强化学习 HW2 – DQN

(The English version is given at the end of the document)

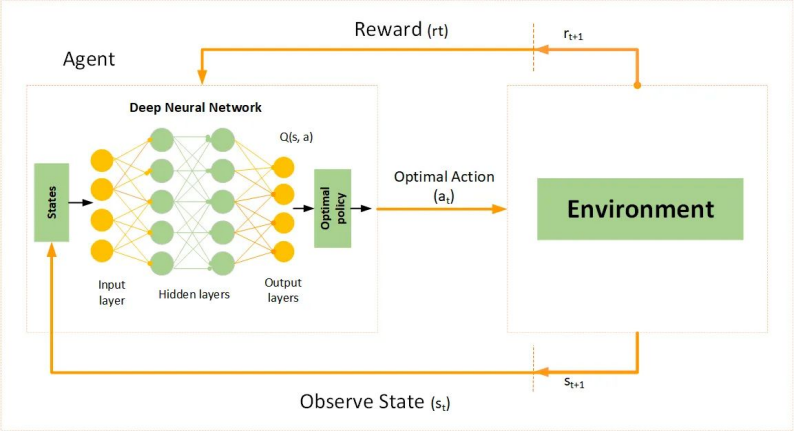
2024/12/3

DQN 介绍

DQN（deep Q-network）是指**基于深度学 习的 Q-learning 算法**，主要结合了**价值函数近似（ value function approximation）**与**神经网络**技术，并采用**目标网络**和**经历回放**等方法进行网络的训练。
传统的 **Q-learning** 算法会使用**表格**的形式存储状态价值函数 V(s) 或动作价值函数 Q(s,a)。但现实任务所面临的状态空间往往是连续的，就不能再使用表格的 方式进行存储。因此引入了**神经网络（Q-network）**来近似的拟合上述两种函数，这个方法被称之 为**价值函数近似（ value function approximation）**。

$$Q_\phi(\boldsymbol{s},\boldsymbol{a}) \approx Q_\pi(\boldsymbol{s},\boldsymbol{a})$$

其中 $\boldsymbol{s}$、$\boldsymbol{a}$ 分别是状态 s 和动作 a 的向量表示，上式左侧的函数表示参数为 ϕ 的函数，比如神经网络，其输出为一个实数。



一个简单的 DQN 算法

A simple DQN Algorithm

1. take action a_i in ENV,then insert (s_i, a_i, s_{i+1}, r_i) in Experience Replay
2. if Experience Replay have enough τ, then randomly sample (s_i, a_i, s_{i+1}, r_i) from Experience Replay
3. $target(s_i) = r_i + \gamma \max_{a'} Q_{\phi^-}(s_{i+1}, a')$
4. $\phi \leftarrow \phi + \alpha(target(s_i) - Q_\phi(s_i, a_i))\frac{dQ_\phi}{d\phi}$
5. every N step, let $\phi^- = \phi$
6. go to step 1

- 初始化函数Q、目标函数$\hat{Q}$，令$\hat{Q} = Q$。
- 对于每一个回合。
  - 对于每一个时间步t。
    - 对于给定的状态s_t，基于Q（ϵ-贪婪）执行动作a_t。
    - 获得反馈r_t，并获得新的状态s_{t+1}。
    - 将(s_t, a_t, r_t, s_{t+1})存储到缓冲区中。
    - 从缓冲区中采样（通常以批量形式）(s_i, a_i, r_i, s_{i+1})。
    - 目标值是$y = r_i + \max_a \hat{Q}(s_{i+1}, a)$。
    - 更新Q的参数使得$Q(s_i, a_i)$尽可能接近于y（回归）。
    - 每C次更新重置$\hat{Q} = Q$。

DQN 中的高估问题

简而言之，高估 问题是 agent 在任何 给定状态下总是选择非最优动作，只是因 为它具有最大 Q 值。
在 **vanilla Q-learning** 中，Agent 的最优策略始终是在任何给定状态下选择最佳动作。这个想法背后的假 设是最佳行动具有最大预期/估计 Q 值。
然而，Agent 一开始对环境一无所知，它需要首先估 计 Q(s, a) 并在每次迭代时更新它们。这样的 Q 值有很多噪音，我们永远无法确定具有最大期望 /估计 Q 值的动作是否真的是最好的 动作。**不幸的是，在大多数情况下，最佳 动作的 Q 值通常比非最佳 动作的 Q 值更小。**

