**强化学习理论学习与代码实现**

# 强化学习及OpenAI Gym介绍

本章内容将介绍强化学习的基本概念、工作原理和监督、非监督学习的不同，并说明如何使用开发和比较强化学习算法的工具Gym。

## 强化学习

当我们思考学习的本质时，我们首先想到的是通过与环境交互来学习。当婴儿玩耍，挥舞手臂或四处张望时，他没有明确的老师，但他可以通过直接的感觉运动与环境联系。他可以通过这种联系获得大量关于因果关系，行为结果结果，以及如何实现目标的信息。在我们的生活中，这样的互动无疑是对环境和我们自己的主要知识来源。无论我们是学开车还是交谈，我们都可以敏锐地意识到我们的环境对我们所做的事情将作出什么样的反应，并且我们试图通过我们的行为来影响所发生的事情。从交互中学习是几乎所有学习和智能理论的基本思想。

在这里，我们探索了一种从交互中学习的计算方法。我们不直接对人或动物如何学习进行理论分析，而是探索理想化的学习情境，评估各种学习方法的效率。也就是说，我们采用人工智能研究人员或工程师的角度。我们去探索设计在科学或经济学领域可以有效解决学习问题，并通过数据分析和计算实验来评估这些设计。我们将这种方法称为强化学习，其与其他机器学习方法相比，更侧重于目标导向的交互学习。

### 强化学习与监督、非监督学习

强化学习就是学习怎么做，即如何建立情境-动作映射，以最大化数值奖励信号。学习器并没有被告知采取什么动作，而是通过不断尝试去发现能最大奖励的动作。在最有趣和最具挑战性的情况中，动作不仅影响瞬时报酬，还会影响下一场景，并由此影响后续的奖励。试错搜素和延迟奖励是强化学习最重要的两个特征。

强化学习与监督学习不同，后者是目前机器学习领域研究较多的一种学习方式。监督学习是从由专家提供的一组带标签的训练集中学习的。每个示例都是对一个情境和说明即标签的描述，该标签就是系统在该情境应该采取的正确动作，通常用于标识该情境所属的类别。这类学习的目的是让系统推断或概括它的响应，以便在未知训练集上也能正确工作。这是一种重要的学习，但仅凭这一点不足以从交互中学习。在交互问题中，代理需要在所有情境中采取正确的典型的可选动作，而获取这样的例子通常是不切实际的。在一个未知的领域，若要使学习收益最大化，代理必须能够从自己的经验中学习。

强化学习也不同于机器学习研究人员所说的无监督学习，后者通常是发现隐藏在未标记数据集合中的结构。监督学习和非监督学习这两个术语似乎囊括了机器学习的范式，但它们并没有。尽管人们可能会倾向于认为强化学习是一种无监督学习，因为它不依赖于正确行为的例子，但强化学习试图最大化奖励信号，而不是试图寻找隐藏的结构。在一个代理的经验中发现结构对于强化学习当然是有用的，但它本身并不能解决最大化奖励信号的强化学习问题。因此，我们认为强化学习是第三种机器学习范式，与监督学习、非监督学习以及其他范式并列，如图1所示。

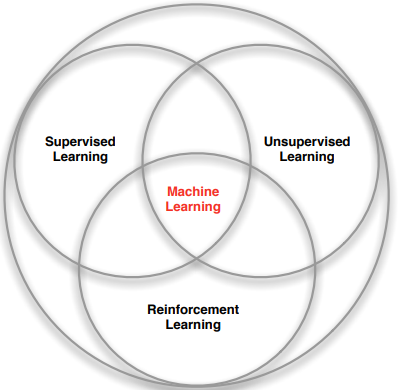


图1 机器学习分支

平衡探索与开发是强化学习中其他学习算法所不具备的挑战之一。为了获得更过的收益，强化学习代理必须倾向于过去已经尝试过并且能够有效获益的动作，但是要发现这样的动作，它又必须去尝试以前没有选择的动作。代理必须充分利用它既有经验以获得收益，但它也必须探索，以便在未来做出更好的工作选择。进退两难的是，要保证任务不失败，不能单一地只探索或利用。代理必须尝试各种各样的动作，并逐步偏向选择那些看起来最好的行动。在随机任务中，每个动作都必须尝试多次，才能获得对期望回报的可靠估计。探索利用困境是数学家们几十年来研究的热点问题，至今仍未解决。现在，我们只是注意到，在有监督和无监督的学习中，甚至不存在探索和开发之间平衡的整个问题，至少在这些范式最纯粹的形式中是这样。

