首行

Spingup
User Documentation

单位：NWPU

https://spinningup.openai.com/en/latest/

时间 2018 年 12 月 7 日星期五

目录

1 Introduction

1.1 What This Is

欢迎来到深 RL 旋转！这是 OpenAI 产生的一种教育资源，它使人们更容易学习深层强化学习(DeepRL)。对于不熟悉的情况：强化学习(RL)是一种机器学习的方法，用于教学 Agent 如何通过尝试和错误来解决任务。深度 RL 是指 RL 与深度学习的结合。本模块包含各种有用的资源，包括：简要介绍 RL 术语、各种算法和基本理论、关于如何成长为 RL 研究角色的文章、按主题组织的重要论文的管理列表、详细记录的关键算法的简短、独立实现的代码回复，以及一些作为热身的练习。

1.2 Why We Built This

我们听到的一个最常见的问题是，如果我想为人工智能的安全做出贡献，我该如何开始呢？在 OpenAI，我们相信深度学习-尤其是深度强化学习-将在强大的人工智能技术的发展中发挥核心作用。为了确保人工智能是安全的，我们必须想出符合这种模式的安全策略和算法。因此，我们鼓励每一个问这个问题的人去研究这些领域。然而，尽管有许多资源可以帮助人们快速提升深度学习，但深入强化学习更具有挑战性。首先，深度 RL 的学生需要有一些数学、编码和定期深入学习的背景。除此之外，他们还需要对这一领域有一个高层次的认识-了解其中研究的主题、它们为什么重要，以及已经做了什么-以及如何将算法理论与算法代码联系起来的仔细指导。高层次的观点是很难得到的，因为这个领域是多么的新。目前还没有一本标准的深层次 RL 教材，因此大部分知识被锁在论文或讲座系列中，需要很长时间才能解析和消化。学习实现深入的 RL 算法通常是痛苦的，因为要么是发布算法的论文遗漏了关键的设计细节，要么是算法的广泛公开实现很难读懂，隐藏了代码与算法是如何排列在一起的。尽管像 rllab、基线和 rllib 这样的出色的回复让已经在该领域的研究人员更容易取得进展，但他们将算法构建到框架中，涉及到许多不明显的选择和权衡，这使他们很难从中学习。因此，深 RL 领域有一个相当高的进入壁垒-对于新的研究人员，以及从业者和爱好者。因此，我们这里的包是为那些对深度 RL 感到兴奋的人而设计的，他们希望学习如何使用 RL 或做出贡献，但是对于学习什么或者如何将算法转化为代码没有一个清晰的概念。我们已经试着让这成为一个尽可能有用的发射点。尽管如此，从业人员并不是唯一能够(或应该)从这些材料中受益的人。解决人工智能安全问题需要具备广泛的专业知识和观点的人，而且许多相关的专业与工程或计算机科学根本没有任何联系。尽管如此，每个参与其中的人都需要对技术有足够的了解，才能做出明智的决定，并且需要几个细分的解决方案。

1.3 How This Serves Our Mission

OpenAI 的使命是确保 AGI 的安全发展和更广泛地从 AI 获得广泛的利益分配。像旋转这样的教学工具帮助我们在这两个目标上取得进展。首先，每当我们帮助人们理解人工智能是什么以及它是如何工作的时候，我们就更接近于广泛的利益分配。这使人们能够批判性地思考我

们预期会出现的许多问题，因为人工智能在我们的生活中变得更加复杂和重要。另外，至关重要的是，我们需要人们帮助我们确保 AGI 是安全的。这需要一套技能，这是目前短缺的，因为该领域是多么新。我们知道很多人对帮助我们很感兴趣，但不知道怎么做-这是你应该学习的！如果你能成为这方面的专家，你可以改变人工智能的安全性。

