RL Crash Course

Eric Ding

November 2022

1 Introduction

In this Project, I followed OpenAI deep reinforcement learning course, and used Spinning Up for exercises and experiments.

2 What's Omitted

Regularization, Observation normalization

3 Key Concepts

The main characters of RL are the agent and the environment. The environment is the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.

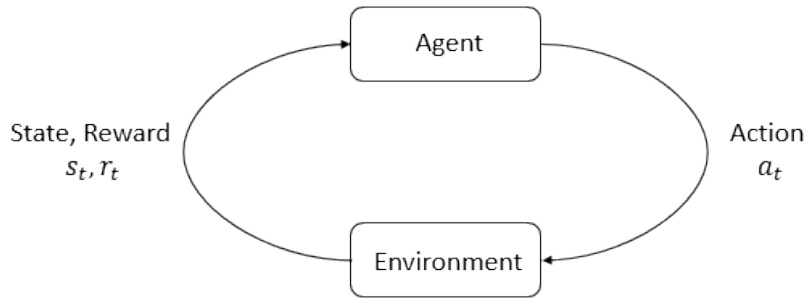

Figure 1: RL framework

The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent

is to maximize its cumulative reward, called return. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.

States and observations: represented by matrix

Action Spaces: the set of all valid actions. Discrete / continuous.

Policies: a rule used by an agent to decide what actions to take. Deterministic: $a_t = \mu_\theta(s_t)$. Stochastic: $a_t \sim \pi_\theta(\cdot|s_t)$. In deep RL, we deal with parametrized policies, with parameters that can be adjusted by optimization.

Stochastic policies: categorical policies in discrete action space and diagonal gaussian policies in continuous action spaces. Two key computations: sampling actions from the policy, and compute log likelihoods of particular actions.

Trajectory: A sequence of states and actions in the world. Can be deterministic or stochastic based on policy.

Reward and return: $r_t = R(s_t, a_t, s_{t+1})$. Maximize cumulative return $R(\tau)$. Finite-horizon undiscounted return: $R(\tau) = \Sigma_{t=0}^{T} r_t$. Infinite-horizon discounted return: $R(\tau) = \Sigma_{t=0}^{\infty} \gamma^t r_t$

The goal in RL is to select a policy which maximizes expected return when the agent acts according to it. The probability of a trajectory:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

The expected return is:

$$J(\pi) = \int_\tau P(\tau|\pi) R(\tau) = E_{\tau\sim\pi}[R(\tau)]$$

The central optimization problem in RL can then be expressed by

$$\pi^* = \arg\max_\pi J(\pi)$$

Value functions: it's useful to know the value of a state, which is the expected return if start in that state. 1) On-Policy Value Function $V^\pi(s)$, start in state s and always acts according to policy π. 2) On-Policy Action-Value Function $Q^\pi(s, a)$, start in state s, take arbitrary action a, and then acts according to policy π. 3) Optimal Value Function $V^*(s)$, start in state s and acts according to the optimal policy. 4) Optimal Action Value Function $Q^*(s, a)$.

The Optimal Q-Function and the Optimal Action: $a^*(s) = \arg\max_a Q^*(s, a)$

Bellman Equations:

