

Introduction to Deep Reinforcement Learning

https://github.com/racousin/rl_introduction

Raphael Cousin

January 22, 2023

Course Objective

- The keys to go by yourself in RL
- Practice coding
- General culture

What do you already know about RL?

0.1)Intro



Figure 1: 2017, AlphaGo beat Ke Jie, the number one ranked player in the world

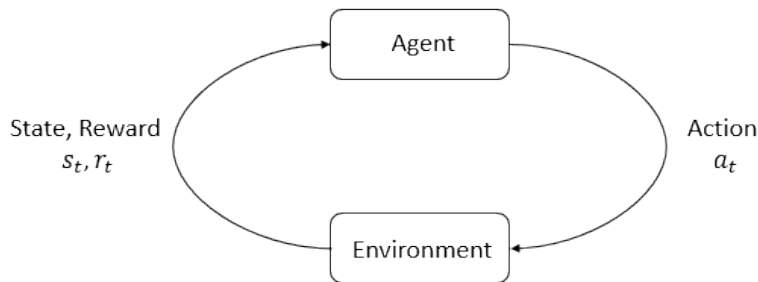
0.2) RL!=ML?

- Supervised learning: $(X, Y) \rightarrow \hat{f}(X) = Y$
- Unsupervised learning: $X \rightarrow \hat{f}(X|\hat{Y})$
- Reinforcement learning (RL): environment/goal \rightarrow optimally interact with the environment to achieve the goal

0.3) Agent takes action in an Environment

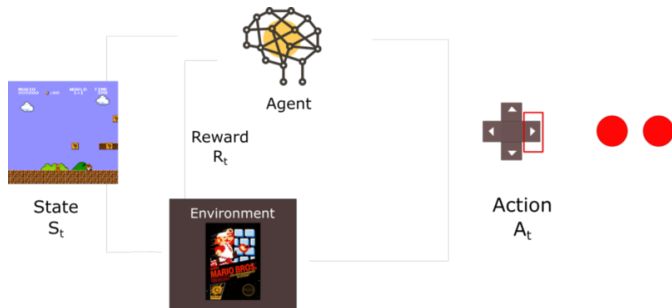
RL framework:

- Unknown “correct” actions → no explicit supervision for a learning algorithm to try to mimic.
- Provide our algorithms only a reward function → indicates to the learning agent when it is doing well, and when it is doing poorly.



0.4) Application

Applications: Autonomous vehicle, robot legged locomotion, cell-phone network routing, marketing strategy, factory control, efficient web-page indexing, video games, Neural Architecture Search, ...



- I) MDP - The RL framework
- II) RL - Model free
- III) Deep RL - Model free

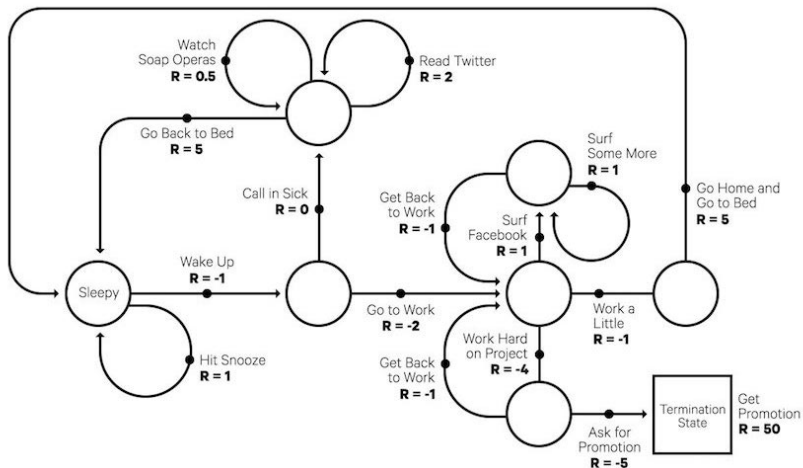
I) MDP and dynamic Programming

I.1) Markov Decision Process

- State space: S
- Action space: A
- Transition model: $P_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- Immediate reward $R_a(s, s') = R_{t+1} \in \mathbb{R}$
- Policy function $\pi(s) \in A$

$\forall s, s \in S, a \in A, t \in \mathbb{N}$

I.2) Example: MDP long term reward



I.3) The MDP Objective

- Find the optimal policy:

$$\pi^* = \arg \max_{\pi} E[G]$$

- Finite-horizon return: $G_t = \sum_{k=0}^T R_{t+k+1}$.

