

# Policy Gradient Methods for Deep Reinforcement Learning

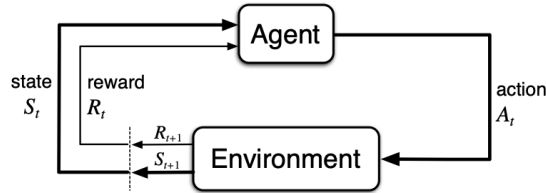
Hongshan Li(hongshal@amazon.com)

August 11, 2021

RL is big and rich, its history traces back to Pavlov's study on the psychology of animal learning (conditional response) back in 1902. The expository article History of Reinforcement Learning (<http://incompleteideas.net/book/first/ebook/node12.html>) is a good resource to learn how RL evolved in the last 100 years.

Policy Gradient Methods is a class of RL algorithms that directly learns a policy from interaction with the environment. In contrast, to this, one can also learn a state value estimate of the current state of the env and the candidate actions. This class of algorithms is called Q-learning.

**Review about setups in RL** The objective of RL is to find a policy  $\pi$  that gets the highest rewards from a *Markov Decision Process*



At time step  $t$ , the env is at state  $S_t$ , the agent takes action  $A_t$ , the env transition into  $S_{t+1}$  and the agent receives a reward of  $R_{t+1}$ .

The critical assumption of MPD is

- The transition probability  $P(S_{t+1}, R_{t+1}|S_t, A_t)$  is independent from the history
- The agent's action does not alter the transition probability, i.e.  $P(S_{t+1}, R_{t+1}|S_t, A_t)$  is the same whenever the agent sees  $S_t$  and takes action  $A_t$

The transition probability  $P(S_{t+1}, R_{t+1}|S_t, A_t)$  is call the *model* of the env. The objective of RL is to find a *optimal policy* (what action to take at  $S_t$ ) that maximize the total reward

If  $P(S_{t+1}, R_{t+1}|S_t, A_t) = 1$  for all  $S$  and  $A$ , then we say the env is deterministic, otherwise stochastic.

The total reward from time  $t$  is

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

The MDP terminates at  $T$ .

To prioritize short term return over long term return, we use a discount factor  $\gamma \in [0, 1]$  and define

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^k R_{t+k+1} + \cdots + \gamma^{T-t-1} R_T$$

A few ways to think about the discount factor

- prioritize short term reward
- A mathematical convenience for infinite episodic task; we want  $G_t$  to be a finite number even if the MDP never stops.

$$\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$$

so

$$G_t \leq \frac{1}{1-\gamma} * R_{\max}$$

- a way to factor in the probability that agent dies. The agent survives with prob  $\gamma$  and die with prob  $1-\gamma$ . i.e reward at time  $t$  is

$$\gamma R_{t+1} + (1-\gamma) * 0$$

Let  $\mathcal{S}$  be the state space, and  $\mathcal{A}$  the action space. A *deterministic policy* is a mapping

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

and a *stochastic policy* is a mapping

$$\pi : \mathcal{S} \rightarrow \text{Dist}(\mathcal{A})$$

where  $\text{Dist}(\mathcal{A})$  denote the space of probability distributions over the action spaces.

Both deterministic env and deterministic policy are special cases of their stochastic counterparts. (all probability concentrate at 1 point). So we only address the stochastic cases in the rest of the note.

Suppose the agent's current policy is  $\pi$ , then at state  $S_t$   $\pi$  yields a probability distribution over the action space

$$\pi(\cdot | S_t)$$

from which the agent samples actions.

So with the model of the env, we can write down the expected total reward if we follow the policy  $\pi$ . Assume the initial state is  $s_0$

$$v_{\pi}(s_0) := \mathbb{E}_{\tau \sim \pi}[G_0 | S_0 = s_0]$$

$$\mathbb{E}_{\tau \sim \pi}(R_1 + \gamma v_{\pi}(s_1))$$

$s_1$  denotes the next state of the env.  $\tau$  denotes a trajectory sampled from the policy  $\pi$ .

We can use Monte Carlo method to estimate  $v_\pi(s_0)$  by running the policy a couple of times and compute the average return  $G_0$ .

**Policy Iteration** How do we improve the policy? That's the entire subject of RL. In supervised learning, the objective is to find a parametric model  $p_\theta$  that maximize the likelihood of the input dataset; in RL, the objective is to train a policy that gets the highest reward from the env.

