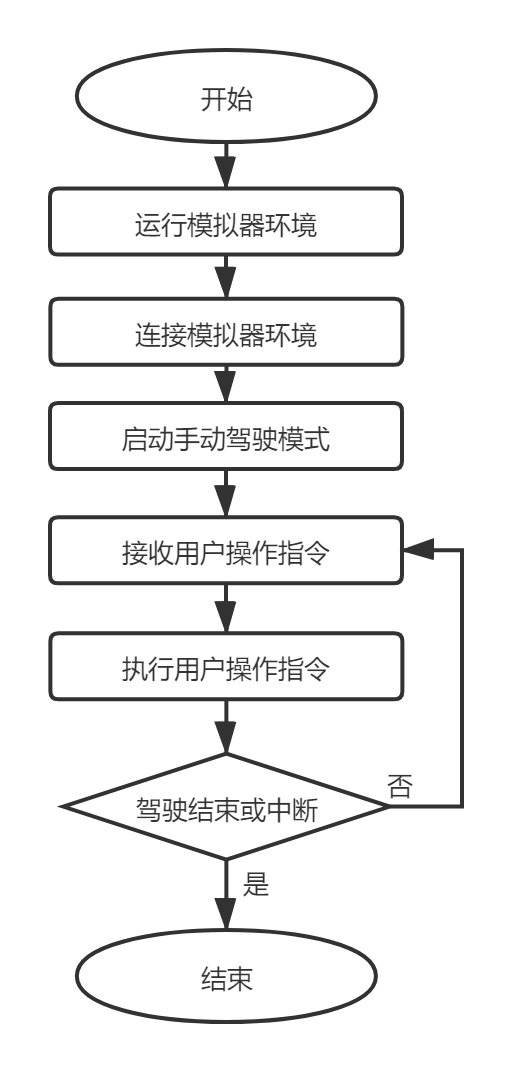


图5.3 无人驾驶系统手动驾驶模块流程图

## 5.4环境感知模块设计

最后是环境感知模块，这个模块主要用于让用户打开深度图像显示与分割图像的显示，具体流程图如图5.4所示。

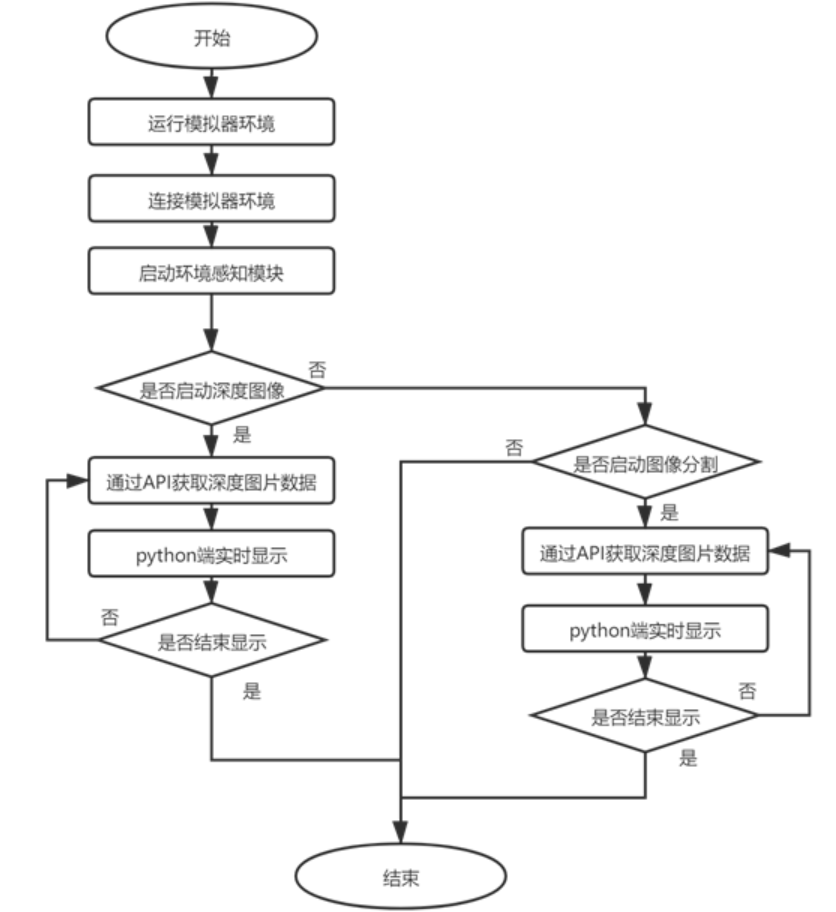


图5.4 无人驾驶系统环境感知模块流程图

## 5.5 本章小结

本章主要对无人驾驶系统中最核心的一些模块的流程进行了设计，主要包括了训练模块、验证模块、手动控制模块和环境感知模块等。在这之中训练模块是本系统最为重要的一个部分，也是在这里体现了本课题研究的几个算法；验证模块用于将训练好的模型应用到驾驶环境中并对其验证，对于一个无人驾驶系统而言，它也是尤为重要的一部分；手动控制模块提供了让用户自主控制车辆进行驾驶的功能；而最后的环境感知模块则是能让用户发觉更多的环境信息。

# 第6章 系统实现

上一章对整个无人驾驶系统及其算法进行了详细的设计，给出了系统重要部分的流程图。在本章中，笔者将对系统中几个重要部分的实现进行说明并对关键代码进行展示。

## 6.1 Gym环境的实现

对于本项目而言，Gym环境是必须的，它能够让多种算法使用统一的接口对模拟器和模拟器中的车辆进行控制和观察。本项目中作者实现的Gym环境除了基本的step、reset等基本gym功能外，还实现了环境信息显示、图像信息显示和按键输入功能，另外本系统的环境感知模块（开启深度图像、语义分割图像）也在这部分代码中实现，具体效果如图6.1、6.2所示。

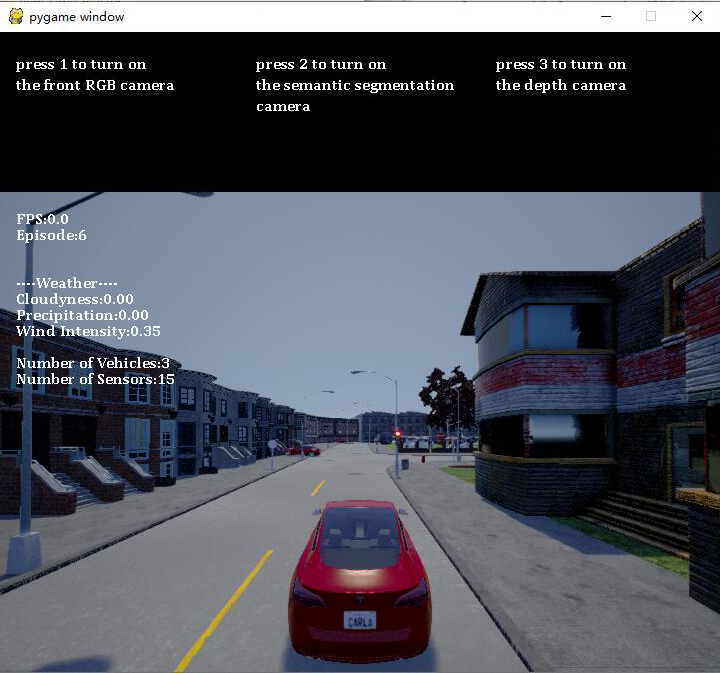


图6.1 基于CARLA的Gym环境



图6.2 启动环境感知模块

完成Gym框架后就能够很方便地将算法接入到模拟器中，只需简单地调用step、reset、render等函数即可，对本框架进行简单调用的代码如下：

env = CarlaEnv(0, render=True, render\_info=True)env.reset()for i in range(MAX\_STEP): env.render(episode=i) obs, reward, done, \_ = env.step(0) if (done): env.reset()

