train4

吴琪

July 2024

这次培训是学习 Isaac Gym，训练一个四足机器人可以正常走路的策略，具体实现就需要借助强化学习。强化学习（Reinforcement Learning）是机器学习的一个分支，主要研究智能体通过观察环境状态、执行动作和接收奖励来学习最优策略。因此，强化学习的核心思想是通过持续与环境的交互来获得反馈并学习到最优的行为序列。

在前面学习的监督学习（supervised learning）中，输入数据是大量有标注的数据，它们之间没有明显的关联关系。那么训练一个分类器，只需要把数据与正确的标签信息传递给神经网络，而当神经网络做出预测时，可以直接得到预测结果是正确还是错误，将结果与构造的损失函数相结合，通过反向传播就能训练神经网络。而在强化学习中，输入的数据并不是独立分布的，上一帧与下一帧之间有很强的连续性，同时也不能得到即时的反馈，也就是无法立刻判断当前操作是否正确，必须等到行动全部结束才能清楚，因此强化学习存在延迟奖励。通过比较两种算法，可以得到：强化学习是一个不断试错探索的过程，它通过探索环境来获取对环境的理解，并且它的输入是连续的，有着很强的关联性，所以在训练过程中，上一步动作也会影响随后得到的结果，也就导致模型只能获取延迟奖励，增加了训练的复杂度。

在强化学习的训练中，会采用序列决策的思想。强化学习是智能体与环境不断交互的过程，智能体把它的动作输出给环境，环境取得这个动作后，把下一步的观测与这个动作带来的奖励返还给智能体。一个简单的强化学习框架如下图所示：

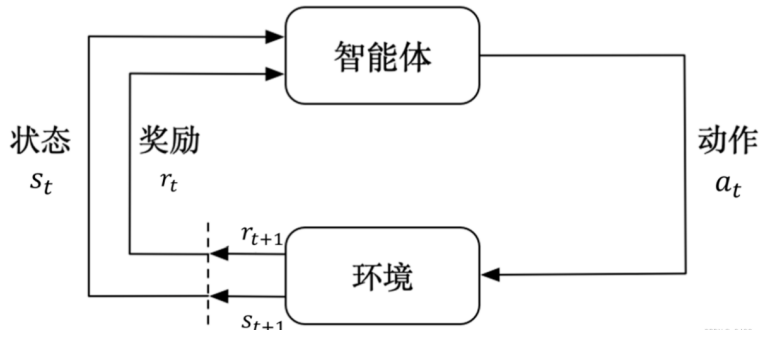

图 1: Structure diagram of reinforcement learning.

奖励是由环境给的一种反馈信号，它可以反映智能体在当前步骤中采取的策略的表现结果。在一个强化学习环境里面，智能体的目的就是选取一系列的动作来最大化奖励，所以这些选取的动作有长期的影响。但在这个过程里面，奖励是被延迟了的，也就是现在选取的某一步

动作，可能要等到很久后才知道产生了什么样的影响，因此需要一段动作序列来记忆每一步，在最终取得结果后返回给该序列的状态。

对于一个智能体，有多个部分组成。策略是智能体的动作模型，它决定了智能体的动作。策略可分为随机性策略和确定性策略两种。价值函数的值是对未来奖励的预测，通过它来评估状态或策略的好坏，目标希望在尽可能短的时间里面得到尽可能多的奖励。Q 函数是一种常见的价值函数，它的表达式如下：

$$Q_\pi(s,a) = E_\pi[G_t|s_t = s, a_t = a] = E_\pi[R_{t+1} + \lambda G_{t+1}|s_t = s, a_t = a] \tag{1}$$

表示为当前状态 s 依据策略 π 执行动作 a 得到的期望回报。下一个组成部分是模型，模型决定了智能体下一步的状态，它取决于当前的状态以及采取的动作。模型由状态转移概率和奖励函数两个部分组成。当有了以上三部分后，就能形成一个基础的马尔可夫决策过程，一个简单的马尔可夫决策过程如图 2 所示。因此可以用马尔可夫决策过程来定义强化学习任务，并将其表示为四元组 <S,A,P,R>，即状态集合、动作集合、状态转移函数和奖励函数。然而通常情况下，状态转移函数和奖励函数难以估计，智能体只能在环境中通过一定的策略来执行动作，等待奖励和状态迁移，然后根据这些反馈信息来更新动作策略，反复迭代直到学习到最优策略。