$$V^\pi(s) = E_{a\sim\pi, s'\sim P}[r(s, a) + \gamma V^\pi(s')]$$

$$Q^\pi(s, a) = E_{s'\sim P}[r(s, a) + \gamma E_{a'\sim\pi}[Q^\pi(s', a')]]$$

$$V^*(s) = \max_a E_{s'\sim P}[r(s, a) + \gamma V^*(s')]$$

$$Q^*(s, a) = E_{s'\sim P}[r(s, a) + \gamma \max_{a'} Q^*(s', a')]$$

Advantage Function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

4 Kinds of RL Algorithm

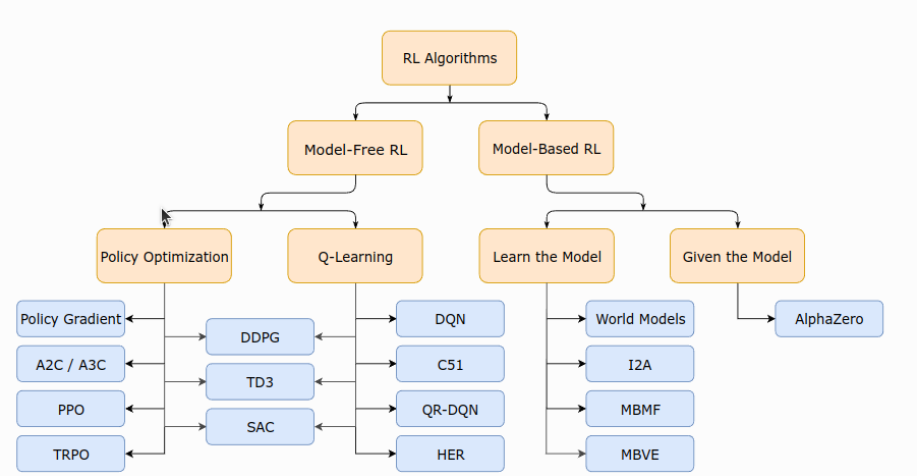

Figure 2: RL Algorithm

4.1 Model-Free vs. Model-Based

Whether the agent has access to (or learns) a model of the environment. The model is a function that predicts state transitions and rewards.

Model allows the agent to think ahead. Such a model is usually not available to the agent. If there is no such models, it has to learn the model purely from experience. The biggest challenge is that bias in the model can be exploited by the agent, resulting in an agent which performs well with respect to the learned model, but behaves sub-optimally (or super terribly) in the real environment.

While model-free methods forego the potential gains in sample efficiency from using a model, they tend to be easier to implement and tune.

4.2 What to Learn

1. Policies
2. Action-value functions
3. Value functions
4. Environment models

4.3 What to Learn in Model-Free RL

1. Policy Optimization: $\pi_\theta(a|s)$. It optimizes the parameters θ either directly by gradient ascent on the performance objective $J(\pi_\theta)$, or indirectly, by

maximizing local approximations of $J(\pi_\theta)$. This is always performed on-policy, each update only uses data collected while acting according to the most recent version of the policy. Also involves learning an approximator $V_\phi(s)$ for the on-policy value function $V^\pi(s)$

2. Q-Learning: Learn an approximator $Q_\theta(s, a)$ for the optimal action-value function $Q^*(s, a)$. This optimization is almost always performed off-policy, each update can use data collected at any point during training. $a(s) = \arg\max_a Q_\theta(s, a)$

Policy Optimization is more stable and reliable, but less sample efficient.

4.4 What to Learn in Model-Based RL

1. Background: Pure Planning. Each iteration, the agent observes the model, computes a plan with respect to the model (use learned value function)

2. Expert iteration: Using and learning an explicit representation of the policy. Uses a planning algorithm to generate candidate actions for the plan by sampling from the current policy.