- Infinite-horizon discounted return: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

$$\gamma \in]0, 1[, t \in \mathbb{N}$$

I.4) Trajectories

- Trajectories: $\tau = (s_0, a_0, s_1, a_1, \dots)$
- States follow:
 $s_0 \sim \rho_0(\cdot)$
 $s_{t+1} \sim P(\cdot | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = P(\cdot | s_t, a_t)$
- Actions follow: $s_t \sim \pi(s_t)$
- So probability of a T -step trajectory is:
 $\mathbb{P}(\tau | \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} | s_t, a_t) \pi(a_t | s_t)$
- $\pi^* = \arg \max_{\pi} E_{\tau \sim \pi} [G(\tau)] = \int_{\tau} \mathbb{P}(\tau | \pi) G(\tau)$

Coding session

Environment and Agent

I.5) Value and Action-Value Functions

- Value Function: $V^\pi(s) = E_{\tau \sim \pi}[G_t | s_t = s]$
- Action-Value Function:
 $Q^\pi(s, a) = E_{\tau \sim \pi}[G_t | s_t = s, a_t = a]$
- The Optimal Value Function:
 $V^*(s) = \max_\pi E_{\tau \sim \pi}[G_t | s_t = s] = E_{\pi^*}[G_t | s_t = s]$
- Optimal Action-Value Function:
 $Q^*(s, a) = \max_\pi E_{\tau \sim \pi}[G_t | s_t = s, a_t = a] = E_{\pi^*}[G_t | s_t = s, a_t = a]$

I.6) Bellman Equations

- **Idea :** The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.

$$V(s) = \mathbb{E}[G_t | S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

$$Q(s, a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a]$$

I.7) Bellman Equations development

$$V_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_{\pi}(s, a)$$

$$Q_{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_{\pi}(s')$$

$$V_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_{\pi}(s') \right)$$

$$Q_{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_{\pi}(s', a')$$

I.8) Bellman Equations Optimality

Bellman equations for the optimal value functions

$$V_*(s) = \max_{a \in \mathcal{A}} Q_*(s, a)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_*(s')$$

$$V_*(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_*(s') \right)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \max_{a' \in \mathcal{A}} Q_*(s', a')$$

I.9) The MDP Solution

Dynamic Programming allows to resolve the MDP optimization problem ($\pi^* = \arg \max_{\pi} E_{\tau \sim \pi}[G(\tau)]$). It is an iterative process:

- Policy initialization
- Policy evaluation
- Policy improvement

I.10) Policy evaluation

Policy Evaluation: compute the state-value V_π for a given policy π :
We initialize V_0 arbitrarily. And we update it using:

$$\begin{aligned} V_{k+1}(s) &= \mathbb{E}_\pi[r + \gamma V_k(s_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} P(s', r|s, a) (r + \gamma V_\pi(s')) \quad (1) \end{aligned}$$

$V_\pi(s)$ is a fix point for (1), so if $(V_k)_{k \in \mathbb{N}}$ converges, it converges to V_π .

I.11) Policy Improvement

Policy Improvement: generates a better policy $\pi' \geq \pi$ by acting greedily. Compute Q from V ($\forall a, s$):

$$\begin{aligned} Q_{\pi}(s, a) &= \mathbb{E}[R_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s', r} P(s', r | s, a) (r + \gamma V_{\pi}(s')) \end{aligned}$$

Update greedily: $\pi'(s) = \arg \max_{a \in \mathcal{A}} Q_{\pi}(s, a)$ ($\forall s$)

I.12) Dynamic Programming

Policy Iteration: iterative procedure to improve the policy when combining policy evaluation and improvement.

$$\pi_0 \xrightarrow{\text{evaluation}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluation}} \dots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluation}} V_* \quad (1)$$

Take home message

Initialize $\pi(s), \forall s$

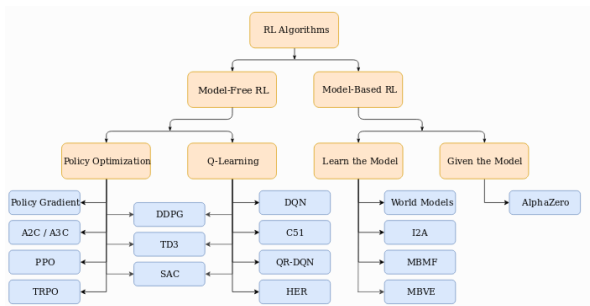
- 1 Evaluate $V_\pi(s), \forall s$ (using $\mathbb{P}_{ss'}^a$)
- 2 Compute $Q_\pi(s, a), \forall s, a$ (using $\mathbb{P}_{ss'}^a$)
- 3 Update $\pi'(s) = \max_a Q_\pi(s, a), \forall s$
- 4 While $\pi'(s) \neq \pi(s)$ do $\pi(s) = \pi'(s)$ and iterate