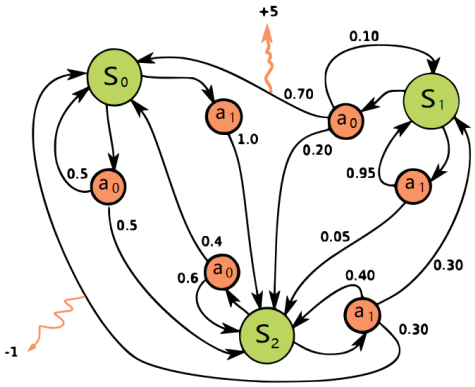

图 2: MDP decision-making process.

接下来介绍一下用于强化学习训练的平台及环境：Isaac Sim 与 Isaac gym。Isaac Sim 是一个自主构建的软件平台，能够更容易地对真实的物理机器人设计、调整、训练和部署自主控制代理。既可以通过软件自带的 UI 界面添加修改机器人，也可以通过代码与软件进行交互，通过代码的方式对机器人实现控制。Isaac Gym 是英伟达提供的强化学习研究的高性能仿真环境，可以通过并行多个模型的方法在 GPU 上快速训练控制模型，它也为创建和填充机器人及物体的场景提供了一个简单的 API，可以对机器人训练与结果进行可视化。

由于 Isaac gym 对于显卡驱动的配置要求不低，因此这次培训通过远程连接实验室的电脑进行实验。首先安装需要的应用，并配置环境，分别安装 IsaacGym，IsaacGymEnvs。实验需要实现四足机器人的正常走路，选取了 Legged Robots，通过在单个工作站 GPU 上使用大规模并行性来实现对现实世界机器人任务的快速策略生成。寻找开源代码 rsl_rl 与 legged_gym，成功配置后就能够对四足机器人实现训练。

训练 iteration 定为 300，在每个 iteration 后，终端会输出相应训练数据，记录 iteration0，iteration150 和 iteration299 的结果如下表所示：

表 1: AThe corresponding indicator data under the specified iteration

index	iteration0	iteration150	iteration299
Value function loss	0.0021	0.0023	0.0024
Surrogate loss	-0.0073	-0.0059	-0.0053
Mean action noise std	1.01	0.54	0.45
Mean reward	0.00	12.15	17.76
Mean episode length	13.24	990.13	993.54
Mean episode *rew_action_rate*	-0.0029	-0.1309	-0.0842
Mean episode *rew_ang_vel_xy*	-0.0030	-0.0714	-0.0465
Mean episode *rew_collision*	-0.0006	-0.0024	-0.0042
Mean episode *rew_dof_acc*	-0.0023	-0.0704	-0.0453
Mean episode *rew_feet_air_time*	-0.0032	-0.2064	-0.0997
Mean episode *rew_lin_vel_z*	-0.0107	-0.0414	-0.0265
Mean episode *rew_orientation*	-0.0013	-0.0233	-0.0174
Mean episode *rew_torques*	-0.0033	-0.2002	-0.1580
Mean episode *rew_tracking_ang_vel*	0.0019	0.4057	0.4412
Mean episode *rew_tracking_lin_vel*	0.0022	0.8627	0.9022

表中分别包括了损失值：用于衡量预测与实际之间的结果；代理损失：衡量新策略与旧策略之间的差异；平均动作噪声：在执行动作时，添加到动作上的噪声的平均值；平均奖励：强化学习过程中产生的奖励的平均值；平均回合长度：每个训练回合中平均时间步数；平均回合动作奖励率：回合期间每个动作所获得的平均奖励，越高表示机器人选择的动作越高效；平均回合在 xy 平面上的角速度奖励：越高表示机器人在控制角速度方面表现更好；平均回合碰撞奖励：衡量机器人避免碰撞的能力；平均回合关节加速度奖励；平均回合脚在空中的时间奖励；平均回合在 z 轴上的线速度奖励：越高表示机器人能够更好地控制其垂直速度；平均回合姿态奖励；平均回合关节力矩奖励；平均回合角速度跟踪奖励；平均回合线速度跟踪奖励。