强化学习的另一个关键特征是，它明确地考虑了目标导向的代理与不确定环境交互的整个问题。这与许多只考虑子问题而不考虑子问题如何融入全局的方法相反。例如，我们已经提到，许多机器学习研究关注监督学习，但没有明确说明这种能力最终将如何发挥作用。其他研究人员已经提出了具有一般性目标的规划理论，但没有考虑规划在实时决策中的作用，也没有考虑规划所需的预测模型从何而来。虽然这些方法已经产生了许多有用的结果，它们一个重要的限制在于过于关注子问题。

### 强化学习案例

* 直升机特技飞行；
* 在西洋双陆棋比赛中击败世界冠军；
* 管理投资组合；
* 控制发电站；
* 让一个人形机器人走路；
* 很多不同的雅达利游戏中玩得比人类更好。

### 强化学习的要素

除了代理和环境，强化学习系统一般有四个主要元素：策略，奖励信号，值函数，和一个可选的环境模型。

策略定义了学习代理在给定时间内的行为方式。粗略地说，策略是将环境中感知的状态映射为在这些状态下采取的行动。它对应于心理学中所谓的一系列刺激反应规则或关联。在某些情况下，策略可能是一个简单的函数或查找表，而在其他情况下，它可能涉及到大量的计算，如搜索过程。该策略是强化学习代理的核心，因为它本身就足以确定行为。一般来说，策略可能是随机的，指定了采取每个动作的概率。

奖励信号定义了强化学习问题的目标。在每一个时间步，环境发给代理的单一数字称为奖励。代理的唯一目标是最大化其长期获得的总奖励。因此，奖励信号定义了对代理而言的好坏事件。在生物系统中，我们可能认为奖励是类似于快乐或痛苦的经历。它们是代理所面临问题的直接精确的特征。奖励信号是改变策略的主要依据；如果策略选择的动作之后是得到了低回报，那么策略可能会被改变，以便在未来的情况下选择其他动作。总之，奖励信号可能是环境状态和所采取的行动的随机函数。

虽然奖励信号表明什么是直接意义上的好东西，但价值函数指明了长期内什么是好的。粗略地说，一个状态的价值是从该状态开始在未来可以预期累积的总奖励。鉴于奖励体现了环境状态的直接价值，价值则考虑了后续状态及奖励，从而反映了状态长期价值。例如，一个状态的瞬时奖励可能很低，但因为其后续状态具有高额奖励，因而也具有很高的价值，反之亦然。以人类作类比，奖励有点像快乐（如果是高的）和痛苦（如果是低的），而价值则对应于特定状态下一种更为精确和远见的对于满意与不满意的判断。

从某种意义上说，奖励是主要的，而价值作为奖励的预测是次要的。没有奖励就没有价值，估计价值的唯一目的是获得更多的奖励。然而，在制定和评估决策时，我们最关心的是价值。行动选择是基于价值判断的。我们寻求带来最高价值的动作，而不是最高奖励的，因为这些行动从长远来看对我们的回报最大。不幸的是，确定价值要比确定奖励难得多。奖励基本上是由环境直接给予的，但是价值必须通过一个代理在其整个生命周期中进行的一系列观察来评估和重新评估。事实上，我们所考虑的几乎所有强化学习算法中最重要的组成部分是一种有效估计值的方法。价值评估的中心作用可以说是我们在过去60年中学习强化学习的最重要的东西。

一些强化学习系统具有第四个也是最后一个要素，既环境模型。这是对环境的模拟，或者说，它对环境的行为做出推断。例如，给定一个状态和动作，该模型可以预测生成的下一个状态和下一个奖励。模型用于规划，规划指的是在实际经历之前考虑未来可能发生的情况来决定行动路线的任何方式。使用模型和规划解决强化学习问题的方法被称为基于模型的方法。更简单的无模型方法正好相反，它通过试错学习。

### 总结

强化学习是一种理解和自动进行目标导向学习和决策的计算方法。它与其他计算方法不同之处在于它强调了代理通过与环境的直接交互进行学习，而不依赖于监督或完整的环境模型。在我们看来，第一个认真处理从与环境的交互中学习以实现长期目标中出现的计算问题的领域就是强化学习。