Result : $\pi = \arg \max_\pi E[G]$

Coding session

Dynamic Programming

I.18) Bestiary of RL Algorithms



To simplify:

- **Model free:** large data + low complexity
- **Model based:** less data + high complexity

II.1) Model-Free

The objective is the same as MDP:

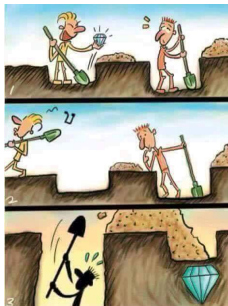
- $J(\pi) = \int_{\tau} \mathbb{P}(\tau|\pi)G(\tau) = E_{\tau \sim \pi}[G(\tau)]$
- The optimization problem:
$$\pi^* = \arg \max_{\pi} J(\pi)$$

But we don't know the transition model P . The constraint is therefore to interact intelligently with the environment to obtain the information needed to solve the problem.

- **Q-learning:** learn the action value function Q
 $(\pi'(s) = \arg \max_{a \in \mathcal{A}} Q_{\pi}(s, a))$
- **Policy Optimization:** learn directly the policy π

II.2) Exploration-Exploitation

Knowledge of the environment comes from interaction. There are trade-offs to be made between using what we know and further exploration.



Naive solution: force exploration with random action (control by an ϵ factor)

II.3) Definition

- **On-policy:** Use the deterministic outcomes or samples from the target policy to train the algorithm.
- **Off-policy:** Training on a distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy.

II.4) Monte-Carlo

- To evaluate $V_\pi(s) = E_{\tau \sim \pi}[G_t | s_t = s]$
- Generate an episode with the policy π $S_1, A_1, R_2, \dots, S_T$ to compute $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$.
- The empirical value function is : $V_\pi(s) = \frac{\sum_{t=1}^T \mathbb{I}[S_t=s] G_t}{\sum_{t=1}^T \mathbb{I}[S_t=s]}$
- As, well, the empirical action-value function is :
- $Q_\pi(s, a) = \frac{\sum_{t=1}^T \mathbb{I}[S_t=s, A_t=a] G_t}{\sum_{t=1}^T \mathbb{I}[S_t=s, A_t=a]}$

II.5) Monte-Carlo Algorithm

Initialize Q $Q(s, a) \forall s, a$.

① Generate an episode with the policy π (extract from Q ϵ -greedy)

② Evaluate Q using the episode:

$$q_{\pi}(s, a) = \frac{\sum_{t=1}^T (\mathbb{1}[S_t=s, A_t=a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1})}{\sum_{t=1}^T \mathbb{1}[S_t=s, A_t=a]}$$

③ Improve the policy greedily: $\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$

④ Iterate

II.6) Temporal difference/bootstrapping

- Remember $V(S_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$
- So $R_{t+1} + \gamma V(S_{t+1})$ is an unbiased estimate for $V(S_t)$
- As well $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ is an unbiased estimate for $Q(S_t, A_t)$
- $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target.
- α improvement:
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

This observation motivates the following algorithm.

II.7) SARSA Algorithm

Initialize Q function $Q(s, a) \forall s, a$

S_t = initial state, act with π to get A_t, R_{t+1}, S_{t+1}

- 1 Act with π to get $A_{t+1}, R_{t+2}, S_{t+2}$
- 2 Evaluate Q using the observation step:
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$
- 3 Improve the policy greedily: $\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$
- 4 $A_t = A_{t+1}, R_{t+1} = R_{t+2}, S_{t+1} = S_{t+2}$. Iterate

II.8) Q-learning

- Remember

$$Q_{\pi^*}(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a]$$

- So $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is an unbiased estimate for $Q(S_t, A_t)$
- $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is called the Q target.
- α improvement:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

This observation motivates the following algorithm.

II.9) Q-learning Algorithm

Initialize Q function $Q(s, a) \forall s, a$

$S_t = \text{initial state}$

- ① $Q : S \times A \rightarrow \mathbb{R}$
- ② $\forall s, a Q(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$
- ③ Improve the policy greedily: $\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$
- ④ Iterate

Coding session

Temporal Difference

II.10) Policy Optimization

- Parametrization of policy, π_θ .
- We aim to maximize the expected return $J(\pi_\theta) = E_{\tau \sim \pi_\theta}[G(\tau)]$.
- Gradient ascent:
$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}.$$
- We can proof that:
$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)]$$

<https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>

I.4) Introduction - Q Learning limitations

- S high dimension \rightarrow low convergence
- It doesn't work for continuous action space