3. Data augmentation for Model-Free methods

4. Embedding planning loops into policies

5 Toy Example

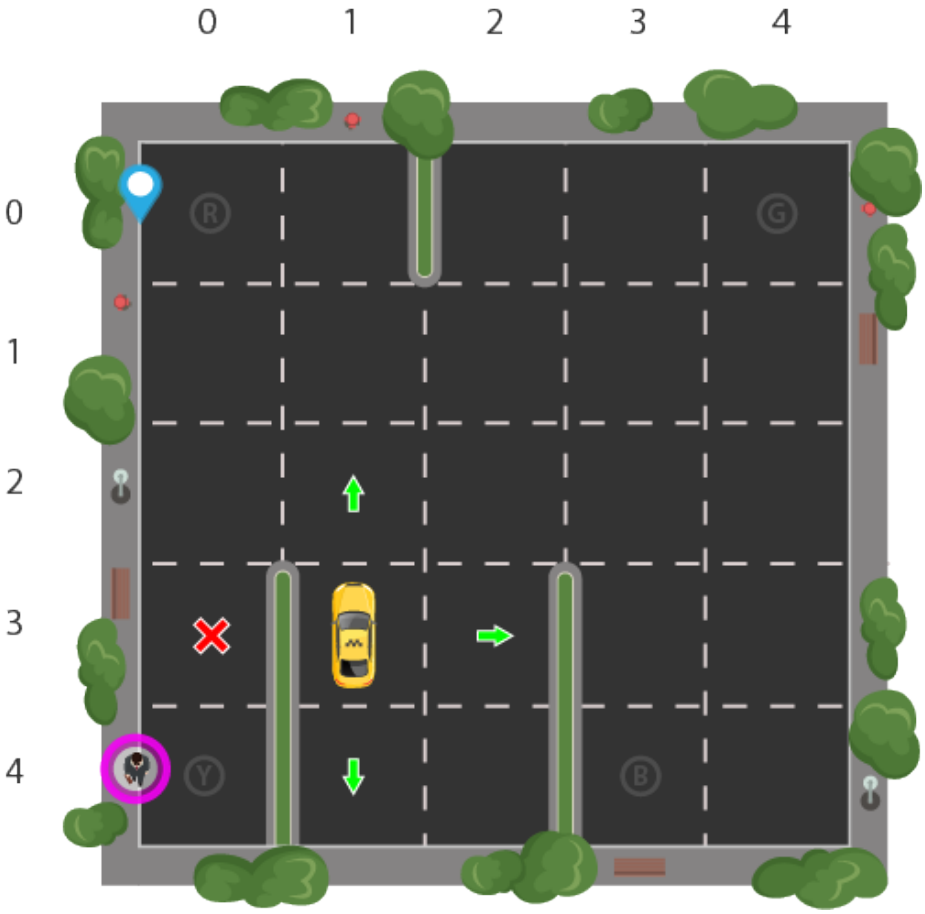

Figure 3: Taxi!

We have an Action Space of size 6 (Four directions, pickup and dropoff) and a State Space of size 500 ($5 \times 5 \times 5 \times 4$).

Here's our restructured problem statement (from Gym docs): "There are 4 locations (labeled by different letters), and our job is to pick up the passenger at one location and drop him off at another. We receive +20 points for a successful drop-off and lose 1 point for every time-step it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions."

Q-learning: A Q-value for a particular state-action combination is representative of the "quality" of an action taken from that state. Better Q-values imply better chances of getting greater rewards.

$$Q^*(s,a) = E_{s' \sim P}[r(s,a) + \gamma \max_{a'} Q^*(s',a')]$$

The Q-value of a state-action pair is the sum of the instant reward and the discounted future reward (of the resulting state). The way we store the Q-values for each state and action is through a Q-table (500×6 matrix)

Explore vs. Exploit

5.1 Training

1. Initialize the Q-table by all zeros.
2. Start exploring actions: For each state, select any one among all possible actions for the current state (S). (Explore vs Exploit)
3. Travel to the next state (S') as a result of that action (a).
4. For all possible actions from the state (S') select the one with the highest Q-value.
5. Update Q-table values using the equation.
6. Set the next state as the current state.
7. If goal state is reached, then end and repeat the process.

5.2 Running

After enough random exploration of actions, the Q-values tend to converge serving our agent as an action-value function which it can exploit to pick the most optimal action from a given state.

5.3 Hyperparameters and Optimization

The values of α, γ, and ϵ were mostly based on intuition and some "hit and trial", but there are better ways to come up with good values.

Ideally, all three should decrease over time because as the agent continues to learn, it actually builds up more resilient priors

5.4 Limitations

Q-table can be large when there are more states.

Solution: 1) Deep neural network to learn action-value function. 2) Use CNN, PCA, etc. to extract state information.

6 Policy Optimization

Want to maximize the expected return $J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$. $R(\tau)$ gives the finite-horizon undiscounted return.

We would like to optimize the policy by gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}.$$

How to numerically compute the policy gradient? This involves two steps: 1) deriving the analytical gradient of policy performance, which turns out to have the form of an expected value, and then 2) forming a sample estimate of that expected value, which can be computed with data from a finite number of agent-environment interaction steps.

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) && \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) && \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) && \text{Log-derivative trick} \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta) R(\tau)] && \text{Return to expectation form} \\
\therefore \nabla_\theta J(\pi_\theta) &= \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right] && \text{Expression for grad-log-prob}
\end{aligned}$$

Figure 4: Policy Gradient

The policy gradient can be estimated with:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau),$$

$\mathcal{D} = \{\tau_i\}_{i=1,\ldots,N}$ is the set of trajectories.

Delete the path rewards in a trajectory, which is unecessary.

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}).$$

The policy gradient has the general form,

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t,$$

where Φ_t can be the form of,

$$\Phi_t = R(\tau),$$

or

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}),$$

or

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t).$$

7 Ideas

7.1 Observation

Fully observed environment vs. partially observed

7.2 Action

Rule out certain actions in action space, can combine both stochastic and deterministic policy.

7.3 Value

How to calculate value functions (future) and adjust policy accordingly (optimize)?

8 What to Explore Next

1. ChatGPT and its implication (Would human programmers, graphic designers, data analysists, etc. all become AI trainers?)
2. Other Reinforcement Learning methods
3. Application in computer hardware and architecture