Crash-Introduction to Reinforcement Learning

GIUSEPPE BONACCORSO

Warning!

This is only a crash introduction. This implies:

- Lack of mathematical rigor
- Limited topic discussion

Agenda

- Constitutive elements
- Value iteration
- Time-Difference algorithms
- Actor-Critic
- SARSA
- Q-Learning
- Policy Gradient

Agent

The agent is the entity that acts into an environment to maximize its total reward and achieve its objectives.

An agent:

- Can repeat the same experience an indefinite number of times
- Operates in a time-sequence $t_1, t_2, \ldots \in T$
- Select an action from a set $a_t \in A$
- Expect a reward r_{t+1} because of the action a_t
- Transition from a state to another $s_t \rightarrow s_{t+1} \in S$

Environment

The environment is the component where the agent can perform actions, receive feedbacks, optimize its policy, and reach its goals.

It can be either **deterministic**:

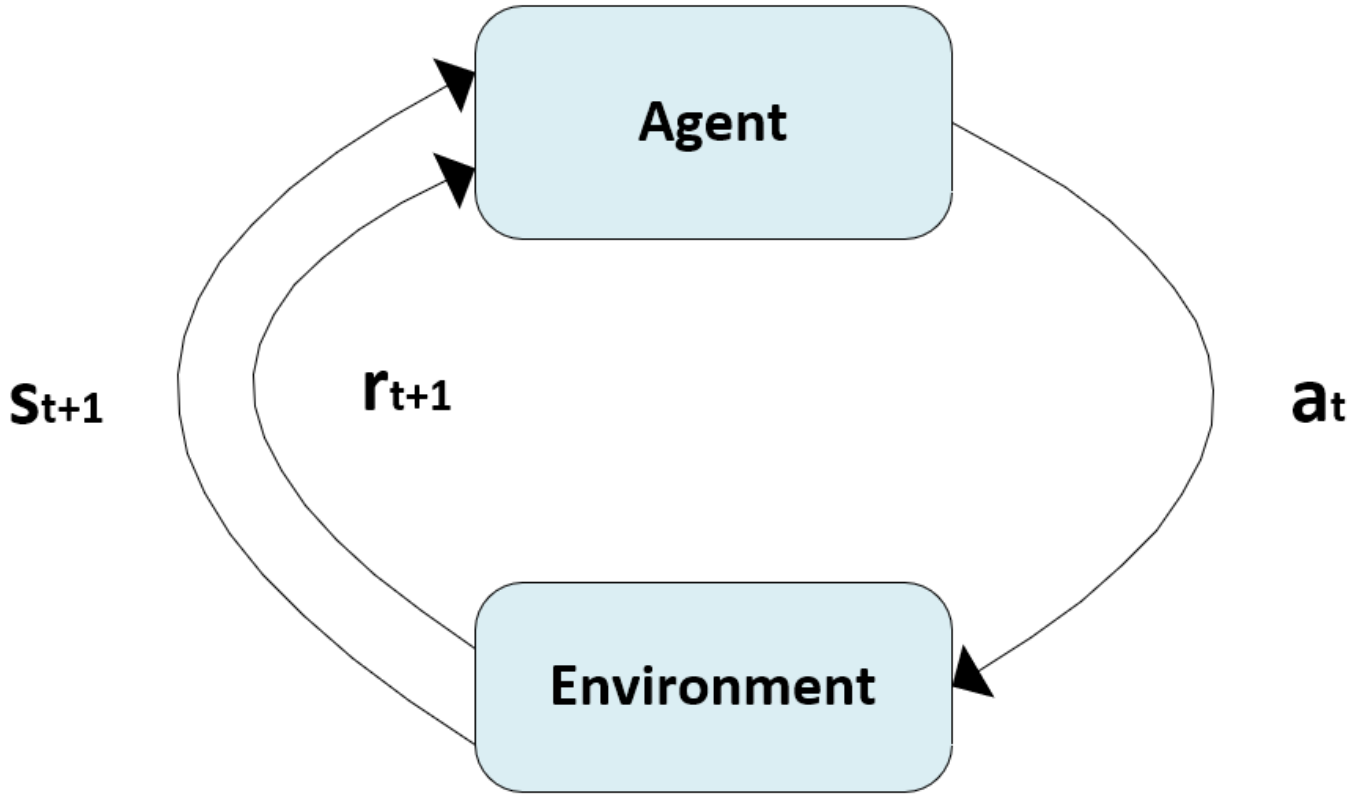
$$(s_{t+1}, r_{t+1}) = f(s_t, a_t) \ \ where \ \ a_t \in A \ \ and \ s_t, s_{t+1} \in S$$