1.4 Code Design Philosophy

旋转回购中的算法实现设计得尽可能简单，但仍然相当好，并且高度一致，以揭示算法之间的基本相似之处。它们几乎是完全独立的，它们之间几乎没有共同的代码共享(除了日志记录、保存、加载和 MPI 实用程序)，这样有兴趣的人就可以单独研究每种算法，而不必挖掘无休止的依赖关系链来查看事情是如何完成的。这些实现是模式化的，因此它们尽可能接近伪代码，以尽量缩小理论和代码之间的差距。重要的是，它们的结构都是相似的，所以如果你清楚地理解其中的一个，那么跳到下一个就没有痛苦。我们试图将每个算法的实现中使用的技巧数量最小化，并尽量减少其他类似算法之间的差异。给出一些删除技巧的例子：我们省略了原来的软角色评论代码中的正则化项，以及所有算法的观察归一化。例如，我们消除了算法之间的差异：DDPG、TD3 和 SAC 的实现都遵循原始 TD3 代码中规定的约定，所有梯度下降更新都在情节结束时执行(而不是在整个集中全部发生)。所有的算法都是“相当好”的，因为它们大致达到了预期的性能，但不一定与文献中关于每项任务的最佳报告结果相匹配。因此，如果使用这些实现进行科学基准比较，请小心。有关每个实现的具体性能级别的详细信息可以在我们的基准页面上找到。

1.5 Support Plan

我们计划支持旋转，以确保它作为学习深入强化学习的有用资源。长期(多年)对分拆的支持的确切性质尚未确定，但在短期内，我们承诺：在发布后的前三周(2018 年 11 月 8 日至 2018 年 11 月 29 日)提供高带宽支持。我们将在错误修复、问题回答和对文档的修改上快速移动，以清除歧义。我们将努力简化用户体验，以便尽可能轻松地自我学习。大约在发布 6 个月后(2019 年 4 月)，我们将根据从社区收到的反馈意见，对软件包的状态进行认真的审查，并宣布任何未来修改的计划，包括一份长期路线图。此外，正如博客文章中所讨论的，我们正在为即将到来的学者和研究员队列使用旋转的课程。任何改变和更新，我们为他们的利益，将立即成为公众的。

2 Installation

2.1 Installing Python

conda create -n spinningup python=3.6
source activate spinningup

2.2 Installing OpenMPI

Ubuntu

sudo apt-get update && sudo apt-get install libopenmpi-dev

2.3 Installing Spinning Up

Installing Spinning Up

git clone https://github.com/openai/spinningup.git
cd spinningup
pip install -e .

2.4 Check Your Install

若要查看是否已成功安装，请尝试在 LunarLander-v2 环境中运行 ppo。
python -m spinup.run ppo --hid "[32,32]" --env LunarLander-v2 --exp_name installtest --gamma 0.999
这可能会运行大约 10 分钟，您可以将它放在后台，同时继续阅读文档。这将不会训练代理完成，但将运行足够长的时间，您可以看到一些学习进展时，结果出来。
After it finishes training, watch a video of the trained policy with
完成培训后，观看一段经过培训的策略的视频，
python -m spinup.run test_policy data/installtest/installtest_s0
然后用下面命令画图
python -m spinup.run plot data/installtest/installtest_s0

2.5 Installing MuJoCo (Optional)

首先，转到 mujoco-py GitHub 页面。按照自述文件中的安装说明，它描述了如何安装 MuJoCo 物理引擎和 mujoco-py 包(它允许使用 Python 中的 MuJoCo)。
您应该知道，为了使用 MuJoCo 模拟器，您需要获得 MuJoCo 许可证。任何人都可以获得免费的 30 天许可证，全日制学生也可以免费获得为期一年的许可证.安装了 MuJoCo 之后，使用

```
pip install gym[mujoco,robotics]
```

然后，通过在 Walker2d-v2 环境中运行 ppo 来检查事情是否正常。

```
python -m spinup.run ppo --hid "[32,32]" --env Walker2d-v2 --exp_name mujocotest
```

3 Algorithms

3.1 What’s Included

以下算法是在旋转包中实现的：

- Vanilla Policy Gradient (VPG)
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)
- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed DDPG (TD3)
- Soft Actor-Critic (SAC)

They are all implemented with MLP (non-recurrent) actor-critics, making them suitable for fully-observed, non-image-based RL environments, eg the Gym Mujoco environments.

