机器学习(进阶)

《四轴飞行器学会飞行》

2018-12-10

吕金波

目录

1. 定义

a. 项目概述

四轴飞行器无论在个人还是专业应用领域都变得越来越热门。它易于操作，并广发英语于各个领域，从最后一公里投递到电影摄影，从搜救到极限运动拍摄，都有着四轴飞行器的影子。

虽然四轴飞行器已经很易于操作了，但还是存在一定的复杂性。所以，这样一个问题诞生了：能不能通过一定方法让它自己飞行，从而解放人力。

随着AlphaGo的带来的世界轰动，强化学习也一跃成为明星算法中的明星。人们很自然的想到，利用强化学习来训练一个智能体，让智能体来操作飞行器便可以完美解决问题。

简单的马尔可夫模型对这种大型状态场来说，效果并不好，随着对算法不断的优化，DQN算法取得了显著的进步。

DQN算法（Mnih等人，2015）能够在许多使用未处理像素输入的视频游戏中进行人类级别的表现。DQN是构建深度神经网络函数逼近器用于估计动作值函数。然而它的局限性非常明显：在高维度，连续动作空间上表现的不好。关于机器人控制方面，关于从原始感知数据输入中学到策略是一个长期的挑战。

教会四轴飞行器飞行》项目的动作空间是连续、高维度的，所以用朴素的DQN是没办法有效的进行探索，因此，在这种环境中，想要成功的训练一个DQN网络是几乎不可能的。但我们从[Contro With Deep Reinforcement Learning](https://arxiv.org/pdf/1509.02971.pdf)一文中介绍了解决问题的方法，这个方法是基于“DQN”，和策略梯度算法的。它的亮点是：

1.从缓存里训练离线样本，来优化样本之间的相关隐变量。

2.在Q网络中利用时间分差法从离线数据中训练target-Q。

b. 问题描述

本次项目要解决的问题，是训练一个DDPG智能体，使得飞行器可以自动完成一些控制行为，任务主要有起飞，着陆，悬浮。

c. 评价指标

对于强化学习来说，评价的指标主要是total_rewards是否收敛，以及通过模拟器来观察agent是否完成任务来评价模型。

2.分析

a.算法和技术

i.DQN

对于连续动作空间上的强化学习任务,我们最先想到的就是 DQN 模型,但是,在本项目中,飞机的状态有 7 个维度,动作有 6 个,也就是说动作空间将无比巨大,在这种情况下,DQN 的训练无法取得很好的效果
而 DeepMind 2015 年提出的 DDPG 可以很好的解决这个问题,所以,本文采取了 DDPG 模型

ii.Actor-Critic

Actor Critic 是将 Policy Gradient 和 Q-Learning 等以值为基础的算法组合。
Actor 是一个 Policy Gradient 模型,以 S 为输入,神经网络输出 actions,在通过策略概率模型,选取和是的 action
Critic 是一个 Q-learning 模型,将 Actor 得到的 action 通过与环境交互,的到新的 s',奖励 r,将 s'作为神经网络的输入,的到 v',而原来的 s 通过 v'由动态规划得到 v。
通过 TD 算法,更新 Actor 网络的参数,实现更新

iii. Buffer-replay

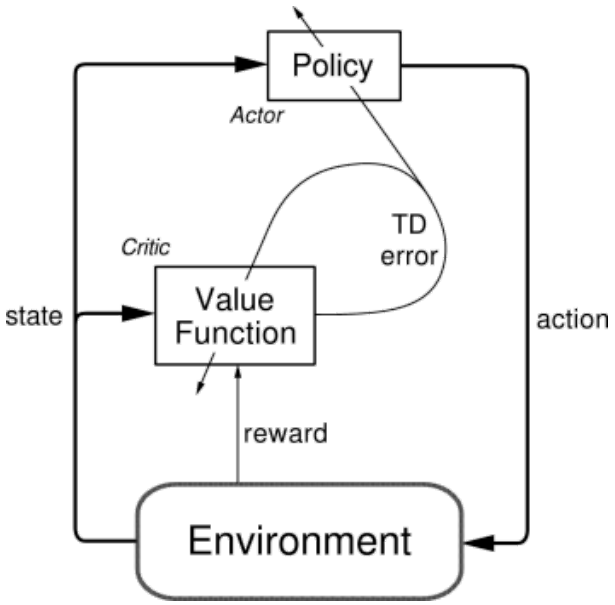
DDPG 主体部分的更新,是更新 Actor-Critic 模型的的参数,这里用到的 Buffer-replay 主要是将没个是 step 的(state, action, reward, next_state)储存,与用 TD 算法更新参数