强化学习使用马尔可夫决策过程的正式框架来定义学习代理与其环境之间的交互作用，包括状态、动作和奖励。这个框架用简单的方法来表示人工智能问题的基本特征。这些特征包括因果性、不确定性和不确定性，以及目标的存在性。

奖励和价值函数的概念是大多数强化学习方法的主要特征。我们认为，在策略空间中，价值函数对于有效搜索是非常重要的。价值函数的使用区分了强化学习方法和根据整个策略的评估直接在策略空间中搜索的进化方法。

## OpenAI Gym教程

Gym是一个开发和比较强化学习算法的工具包。它对代理的结构没有任何假设，并且兼容于任何数值计算库(如TensorFlow或Theano)。

Gym库中包含许多可以用于制定强化学习算法的测试问题（即环境），这些环境有共享接口，允许编写通用的算法。

### 安装

首先，需要安装Python 3.5+，只需使用pip安装gym：

pip install gym

这里需要更新pip至19.2.2版本，直接用以下命令即可：

python -m pip install --upgrade pip

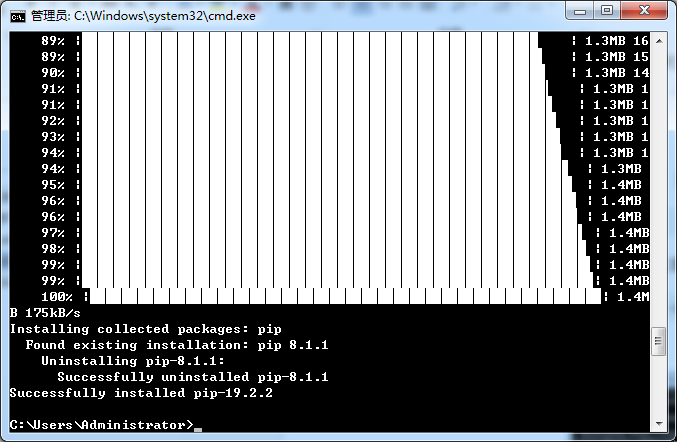


图2 更新pip

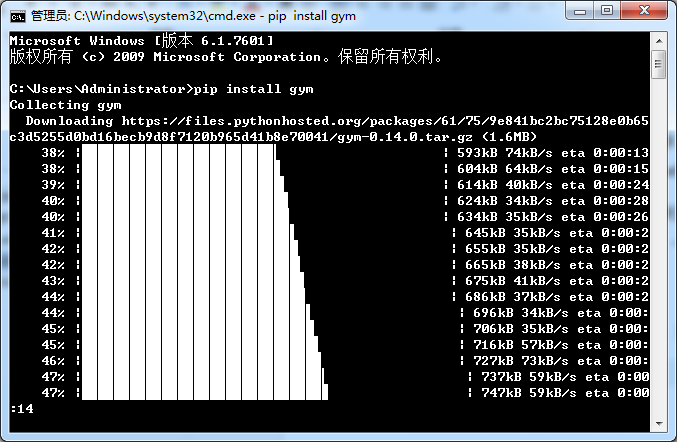


图3 安装gym

安装成功后可查看安装路径（D:\Python35\Lib\site-packages）。

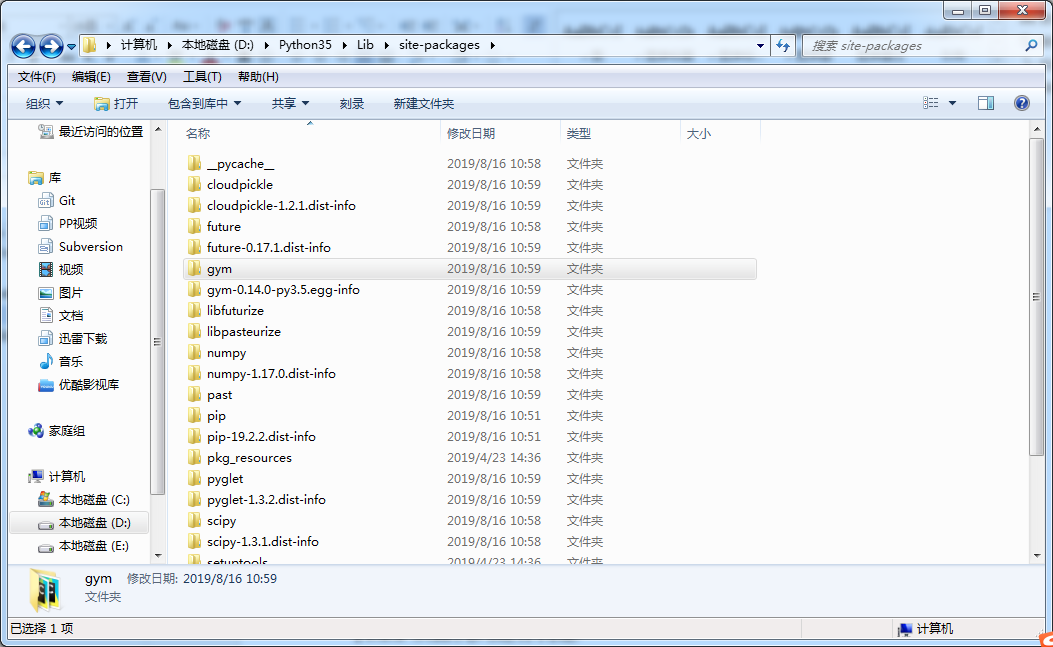


图4 安装路径

### 从源代码进行构建

如果喜欢的话，也可以直接克隆gym git库，当需要更改gym或者添加新的环境时这非常有用。用以下进行下载和安装：

git clone https://github.com/openai/gym

cd gym

pip install -e .

### 完全安装

为了安装整个环境集，需要先安装部分系统包。

MuJoCo安装教程见

<https://github.com/openai/mujoco-py#obtaining-the-binaries-and-license-key>。

Pip版本

要求Pip版本至少为1.5.0，可执行pip install --ignore-installed pip进行pip升级。

