

# **Reinforcement Learning**

– Course notes –

# Chapter 1

## Dynamic programming

### 1.1 The value function

The value function is a staple from the literature on dynamic programming, whether it be for discrete or continuous problems (as in control theory). It measures just how good a control  $u$  – or, in our case, a policy  $\pi$  – is regarding the desired target of our problem.

**Definition 1 (Value function)** *The value function  $V^\pi: \mathcal{S} \rightarrow \mathbb{R}$  of a policy  $\pi$  is the expectation of the cumulative (discounted) future rewards starting from a point  $s_0 = s$*

$$V^\pi(s) := \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=0}^T \gamma^t r(s_t, a_t) \mid s_0 = s \right] \quad (1.1)$$

where the trajectory  $\tau$  is generated under the policy  $\pi$ .

This notion can be generalized to the case where the rewards are generated by the transitions  $(s_t, a_t, s_{t+1})$  rather than the (state, action) couple:

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \mid s_0 = s \right]$$

Under a deterministic policy  $\pi: \mathcal{S} \rightarrow \mathcal{A}$  and associated decision rule  $d^\pi(s) = \pi(s)$ , the dynamic programming principle leads to a dynamic programming equation called the **Bellman equation**:

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') V^\pi(s') \quad (1.2)$$

With generalizations:

- for stochastic policies  $\pi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_+$ , the sum becomes  $\sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(s, a) p(s, a, s') V^\pi(s')$
- non-discrete state space, the sum can be replaced by an integral with respect to a measure  $p(s, \pi(s), ds')$  – see Sutton’s book [1]
- if the rewards are given for transitions as  $r(s, a, s')$ , we introduce  $r(s, a) = \sum_{s' \in \mathcal{S}} r(s, a, s')$ .

**Remark 1 (Notational abuse)** Often, we will write the value function at a state  $s_t$ , where it is implied we are at the  $t$ -th step in a trajectory, as

$$V(s_t) = \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t'=t}^T \gamma^{t'-t} r(s_{t'}, a_{t'}) \mid s_t \right]$$

This is coherent with the notion of cumulative (discounted) future rewards which defined the value function (1.1), but not with the notation; a more correct notation would be to write  $V(t, s_t)$  and make the dependence on the starting time  $t$  of the discounting explicit.

### 1.1.1 The optimal value function

Solving a Markov Decision Process involves finding the **optimal** policy  $\pi^*$  that will maximize the expected rewards when starting at a point  $s_0$ .

**Definition 2 (Optimal policy and value function)** Given a set of policies  $\Pi$ , an optimal policy  $\pi^*$ .

The associated optimal value function is

$$V^* := V^{\pi^*}$$

### 1.1.2 The Value Iteration algorithm

## 1.2 The $Q$ -function

**Definition 3 (State-action value function)** The state-action value function of a policy  $\pi$  is the function  $Q^\pi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is defined by

$$Q^\pi(s, a) := \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=0}^T \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \quad (1.3)$$

where the trajectory  $\tau$  is generated under the policy  $\pi$ . The horizon  $T$  of the problem can be finite or infinite ( $T$  can be a stopping time).

The state-action value function  $Q^\pi$  has an obvious link to the value function  $V^\pi$ .

**Proposition 1** For any state  $s \in \mathcal{S}$ , it holds that

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s, \cdot)} [Q^\pi(s, a)] \quad (1.4)$$

## 1.3 Temporal-difference estimation – TD(0)

The real value function  $V^\pi$  satisfies the Bellman equation. This means that the **temporal difference error** of a good estimate  $\hat{v}^\pi$  of  $V^\pi$ , defined as

$$\delta_t = r_t + \gamma \hat{v}^\pi(s_{t+1}) - \hat{v}^\pi(s_t),$$

should be small.

## Chapter 2

# Approximate solving of Markov Decision Processes

Solving MDPs is seeking the maximizing policy of the value function. For approximate solving of MDPs, we target what could be a more general **policy performance metric**. Often, it is connected to the value function: the expected (discounted) cumulative reward of the policy“

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=0}^T \gamma^t r_t \right] = \mathbb{E}_{\tau \sim \pi} [R(\tau)] \quad (2.1)$$

where  $\tau = \{s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$  and  $R(\tau) = \sum_{t=0}^T \gamma^t r_t$ .