## 6.2 A3C算法实现

A3C本质上就是在A2C的基础上实现了多线程的异步（Asynchronous），每个线程运行自己的worker，定期将自己网络中的参数软更新到Main worker的网络中。A3C的Actor网络结构如图6.3所示。

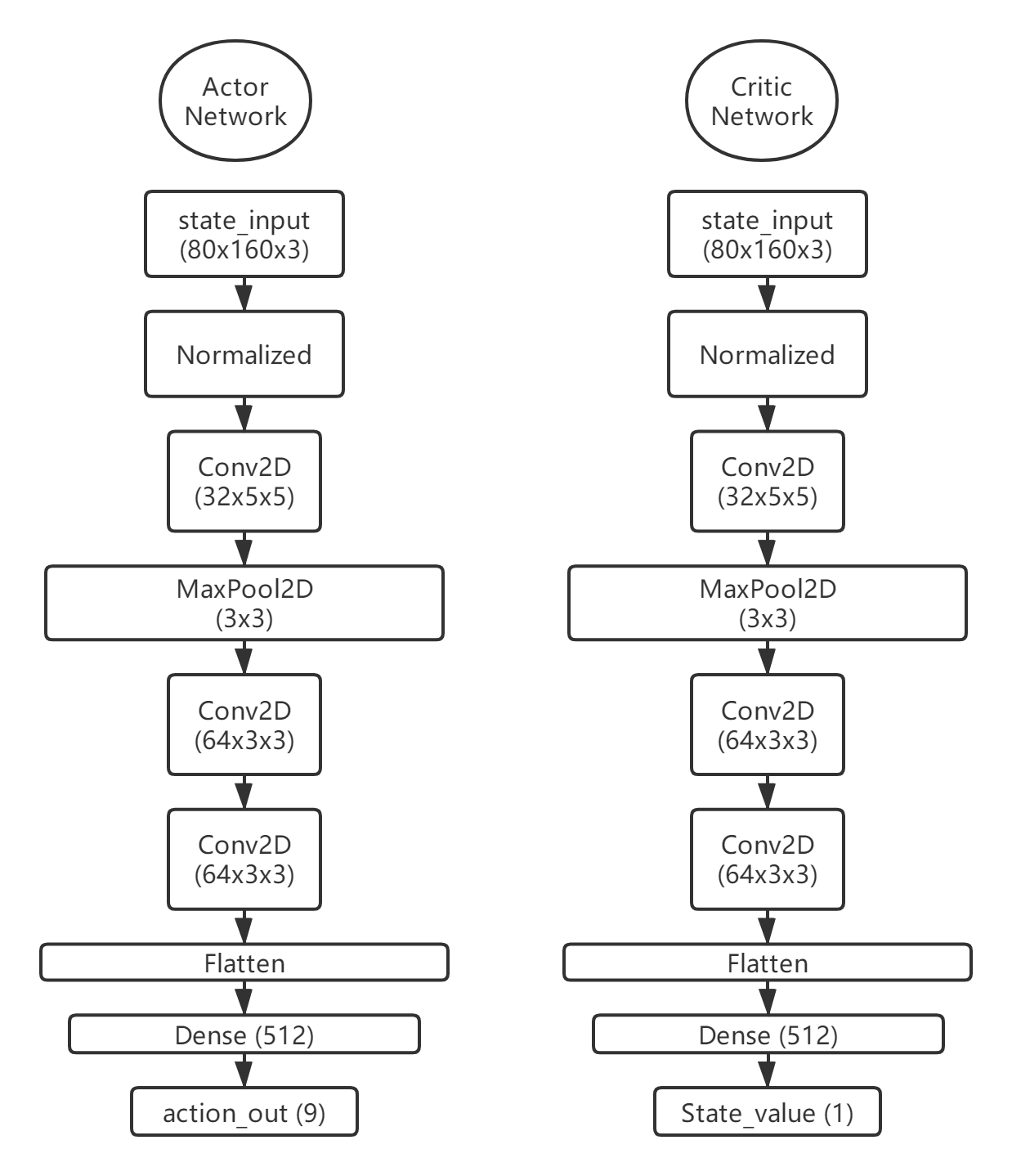


图6.3 A3C中每个worker的网络结构

如上图，在离散的动作空间下，A3C中Actor和Critic的网络是很相似的，Actor网络输出每个动作的概率，Critic网络输出当前状态的V值。但是损失函数有区别，Critic用reward和衰弱后的未来状态值累加得到的Target值直接计算MSE（Mean Square Error），而Actor的损失函数如下：

def actor\_loss(self, y\_true, y\_pred): y\_pred = tf.clip\_by\_value(y\_pred, 0.001, 0.999) log\_prob = tf.log(y\_pred) loss = log\_prob \* y\_true return loss

Actor的目标Target和Critic的目标Advantage的计算过程如下：

advantage = np.zeros(shape=(1, n\_actions))target = np.zeros(shape=(1, 1))current\_value = self.critic\_model.predict(state, batch\_size=1)next\_value = self.critic\_model.predict(new\_state, batch\_size=1)if dead: advantage[0][action] = reward - current\_value target[0][0] = rewardelse: advantage[0][action] = reward + gamma \* next\_value - current\_value target[0][0] = reward + gamma \* next\_value

为了增强A3C的探索性，我们需要加入Epsilon-Greedy算法：

def get\_action(self, state, model): if np.random.rand() <= self.epsilon: return random.randrange(n\_actions) else: action\_output = model.predict(state) return np.random.choice(np.array(range(n\_actions)), size=1, p = action\_output.ravel())

## 6.3 DDPG算法实现

DDPG拥有四个网络，即两个Ctiric网络和两个Actor网络，在连续动作空间的情况下，它们的网络结构如图6.4所示。

DDPG在训练时需要将估计网络中的参数软跟新到目标网络上，具体代码如下：

def update\_target\_net(self, target\_net, evaluate\_net, learning\_rate=0.1): target\_weights = target\_net.get\_weights() evaluate\_weights = evaluate\_net.get\_weights() average\_weights = [(1. - learning\_rate) \* t + learning\_rate \* e for t, e in zip(target\_weights, evaluate\_weights)] target\_net.set\_weights(average\_weights)

