实验 3. 强化学习实践

2017 年 12 月 1 日

综述

与传统的监督、无监督学习不同，强化学习的过程需要与学习环境进行交互，利用环境反馈信息进行学习。因此在本次实验中，我们将脱离从训练数据集中获取学习模型的学习框架，转向与动态的学习环境打交道。本次实验，我们将从学习环境的安装、常用强化学习算法的实现，强化学习算法改进这些方面完整的体会一次强化学习研究的过程。

本次实验使用 Python 语言完成，请未使用过 Python 语言的同学自行安装和熟悉 Python 语言。

实验一. 实验环境安装 (10%)

强化学习的实验环境有很多，目前集成较为完善的环境为 OpenAI 开源的 Gym。Gym 中集成了很多强化学习的实验环境如 Atari、MuJoCo 等，并为测试强化学习方法提供了很多学习任务。本次实验环境的安装分为如下几部分：

1. 安装 OpenAI Gym 环境，安装方法见`http://gym.openai.com/docs/`，安装完成后便能够使用 Gym 提供的基础学习任务，如 CartPole 和 MountainCar；
2. 本次实验将会简单使用深度学习工具，推荐使用集成深度学习工具 Pytorch，安装和使用方法见`http://pytorch.org/`，同学们也可以自由选择其他深度学习工具。本次实验网络结构简单，仅需 CPU 就能训练，无需使用 GPU；
3. 熟悉 Gym 的使用，关注如何为 agent 指派动作，以及获得 agent 完成动作后的状态和 reward。

实验二. 离散环境下强化学习实现 (40%)

Gym 为用户提供了一些基础强化学习任务，如 CartPole-v0、MountainCar-v0 和 Acrobot-v1。其状态空间是连续的但是动作空间是离散的，对此类问题，一种直接的方法是将状态空间离散化，用离散的强化学习方法处理。实验二要求在状态空间离散化的基础上用 Q-learning 求解 CartPole-v0、MountainCar-v0 和 Acrobot-v1 三个学习任务。

因此本次实验的首要任务是对状态空间离散化，在离散化之前我们需要了解每个任务的状态空间设置，Gym 文档中已经给出了状态-动作空间设置的查询命令。在已知状态空间

的设置后，请同学们自行设计连续空间离散化的方式。对于离散的状态-动作空间，我们便可以使用 Q-learning[1]算法去学习。实验二的具体要求如下：

1. 在 MyQLearning.py 中实现 Q-learning 算法，并用于求解 CartPole-v0、MountainCar-v0 和 Acrobot-v1 三个学习任务；
2. CartPole 一条轨迹最大长度设为 20000，MountainCar 和 Acrobot 一条轨迹最大长度设为 2000；
3. 对状态空间离散化，在实验报告中说明离散化的方法；
4. 对 Q-learning 算法，在实验报告中简要报告算法的实现思路，对每个任务说明学习时算法的超参设置；
5. 对每个任务算法均能获得一个最好的策略，请对策略测试多条轨迹，在实验报告中报告多条轨迹上 reward 之和的均值和标准差；
6. Gym 环境下，学习得到的策略对任务是否有效可以通过可视化的方式直观的展示出来，请展示 CartPole 任务下的可视化结果。

实验三. 连续环境下强化学习实现 (30%)

对于连续的状态-动作空间，另一种更为合理的方式是直接在连续的空间中进行学习。仍然对 CartPole-v0、MountainCar-v0 和 Acrobot-v1 这三个任务，在实验三中我们直接在连续空间中学习。由于 Q-learning 需要穷举状态-动作对，所以面对连续状态-动作空间 Q-learning 无法适用。但是值函数近似 Q-learning 方法能够解决此类问题。若将值函数近似模型选取为一个深度网络，此方法就是著名的 Deep Q-learning (DQN) 算法。

Deep Q-network (DQN)

DQN 算法基于 Q-learning，算法框架和 Q-learning 完全相同，不同之处在于 Q 函数的表示和更新方式上。DQN 中 Q 函数用一个深度网络表示，记为 $Q(s,a;\theta)$，其中 s 表示状态，a 表示执行的动作，θ 表示网络参数。DQN 的目的是得到一个 Q 函数网络，对于一组状态动作对 (s,a)，使得 Q 函数网络的输出 $Q(s,a;\theta)$ 等于 (s,a) 下真实的 Q 值。具体到深度网络表示的 Q 函数模型上，Q 函数的学习问题转化为了如何学习一组深度网络参数 θ 使网络输出近似于真实的 Q 值。

DQN 算法伪代码如 Algorithm 1所示，Algorithm 1由 [1] 中 Algorithm 1 稍作修改得到。在 [1] 中观察的状态是一帧帧的图像，一个 CNN 网络被用来提取抽象特征。在本次实验中，选取的任务的观察值均为实数值，所以抽象特征提取步骤被省略。请大家仔细观察此算法，体会此算法与 Q-learning 的联系与不同。