I.4) Introduction - Q Learning limitations

- S high dimension \rightarrow low convergence
- It doesn't work for continuous action space

II.1) Deep Q Learning

Parametrize Q with θ , initialize $\theta \in \mathbb{R}^d$

- $Q_\theta : S \times A \rightarrow \mathbb{R}$
- Objective find $\theta^* \in \mathbb{R}^d \forall s, a \ Q_{\theta^*}(s, a) = \mathbb{E}_\pi^*[G_t | S_t = s, A_t = a]$
- $Q_{\pi^*}(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a]$
- $y = R_{t+1} + \gamma \max_{a'} Q_\theta(S_{t+1}, a')$ is called the Q target.
- Loss (eg MSE): $L(\theta) = \mathbb{E}_{s, a \sim Q}[(y - Q(s, a, \theta))^2]$

II.2) Deep Q Learning Algorithm

Initialize $\theta \in \mathbb{R}^d$, get initial state S_t

- ① $\forall a$ predict/compute $Q_\theta(S_t, a)$
- ② Act greedily $A_t = \arg \max([Q_\theta(s, a_0), Q_\theta(s, a_1), \dots, Q_\theta(s, a_{\dim(A)})])$
and get R_{t+1}, S_{t+1}
- ③ $\forall a$ predicts $Q_\theta(S_{t+1}, a)$
Compute the target $y = R_{t+1} + \gamma \max_{a \in A} Q_\theta(S_{t+1}, a)$
- ④ Evaluate + improve Q by minimizing the loss
 $L(\theta) = (y - Q(S_t, A_t, \theta))^2$
- ⑤ Iterate

II.3) Convergence and stability improvements

Experience replay

- At every step t , we get $S_t, A_t, R_{t+1}, S_{t+1} \rightarrow$ memory
- New loss at each step

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

- Other improvements: Epsilon decay, Clipping, Double Q learning

Coding session

Deep Q - learning

III.1) Policy Optimization

- Parametrize the policy, π_θ .
- We aim to maximize the expected return $J(\pi_\theta) = E_{\tau \sim \pi_\theta}[G(\tau)]$.
- Gradient ascent:
$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}.$$
- We can proof that:
$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)]$$

<https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>

III.2) Reinforce/VPG algorithm

Initialize policy π_θ

- 1 Generate episodes $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ with the policy π_θ
- 2 Compute gradient approximation
$$\hat{\nabla} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$
- 3 Update policy (apply gradient ascent) $\theta \leftarrow \theta + \alpha \hat{\nabla}$
- 4 Iterate

Coding session

Policy gradient - reinforce

III.3) Improvement Actor-Critic

We can rewrite the policy gradient

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

whith Φ_t could be any of

- $\Phi_t = G_t$
- $\Phi_t = \sum_{t'=t}^T R_{t'+1} - V(s_t)$
- $\Phi_t = \sum_{t'=t}^T R_{t'+1} - Q(s_t, a_t)$

For the last 2 cases we need to estimate V or Q (the critics).

III.4) Actor-Critic Algorithm

Initialize $\theta, \in \mathbb{R}^{d_1}, \phi \in \mathbb{R}^{d_2}$ Get start State S_t

- ① Generate A_t R_{t+1} following π_{θ_t}
- ② Update actor (apply gradient ascent)
$$\theta \leftarrow \theta + \alpha \hat{\nabla} = \theta + Q_{\phi}(a_t, s_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$
- ③ Compute critic target $y = R_{t+1} + \gamma Q_{\phi}(S_{t+1}, a')$
- ④ Evaluate/Improve Q_{ϕ} by minimizing the loss
$$L(\phi) = (y - Q(S_t, A_t, \phi))^2$$
- ⑤ Iterate

References

Course:

<http://incompleteideas.net/book/the-book-2nd.html>

<https://lilianweng.github.io/lil-log/2018/02/19/>

[a-long-peek-into-reinforcement-learning.html](#)

<http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/>

<https://github.com/kengz/awesome-deep-rl>

Framework:

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

<https://gym.openai.com/envs/#atari>

<https://github.com/deepmind/bsuite>

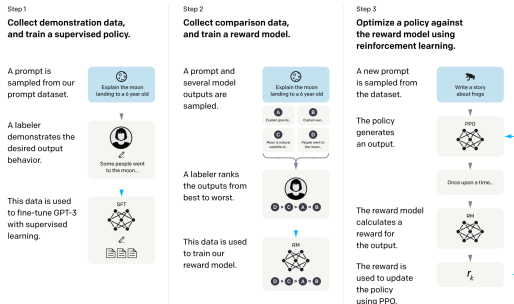


Figure 2: instructgpt-chart-openai