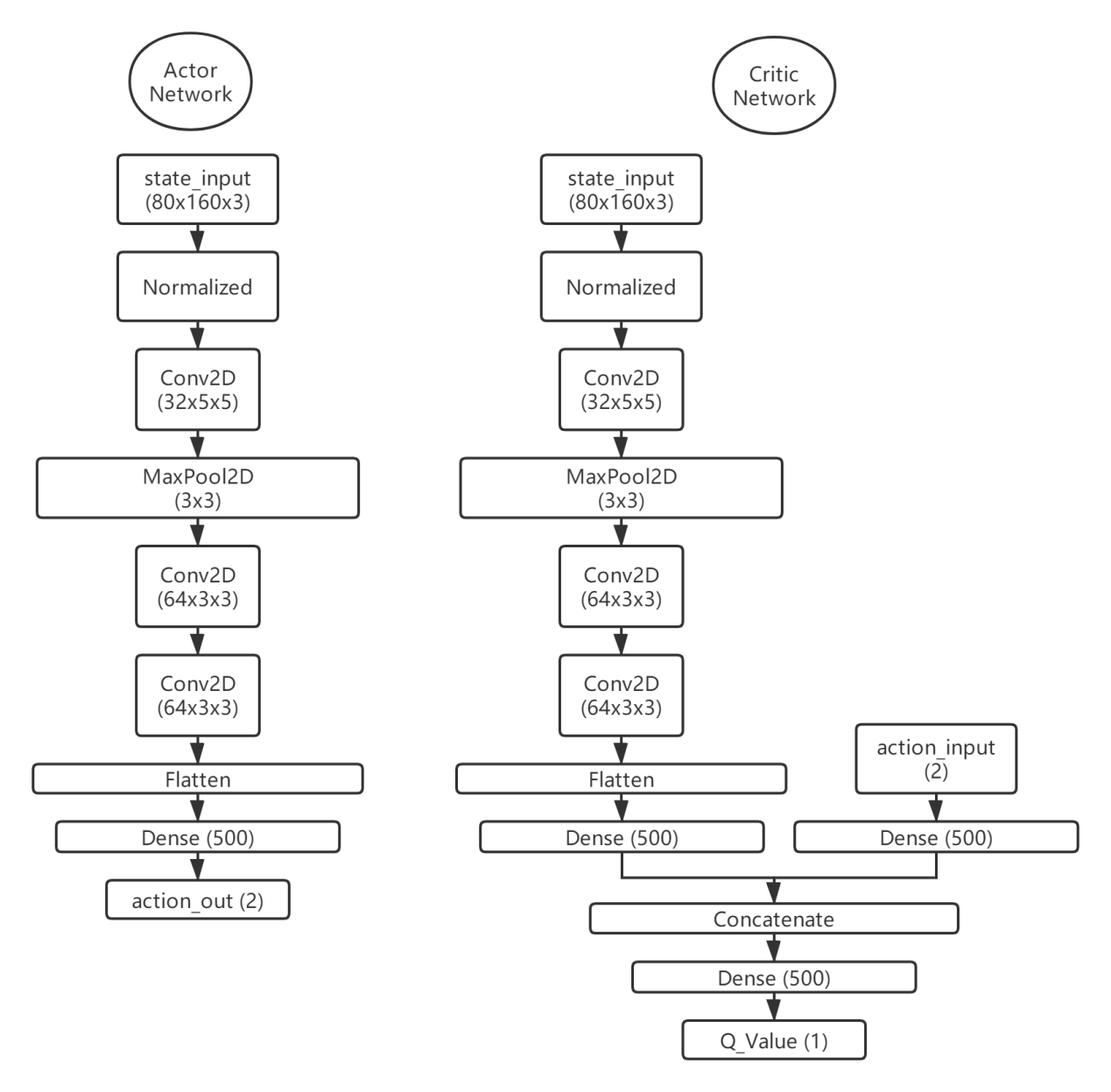


图6.4 DDPG网络结构

DDPG中的Critic估计网络需要用未来奖励作为目标来优化，而Actor则使用Q估计网络的损失值：

target\_actions = self.actor\_target\_net.predict(new\_states)future\_rewards = self.q\_target\_net.predict([new\_states, target\_actions])future\_rewards = future\_rewards.reshape(self.batch\_size, )rewards += self.gamma \* future\_rewards \* (1 - np.array(dones))evaluation = self.q\_evaluate\_net.fit([cur\_states, actions], rewards, verbose=0)predicted\_actions = self.actor\_evaluate\_net.predict(cur\_states)grads = self.sess.run(self.critic\_grads, feed\_dict={ self.critic\_state\_input: cur\_states, self.critic\_action\_input: predicted\_actions})[0]self.sess.run(self.optimize, feed\_dict={ self.actor\_state\_input: cur\_states, self.actor\_critic\_grad: grads})

由于DDPG是确定性策略，面临着探索性低下的问题，为缓解这个问题，我们需要引入Ornstein-Uhlenbeck 随机过程，代码如下：

class OrnsteinUhlenbeckProcess: def \_\_init\_\_(self, mu=0., theta=.15, sigma=1., dt=.01): self.mu = mu self.sigma = sigma self.theta = theta self.dt = dt def \_\_call\_\_(self): W = np.random.normal() self.x += (self.theta \* (self.mu - self.x) \* self.dt + self.sigma \* np.sqrt(self.dt) \* W) return self.x def reset(self, x=0.): self.x = x

通过上述代码中实现的类，我们能够生成稳定且分布均匀的噪声，用于对环境进行探索，下图6.5为通过Ornstein-Uhlenbeck 产生的一组噪声数据。



图6.5 通过Ornstein-Uhlenbeck产生的噪声数据

## 6.4 SAC算法实现

SAC算法采用了双重学习和Target Network方法，一共有五个网络，分别为Actor网络、Q1网络、Q2网络、V估计网络和V目标网络，其中两个Q网络拥有同样的结构，两个V网络也拥有同样的结构。在离散动作空间的情况下，三种网络的结构如图6.6所示。