iv.Noise

我们将使用一个具有所需特性的特定噪点流程，称之为 Ornstein–Uhlenbeck 流程。它会根据高斯（正态）分布生成随机样本，但是每个样本都会影响到后续样本，使两个连续样本很接近。因此本质上是马尔可夫流程。

v.算法设计总结

算法核心思路：DDPG



- 由指定特定策略来持续动作者的参数化，监督者用贝尔曼方程来Q-learning
- 用一个缓存，通过在缓存里取样的优化来更新actor and critic
- 用“soft-update”来更新目标函数。以求增加模型的stable

1.Create a copy of the actor and critic networks,Q'(s, a |Q') and '(s|') respectively, that are used for calculating the target values.

2.update the weights of these target networks slowly。

$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau << 1$

-训练策略时增加噪音，来增加鲁棒性

算法步骤：

- 设置Actor：loss为参数化的策略（状态到动作的映射),Input 为飞行器当前state
- 设置Critic：loss 为参数化的Q-value，Input为飞行器当前state和所采取action

- 设置RelpayBuffer 其作用为将每个step得到的（state,action,reward,next_state）储存，用于随机取样更新目标逼近器

- 完善DDPG主体： Actor，Critic都有是一个local用于训练参数，一个target作为label用于提供固定参照又时间分差法来更新local模型。target是由Buffer中的变量对预测而来。

- 噪音部分：将actions中加入噪音增加其鲁棒性

- 统计部分：编写统计函数，和可视化部分，评估模型

3.具体方法

a.状态空间和动作空间分析

关于强化学习，我们用马尔可夫决策模型解决问题，这便涉及到了状态空间S，动作空间A和一个初始的状态分布P。我们利用飞行器从环境中反馈的状态来由策略来得到动作，状态s包括7个：空间的（x,y,z）坐标，和四元数：

` state = np.array([
        pose.position.x, pose.position.y, pose.position.z,
        pose.orientation.x, pose.orientation.y, pose.orientation.z, pose.orientation.w])`

首先，需要明确我们的状态，也就是无人机每个时刻的姿态是什么：

若通过规则来实现操控无人机，我们的主要作业便是控制四个电机的转速来控制无人机的升力，从而控制飞行器的上升，下降悬浮的等，即将无人机看做一个质点，仅考虑他在（X，Y,Z）坐标系中的运动，而不考虑自身变换

而垂直运动时四个电机输出匹配，，仰俯运动，偏航运动等，则需要更复杂的微分方程来实现力的旋转，即四元数法，所以需要考虑的状态是：坐标系+四元数。

同时，上面也提到了，若是只考虑飞行器的升降及悬浮，我们便不需要引入四元数增加运算参数及运算量，故，在我们的项目中，主要考虑的便是(X,Y,Z)三个方向的状态（速度和力）

动作是由3个线性推力和三个扭矩组成：

`action = [force_x, force_y, force_z, torque_x, torque_y, torque_z]`

我们的状态空间是模拟器box，并且飞行器是在box中学习飞行的，所以，本次任务拥有连续的动作域。

4.任务

对于几个 task,重点在于 reward 的设置:
Task1:想让无人机飞起来是比较容易的,无人机只有受到正向的力且大于重力他才会起飞,而负向力不做移动(当然只是在模拟器中),我们的 reward 设置便可以根据这一点,只要达到目标高度,便基于高额回报。
Task2,Task3:这两个 Task 放到一起说,是因为他俩的思路是一致的,与起飞不同的地方是在于,悬浮我们不希望他在控制某一高度乱飞,降落也不希望
他以最大速度冲刺下来
对于 Task2 要做的便是限制他的活动范围和速度范围(理想状态当然是保持在目标高度,速度为 0),所以,我们要设置一个容错范围,规定时间内超过这个范围,给予惩罚
对于 Task3 要做的