Or **stochastic**:

$$T\left(s_t, s_{t+1}^i, a_t\right) = (p(s_t, s_{t+1}^1, a_t), \dots, p\left(s_t, s_{t+1}^i, a_t\right), \dots)$$

Where $T\left(s_t, s_{t+1}^i, a_t\right)$ is the transition probability from $s_t \ to \ s_{t+1}^i$ if the action a_t is chosen

The interaction between Agent and Environment



Markov Decision Process

Given a stochastic environment, we can consider the sequence:

$$s_1 \rightarrow a_1 \rightarrow (s_2, r_2) \rightarrow \ldots \rightarrow (s_n, r_n) \rightarrow \cdots$$

The transition between stated is governed by $T\left(s_t, s_{t+1}^i, a_t\right)$ which is very similar to a **Markov Chain**.

In this case, the presence of the **action** transforms the evolution of the process into a **decision** one.

Remember that every action is selected after observing the result of the previous transition and the reward.

Rewards

Immediate rewards r_{t+1} are helpful only considering the **shortest horizon**

In order to extend the horizon, it's helpful to include a discount factor $\gamma \in (0,1)$:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{i+t+1} = r_{t+1} + \gamma r_{t+1} + \cdots + \gamma^k r_{k+t+1} + \cdots$$

R_t denotes the discounted reward with an **infinite horizon**.

If $|r_{t+1+i}| \leq |r_{max}|$:

$$|R_t| = \left|\sum_{i=0}^{\infty} \gamma^i \, r_{i+t+1}\right| \leq \sum_{i=0}^{\infty} \gamma^i \, |r_{i+t+1}| \leq |r_{max}| \sum_{i=0}^{\infty} \gamma^i = \frac{|r_{max}|}{1-\gamma}$$

Policy

A policy encodes the behavior of the agent and provides the best action to perform in each state $s_t \in S$.

It can be either **deterministic**:

$$a_{t+1} = \pi(s_t)$$

Or **stochastic**:

$$\pi(s_t) = \left(p\left(a_{t+1} = a^{(1)}\right), \dots, p\left(a_{t+1} = a^{(n)}\right), \dots\right)$$

For our purposes, a policy represents an agent and vice versa. To goal of a RL algorithm is to find an optimal policy for a given environment.

Exploitation vs. Exploration

A stochastic policy allows to:

- Exploit the learned policy
- Explore the environment

If the policy is optimized too quickly, the agent loses the ability to explore the environment and discover better alternatives.

To solve this problem it's possible to employ an ϵ-**greedy policy**:

$$\pi(s_t) = \begin{cases} \text{argmax}\,\pi(s_t) \;\; with\; p = 1 - \epsilon \\ s_t \sim \pi \;\; with\; p = \epsilon \end{cases} \quad and \;\; \lim_{t \to \infty} \epsilon_t = 0$$

Value of a state

A state s_t can be associated with a **value**:

$$V(s_t;\pi) = E_\pi[R_t; s_t] = E_\pi\left[\sum_{i=0}^{\infty} \gamma^i \, r_{i+t+1}; s_t\right]$$

Intuitively: finding the value of each s_t under π allows to optimize the policy

More formally: It's possible to turn the expression into a Bellman equation and use a dynamic programming approach to iteratively optimize it (*proof is omitted*)

Value iteration

Given a generic stochastic environment characterized by $T(s_t, s_k; a_k)$, a simple algorithm is based on the following update rule:

$$V^{(i+1)}(s_t) = \max_{a_t} \sum_{s_k} T(s_t, s_k; a_k)\left[E[r_{t+1}; s_k, a_k] + \gamma V^{(i)}(s_k)\right]$$

It's possible to prove that $V^{(\infty)} \rightarrow V^{(opt)}$ (*proof is omitted*)

The main drawback of value iteration is the absence of the action in the value function.