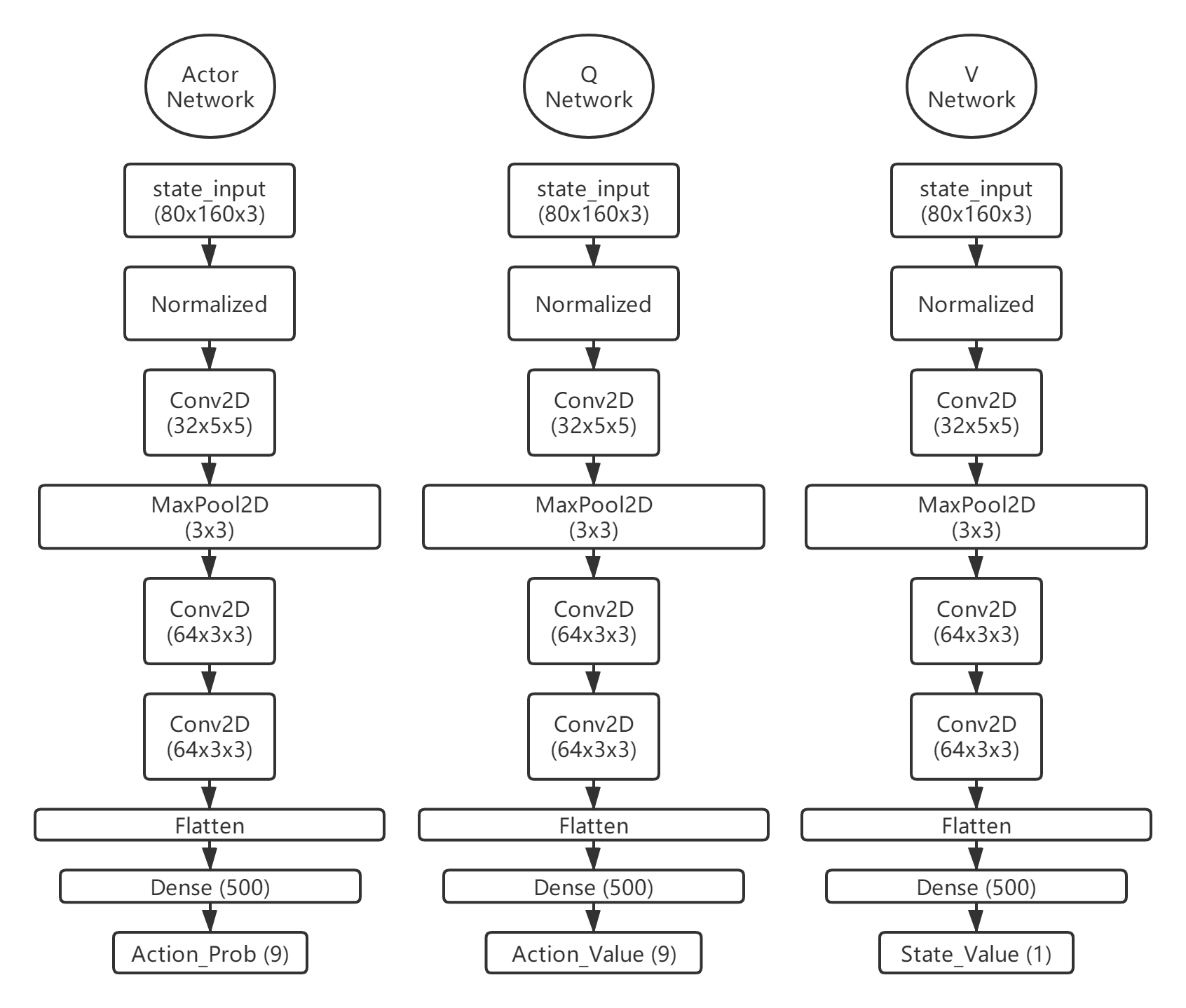


图6.6 SAC网络结构

相对于前文中的A3C和DDPG，SAC由于引入了最大化熵的目标，它在训练过程中始终会保持良好的探索性，因此不需要额外的探索策略，仅对Actor网络输出的动作概率直接进行采样即可，代码如下：

def decide(self, observation): probs = self.actor\_net.predict(observation[np.newaxis])[0] action = np.random.choice(self.action\_n, p=probs) return action

SAC的优化过程比较繁琐，具体参照2.7中的公式和原理分析，此处不再赘述，优化代码如下：

pis = self.actor\_net.predict(observations)q0s = self.q0\_net.predict(observations)q1s = self.q1\_net.predict(observations)# 训练Actorself.actor\_net.fit(observations, q0s, verbose=0)# 训练Vq01s = np.minimum(q0s, q1s)entropic\_q01s = pis \* q01s - self.alpha \* \ scipy.special.xlogy(pis, pis)v\_targets = entropic\_q01s.sum(axis=-1)self.v\_evaluate\_net.fit(observations, v\_targets, verbose=0)# 训练Qnext\_vs = self.v\_target\_net.predict(next\_observations)q\_targets = rewards + self.gamma \* (1. - dones) \* \ next\_vs[:, 0]q0s[range(self.batch\_size), actions] = q\_targetsq1s[range(self.batch\_size), actions] = q\_targetsself.q0\_net.fit(observations, q0s, verbose=0)self.q1\_net.fit(observations, q1s, verbose=0)# 更新目标网络self.update\_target\_net(self.v\_target\_net, self.v\_evaluate\_net, self.target\_learning\_rate)

## 6.5 奖励函数实现

由于本项目的大部分算法测试在Donkey Car Simulator中进行，因此笔者根据这个模拟器的特性和可获取的环境数据定制了其奖励函数，具体代码如下：

def calc\_reward(self, done):

if done:

norm\_throttle = (self.last\_throttle - MIN\_THROTTLE) / (MAX\_THROTTLE - MIN\_THROTTLE)

return REWARD\_CRASH - CRASH\_SPEED\_WEIGHT \* norm\_throttle

throttle\_reward = THROTTLE\_REWARD\_WEIGHT \* (self.last\_throttle / MAX\_THROTTLE)

if self.speed < EXPECT\_SPEED:

speed\_reward = (self.speed - EXPECT\_SPEED) \* LOW\_SPEED\_PUNISH

else:

speed\_reward = (self.speed - EXPECT\_SPEED) \* FAST\_SPEED\_WEIGHT

cte\_punish = - math.fabs(self.cte) / self.max\_cte\_error \* CTE\_PUNISH\_WEIGHT

return 1 + throttle\_reward + speed\_reward + cte\_punish

上述代码将整个奖励函数分为四个部分，分别是基础奖励、终止惩罚、速度奖励以及CTE轨道误差惩罚。基础奖励是一个常数，这让智能体有了继续进行驾驶的动力；终止惩罚是对小车离开车道或发生碰撞时进行惩罚，这里加上了对油门的惩罚，这是在指导小车即使发生了偏离车道或碰撞的事件，这个时候也不应该处于加速状态；速度奖励很好理解，在低于期望速度的时候进行惩罚，达到期望速度时进行奖励；最后的CTE轨道误差惩罚则是以小车与车道的偏离程度为基准，给出一个持续性的惩罚。

本系统的奖励函数的每个部分都有相应的权重因子，可以在setting.py中进行手动调整。

## 6.6 本章小结

本章对无人驾驶系统的具体实现进行了介绍。首先对自定义gym环境进行介绍，它不仅实现了gym框架的基本函数，同时也实现了许多实时的图像信息、环境信息展示功能，在运行的同时也可以监听键盘事件，让用户能够在使用本系统时进行实时交互和信息获取。然后笔者介绍了A3C、DDPG、SAC三种算法的具体实现，并给出了关键部分的代码。

# 第7章 分析与测试

上一章对整个无人驾驶系统及其算法进行了详细的设计与实现，在本章中，笔者将对各个算法进行分析，对比它们的特性，并对本系统进行测试，最后尝试在真实环境的小车上进行测试。

## 7.1 各算法训练结果分析

本节主要对各算法的训练结果进行分析，通过对比阐明每个算法的特性和优缺点。

### 7.1.1 A3C性能分析

由于笔记本电脑的硬件限制和模拟驾驶环境的限制，在最初的测试中，我们只能将A3C的线程数设置为1才能正常运行，也就是相当于变成了单线程的A2C。此次训练过程中reward的表现如图7.1所示。

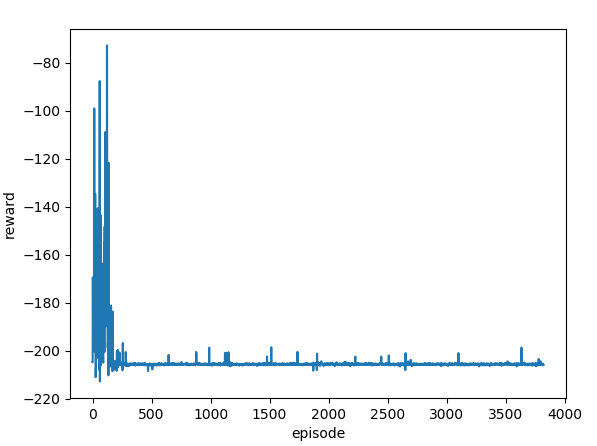


图7.1 A3C中reward随episode的变化趋势

在本次训练中，笔者将参数设置为：衰弱因子gamma为0.99，网络学习率为0.000025，batch size为16。可以看出在单线程的情况下A3C退化为了A2C，自身收敛效果比较差，同时由于无人驾驶的特殊性，样本相关性太大，在前期的探索结束后直接收敛到了局部最优，导致其无法继续优化。

