Deep RL with PyTorch and Gymnasium

John Wikman
jwikman@kth.se

KTH Royal Institute of Technology

2023-10-31

What to Expect

- Only formulas and notation only necessary for the implementation. Warm-up before going through the code.
- **Will gloss over A LOT of theory.**
- See the book by Sutton and Barto (theory) and DQN Paper:

Richard S. Sutton and Andrew G. Barto
Reinforcement Learning, An Introduction.
The MIT Press, 2018.
Available for free online:
http://incompleteideas.net/book/RLbook2020.pdf

Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M.
Playing Atari with Deep Reinforcement Learning.
NIPS Deep Learning Workshop 2013
http://arxiv.org/abs/1312.5602

Outline

Recap of Reinforcement Learning

What are we optimizing?

Deep Q-Learning

Practical RL Tools

Overview

Gymnasium

The Agent and the Environment

- Know state space $s \in S$ and action space $a \in A$.
- Functions $p(s'|s,a)$ and $r(s,a)$ assumed unknown, only observe their output in transitions (s, a, r, s').
- Policy performance metric, the Q-function.
- $Q^\pi(s,a)$: *How good is action a in state s on policy π?*

$$Q^\pi(s,a) = \begin{cases} r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a), a' \sim \pi(\cdot|s')} [Q^\pi(s', a')] \\ r(s,a) \qquad \text{if } s' \text{ is terminal} \end{cases}$$

  - The *discount* $0 < \gamma < 1$ ensures convergence
- $Q^*(s,a)$: *How good is a in s under an optimal policy?*

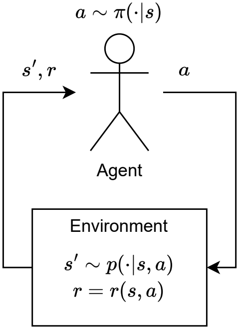

Figure: Agent-Env Interaction

Q-Function Approximator

- If known $Q^*(s, a)$, then $\arg\max_a Q^*(s, a)$ will most likely give an optimal policy
- Cannot compute $Q^*(s, a)$ since we do not know $p(s'|s, a)$ and $r(s, a)$
- But can estimate it over observations (s, a, r, s'): *(equal on average)*

$$Q^*(s, a) \approx r + \gamma Q(s', \arg\max_{a'} Q^*(s', a')) \tag{1}$$

- If finite A, parameterized function $f_\theta(s) : \mathbb{R}^d \times S \to \mathbb{R}^{|A|}$ can estimate $Q^*(s, a)$ as

$$f_\theta(s) \approx [Q^*(s, a_0), Q^*(s, a_1), ..., Q^*(s, a_{|A|-1})] \tag{2}$$

- **Benefit:** $f_\theta(s)$ allows for trivial argmax
- Will use notation $Q_\theta(s, a) = f_\theta(s)[a]$ from now on

Learning the Q-Function

- Recall what we want to approximate:

$$Q_\theta(s,a) \approx Q^*(s,a) = r + \gamma Q(s', \arg\max_{a'} Q^*(s',a')) \tag{3}$$

- Substitute and subtract to get the approximation error:

$$err = r + \gamma Q_\theta(s', \arg\max_{a'} Q_\theta(s',a')) - Q_\theta(s,a) \tag{4}$$

- We denote $r + \gamma Q_\theta(s', \arg\max_{a'} Q_\theta(s',a'))$ as the **target** that $Q_\theta(s,a)$ should approximate.

Exploring the Environment

- **Catch 22:**
  1. Get a policy by probing $Q^*(s, a)$
  2. Approximate $Q^*(s, a)$ by observing transitions
  3. Observe transition by acting on the policy
- **Solution: Bootstrapping with ϵ-greedy policy**
  - Start with some function $Q_\theta(s, a)$
  - With a small probability ϵ, choose a random action
  - Otherwise act by $\arg\max_a Q_\theta(s, a)$
- As we observe more data, $Q_\theta(s, a)$ should become more accurate, which gives better actions, which hopefully converges to an optimal policy.

  (No guarantees in this setting.)

Deep Q-Learning (DQN)

- $Q_\theta(s, a)$ is represented as a neural network.
- Split into two networks, the training network θ and the **target** network $\bar{\theta}$.
- Optimize by gradient descent of the Mean Squared Error (MSE) of training estimate and the target estimate

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}_{(s,a,r,s') \sim B} \left[\left(r + \gamma Q_{\bar{\theta}}(s', \arg\max_{a'} Q_{\bar{\theta}}(s', a')) - Q_\theta(s, a) \right)^2 \right] \tag{5}$$

where α is the learning rate and B is a replay memory of previous interactions.

- Every so often, update target weights: $\bar{\theta} \leftarrow \theta$

Overview of RL Tools in Practice

- Python is standard in ML, but often just a wrapper around optimized low-level code
- Tools that cover three aspects of RL: *environments*, *learning*, and *statistics*. Examples:
  - Environments: **Gymnasium**
  - Learning: **PyTorch**
  - Statistics: **TensorBoard**
- Will briefly cover Gymnasium on the next slide. PyTorch and TensorBoard only shown in practice.

Gymnasium

Overview

- A standardized API for RL interaction
- Provides a set of standard environments
  - Analogous to MNIST, CIFAR-10 (etc.) datasets for supervised learning
- Originally provided by OpenAI
- Recently handed off to the Farama Foundation
  `https://gymnasium.farama.org`

Gymnasium

Interface example

```python
import gymnasium as gym
env = gym.make("Pendulum-v1") # Off-the-shelf environment

s, _ = env.reset() # Reset environment, get initial state
while True:
  a = your_policy(s) # pi(a|s)

  # Observe p(s'|s,a) and r(s,a) from environment
  s_next, r, terminated, truncated, _ = env.step(a)
  if terminated or truncated:
    break

  s = s_next
```