- 香草策略梯度(VPG)
- 信任区域策略优化(TRPO)
- 最大策略优化(PPO)
- 深度确定性策略梯度(DDPG)
- 双延迟 DDPG(TD 3)
- 软演员-批评家(SAC)

它们都是由 MLP(非经常性的)演员评论家实现的，使得它们适合于完全观察的、非基于图像的 RL 环境，例如健身房的 Mujoco 环境。

3.2 Why These Algorithms?

我们在这个软件包中选择了核心的深度 RL 算法，以反映该领域最近历史上有用的思想进展，特别是两种算法-PPO 和 SAC-它们在策略学习算法的可靠性和样本效率方面接近 SOTA。他们还揭露了在深层 RL 中设计和使用算法时所做的一些权衡。

3.2.1 The On-Policy Algorithms

香草策略梯度是深 RL 空间中最基本的入门级算法，因为它完全早于深度 RL 的出现。VPG 的核心元素可以追溯到 80 年代末 90 年代初。它开始了一系列的研究，最终导致了更强大的算法，如 TRPO 和随后不久的 PPO。这一行工作的一个关键特点是，所有这些算法都是 ON 策略的：也就是说，它们不使用旧数据，这使得它们在样本效率方面更弱。但这是有原因的：这些算法直接优化了您所关心的目标-策略性能-并且从数学上计算出您需要在策略上的数据来计算更新。因此，这一系列算法通过牺牲样本效率来促进稳定性-但是您可以看到技术的进步(从 VPG 到 TRPO 到 PPO)，以弥补样本效率上的不足。

3.2.2 The Off-Policy Algorithms

DDPG 是 VPG 的一个类似的基础算法，尽管它要年轻得多-导致 DDPG 的确定性策略梯度理论直到 2014 年才出版。DDPG 与 Q 学习算法有着密切的联系，它同时学习 Q 函数和策略，并对它们进行改进。像 DDPG 和 Q-学习这样的算法是非策略的，因此它们能够非常有效地重用旧数据。他们通过利用 Bellman 的最优性方程来获得这一好处，q 函数可以训练成使用任何环境交互数据来满足(只要环境中的高回报区域有足够的经验)。但问题是，做好满足 Bellman 方程的工作并不能保证政策表现良好。从经验上看，我们可以获得很好的性能-当它发生的时候，样本效率是很好的-但是缺乏保证使得这个类中的算法可能脆弱和不稳定。TD3 和 SAC 是 DDPG 的后代，DDPG 利用各种见解来缓解这些问题。

3.3 Code Format

旋转过程中的所有实现都遵循一个标准模板。它们被分成两个文件：一个包含算法核心逻辑的算法文件和一个包含运行该算法所需的各种实用程序的核心文件。

3.3.1 The Algorithm File

算法文件总是从经验缓冲区对象的类定义开始，该类定义用于存储来自代理-环境交互的信息。接下来，有一个函数运行该算法，执行以下任务(按此顺序)：

- Logger setup
- Random seed setting
- Environment instantiation
- Making placeholders for the computation graph
- Building the actor-critic computation graph via the actor_critic function passed to the algorithm function as an argument
- Instantiating the experience buffer
- Building the computation graph for loss functions and diagnostics specific to the algorithm
- Making training ops
- Making the TF Session and initializing parameters
- Setting up model saving through the logger
- Defining functions needed for running the main loop of the algorithm (eg the core update function, get action function, and test agent function, depending on the algorithm)
- Running the main loop of the algorithm:
- Run the agent in the environment
- Periodically update the parameters of the agent according to the main equations of the algorithm
- Log key performance metrics and save agent
- Finally, there’ s some support for directly running the algorithm in Gym environments from the command line.