### 7.1.2 DDPG性能分析

由于DDPG的特性，我们使用了连续的动作空间作为输入，通过在模拟器上的训练，我们得到了DDPG在训练过程中的表现情况如图7.1所示。

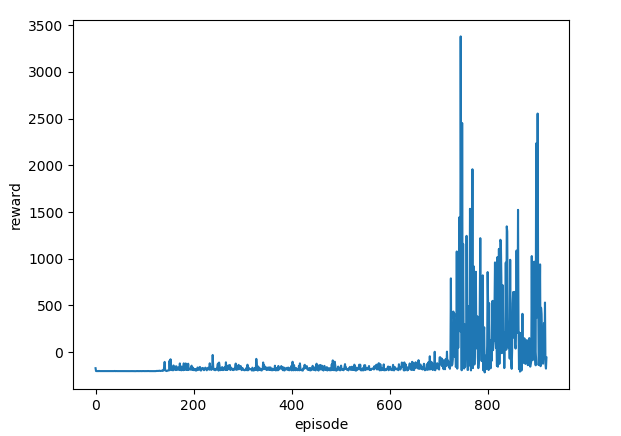


图7.2 DDPG中reward随episode的变化趋势

在本次训练中，笔者采用的参数为：更新方式为单步更新，衰弱因子gamma为0.99，每次学习时的采样数量batch size为8，神经网络学习率learning rate为0.0001。在探索方面，开始学习之前先以纯粹的Ornstein-Uhlenbeck随机策略来获取5000个step的数据，在5000个step之后采取输出动作加Ornstein-Uhlenbeck噪声的形式进行学习。

通过图7.1我们可以发现，DDPG作为一种确定性策略算法，在前面700个episode中还是无法快速收敛的，并且在训练中我也发现它常常会表现出陷入局部最优的现象，比如经常在同一个转弯处连续上百个episode重复一样的出界动作。但是我们也能够从上图中发现它在约750个episode时开始掌握了一些驾驶技巧，但是作为确定性策略，它的表现不稳定的缺点也暴露了出来——在后续的训练中它的reward有较大的波动。

### 7.1.3 SAC性能分析

SAC相对于A3C和DDPG，有着明显更好的收敛性和探索性，且对超参数不是特别敏感，本次训练中笔者将参数设置为：更新方式为单步更新，衰弱因子gamma为0.99，batch size为4，学习率learning rate为0.000025。图7.3和图7.4分别是将熵权重因子alpha设置为0.9和0.3两种情况下的表现：

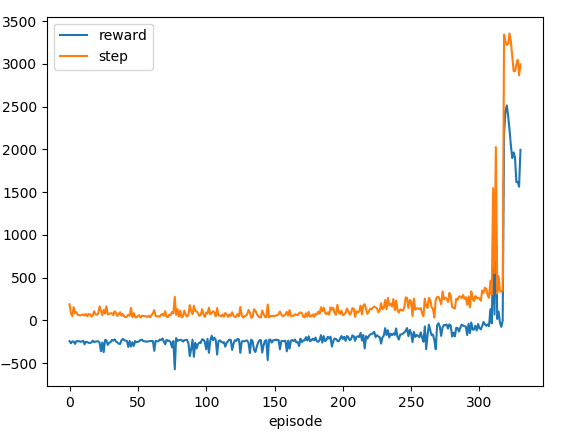


图7.3 SAC在alpha为0.9时reward和step随episode的变化趋势

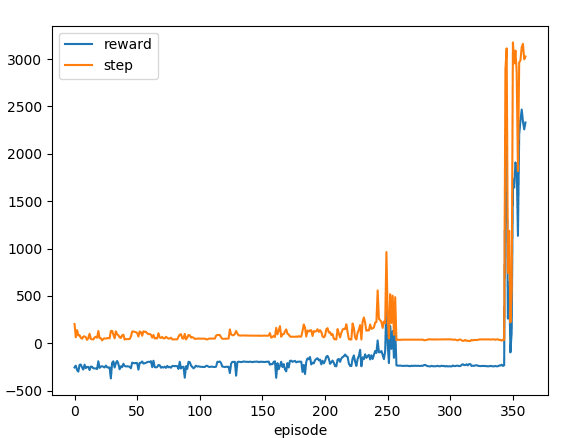


图7.4 SAC在alpha为0.3时reward和step随episode的变化趋势

alpha是最大化熵项的权重因子，它控制了学习过程中熵的重要程度，也就是说设置得越高算法就会越追求输出动作概率的平均，也就具有了更高的探索性，而在设置成较低值时，算法就会将最大化奖励值作为目标，降低一部分的探索能力。根据图7.3和图7.4中的数据，我们可以发现将alpha设置为0.9与0.3在训练时长上并没有太大差异，都在约350个episode时成功学习到了驾驶方法并跑完全程。它们的区别主要在于训练过程中的震荡，alpha设置为0.9时Agent表现出了较高的探索性，整个训练过程中都在不停地进行尝试并产生震荡，而alpha设置为0.3时的整个训练过程就平缓了许多，且经常会出现一些趋于水平的部分，这样的部分意味着Agent正在利用现有策略寻找更优解。

## 7.2 奖励函数对训练的影响

对于强化学习而言，奖励函数是一个非常重要的部分，因为它代表着智能体学习的目标，一个奖励函数的优劣对于无人驾驶的训练会有决定性的作用。但是无人驾驶其实并没有一个非常明确的目标，人类对于驾驶技术的好坏也很难作出一个量化的评判，因此对于无人驾驶问题，奖励函数的设计就很考验设计者对于无人驾驶评判的认知。本项目中的无人驾驶系统对模拟器进行了奖励函数的定制，具体见6.4章节。

为了测试奖励函数的不同对训练过程的影响，笔者选择对奖励函数中CTE（Cross Track Error，轨道交叉误差）的影响因子进行修改，取两个不同的值来分别进行训练。将它们都训练到能够跑完赛道全程后，二者在测试驾驶时的表现如下：

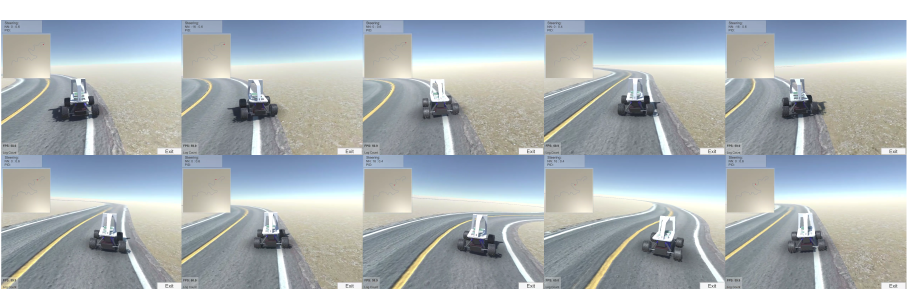


图7.5 CTE影响因子设置为1时的训练结果

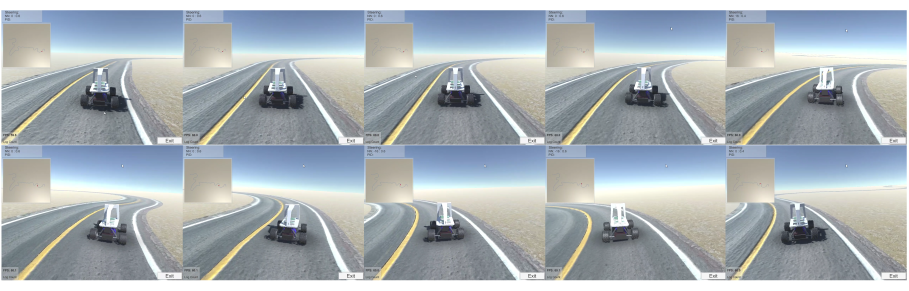


图7.6 CTE影响因子设置为5时的训练结果

从图7.5和7.6的结果来看，很明显当我们把CTE的影响因子设置为5时，训练出的模型在驾驶时会更加地靠近道路中间。这其实很容易理解，当我们把CTE的影响因子设置得较大时，训练过程中如果小车的CTE较大（偏离赛道的程度较大），Agent就会得到一个较大的惩罚，这个惩罚会激励我们的Agent，使它在行驶时更加偏向于处在道路的中间位置。而当我们将CTE的影响因子设置得较小时，训练过程中CTE造成的惩罚就会比较小，相对于其它部分的奖励值或惩罚值就会显得微不足道，这就造成了Agent没有让自己保持在车道中间的动机，从而导致了图7.5的情况——Agent在保证自己不会驶离跑道的前提下，并不会很在意自己是不是处于道路的中间。