A fundamental difference supervised learning and RL is that for SL data is completely determined before any training, whereas for RL data is dynamically generated through interaction with the environment. This means crappy policy could generate useless data by visiting the useless state space. The distribution of data for RL changes over time, whereas for SL the distribution is static.

This might sounds pretty hopeless, how do you overcome the garbage in and garbage out positive feedback loop? Just hang on.

Suppose we have a value function  $v_\pi$  for the current policy  $\pi$ , one naive way to improve  $\pi$  is to "look ahead one step of time". At  $t$ , we are at  $S_t = s$ , the policy results an action  $\pi(s)$ , we want whether it is a good idea to choose a different action  $a \neq \pi(s)$ ? We can simply take  $a$  and follow  $\pi$  afterwards. This is defined as the state-action value

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

Recall

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

If  $q_\pi(s, a) - v_\pi(s) > 0$ , then it is a good idea to assume action  $a$  at  $s$ .

This gives us a template for generalized policy iteration

**Algorithm 1:** Generalized Policy Iteration

- 1: Input: initial policy  $\pi$
- 2: **repeat**
- 3:   evaluate the policy  $\pi$  and get  $v_\pi$
- 4:   **for** each  $s \in \mathbb{S}$  **do**
- 5:      $oldAction \leftarrow \pi(s)$
- 6:      $\pi(s) \leftarrow \operatorname{argmax}_a \mathbb{E}[r + \gamma v(s_1)]$
- 7:   **end for**
- 8: **until**  $\pi$  stops improving, i.e  $oldAction = \pi(s) \forall s$

So this is what you do to overcome the garbage in and garbage out. You can get a good policy by iterating one-step a time.

# 1 Value Iteration and Exact Methods

## 2 Function Approximation and Policy Gradient Methods

The above generalized policy iteration works pretty well if the underlying MDP is finite (finitely states). One gets  $v_\pi$  by dynamic programming. But this becomes intractable if the state space is big. Suppose your state is given as a 2D image with dimension 512x512, then your state space is of size  $(3 \times 512 \times 512)^{256} \sim 1.5 \times 10^{1510}$ .

Function approximation is the idea that addresses the curse of dimension problem when the state space gets big (or even infinite). The idea is if two states are "similar" the agent should take the similar actions and the value of that states should be similar as well. So we can use parametrized functions  $v_\theta$  and  $\pi_\theta$  to represent the value and policy tables. The differentiability of  $v_\theta$  and  $\pi_\theta$  ensures that similar states have similar value and the agent acts similarly.

Now we are officially in the Deep Reinforcement Learning (DRL) world. **Policy gradient methods** concerns about the following question: How do we move the parameters of  $\pi_\theta$  so that  $\pi_\theta$  improves. "Move" in this context means updating  $\theta$  according to the *policy gradient*.

In tabular method, we have defined the value of a policy. Analogous definition carries over

$$J(\theta) = v_{\pi_\theta}(s)$$

To improve  $J(\theta)$ , we do the usual calculus trick (gradient ascend)

$$\theta \leftarrow \theta + \alpha \nabla J(\theta)$$

This is basically the entirety of policy gradient methods.

So the next thing is how to compute  $\nabla J(\theta)$

**Theorem 1 (Policy Gradient Theorem)** *Suppose the state space  $\mathbb{S}$  and action space  $\mathbb{A}$  are discrete*

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a)$$

where  $\mu(s)$  is the visitation probability (how often the agent visits state  $s$ )  
If  $\mathbb{S}$  and  $\mathbb{A}$  are continuous

$$\nabla J(\theta) \propto \int_{\mathbb{S}} \mu(s) \int_{\mathbb{A}} \nabla \pi(a|s) q_\pi(s, a) ds da$$

More details about  $\mu(s)$  before moving on the proof. Consider the visitation sequence from state  $s$  to  $x$  following the policy  $\pi$  after  $k$  steps

$$s \xrightarrow{a \sim \pi(\cdot|s)} s_1 \xrightarrow{a \sim \pi(\cdot|s_1)} s_2 \dots$$

(Markov Chain) denote the probability of transitioning from state  $s$  to  $x$  in  $k$  steps as  $\rho^\pi(s \rightarrow x, k)$

Assume for simplicity that we only deal with finite episodic tasks, i.e. only allow the agent to take  $n$  steps of actions for  $n \leq \infty$ .