记录器设置
随机种子设置
环境实例化，
为计算图创建占位符，
通过传递给算法函数的
参与者-批判性计算图作为参数，
实例化经验缓冲区，
构建损失函数的计算图和特定于算法的诊断，
进行训练操作，
进行 TF 会话，并通过记录器定义所需的函数初始化参数，
建立模型保存。
运行算法的主循环(例如核心更新函数)，获取动作函数，并根据算法测试代理函数)
运行算法的主循环：
在环境中运行代理，
根据算法的主要方程定期更新代理的参数，
记录密钥性能指标，保存代理。

最后，对在 Gym 环境中从命令行直接运行该算法提供了一些支持。

3.3.2 The Core File

核心文件不像算法文件那样依附于模板，但确实有一些近似的结构：
与生成和管理占位符函数相关的函数，
用于构建与某个特定算法的参与者_批评家方法相关的计算图部分-
对于与该算法兼容的 MLP 参与者-批评家的任何其他有用的函数实现，其中，策略和值函数都由简单的 MLP 表示。

3.4 Running Experiments

获得深度 RL 的最佳方法之一是运行这些算法，并查看它们在不同任务上的执行情况。编出的代码库使小规模(本地)实验变得很容易，在本节中，我们将讨论两种运行它们的方法：要么从命令行运行，要么通过脚本中的函数调用。

3.4.1 Launching from the Command Line

https://spinningup.openai.com/en/latest/user/running.html#launching-from-scripts
从命令行启动，使用 SPOUP/run.py 旋转工具，这是一个方便的工具，可以让您轻松地从命令行启动任何算法(有任何超参数的选择)。它也是监视经过训练的策略和绘图的实用程序的薄包装器，尽管我们不会在这个页面上讨论这个功能(关于这些细节，请参阅关于实验输出和绘图的页面)。从命令行运行旋转算法的标准方法是
python -m spinup.run [algo name] [experiment flags]

例如：Python-mspinup.run ppo-envWalker2d-v2-exp_name walker

如果使用 ZShell：ZShell 将方括号解释为特殊字符。使用方括号的方法有几种，用于命令行参数；如果您想默认情况下转义它们，请确保转义它们，或者尝试这里推荐的解决方案。

3.4.2 Launching from Scripts

有关可能的参数的完整说明，请参阅每个算法的文档页。这些方法可用于建立专门的自定义实验，例如：

```
from spinup import ppo
import tensorflow as tf
import gym

env_fn = lambda : gym.make('LunarLander-v2')

ac_kwargs = dict(hidden_sizes=[64,64], activation=tf.nn.relu)

logger_kwargs = dict(output_dir='path/to/output_dir', exp_name='experiment_name')

ppo(env_fn=env_fn, ac_kwargs=ac_kwargs, steps_per_epoch=5000, epochs=250,
logger_kwargs=logger_kwargs)
```

3.4.3 ExperimentGrid

在机器学习研究中，在可能的超参数情况下运行相同的算法是非常有用的。用一个简单的工具来帮助实现这一点，叫做 ExperimentGrid。

Consider the example in spinup/examples/bench_ppo_cartpole.py:

```
from spinup.utils.run_utils import ExperimentGrid
from spinup import ppo
import tensorflow as tf

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--cpu', type=int, default=4)
    parser.add_argument('--num_runs', type=int, default=3)
    args = parser.parse_args()

    eg = ExperimentGrid(name='ppo-bench')
```

```
    eg.add('env_name', 'CartPole-v0', '', True)
    eg.add('seed', [10*i for i in range(args.num_runs)])
    eg.add('epochs', 10)
    eg.add('steps_per_epoch', 4000)
    eg.add('ac_kwargs:hidden_sizes', [(32,), (64,64)], 'hid')
    eg.add('ac_kwargs:activation', [tf.tanh, tf.nn.relu], '')
    eg.run(ppo, num_cpu=args.cpu)
```

在生成 ExperimentGrid 对象之后，将参数添加到

eg.add(param_name, values, shorthand, in_name)

in_name 强制一个参数出现在实验名称中，即使它在所有实验中都具有相同的值。在添加了所有参数之后，

eg.run(thunk, **run_kwargs)