之后（[mujoco-py需要单独安装](https://github.com/openai/mujoco-py#obtaining-the-binaries-and-license-key)）可以运行以下命令对所有环境进行完成安装：

pip install -e .[all]

### 环境

下面是运行程序的简单案例，运行环境[CartPole-v0](https://gym.openai.com/envs/CartPole-v0) 1000步，每一步都渲染环境，然后弹出的窗口将呈现经典的小车倒立摆问题。

import gym

env = gym.make('CartPole-v0')

env.reset()

for \_ in range(1000):

env.render()

env.step(env.action\_space.sample()) # take a random action

env.close()

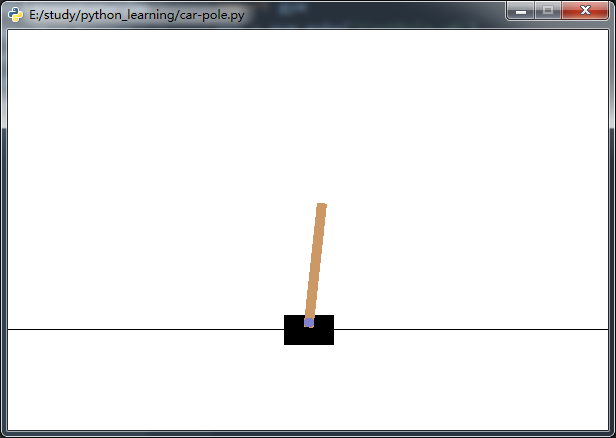


图5 运行示例

一般情况下，我们会在卡杆离开屏幕之前结束模拟，稍后会详细介绍。现在，请忽略关于即使这个环境已经返回done = True仍然调用step()的警告。

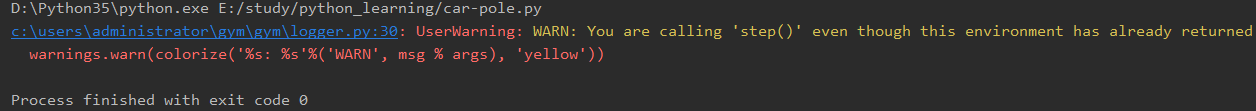


图6 警告

如果想看到其他环境的运行效果，可以将上面的[CartPole-v0](https://gym.openai.com/envs/CartPole-v0)替换为[MountainCar-v0](https://gym.openai.com/envs/MountainCar-v0)、[MsPacman-v0](https://gym.openai.com/envs/MsPacman-v0)(需要[Atari依赖](https://github.com/openai/gym#atari))或[Hopper-v1](https://gym.openai.com/envs/Hopper-v1)(需要[MuJoCo](https://github.com/openai/gym#mujoco)依赖)。