TD-target is UNKNOWN! Current Q-value

$$LostFunction = Q_{best}(s_t, a_t) - Q(s_t, a_t)$$

Estimated TD-target

Discount factor

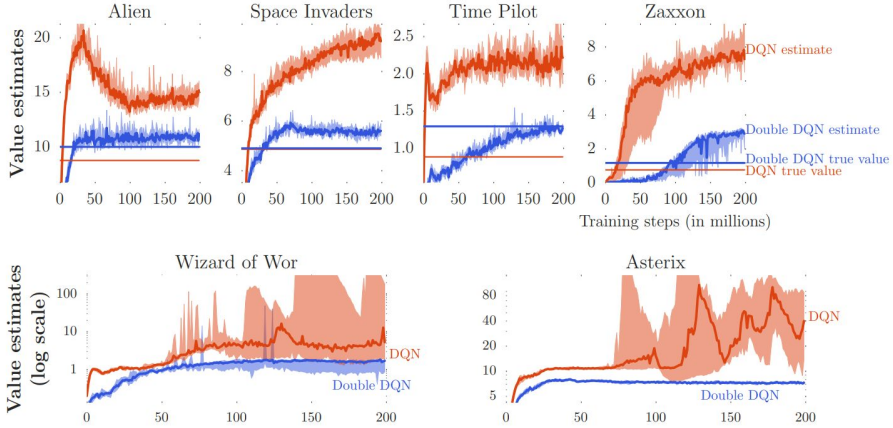
$$Q_{best}(s_t, a_t) = R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$

Reward Maximum next-state Q-value

Update Current Q-value

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Current Q-value



Double DQN 算法

首先从 Double Q-Learning 入手，使用两个不同的**动作价函数（action-value functions）**，**Q** 和 **Q'**，这样噪声就可以被视作是均匀分布，解决了 动作价值高估的问题。其 Q-network 的更新过程：

1. **Q 函数**用于选择下一状态 Q 值最大的最佳动作 a。
2. **Q' 函数**用于通过上面选择的 a 来计算预期 Q 值。

不正式的说，**Double DQN** 是将里面的 Q 函数用神经网络来表示：

1. **深度 Q 网络（Deep Q Network）**用于选择下一个状态具有最大 Q 值的最佳动作 a 。
2. **目标网络（Target Network）**用于使用上面选择的 a 计算估计的 Q 值。

Double Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma\, Q'(s_{t+1}, a) - Q(s_t, a_t))$$

estimated/expected Q-value

$$a = \max_a Q(s_{t+1}, a)$$

$$q_{estimated} = Q'(s_{t+1}, a)$$

Double Q Network

estimated/expected Q-value

$$Q_{qnet}(s_t, a_t) \leftarrow R_{t+1} + \gamma\, Q_{tnet}(s_{t+1}, a)$$

$$a = \max_a Q_{qnet}(s_{t+1}, a)$$

$$q_{estimated} = Q_{tnet}(s_{t+1}, a)$$

根据选择的参数更新 Target Network 的参数

$$Q_{tnet}(s, a) = Q_{qnet}(s, a)$$

实验介绍

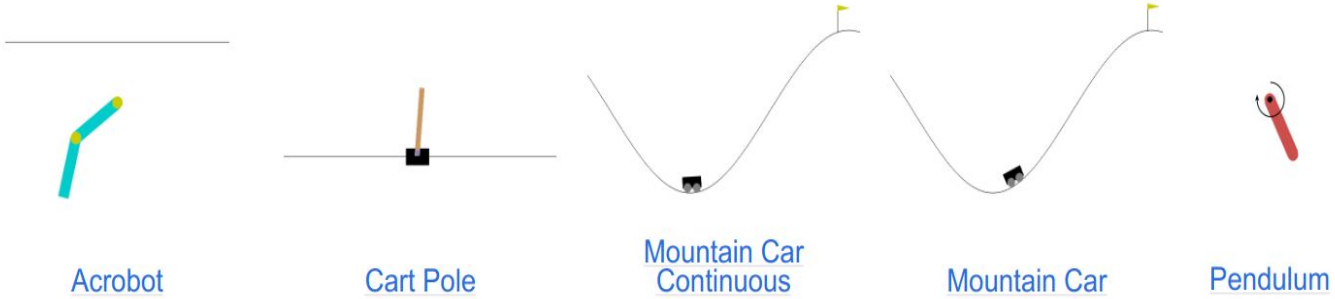
实验准备：

- 安装 anaconda/miniconda
  - 配置镜像源 ：https://mirror.tuna.tsinghua.edu.cn/help/anaconda/
- 安装 pytorch ~= 1.12，根据自身情况 选择GPU或CPU版本
  - 30系及以上的 显卡，不要 选择CUDA10.x的版本
  - https://pytorch.org/get-started/previous-versions/
- 安装 tensorboard 用于可 视化训练过程
  - pip install tensorboard
- 安装 Gymnasium（Gym被OpenAI转手出去以后 处于欠维护状态）
  - https://github.com/Farama-Foundation/Gymnasium
  - 官方手册：https://gymnasium.farama.org/

环境介绍(这一页中提到的 5 种中任选两种实现)

Classic Control https://gymnasium.farama.org/environments/classic_control/ :

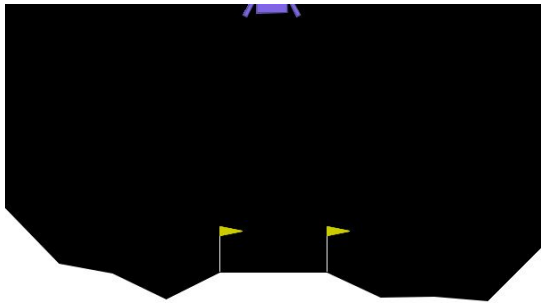
- Choose any two out of the five environments mentioned on this page to implement.
- Install the unique dependencies of this set of environments
  - pip install gymnasium[classic-control]