在网格中运行所有实验(每个有效配置一个实验)，方法是将这些配置作为 kwargs 提供给函数 thunk.ExperimentGrid.run 使用一个名为 Call_test 的函数来启动 thunk，而*run_kwargs 指定调用_实验的行为。有关详细信息，请参阅文档页。除了没有快捷的 kwargs(不能使用 HID forac_kwargs：隐号在实验网格中)之外，ExperimentGrid 的基本行为与从命令行运行事物相同。(实际上，spinup.run 在引擎盖下使用了一个 ExperimentGrid。)

3.5 Experiment Outputs

在本节中，我们将介绍旋转算法实现产生的输出、存储在哪些格式中以及它们是如何组织的、存储在哪里以及如何更改这些输出，以及如何加载和运行经过训练的策略。你应该知道，旋转实现目前无法恢复对部分培训的代理的培训。如果你认为这个功能很重要，请告诉我们-或者说它是一个黑客项目！

3.5.1 Algorithm Outputs

每个算法都是为了保存训练运行的超参数配置、学习进度、经过训练的代理函数和值函数，并尽可能保存环境的副本(以便于同时加载代理和环境)。输出目录包含以下内容：

Output Directory Structure

◆ simple_save/

A directory containing everything needed to restore the trained agent and value functions. (Details below.)

包含恢复经过训练的代理和值函数所需的所有内容的目录。(详情如下)

◆ config.json

A dict containing an as-complete-as-possible description of the args and kwargs you used to launch the training function. If you passed in something which can't be serialized to JSON, it should get handled gracefully by the logger, and the config file will represent it with a string. Note: this is meant for record-keeping only. Launching an experiment from a config file is not currently supported.

一个包含一个尽可能完整的描述你用来启动训练功能的 args 和 kwargs 的数据集。如果您传入一些不能序列化到 JSON 的内容，它应该由记录器优雅地处理，并且配置文件将用一个字符串来表示它。注意：这只是为了记录。当前不支持从配置文件启动实验。

◆ progress.txt

A tab-separated value file containing records of the metrics recorded by the logger throughout training. eg, Epoch, AverageEpRet, etc.

一个选项卡分隔的值文件，包含记录器在整个培训过程中记录的指标记录。Epoch，AverageEpRet 等

◆ vars.pkl

A pickle file containing anything about the algorithm state which should get stored. Currently, all algorithms only use this to save a copy of the environment.

包含应该存储的算法状态的任何信息的泡菜文件。目前，所有算法只使用此方法保存环境的副本。

您应该知道，环境节约有时会失败，因为环境不能被腌制（pickled），vars.pkl 是空的。这是已知的一个问题的健身房 Box2D 环境在较早版本的健身房，这是无法保存的方式。

SIMPLE_SAVE 目录包含：

variables/

一个包含来自 TensorFlow 保护程序的输出的目录。参见 TensorFlow 保存模型的文档。

model_info.pkl

一个包含信息(从键到张量名称的映射)的数据集，它帮助我们在加载后解压缩保存的模型。

saved_model.pb

一个协议缓冲区，需要一个 TensorFlow 保存模型。

您应该知道这里唯一需要“手工”使用的文件是.json 文件。我们的代理测试实用程序将从 Simple_Save/目录和 vars.pkl 文件中加载内容，我们的绘图仪将解释 Progress.txt 的内容，这些都是与这些输出进行接口的正确工具。但是没有 config.json 的工具-它就在那里，这样如果您忘记了使用哪个超级参数进行了实验，就可以反复检查。

3.6 Save Directory Location

Experiment results will, by default, be saved in the same directory as the Spinning Up package, in a folder called data:

```
spinningup/
    data/
        ...
    docs/
        ...
    spinup/
        ...
```

LICENSE
setup.py

You can change the default results directory by modifying DEFAULT_DATA_DIR in spinup/user_config.py.