参数：

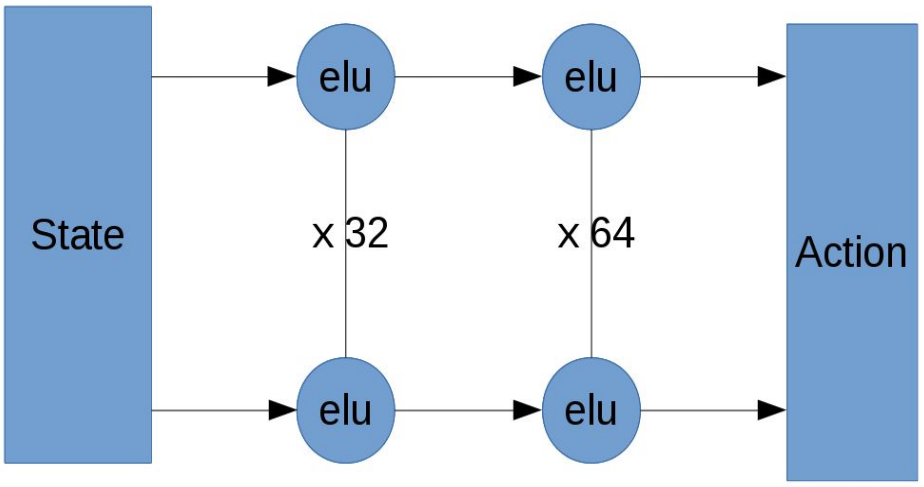
rl的设定就是借鉴Contro With Deep Reinforcement Learning第七部分的调参经验，actor的rl设置为0.0001,Critic的rl设置为0.001，并且对Critic进行l2正则化

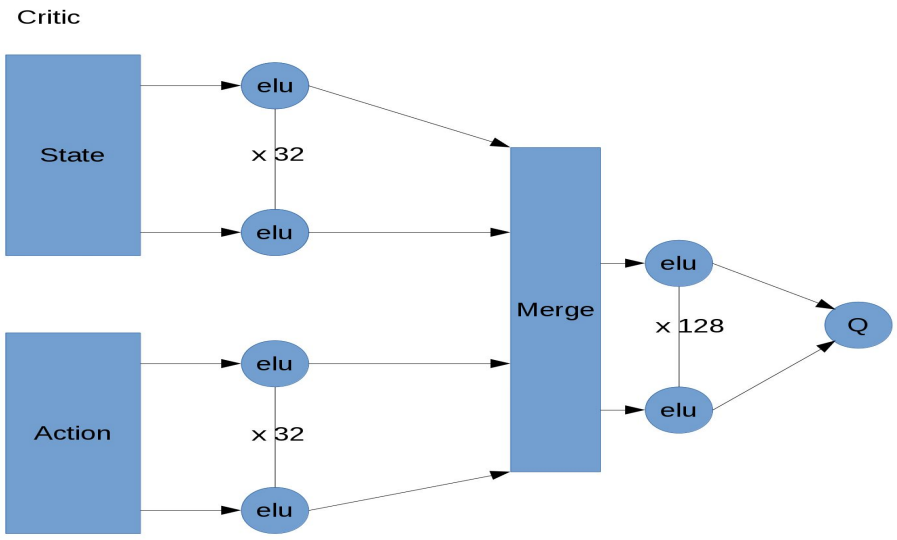
2.因为我们的项目更看中长期回报，所以gamma设为0.99

3.tua是更新权重时local_weights和targe_weights的比率参数，，这样做的好处是用更温和的方法增加权重，来取代直接增加。tua越大，权重越偏倚local_weights，更新的幅度越大。我试了，0.01, 0.005, 0.001