### 观察

如果想要在每一步比随机行为做的更高，就需要知道采取的动作是如何在环境中进行交互的。

环境的step函数返回的值就是我们所需要的，实际上，每一步环境都会返回四个值：

observation(object)：一个特定的环境对象，代表了从环境中得到的观测值，例如从摄像头获得的像素数据，机器人的关节角度和关节速度，或者棋盘游戏的棋盘状态。

reward(float)：前一行为所获得的奖励。奖励大小因环境而异，但目标总是提高总奖励。

done(boolean)：决定是否再次初始化环境。大多数（不一定所有）任务都被定义好了什么情况该结束这个回合。(举个例子，倒立摆倾斜地太远，或失去最后一条命)

info(dict)：调试过程中诊断信息，有时它会对我们的强化学习学习过程很有用（例如，有时它会包含最后一个状态改变后的原始概率），然而在评估你的智能体的时候你是不会用到这些信息去驱动你的智能体学习的。

一个经典的强化学习智能体与环境交互的过程可以被描述成如下方式：每次迭代，智能体选择一个动作，这个动作输入到环境中去，智能体会得到下一个观察(也就是下一个状态)和奖励。

程序开始时先调用reset()，它会返回一个初始的观测值，一个合适的方式编写代码如下所示：

import gym

env = gym.make('CartPole-v0')

for i\_episode in range(20):

observation = env.reset()

for t in range(100):

env.render()

print(observation)

action = env.action\_space.sample()

observation, reward, done, info = env.step(action)

if done:

print("Episode finished after {} timesteps".format(t+1))

break

env.close()

通过运行这段代码，可以输出以下视频和结果，可以清楚地看到在哪进行了reset。

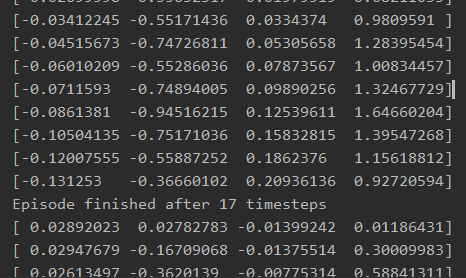
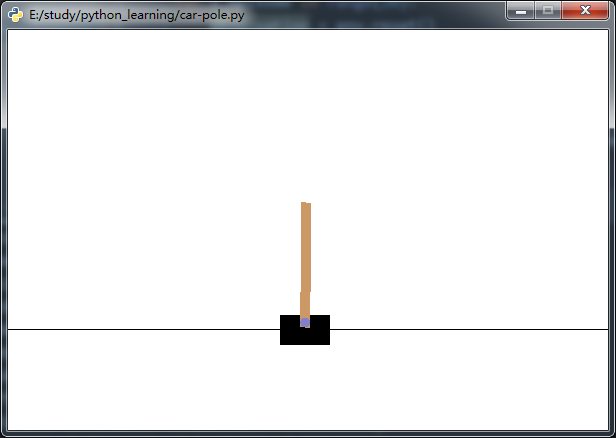


图7 输出结果

### 空间

在上面的例子中，我们一直在从环境的动作空间中随机抽取动作。但这些动作究竟是什么呢?每个环境都有一个action\_space和一个observation\_space。这些属性属于Space类型，它们描述了有效动作和观察的格式：

import gym

env = gym.make('CartPole-v0')

print(env.action\_space)

#> Discrete(2)

print(env.observation\_space)

#> Box(4,)

离散空间允许一个固定的非负数范围，所以在这种情况下有效的动作为0或1。Box空间表示一个n维的box，因此有效的观察值将是一个由4个数字组成的数组。可以通过以下来检查动作的边界范围：

print(env.observation\_space.high)

#> array([ 2.4 , inf, 0.20943951, inf])

print(env.observation\_space.low)

#> array([-2.4 , -inf, -0.20943951, -inf])

这种方式可以很有效地帮助我们编写不同环境下的代码，Box和离散的space是最常见的space。你可以从space中进行采样，或者查看一些信息：

from gym import spaces

space = spaces.Discrete(8) # Set with 8 elements {0, 1, 2, ..., 7}

x = space.sample()

assert space.contains(x)

assert space.n == 8

### 可用的环境

Gym配有多种从难到易的环境，也包含多种不同类型的数据，[full list of environments](https://gym.openai.com/envs)中可以查看概览。