- $k = 0, \rho^\pi(s \rightarrow s, k = 0) = 1$
- $\rho^\pi(s \rightarrow s_1, k = 1) = \sum_a \pi(a|s)P(s_1|s, a)$
- $\rho(s \rightarrow x, k + 1) = \sum_{s_1} \rho(s \rightarrow s_1, k)\rho(s_1 \rightarrow x, 1)$
- $\eta(x) := \sum_k \rho(s_0 \rightarrow x, k)$
- Define  $\mu(x) := \frac{\eta(x)}{\sum_x \eta(x)}$  if  $\mathbb{S}$  is discrete and  $\mu(x) := \frac{\eta(x)}{\int_{\mathbb{S}} \eta(x)dx}$  if  $\mathbb{S}$  is continuous. So  $\mu(x)$  becomes a probability mass function (or probability distribution). It can be understood as a measure of how frequent the agent lands at state  $x$

With the understanding of  $\mu(x)$  as the visitation distribution (relative importance of the state  $x$ ), how we do interpret the policy gradient?

**Policy gradient is the weighted average of gradient of  $\pi(a|s)$  by the value of action  $a$  and relative importance of the state  $s$**

**Proof 1 (policy gradient theorem)** For notational convenience, write  $v(s)$  for  $v_\pi(s)$  and  $\nabla v(s)$  to mean  $\nabla_\theta v_\pi(s)$ , i.e. gradient of  $v(s)$  with respect to the parameter  $\theta$  of  $\pi$ .

$$\begin{aligned} \nabla v(s) &= \nabla \left[ \sum_a \pi(a|s) q_\pi(s, a) \right] \\ &= \sum_a [\nabla \pi(a|s) q_\pi(s, a)] + \sum_a \pi(a|s) \nabla q_\pi(s, a) \\ q_\pi(s, a) &= \sum_{s_1, r} p(s_1, r|s, a)(r + v(s_1)) \end{aligned}$$

so

$$\begin{aligned} \nabla v(s) &= \nabla \left[ \sum_a \pi(a|s) q_\pi(s, a) \right] \\ &= \sum_a [\nabla \pi(a|s) q_\pi(s, a)] + \sum_a \pi(a|s) \sum_{s_1} p(s_1|s, a) \nabla v(s_1) \end{aligned}$$

**Note the recursive relation between  $\nabla v(s)$  and  $\nabla v(s_1)$**

Write  $\phi(s) = \sum_a \nabla \pi(a|s) q(s, a)$  and  $\rho(s \rightarrow s_1, 1) = \sum_a \pi(a|s) p(s_1|s, a)$

So

$$\begin{aligned}
\nabla v(s) &= \phi(s) + \sum_{s_1} \rho(s \rightarrow s_1, 1) \nabla v(s_1) \\
&= \phi(s) + \sum_{s_1} \rho(s \rightarrow s_1, 1) \phi(s_1) + \sum_{s_2} \rho(s \rightarrow s_2, 2) \nabla v(s_2) \\
&= \phi(s) + \sum_{s_1} \rho(s \rightarrow s_1, 1) \phi(s_1) + \sum_{s_2} \rho(s \rightarrow s_2, 2) \phi(s_2) + \sum_{s_3} \rho(s \rightarrow s_3, 3) \nabla v(s_3) \\
&\vdots \\
&= \phi(s) + \sum_{s_1} \rho(s \rightarrow s_1, 1) \phi(s_1) + \cdots + \sum_{s_n} \rho(s \rightarrow s_n, n) \phi(s_n)
\end{aligned}$$

The unrolling stops after  $n$  steps, because we only let the agent to act  $n$  steps. so  $v(s_{n+1})$  becomes a constant function and it has no derivative.

$$\begin{aligned}
\nabla v(s) &= \sum_x \sum_{k=0}^n \rho(s \rightarrow x, k) \phi(x) \\
&= \sum_x \eta(x) \phi(x) \\
&= \left( \sum_x \eta(x) \right) \sum_x \frac{\eta(x)}{\sum_x \eta(x)} \phi(x) \\
&= \left( \sum_x \eta(x) \right) \sum_x \mu(x) \phi(x) \\
&\propto \sum_x \mu(x) \nabla \pi(a|s) q(s, a)
\end{aligned}$$

Talk about value iteration and policy iteration; Talk about how tabular methods can be used to achieve value/policy iteration; Talk about why tabular method is not sufficient and function approximator; then talk about parametrized policy and policy gradient theorem

## REINFORCE

History of Reinforcement Learning <http://incompleteideas.net/book/first/ebook/node12.html>