Q-function

The value of a state s_t has the disadvantage not to include the action a_t that triggered the transitions. The problem is solved by defining another proxy function:

$$Q(s_t, a_t; \pi) = E_\pi[R_t; s_t, a_t] = E_\pi\left[\sum_{i=0}^{\infty} \gamma^i r_{i+t+1}; s_t, a_t\right]$$

Also in this case, we can turn $Q(s_t, a_t; \pi)$ into a Bellman equation:

$$Q(s_t, a_t; \pi) = \sum_{s_k} T(s_t, s_k; a_t)[E[r_{t+1}; s_k, a_t] + \gamma V(s_k; \pi)]$$

Policy from Q-function

Given a Q-function, the update step can be obtained as:

$$\pi^{(k+1)}(s_t) = \underset{a_t}{\operatorname{argmax}}\, Q\left(s_t, a_t; \pi^{(k)}\right)$$

Clearly, to optimize the Q-function is necessary to update the $[\ldots + \gamma V(s_k; \pi)]$ term representing the values of the states.

When $k \to \infty$, $\pi(s_t)$ will converge to a fixed point (Policy improvement theorem – *proof is omitted*) representing an optimal policy.

Time Difference (TD) Algorithm

Another simple but effective algorithm is based on the update of the values in a way proportional to their time difference:

$$V^{(t+1)}(s_i) = V^{(t)}(s_i) + \alpha\left(r_{ij} + \gamma V^{(t)}\left(s_j\right) - V^{(t)}(s_i)\right)$$

The constant $\alpha \in (0,1)$ is a *pseudo*-learning rate.

This version is called **TD(0)** as it considers only the immediate reward r_{ij}.

A more generic version is called **TD(λ)** and it's based on more time steps:

$$r_{ij} + \gamma V^{(t)}\left(s_j\right) \;\rightarrow\; R_t^k = r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{k-1} r_{k-1} + \gamma^k V^{(t)}(s_{t+k})$$

Actor-Critic

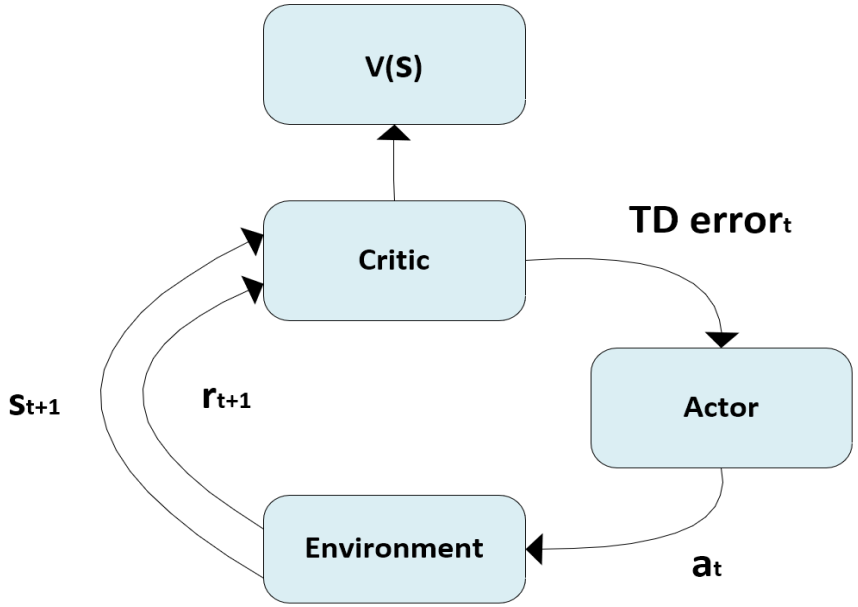
It’s possible to improve the effectiveness of a TD algorithm using a particular approach called Actor-Critic:

Defining the TD error as:

$$TD_{error} = r_{ij} + \gamma V(s_j) - V(s_i)$$

The policy (agent) is split into:

- An **Actor** that selects and performs an action
- A **Critic** that evaluates the estimation of the value



Actor-Critic (Dynamics)

Consider a matrix P, called **Policy Importance**, where each entry $p_i(s,a)$ represents the preference for an action a in every state s.

Let's define an ϵ**-greedy soft policy**:

$$\pi(s,a) = \frac{e^{p_i(s,a)}}{\sum_{a_k} e^{p_i(s,a_k)}} = \frac{e^{-\max_a p_i(s,a)} e^{p_i(s,a)}}{e^{-\max_a p_i(s,a)} \sum_{a_k} e^{p_i(s,a_k)}} = \frac{e^{p_i(s,a)-\max_a p_i(s,a)}}{\sum_{a_k} e^{p_i(s,a_k)-\max_a p_i(s,a)}}$$

Actor-Critic (Dynamics)

Let's now reconsider a transition $s_i \rightarrow (s_j, r_{ij})$ the TD error:

$$TD_{error} = r_{ij} + \gamma V(s_j) - V(s_i)$$

If $V(s_i) < r_{ij} + \gamma V(s_j)$ the Critic considers the action as **positive** and vice versa.