We use the shorthand  $r_t$  for the reward  $r(s_t, a_t)$  – or more generally a transition reward  $r(s_t, a_t, s_{t+1})$  or stochastic reward distributed as  $r_t \sim p(\cdot, s_t, a_t, s_{t+1})$ .

We seek to compute the maximizing policy in a parametric search space  $\{\pi_\theta : \theta \in \Theta\}$ :

$$\max_{\theta} J(\pi_\theta)$$

The expectation  $J$  could be computed if we are given the complete structure of the Markov decision process: the transition probabilities  $p(s, a, s')$  and reward function  $r(s, a)$ . But then we could just use the usual  $Q$ -learning algorithm.

Instead, we can use a gradient ascent method, by iteratively updating the policy parameter  $\theta$  using a direction provided by the gradient.

**Proposition 2 (Gradient under a parametric law)** *Given a set of probability models  $\{P_\theta : \theta \in \Theta \subseteq \mathbb{R}^d\}$  on a set  $\mathcal{X}$  and a function  $f: \mathcal{X} \rightarrow \mathbb{R}$ , we have that*

$$\nabla_\theta \mathbb{E}_{X \sim P_\theta} [f(X)] = \mathbb{E}_{X \sim P_\theta} [f(X) \nabla_\theta \log P_\theta(X)]$$

*This is a useful property for deriving estimators of the derivatives in optimization problems with stochastic objectives.*

This can be shown either by either writing the expectation as an integral, or by a change of measures with a Radon-Nikodym derivative.

Proposition 2 allows us to write the gradient of (2.1), called the **policy gradient** as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ R(\tau) \sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) \right] \quad (2.2)$$

and we will need to derive estimations for this quantity.

This makes sense in the finite (or almost surely finite) horizon.

## 2.1 Monte Carlo policy gradient: the REINFORCE algorithm

**The idea.** The policy performance  $J$  using a Monte Carlo approximation:

$$\hat{J}(\pi_\theta) = \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T_i} \gamma^t r_t^i = \frac{1}{M} \sum_{i=1}^M R(\tau_i) \quad (2.3)$$

where  $\tau_i$  are simulated trajectories under the policy  $\pi_\theta$ , and eq. (2.2)

We obtain the following estimate:

$$\widehat{\nabla_\theta J}(\pi_\theta) = \frac{1}{M} \sum_{i=1}^M R(\tau_i) \sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(s_t^i, a_t^i) \quad (2.4)$$

This is an unbiased Monte Carlo estimate of the policy gradient. It only requires suitable regularity of the parametric policy model  $\theta \mapsto \pi_\theta$ .

**Remark 2** *The expression (2.4) can be used as-is for functions with simple closed-form derivatives. In an automatic differentiation framework such as PyTorch, we can instead get the policy gradient from a computational graph with the following pseudo-loss function:*

$$\tilde{J}(\theta) = \frac{1}{M} \sum_{i=1}^M R(\tau_i) \sum_{t=0}^{T_i} \log \pi_\theta(s_t^i, a_t^i) = \frac{1}{M} \sum_{i=1}^M \left( \sum_{t=0}^{T_i} \gamma^t r_t \right) \sum_{t=0}^{T_i} \log \pi_\theta(s_t^i, a_t^i) \quad (2.5)$$

### 2.1.1 Variance reduction: temporal structure and baselines

We can re-weight the log-probability gradients in eq. (2.2) by exploiting the fact that, for any time  $t$ , the cumulative rewards  $\sum_{t'=0}^{t-1} \gamma^{t'} r_{t'}$  from 0 to  $t-1$  are measurable with respect to the trajectory up to  $t$ ,  $\tau_{0:t}$ :

**Proposition 3** *The policy gradient can be rewritten as*

$$\nabla_\theta J(\pi_\theta) = \mathbb{E} \left[ \sum_{t=0}^T \sum_{t'=t}^T \gamma^{t'} r_{t'} \nabla_\theta \log \pi_\theta(s_t, a_t) \right] \quad (2.6)$$

*which leads to the policy gradient estimate*

$$\widehat{\nabla_\theta J}(\pi_\theta) = \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T_i} \gamma^t \hat{q}_t^i \nabla_\theta \log \pi_\theta(s_t^i, a_t^i) \quad (2.7)$$

where<sup>a</sup>  $\hat{q}_t^i = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ .

<sup>a</sup> $\hat{q}_t^i$  can be seen as an estimate of the  $Q$ -function  $Q^\pi(s_t, a_t) = \mathbb{E}[\sum_{t'=t}^T \gamma^{t'-t} r_{t'} \mid s_t, a_t]$ .

Given any **baseline** function  $b: \mathcal{S} \rightarrow \mathbb{R}$ , we can rewrite the policy gradient again as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[ \sum_{t=0}^T \left( \sum_{t'=t}^T \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(s_t, a_t) \right] \quad (2.8)$$

The resulting policy gradient estimate we get is

$$\widehat{\nabla}_\theta J(\pi_\theta) = \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T_i} (\hat{q}_t^i - b(s_t^i)) \nabla_\theta \log \pi_\theta(s_t^i, a_t^i) \quad (2.9)$$

which is an unbiased estimate.

It can be shown that the “best” baseline in terms of variance reduction  $b^*$  is the value function:

$$b^*(s_t) = \mathbb{E}_\pi \left[ \sum_{t'=t}^T \gamma^{t'-t} r_{t'} \mid s_t \right]$$

...which we are already trying to approximate. This suggests that we use some kind of **bootstrap** estimate for the baseline.

An obvious baseline is Monte Carlo estimation:

$$\hat{b}_t = \frac{1}{M} \sum_{i=1}^M \hat{q}_t^i$$

This is an unbiased estimate, as the expectation of the  $Q$ -function estimate under the policy  $\pi_\theta$

$$\mathbb{E}_{a_t^i \sim \pi_\theta(s_t^i, \cdot)} [\hat{q}_t^i] = \mathbb{E}_{a_t \sim \pi_\theta(s_t, \cdot)} \left[ \sum_{t'=t}^T \gamma^{t'-t} r_{t'} \mid s_t \right] = V^\pi(s_t)$$

## 2.1.2 Parametric Bootstrapping of the baseline

We define the bootstrap estimate  $\hat{b} = \hat{v}_\nu(\cdot)$ , where  $\hat{v}_\nu$  is in a parametric search space with parameter  $\nu \in \mathcal{V}$ . The parameter can be iteratively updated using gradient steps by alternating with the policy optimization steps.

For a given trajectory sample  $\tau = \{s_0, a_0, r_0, \dots\}$ , introduce the mean-squared error between the forward cumulative rewards (a nonparametric estimate of the value function) and the output of the value model:

$$\mathcal{L}(\nu; \tau) = \sum_{t=0}^T \left( \sum_{t'=t}^T \gamma^{t'-t} r_{t'} - \hat{v}_\nu(s_t) \right)^2$$

Then before each update of the policy  $\pi_\theta$ , update the value parameter  $\nu$  using either the gradient of  $\mathcal{L}$ .

## 2.2 Parametric approximation: Actor-Critic algorithms

**The idea.** The REINFORCE algorithm builds estimates of the  $Q$ -function to compute the policy gradient as it runs: this is computationally expensive and may lead to high variance. To combat this, it might be a good idea to *learn* from the  $Q$ -function estimates in a way that gives a consistent estimate that follows the policy gradient updates.

The class of actor-critic methods introduces a second search space for approximation of the state(-action) value function.

### 2.2.1 Actor-critic

The policy learning is still done by gradient ascent following a policy gradient estimate of the form eq. (2.9) – but this time, we replace the Monte Carlo estimate  $\hat{q}_t^i$  of the  $Q$ -function by a parametric estimator  $\hat{q}_\omega(s_t, a_t)$

$$\widehat{\nabla_\theta J}(\pi_\theta) = \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T_i} \hat{q}_\omega(s_t^i, a_t^i) \nabla_\theta \log \pi_\theta(s_t^i, a_t^i) \quad (2.10)$$

### 2.2.2 Actor-critic with baselines: advantage

As we've seen before, a good baseline to introduce in the policy gradient (2.2) is the value function.



# Bibliography

- [1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: <http://incompleteideas.net/book/the-book-2nd.html>.