在进行上述测试后，笔者又使用这两个模型在同一个赛道上进行了一次测试，并记录了它们在行驶中产生的速度值与CTE值，如下图：

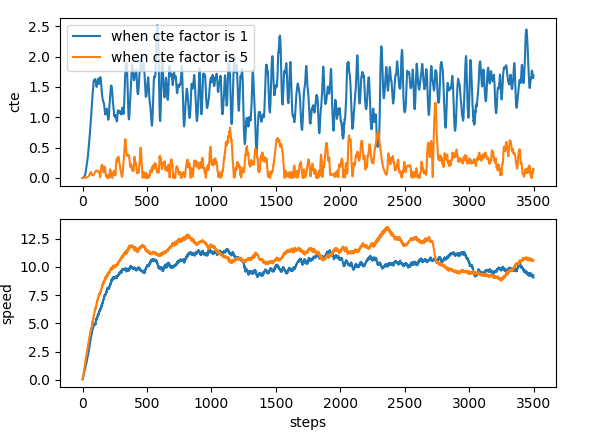


图7.7 不同CTE影响因子下训练的模型测试时CTE和速度的差异

图7.7中CTE值随着step的变化情况也佐证了前文的观点，在CTE影响因子设置为5时，训练出来的模型在驾驶时明显会保持着较低的CTE，两者的CTE平均值分别为0.24和1.41。而从第二张表我们能够发现，CTE影响因子为5时训练出的模型在大部分时候的速度都比CTE影响因子为1时训练出的模型更快一些，二者的平均值分别为10.79和9.95。这是因为只有当小车在路中间行驶时才能有余力去加速，如果一直处于驶离道路的边缘的话就不得不以较低的速度行驶。

得到上述测试结果后，我们还可以对训练过程中产生的数据进行分析。通过对比训练产生的数据，我们可以发现将CTE影响因子设置为5相对于设置为1而言还有其他的优势，如下图：

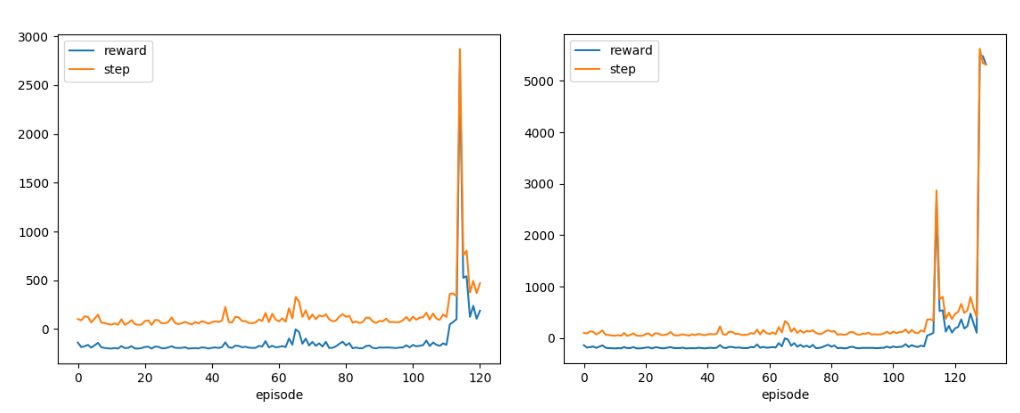


图7.8 CTE影响因子设置为1时的训练数据

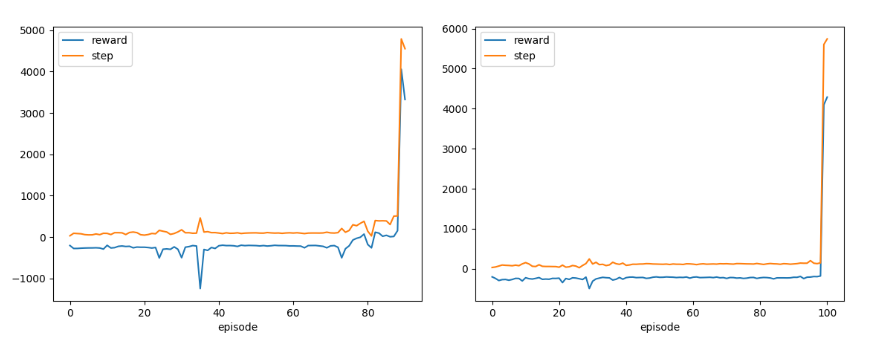


图7.9 CTE影响因子设置为5时的训练数据

为了能够快速地得到实验结果，本次实验使用了实验室中搭载了GTX1080Ti显卡的高配置计算机进行了训练，并将batch size调到了32。从图中数据我们也能看出训练的速度非常的快，只要100个左右的episode就可以让Agent学习到能够驾驶完赛道全程的程度。同时为了实验的严谨性，笔者也对这两种情况进行了多次测试，这里取每种情况各两张图片来进行对比。

从图7.8和图7.9中的训练数据我们能够发现，当CTE影响因子设置为5时，Agent训练至能够驾驶完赛道全程所使用的episode会更少，也就是说它会更快地学会驾驶策略。根据实验结果统计，相对于把CTE影响因子设置为1的情况，设置为5时只需要花费前者77.41%的时间就能够学习到相同的程度，并且其训练出的Agent在驾驶时会尽量把小车保持在赛道中心，在速度上也有略微的提升。

## 7.3 真实环境下的驾驶测试

为了测试算法模型在真实环境下的驾驶能力，需要将整个无人驾驶系统移植到真实环境的Donkey Car小车上进行测试。目前已经对Donkey Car小车完成了安装工作，其外观如图7.10所示，它主要由一块Raspberry Pi 4B、一块PWR.A53拓展板、摄像头、可充电电池、两个马达、一个舵机和一些其他零配件组装而成，全部零件如图7.11所示。其中，Raspberry Pi 4B为它的计算核心，用于运行无人驾驶系统的程序和一些辅助应用，PWR.A53拓展板用于输出控制信号和电源管理，摄像头用于图像数据采集，电池保证其能够在没有有线电源的情况下运行，两个马达用于驱动小车前进或后退，舵机配合零配件实现小车的左右转弯。