The role of the Critic is immediately reflected in the Policy Importance:

$$p_i(s_i, a) = p_i(s_i, a) + \rho TD_{error}$$

Where ρ is weight parameter. The Actor selects an action as $\pi(s) = \underset{a}{\operatorname{argmax}}\, \pi(s, a)$

SARSA Algorithm

SARSA is a natural extension of TD(0) for the estimation of the Q-function. Given a transition $s_t \rightarrow a_t \rightarrow r_{t+1} \rightarrow s_{t+1} \rightarrow a_{t+1}$, we have:

$$Q(s_t, a_t; \pi) = Q(s_t, a_t; \pi) + \alpha\big(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \pi) - Q(s_t, a_t; \pi)\big)$$

SARSA converges to the optimal policy with p = 1, $p\left(\lim_{k\to\infty} \pi^{(k)}(s) = \pi^{opt}(s)\right) = 1 \ \forall s \in S$ if the following conditions are met:

- The learning rate $\alpha \in (0,1)$ with the constraints $\sum \alpha = \infty$, $\sum \alpha^2 < \infty$
- The variance of the rewards must be finite

Q-Learning

Q-Learning can be considered as an extension/alternative to SARSA. Given the transition $s_t \rightarrow a_t \rightarrow r_{t+1} \rightarrow s_{t+1}$, we have:

$$Q(s_t, a_t; \pi) = Q(s_t, a_t; \pi) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a; \pi) - Q(s_t, a_t; \pi))$$

As you see, in this case, we are considering the term $\max_a Q(s_{t+1}, a; \pi)$, which is the maximum Q obtainable with a potential action (a.k.a. Next Best Action).

Q-Learning converges under the same condition of SARSA

Policy Gradient (Structure)

Policy Gradient is a **direct policy search** algorithm. We assume to have:

- A parametrized policy $\pi(s, \bar{\theta})$
- N sequences of states $S_k = (s_1, s_2, \dots, s_k)$
- A transition probability $p\left(s_i \rightarrow s_j \middle| a_k\right) \forall\, s_i, s_j \in S \; and \; a_k \; in \; A$

We can define the conditional probability as $p(S_k|\pi)$:

$$p(S_k|\pi) = p(s_1) \prod_i p(s_i \rightarrow s_{i+1}|a_i)\pi(a_i|s_i;\, \bar{\theta})$$

Policy Gradient (Objective)

We can define a cost function as:

$$L(\bar{\theta}) = \int p(S_t|\pi)R(S_t)dS_t$$

An agent that maximizes $L(\bar{\theta})$ will find an optimal policy. After discretizing it, we get:

$$L(\bar{\theta}) \approx \frac{1}{N}\sum p(S_t|\pi)R(S_t)$$

At this point, we can compute the gradient:

$$\nabla L(\bar{\theta}) \approx \frac{1}{N}\sum \nabla p(S_t|\pi)R(S_t)$$

Policy Gradient (Optimization)

We can simplify the gradient:

$$\nabla L(\bar{\theta}) \approx \frac{1}{N}\sum \nabla p(S_t|\pi)R(S_t)$$

Consider that:

$$\frac{d}{dx}f(x) = f(x)\frac{1}{f(x)}\frac{d}{dx}f(x) = f(x)\frac{d}{dx}\log f(x)$$

So, by applying this transformation, we get:

$$\nabla L\left(\bar{\theta}\right) \approx \frac{1}{N}\sum p(S_t|\pi)\nabla \log p(S_t|\pi)\, R(S_t) = \frac{1}{N}\sum p(S_t|\pi)\sum_i \nabla \log \pi\left(a_i\middle|s_i;\ \bar{\theta}\right) R(S_t)$$

Thank you!

Sorry for the boredom!

References

- *Sutton R. S., Barto A. G., Reinforcement Learning, The MIT Press, 1998*
- *R. A. Howard , Dynamic Programming and Markov Process, The MIT Press, 1960*
- *Dayan P., Abbott L. F., Theoretical Neuroscience, The MIT Press, 2005*
- *Bonaccorso G., Mastering Machine Learning Algorithms (2nd ed.), Packt, 2020*