开启训练后，首先对模型进行初始化，创建仿真所需环境：包括地形与多个模型，并初始化数据，然后进入每一步的迭代，渲染仿真画面，在每一步的物理迭代中，计算观测值与奖励值，根据评估结果调整训练模型。设定的策略接收机器人本体测量以及机器人基座周围的地形信息，观测结果包括: 基准线速度和角速度、重力矢量测量、关节位置和速度、策略选择的先前动作。总奖励设定为九项的加权和，希望机器人可以遵循指令速度。为了创造一个更平滑更自然的运动，还添加了对于关节扭矩，关节加速度，关节目标变化和碰撞的惩罚。最后还添加了一个额外的奖励条款，鼓励机器人采取更长的步骤。

在训练结束后进行测试，会输出相应指标曲线，如下图所示：

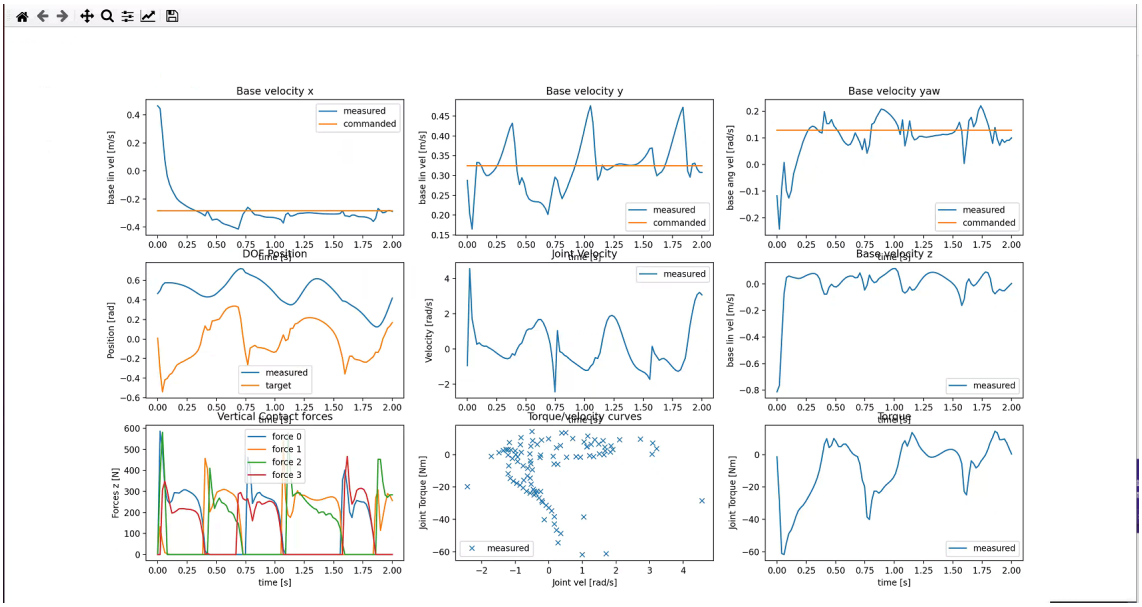

图 3: The result curve at the end of training.

图中的数据包括在 x 方向/y 方向/z 方向的基准速度，自由度的位置，关节速度，垂直接触力，转矩。

在训练过程中选用的强化学习算法是 PPO 算法，PPO 算法是在 Policy Gradient 算法的基础上产生的，Policy Gradient 是一种 on-policy 的方法，它利用现有策略和环境互动，产生学习资料，再利用这些资料，按照 Policy Gradient 的方法更新策略参数，最后用新的策略去交互、更新、重复。这其中有很多的时间都浪费在了产生资料的过程中，所以 PPO 算法优化为 off-policy，更加充分的利用产生的交互资料，增加学习效率，即添加了重要性采样（Importance Sampling），使用一些已经训练好的旧策略来采集样本，公式表示为：

$$E_{x\ p}[f(x)] = E_{x\ q}[f(x)\frac{p(x)}{q(x)}] \tag{2}$$

基于梯度更新的方法，PPO 算法又分为两种：近端策略优化惩罚（PPO-penalty）和近端策略优化裁剪（PPO-clip）。对于前者，将惩罚项添加到 PPO 算法的损失函数中，惩罚不符合约束条件的行为；后者则对函数进行一定程度的裁剪，限制新旧策略之间的差异。