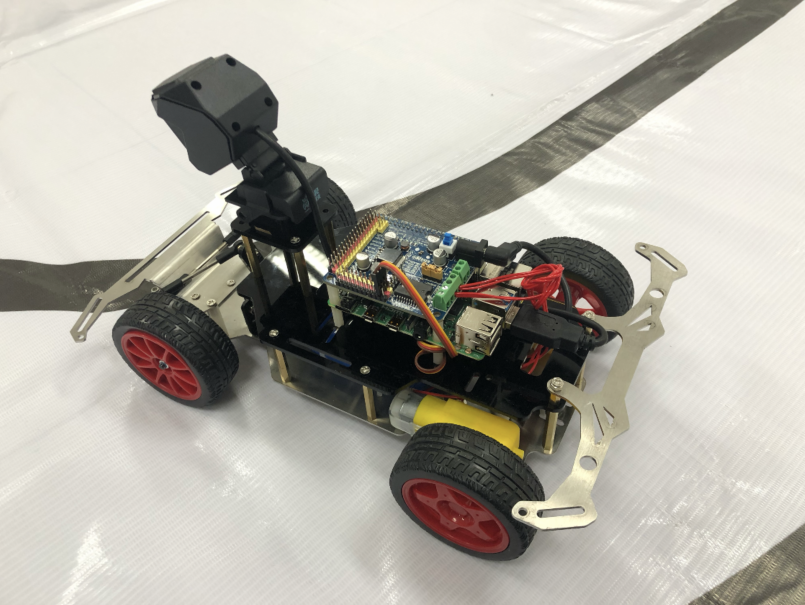


图7.10 Donkey Car小车完成品



图7.11 Donkey Car小车零部件



图7.12 Donkey Car小车赛道

通过简单的配置并将开源的Donkey Car项目拷贝到树莓派上之后，成功实现了通过手柄和键盘的手动控制功能。而要在真实小车上实现无人驾驶，就需要将算法和模型移植到Donkey Car小车搭载的树莓派上，但是考虑到树莓派的算力不足以支撑训练，直接用小车来进行训练并不现实，那么就需要一个能够直接将模型应用到小车上的方法。这里我们可以使用(Conditional Generative Adversarial Network)[31-34]的技术来将输入图像转换成语义图像，实现虚拟到现实的转换[35]，只要从模拟器得到的图像和真是环境能够转换成一致的语义图像，我们的DRL模型就能够在模拟器与真实环境二者中正常运行。

另外，考虑到树莓派的算力低下，我们可以使用VAE(Variational Autoencoder)[36-37]技术来将语义图转换成特征向量，再输入到算法模型中进行计算，这个方法也能够让DRL模型在训练时避免图片卷积，大幅提升训练速度，但是如果VAE的模型本身没有被训练完善，则会导致DRL模型无法训练至最优。

## 7.4 本章小结

本章对整个无人驾驶系统进行测试，对几种算法的性能进行了分析与总结，也给出了不同情况与参数下的训练差异以及各算法的特性。最后笔者实现了Donkey Car小车的安装并初步实现了对它的运行和控制。

# 第8章 总结与展望

上一章对整个无人驾驶系统及其算法进行了测试与分析，在本章中，笔者将对整个课题进行总结，并提出本系统的不足和改进方向。

## 8.1 总结

本论文首先介绍了无人驾驶的发展史，并对本论文研究的核心——深度强化学习算法的各部分进行了研究与探讨，最终依靠DRL算法在模拟器上基本实现了L4级别的驾驶，并对各算法的性能与优缺点做了总结。最后本文也初步完成了真实环境中的车辆控制，并提出将无人驾驶系统移植到真实环境中运行的方案。

本文的主要贡献如下：

1. 实现了较为完善的无人驾驶系统，具备完善的功能与算法模型的兼容能力。定义了无人驾驶系统的基本流程，为后续研究打下了基础。
2. 对多种DRL算法进行了研究，证实了它们的可行性，并对它们总结了特点和在不同情况下的表现。实验证明对于无人驾驶这个问题而言，SAC相比其他算法能够更快地进行收敛且能够表现出稳定的效果。
3. 分析了在无人驾驶任务中，奖励函数中的CTE影响因子、SAC的最大化熵项的权重因子alpha等参数的差异对于训练过程与训练结果所造成的影响。
4. 研究了将模拟器中训练得到的模型移植到真实环境中并直接使用的方案，即在系统的环境感知模块通过CGAN将输入图像转换成予以分割图像，以屏蔽模拟器环境与真实环境的图像差异，使算法模型能够通用。
5. 探讨了使用VAE将输入图像转换为特征向量以提高训练效率和真实环境中验证效率的方案，即在训练DRL模型之前先训练一组VAE以实现输入图像到特征向量的双向映射，使训练DRL模型时能够避免图像卷积。

## 8.2 展望

随着人工智能的发展，无人驾驶已经处于大众关注的焦点，近几年无人驾驶的发展也非常迅猛，许多研究机构和公司都相继推出了自己的成果。但是要真正实现L4级以上的无人驾驶，其实还是有许多需要做的工作和需要解决的问题。

1. 模拟环境和真实环境的差别，真实环境中天气、光线、场景的差别，这些差别都会导致图片信息的大幅度改变，正因如此，引入CGAN等技术来产生语义分割图像就变得非常有必要。因为这样能够帮助算法模型屏蔽掉这些信息的差异，只提取必要的信息并且对它们进行统一化，就能够让模型具有更强的普适性和鲁棒性。
2. 无人驾驶问题很难确定一个明确且可量化的目标，因此奖励函数如果仅凭人类的经验来确定其实未必是好的，甚至可能会在人类意想不到的情况下发生巨大的错误。而通过逆强化学习[38-40](Inverse Reinforcement Learning)来学习人类的行为，以得到一个更合理奖励函数将是一个很好的方法。
3. 深度神经网络目前对于人类而言还是有太多的未知和无法解释的地方，人类无法仅从验证结果来断定一个模型是否完美，因为神经网络的参数实在太过庞大（特别是涉及图片运算时），我们很难知道一个模型是否会在什么时候识别出错误的特征。事实上学术界也已经有很多学者对神经网络的一些弱点进行了研究[41-42]。而在无人驾驶中，核心决策算法是否会被攻击是非常关键的一点，因为在驾驶过程中如果出现决策错误将会是灾难性的，甚至可能会对乘客造成生命危险。因此现在的无人驾驶算法也应该针对模型是否会被攻击的问题进行考虑。
4. 为了提升训练效率，减少运算单元的负担，前文7.3中提到的VAE可以对无人驾驶系统进行一个很好的优化，对图片进行特征提取可以避免神经网络的卷积运算，甚至对于数据存储而言，使用VAE也能够节省非常多的空间。
5. 本文中实现的无人驾驶系统是基于端到端的，仅通过摄像头得到的图片来进行决策，实际环境中我们应该考虑更多方面的传感器，并且要考虑当部分传感器失效时的决策，对于无人驾驶而言，这也是尤为重要的一个课题。

本文对于深度强化学习算法和无人驾驶进行了研究与探讨，也提出了一些问题，相信在人工智能领域飞速发展的现在，世界各地的研究者们将会解决这些问题，甚至提出新的更完善、更适合无人驾驶的解决方案。

# 参考文献

[1] 张茜.无人驾驶系统的设计与实现[J].电子技术与软件工程 . 2019

[2] 张宇航.无人驾驶汽车的起源、现状及展望[J].电子技术与软件工程. 2017

[3] LeCun Y, Boser B, Denker J S, et al. Backpropagation applied to handwritten zip code recognition[J]. Neural computation.1989.

[4]万里鹏,兰旭光,张翰博,郑南宁.深度强化学习理论及其应用综述[J].模式识别与人工智能,32(01):67-81.2019.

[5]赵星宇,丁世飞.深度强化学习研究综述[J].计算机科学,2018,45(07):1-6

[6]V. Mnih et al., Playing Atari with Deep Reinforcement Learning[J]，arXiv preprint arXiv:1312.5602, 2013.

[7]V. Mnih et al.,Human-level control through deep reinforcement learning[J],Nature, 2015.

[8]Kingma, D.P., Ba, J. Adam: a method for stochastic optimization[J]. CoRR abs/1412.6980. 2014.

[9]Howard R.A.Dynamic Programming and Markov Processes [M]. MIT Press, Cambridge, MA.1960.

[10]郭宪,方勇纯.深入浅出强化学习:原理入门[Ｍ].电子工业出版社,2018.

[11]Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.

[12]基于值函数和策略梯度的深度强化学习综述[J]. 刘建伟,高峰,罗雄麟.计算机学报. 2019.

[13]D. Silver, T. Hubert, J. Schrittwieser et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play[J]. Science, 362(6419):1140–1144, 2018.