Take off model:

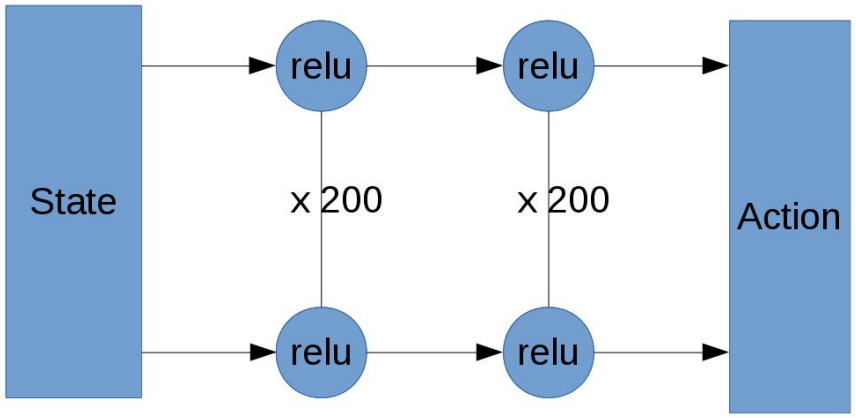
Actor



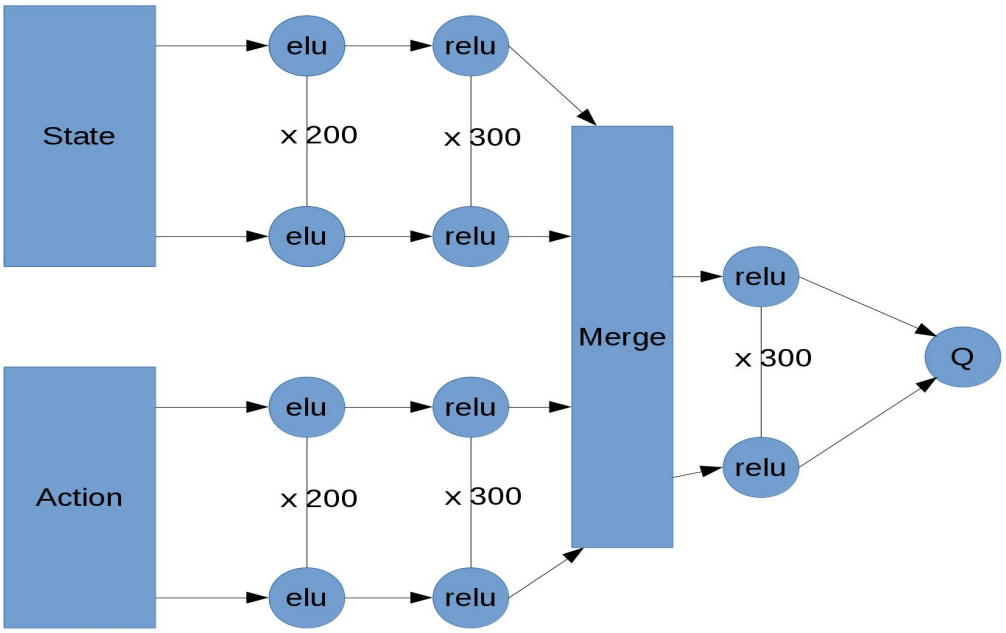


Hover and Landing Model

Actor



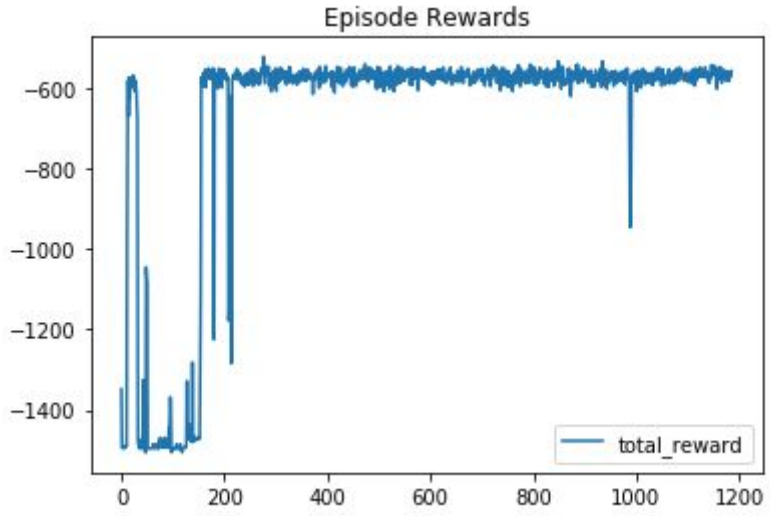
Critic



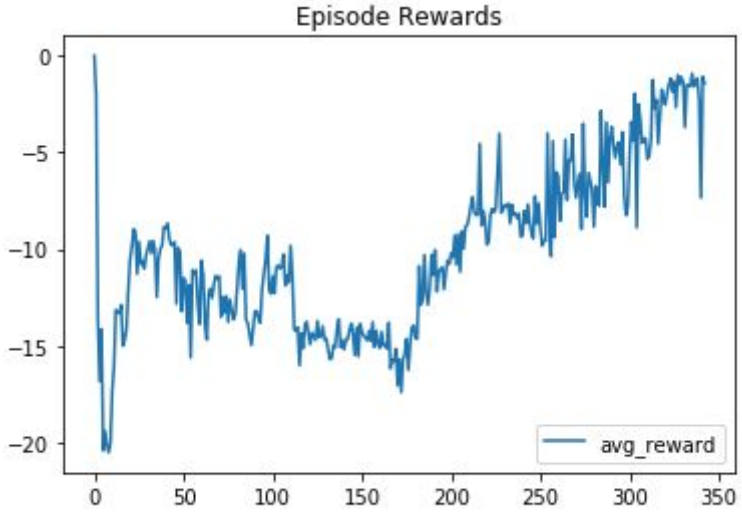
5.结论

经过训练，通过对模拟器的观测，模型可以成功完成目标任务，并且total_rewards收敛到了一个稳定的水平。