* [Classic control](https://gym.openai.com/envs#classic_control)和[toy text](https://gym.openai.com/envs#toy_text)：完整的小规模任务，大多来自于强化学习文献，适合于入门。
* [Algorithmic](https://gym.openai.com/envs#algorithmic)：执行计算例如多位数加法和反转序列。一般认为这些任务对于计算机来说很容易，但是挑战在于纯粹从例子中去学习这些算法。这些任务有一个很好的特性，即通过改变序列长度很容易改变难度。
* [Atari](https://gym.openai.com/envs#atari)：玩经典的Atari游戏。我们以一种[易于安装](https://github.com/openai/gym#atari)的形式集成了[Arcade学习环境](https://github.com/mgbellemare/Arcade-Learning-Environment)(这对强化学习研究产生了很大的影响)。
* [2D and 3D robots](https://gym.openai.com/envs#mujoco)：控制仿真机器人。这些任务使用MuJoCo物理引擎，用于快速准确的仿真。包含了一些来自由UC Berkeley研究人员提供的benchmark环境。MuJoCo是一款私有软件，但也提供了[免费试用许可证](https://www.roboti.us/trybuy.html)。

### 注册

gym的主要目的是提供大量的环境集合，这些环境暴露了一个公共接口，并进行了版本控制以便进行比较。要列出已安装可用的环境，只需询问gym.env .registry：

from gym import envs

print(envs.registry.all())

#> [EnvSpec(DoubleDunk-v0), EnvSpec(InvertedDoublePendulum-v0), EnvSpec(BeamRider-v0), EnvSpec(Phoenix-ram-v0), EnvSpec(Asterix-v0), EnvSpec(TimePilot-v0), EnvSpec(Alien-v0), EnvSpec(Robotank-ram-v0), EnvSpec(CartPole-v0), EnvSpec(Berzerk-v0), EnvSpec(Berzerk-ram-v0), EnvSpec(Gopher-ram-v0), ...

这将给出EnvSpec对象的列表。这些对象定义了特定任务的参数，包括要运行的试验数量和最大步数。例如，EnvSpec(Hopper-v1)定义了一个环境，其中的目标是让一个2D模拟机器人跳跃：EnvSpec(Go9x9-v0)在9x9棋盘上定义了围棋游戏。

这些环境id被视为不透明的字符串。为了确保将来进行有效比较，环境永远不会以影响性能的方式更改，只会被更新的版本替换。我们现在给每个环境加上一个v0后缀，以便将来的替换可以自然地称为v1、v2等。将

将自己的环境添加到注册表中非常容易，从而使它们对gym.make()可用。make():只需在加载时注册[register()](https://github.com/openai/gym/blob/master/gym/envs/__init__.py)它们。

### 背景：为什么有Gym？

强化学习(RL)是机器学习中涉及决策和电机控制的子领域。它研究代理如何在复杂、不确定的环境中学习如何实现目标。令人兴奋的原因有两个：

RL非常普遍，包括所有涉及到做出一系列决策的问题：例如，控制机器人的马达使其能够跑和跳；做出商业决策，如定价和库存管理；或玩视频游戏和棋盘游戏。RL甚至可以应用于具有顺序或结构化输出的监督学习问题。

RL算法已经开始在许多困难的环境中取得良好的效果。RL有着悠久的历史，但直到最近在深度学习方面取得的进展之前，它还需要许多针对特定问题的工程。DeepMind的Atari results、Pieter Abbeel小组的BRETT和AlphaGo都使用了深度RL算法，这些算法没有对环境做太多假设，因此可以应用于其他设置。

然而，RL的研究也受到两个因素的影响：

需要更好的benchmas。在监督学习中，像ImageNet这样的大型标记数据集驱动了其进步。在RL中，类似的就是大量多样的环境集合。然而，现有的RL环境的开源集合没有足够的多样性，而且它们通常很难设置和使用。

缺乏环境的标准化。在问题定义上的细微差别，如奖励函数或动作集合，可以极大地改变任务的难度。这个问题使得复制已发表的研究和比较不同论文的结果变得困难。

Gym正试图解决这两个问题。