[1]Q-learning 算法见 [3] 16.4.2 章节，算法伪代码 (p.388，图 16.13) 经过修正如下：步骤 4，$\pi^{\epsilon}(x) \to a = \pi^{\epsilon}(x)$，详情见`https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/MLbook2016.htm`。

Algorithm 1 Deep Q-learning with Experience Replay

Procedure:

Initialize replay memory $\mathcal{D}$ to capacity N

Initialize action-value function Q with random weights θ

for episode $= 1$ to M **do**

  Initialize first state s_1

  for $t = 1$ to T **do**

    $$a_t = \begin{cases} \text{Select from } A \text{ randomly} & \text{w.p. } \epsilon \\ \max_a Q(s_t, a; \theta) & \text{w.p. } 1-\epsilon \end{cases}$$

    Execute a_t in emulator to observe reward r_t and next state s_{t+1}

    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$

    Sample random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$

    $$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$$

    Update θ by gradient descent with loss function $(y_j - Q(s_j, a_j; \theta))^2$

  end for

end for

实验要求

在实验三中，要求实现 DQN 算法，并将此算法应用于 CartPole-v0、MountainCar-v0 和 Acrobot-v1 三个动作控制任务中，具体实现细节和要求如下：

1. CartPole-v0、MountainCar-v0 和 Acrobot-v1 的参数设置同实验二；
2. Q 值网络可选用 Multi-layer Perceptron (MLP)，网络结构请自行调参决定；
3. 在 MyDQN.py 中实现该算法，训练过程中，每个任务 DQN 中的相关超参数如 M, T, N, ϵ，MLP 结构等的设置可能不尽相同，请大家针对不同的任务自行调参，并在实验报告中详细报告各个超参数的设置；
4. 实验报告还需报告算法实现细节并用图示的形式展示算法的训练结果，主要报告两幅图，一、网络训练误差随训练轮数的变化关系 (x 轴为训练轮数，y 轴为每轮中训练误差的均值)，二、每轮训练 reward 之和随训练轮数的变化关系 (x 轴为训练轮数，y 轴每轮 reward 之和)；
5. 对于每个任务，多次测试训练得到的最好的策略，报告 reward 之和的均值和标准差；
6. 请展示 DQN 在 CartPole 上的可视化实验效果。

实验四. DQN 的改进 (20%)

在 DQN 中，每在任务环境中探索一步均要更新一次网络权重，在 [2] 中，研究者指出这样的更新方式是不稳定的，对学习 Q 值网络不利，文中给出的改进版本 DQN 如算法 2所示。请仔细阅读此算法伪代码，体会其中的改进之处，完成以下实验：

Algorithm 2 Improved Deep Q-learning with Experience Replay

Procedure:

Initialize replay memory $\mathcal{D}$ to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

for episode $= 1$ to M **do**

  Initialize first state s_1

  for $t = 1$ to T **do**

    $a_t = \begin{cases} \text{Select from } A \text{ uniformly} & \text{w.p. } \epsilon \\ \max_a Q(s_t, a; \theta) & \text{w.p. } 1 - \epsilon \end{cases}$

    Execute a_t in emulator to observe reward r_t and next state s_{t+1}

    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$

    Sample random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$

    Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) & \text{for non-terminal } s_{t+1} \end{cases}$

    Update θ by gradient descent with loss function $(y_j - Q(s_j, a_j; \theta))^2$

    Every C steps reset $\hat{Q} = Q$

  end for

end for

1. 在 DQN 的代码基础之上，在 MyImprovedDQN.py 文件中实现算法 2；
2. 用实现的算法 2完成 CartPole-v0、MountainCar-v0 和 Acrobot-v1 的训练，强化学习任务的参数设置同实验二；
3. 实验报告内容同实验三，除此之外请简要说明此改进算法是如何增加 Q 值网络训练时的稳定性的；
4. 试比较 DQN 和改进版本 DQN 的实验效果，从实验效果上尽可能说明两种方法的异同；
5. 请展示改进后的 DQN 在 CartPole 上的可视化实验效果。

实验结果提交

提交的实验结果.zip 文件中需包含如下内容：

1. 实验源码，MyQLearning.py、MyDQN.py、MyImprovedDQN.py 和其他完成实验所需的代码文件，此外需包含一个 ReadMe.txt 文件，在此文件中描述如何运行提交的代码；
2. 实验报告，一个.pdf 文件，报告中需包含实验二、三、四中要求报告的内容，实验报告的 Latex 模板可从课程主页下载；

3. 3 个.mp4 文件，内容为实验二、三、四中 CartPole 任务的可视化效果；

实验结果的提交方法和命名规则见课程主页。

参考文献

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] 周志华. 机器学习. 清华大学出版社, 2016.