3.7 Loading and Running Trained Policies

3.7.1 If Environment Saves Successfully

3.7.2 Environment Not Found Error

3.7.3 Using Trained Value Functions

3.8 Plotting Results

用一个简单的绘图工具来解释结果。运行它时：

python -m spinup.run plot [path/to/output_directory ...] [--legend [LEGEND ...]]
[--xaxis XAXIS] [--value [VALUE ...]] [--count] [--smooth S]
[--select [SEL ...]] [--exclude [EXC ...]]

位置参数：

logdir 字符串。

尽可能多的日志目录(或用于日志目录的前缀，绘图仪将在内部自动完成这些目录)，就像您想要绘制的那样。将递归地搜索 Logdirs 以寻找实验输出。

你应该知道内部自动完成真的很方便！假设你做了几个实验，目的是比较不同算法的性能，生成的日志目录结构为：

data/
bench_algo1/
bench_algo1-seed0/
bench_algo1-seed10/
bench_algo2/
bench_algo2-seed0/
bench_algo2-seed10/

您可以很容易地生成一个图表，将 algo 1 和 algo 2 与：

python spinup/utils/plot.py data/bench_algo

依赖于自动完成来找到这两个 data/bench_algo1 和 data/bench_algo2.

可选参数:

-l, --legend=[LEGEND ...]

strings. 为情节指定图例（legend ）的可选方法。绘图仪图例（plotter legend ）将自动使用 config.json 文件中的 exp_name，除非通过此标志告诉它。只有当您为将被绘制的每个目录提供一个名称时，这才有效。(注意：这可能与您提供的 logdir args 的数量不一样！)回想一下，绘图仪查找 logdir ARGS 的自动完成：给定的 logdir 前缀可能有多个匹配项，您需要为每个匹配提供一个图例字符串-除非您已经通过选择或排除规则将其中的一些作为候选项删除(如下所示)。

-x, --xaxis=XAXIS, default='TotalEnvInteracts'

string. Pick what column from data is used for the x-axis.

-y, --value=[VALUE ...], default='Performance'

strings. Pick what columns from data to graph on the y-axis. Submitting multiple values will produce multiple graphs. Defaults to Performance, which is not an actual output of any algorithm. Instead, Performance refers to either AverageEpRet, the correct performance measure for the on-policy algorithms, or AverageTestEpRet, the correct performance measure for the off-policy algorithms. The plotter will automatically figure out which of AverageEpRet or AverageTestEpRet to report for each separate logdir.

从数据到 y 轴的图表中选择哪些列。提交多个值将产生多个图形。默认为性能，这不是任何算法的实际输出。相反，性能是指 AverageEpRet，这是对 ON 策略算法的正确性能度量，或者是 AverageTestEpRet，是对非策略算法的正确性能度量。绘图仪将自动确定哪一个 AverageEpRet 或 AverageTestEpRet 为每个单独的日志报告。

--count

Optional flag. By default, the plotter shows y-values which are averaged across all results that share an exp_name, which is typically a set of identical experiments that only vary in random seed. But if you’d like to see all of those curves separately, use the --count flag.

默认情况下，绘图仪显示 y-值，这些值在所有共享 exp_name 的结果中都是平均的，而 exp_name 通常是一组相同的实验，仅在随机种子中变化。但是如果你想单独看到所有这些曲线，请使用-计数标志。

-s, --smooth=S, default=1

int. Smooth data by averaging it over a fixed window. This parameter says how wide the averaging window will be.

通过固定窗口的平均值来平滑数据。这个参数表示平均窗口的宽度。

--select=[SEL ...]

strings. Optional selection rule: the plotter will only show curves from logdirs that contain all

of these substrings.

可选择规则：绘图仪将只显示包含所有这些子字符串的日志中的曲线。

--exclude=[EXC ...]

strings. Optional exclusion rule: plotter will only show curves from logdirs that do not contain these substrings.

可选排除规则：绘图仪将只显示不包含这些子字符串的日志文件中的曲线。