[14]Silver David, et al. "Mastering the game of go without human knowledge[J]" Nature 550.7676: 354-359. 2017.

[15]唐振韬, 邵坤, 赵冬斌. 深度强化学习进展:从AlphaGo到 AlphaGo Zero.控制理论与应用[J].pp1529-1546, 2017.

[16] 陈兴国, 俞扬. 强化学习及其在电脑围棋中的应用 [J]. 自动化学报, 42(05):685-695. 2016.

[17]Christopher J. C. H. Watkins, Peter Dayan, Q-learning , Machine Learning[M], Volume 8, Issue 3–4, pp. 279–292. May 1992.

[18]Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning[J]" Nature 518.7540: 529-533. Feb 2015.

[19]K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image

Recognition[C]" 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, pp. 770-778. 2016.

[20]John S, Filip W, Prafulla D, et al. Proximal Policy Optimization Algorithms[J]. arXiv preprint arXiv: 1707.06347, 2017.

[21]Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J].arXiv preprint arXiv:1509.02971, 2015.

[22]Volodymyr M, Adrià Puigdomènech B, Mehdi M et al. Asynchronous Methods for Deep Reinforcement Learning[J]. arXiv preprint arXiv: 1602.01783. 16.

[23]Tuomas H, Aurick Z et al. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor[J]. arXiv preprint arXiv: 1801.01290, 2018.

[24]Tuomas Haarnoja, Sergey Levine,et al. Soft Actor-Critic Algorithms and Applications[J]. arXiv preprint.2018.

[25]G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym[J]. arXiv preprint.2016.

[26]Dosovitskiy A, Ros G, Codevilla F, et al. CARLA: An open urban driving simulator[J]. arXiv preprint arXiv:1711.03938, 2017.

[27]B. Wymann, E. Espie, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. TORCS, The Open Racing Car Simulator[DB/OL]. available: http://www.torcs.org, 2014.  
[28]W. Roscoe, Donkey Car: An opensource DIY self driving platform for small scale cars, [DB/OL] Available: http://www.donkeycar.com.2016.

[29]Q. Zhang and T. Du. Self-driving scale car trained by deep reinforcement learning[J]. arXiv preprint arXiv:1909.03467, 2019.

[30]Yu, F. Train donkey car in unity simulator with reinforcement learning [DB/OL]. Available: https://flyyufelix.github.io/2018/09/11/donkey-rl-simulation.html .2018.

[31]陈文兵,管正雄,陈允杰.基于生成式对抗神经网络的数据增强方法[J/OL].计算用:1-8. 2019.12.

[32]陆萍,董虎胜.基于条件生成对抗网络的图像生成[J].甘肃科技纵横:1-3.2019

[33]贾玉福,胡胜红,刘文平,王超,向书成.使用条件生成对抗网络的自然图像增强方法[J].南京师大学报(自然科学版):88-95.2019.

[34][Mehdi Mirza](https://arxiv.org/search/cs?searchtype=author&query=Mirza,+M), [Simon Osindero](https://arxiv.org/search/cs?searchtype=author&query=Osindero,+S). Conditional Generative Adversarial Nets[C] . arXiv preprint arXiv:1411.1784. 2014.

[35]Pan Xinlei, et al. Virtual to real reinforcement learning for autonomous driving[J]. arXiv preprint arXiv:1704.03952. 2017.

[36]Kingma D P, Welling M. Auto-Encoding Variational Bayes[J]. stat, 2014.

[37]DOERSCH C. Tutorial on Variational Autoencoders[J]. stat, 2016.

[38]Ng, A. Y., & Russell, S. Algorithms for inverse reinforcement learning[J]. Proc. ICML,2000.

[39]Ziebart, BD, Maas, A, Bagnell, JA, Dey, AK. Maximum entropy inverse reinforcement learning[J]. AAAI,2008.

[40]Sharifzadeh, Sahand, et al. Learning to drive using Inverse Reinforcement Learning and Deep Q-Networks[C]. NIPS workshop on Deep Learning for Action and Interaction, Dec 2016.

[41] J. Su, D. V. Vargas, S. Kouichi, One pixel attack for fooling deep neural networks,[DB/OL]. Available: [https://arxiv.org/abs/ 1710.08864](https://arxiv.org/abs/%201710.08864). 2017.

[42] Akhtar N, Mian A .Threat of adversarial attacks on deep learning in computer vision: a survey[J]. [arXiv:1801.00553](http://arxiv.org/abs/1801.00553), 2018.

# 致 谢

四年的大学生活马上就要结束，对于我而言，在金陵科技学院的四年是充满回忆的四年，我学到了很多，悟到了很多，但最重要的，是我确信了自己对计算机科学的热爱与执着，确信了自己会在计算机科学的道路上越走越远。

一路走来，我感慨万千，我要感谢那些带领我、指引我的人们，而在他们之中我首先应该感谢的便是我的论文指导老师——龚如宾老师，是他帮我打开了机器学习的大门。在我还对机器学习一无所知时，他带我们去空教室为我们讲解论文中需要用到的技术，每当我对论文内容抱有疑问时，他也会在办公室和我一起研读论文，耐心的为我讲解每一个知识点。在这个过程中，我不仅学到了机器学习领域的知识，更学到了龚老师严谨认真的思维方式。

此外，我也非常感谢在软件研究院指导我完成多次项目研发的徐尚瑜老师，他既是我的良师，也是我的益友，在每次项目研发的过程中他都会耐心地指导我们，有时也会带我们一起阅读外文文献，逐句地翻译并解释给我们听。而对于项目中的硬件需求他也是有求必应，帮助我们申请科研经费，最后还协助我们取得了两个项目的软件著作权。在研发室做项目的日子，我们经常会在研发室做项目待到很晚，每当这时候，我们就会看到办公室里孤身一人的徐老师，他的勤劳刻苦感染了我，让我也能够在夜晚空荡荡的教室里静下心来，努力奋斗。

另外，我也很感谢学校给予我在卓越班学习的机会，软件研究院的老师们细心、耐心地教导我们，监督我们，让我们变得自律，在很多方面也提供了非常多的帮助。在卓越班的两年不仅让我积累了庞大的知识量，更磨炼了我的意志。而卓越班的同学们也都十分优秀和有趣，为我的生活增添了许多乐趣，陪伴我度过了美好的大学生活。

最后，我也特别感谢我的家人，感谢你们在物质和精神上对我的支持，感谢你们愿意在很多时候倾听我的诉求和想法。

学无止境，在大学生活的终点，我会带着龚老师带给我的严谨、徐老师带给我的努力、卓越班带给我的自律以及我心中对计算机科学的热爱，向着新的目标起航。

# 附 录

**王家骅本科阶段成果**

**获奖情况**

2019.11 金陵科技学院第三节程序设计大赛 优秀奖

2019.07 卓越贡献奖 特等奖

2019.07 卓越晋级奖

2019.03 蓝桥杯省赛 二等奖

2019.01 卓越贡献奖 特等奖

2019.01 卓越晋级奖

2018.12 蓝桥杯校选赛 一等奖

2018.04 中国工程机器人大赛暨国际公开赛 二等奖

2018.03 优秀地铁志愿者

2017.04 “吾曰思程杯”程序设计大赛 优胜奖

**软件著作权**

软件名称：Spider IoT通用物联网管理平台

著作权人：金陵科技学院

申请人：王家骅、李科辰、顾宇庆、张皓玥、赵泽亮

登记号：2020SR0267710

日 期：2020.03

软件名称：沃基室内安全监测系统

著作权人：金陵科技学院

申请人：李科辰、王家骅、赵泽亮、王鹏、印沛锜

登记号：2019SR0711107

日 期：2019.07