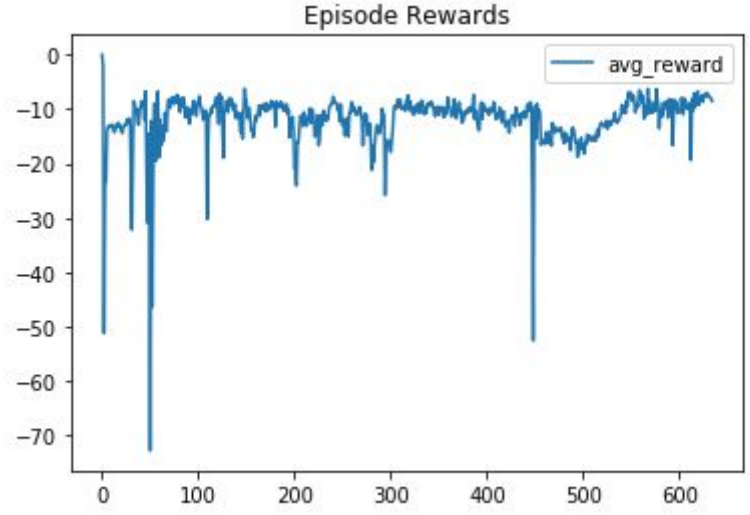
take off:



Hover:



Landing:



6.改进及思考

1.对比之前做的模型，主要改进是：

i.DDPG模型对于Takeoff和其他两个任务分别设定。（后两个任务涉及到的状态比较多，需要的结构也比Takeoff复杂）

ii.对Combine的优化，上一个版本是设计了一套整体的rewards来运行联合任务，但是不能很好的完成，有时会出现停滞。改进后使用的是将训练好的三个模型存储下来，按顺序执行。简单了很多。

iii.我对老模型进行了，改进，但效果没什么用，不知到怎么回事