环境介绍(必选)

LunarLander https://gymnasium.farama.org/environments/box2d/lunar_lander/:

- It's mandatory to using this environment.
- Install the unique dependencies of this set of environments
  - pip install gymnasium[box2d]
  - 如果出现找不到 swig 的错误，尝试用apt安装swig（If an error indicating that swig cannot be found occurs, try to install swig using apt.）



实验要求

- **参考给出的 demo_DQN.py, 完善 DQN 算法**：
  - **环境**：在任选的 2 个 Classic Control 环境（如 CartPole、MountainCar 等）和 LunarLander 环境中实现 DQN 算法。
  - **可视化**：绘制 Reward 曲线（可以直接从 Tensorboard 的界面截图），为了更好的视觉效果，可以从Tensorboard中导出CSV，用seaborn等重绘（不重要）。
  - **实现方式**：在提供的 demo_DQN.py 中完成需要补全的部分，这些部分已使用 #TODO 标出。
  - 不需要太过关注训练的分数（波动很大或者难以收敛都很正常）。
- **实现 Double DQN (DDQN) 并与 DQN 的表现进行比较**：
  - **环境和可视化**：使用与 DQN 相同的环境和可视化方法。
  - **实现方式**：在 DQN 的基础上修改实现，，重点是基于实验结果对 DQN 和 Double DQN 算法进行简要分析和比较。比较内容应包括收敛速度、最优性、稳定性等方面。
- **录制每种方法在每种 环境下最好策略的 视频**（ubuntu下可使用kazam）。

实验要求（bonus）

- **在 DQN 的基础上实现 Dueling DQN：**
  - **环境、可视化和实现方式**同 DDQN。
  - 需要在实验报告中说明其改进点是如何在代码上体现的（直接给出代码和公式的对照即可，不必长篇幅的介绍）。
  - 需要在实验报告中将 Dueling DQN 加入对比。

实验提交

- 提交内容中需要包含的内容：
  - 一份 PDF 格式的 实验报 告。
  - 视频压缩 包(.zip)（每个 环境每种模型一个 视频）。
- 代码压缩包(.zip)：
  - 代码压缩包中需包含：DQN、DDQN、Dueling DQN 的代码（也可以写在一份代 码中）。
  - 一份 readme.md，给出可以用于 测试的命令（如果有使用到 demo_DQN.py 之外的 库，需要在里面 说明）。
  - 一个用于存放 权重文件的文件 夹（只需保留用于 测试的最好 权重即可）。
- **作业发布地址 / 提交地址：BB系统。**
- **截止 时间：2025/1/1 23:59 UTC+8（元旦假期 结束前）**

Experimental requirements

- **DQN Algorithm Implementation::**
  - **Environment:** Implement the DQN algorithm in two of the Classic Control environments (e.g., CartPole, MountainCar) and the LunarLander environment.
  - **Visualization:** Plot the reward curve (this can be directly captured from TensorBoard's interface). For better visualization, you may export data from TensorBoard as a CSV and then use tools like Seaborn to replot the curve. (This step is optional.)
  - **Implementation:** Complete the parts marked with #TODO in the provided demo_DQN.py.
  - Don't focus too much on achieving high training scores—fluctuations or difficulty in convergence are expected.
- **Double DQN (DDQN) Implementation:**
  - **Environment and Visualization: Use the same environments and visualization methods as in the DQN task.**
  - **Implementation:** Modify the DQN implementation to incorporate Double DQN and compare its performance with that of DQN.
- **Experimental report:** The report should not include detailed explanations of code implementation or the underlying theory. Instead, focus on a brief analysis and comparison of the performance of DQN and Double DQN based on experimental results. Key comparison points should include convergence speed, optimality, stability, and other relevant aspects.
- **Record a video showing the best strategy learned by each method (DQN and Double DQN) in each environment.(on Ubuntu, you can use Kazam).**

Experimental requirements (bonus)

- **Implement Dueling DQN based on DQN:**
  - **The environment, visualization, and implementation methods are the same as those of DDQN.**
  - It is required to illustrate in the **experiment report** how the improvement points of Dueling DQN are reflected in the code (it is sufficient to directly present the comparison between the code and the formula; there is no need for a lengthy introduction).
  - Dueling DQN should be added for methods comparison in the **experiment report**.

Experiment Submission

- **The following contents are required to be included in the submission:**
  - **PDF Format Experiment Report.**
  - **A video compression package (.zip) which should contain one video for each environment and each model.**
- **Code Compression Package(.zip) :**
  - The code compression package should include **the codes of DQN, DDQN (and Dueling DQN)**, (it is also acceptable to write them in one single piece of code).
  - A **readme.md file** should be provided, which gives the **commands that can be used for testing**. If any libraries other than demo_DQN.py are used, they need to be specified in this file.
  - A folder for storing **weight files** is required. **Only the best weights** used for testing need to be retained.
- **Assignment Release Address / Submission Address : The BB system.**
- **Deadline : 2025/1/1 23:59 UTC+8 (before the end of the New Year's Day holiday).**