

AA274A: Principles of Robot Autonomy I

Course Notes

The previous chapter introduced the deterministic and stochastic sequential decision making problems, and demonstrated how these problems can be solved by dynamic programming. While dynamic programming is a powerful algorithm, it also suffers from several practical challenges. This chapter briefly introduces some of the key ideas in *reinforcement learning*, which encompasses a set of ideas that can generally be viewed as extensions or approximations of the dynamic programming approach, which make it more practical for many robotic applications.

22 Reinforcement Learning

Reinforcement learning (RL) is a broad field that studies autonomous decision making, including several more general and challenging problems than have been considered in previous chapters. In particular, the standard reinforcement learning problem is to determine closed-loop control policies that drive an agent to maximize an accumulated reward¹. However, in the most general case it is not required that a *system model* be known! This paradigm can be represented by Figure 1, where it can be seen that given a control input the environment specifies the state and reward, and the environment can be considered to be a black box (it is not necessarily known how the state is generated).

Reinforcement learning is a highly active field of research, and has seen successes in several applications including acrobatic control of helicopters, games, finance, and robotics. In this chapter the fundamentals of reinforcement learning are introduced, including the formulation of the RL problem, RL algorithms that leverage system models (value iteration/dynamic programming and policy iteration), and a few RL algorithms that do not require system models (Q-learning, policy gradient, actor-critic).

¹Note that the maximization of “reward” in the context of reinforcement learning is essentially equivalent to minimization of “cost” in optimal control formulations

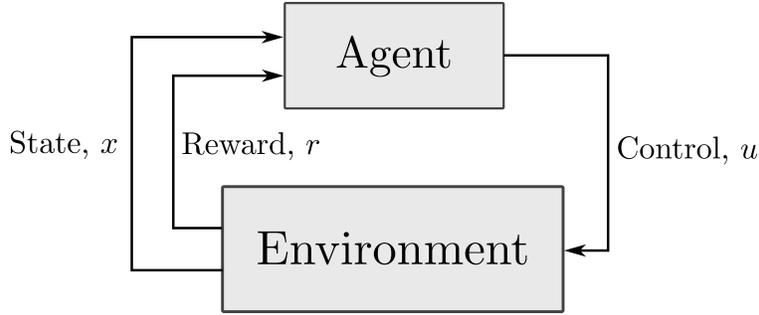


Figure 1: In reinforcement learning problems, the robot (agent) learns how to make decisions by interacting with the environment.

22.1 Problem Formulation

The reinforcement learning problem is effectively the same as the stochastic decision making problem from the previous chapter, but a slightly different notation will be used². The state and control input for the system is denoted as \mathbf{x} and \mathbf{u} (as in previous chapters), and the set of admissible states and controls are denoted as X and U . However, the stochastic state transition model will now be written as:

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1}), \quad (1)$$

which is the conditional probability distribution over \mathbf{x}_t , given the previous state and control.

As was the case for the stochastic decision making problem from last chapter, RL algorithms seek to compute *policies* $\pi = \{\pi_0, \pi_1, \dots, \pi_{T-1}\}$ that define the control:

$$\mathbf{u}_t = \pi_t(\mathbf{x}_t), \quad (2)$$

which results in a closed-loop control law. However, instead of a “cost” function (that was used in the previous chapter), reinforcement learning problems are typically formulated with a *reward* function:

$$r_t = R(\mathbf{x}_t, \mathbf{u}_t), \quad (3)$$

where r_t is the reward for being at state \mathbf{x}_t and applying a control \mathbf{u}_t . The goal of the reinforcement learning problem is to maximize the accumulated reward over a number of time steps (in contrast to minimizing cost in the previous chapter). Specifically, the total expected (discounted) reward over T time steps is defined for a particular policy π as:

$$V_T^\pi(\mathbf{x}) = E \left[\sum_{t=0}^{T-1} \gamma^t R(\mathbf{x}_t, \pi_t(\mathbf{x}_t)) \mid \mathbf{x}_0 = \mathbf{x} \right], \quad (4)$$

²The fields of optimal control and reinforcement learning have significant overlap, but each community has developed its own standard notation. Most often, the state in the optimal control community is represented by \mathbf{x} and in the RL community as \mathbf{s} . Similarly, in control theory the control input is \mathbf{u} while in the RL community it is referred to as an action \mathbf{a} .

where $\gamma \in (0, 1]$ is referred to as a *discount factor*, and the function $V_T^\pi(\mathbf{x})$ is referred to as the *value function*³ for the policy π (starting from \mathbf{x} and with T steps to go). The discount factor γ was not used in the decision making problems from the previous chapter, and can be chosen as $\gamma = 1$ for finite horizon problems. However, it is common in RL problems to consider infinite horizons $T = \infty$, and in this case the discount factor must be chosen such that $\gamma < 1$ to ensure the expected reward is finite⁴.

The reinforcement learning problem is therefore to find an optimal policy π^* that maximizes the expected reward (i.e. maximizes the value function):

Definition 22.1 (Reinforcement Learning Problem). *The reinforcement learning problem for the system model (1), admissible states $\mathbf{x} \in X$ and controls $\mathbf{u} \in U$, and reward function (3) is to find an optimal policy π^* that satisfies:*

$$\pi^*(\mathbf{x}) = \arg \max_{\pi} V_T^\pi(\mathbf{x}), \quad (5)$$

for all states $\mathbf{x} \in X$.

22.2 Value Iteration (Dynamic Programming)

While the dynamic programming algorithm was covered in the last chapter, it will also be included here in the context of the reinforcement learning problem formulation. In this case, the “principle of optimality” again says that the optimal *tail* policy is optimal for *tail* subproblems, which leads to the recursion:

$$V_{k+1}^*(\mathbf{x}) = \max_{\mathbf{u} \in U} R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) V_k^*(\mathbf{x}'), \quad (6)$$

which is commonly referred to as the *Bellman equation* (note that a shorthand $V_k^*(\mathbf{x}) = V_k^{\pi^*}(\mathbf{x})$ is used for the optimal value function). In words, this expression says that the optimal reward associated with starting at the state \mathbf{x} and having $k + 1$ steps to go can be found as an optimization over the immediate control by accounting for the (expected) optimal tail rewards. The full dynamic programming algorithm for solving the RL problem (5) is given in Algorithm 1.

In the context of reinforcement learning, this procedure is commonly referred to as *value iteration* and in many cases it is assumed that the horizon T is infinite. For infinite-horizon problems the “value iteration” in Algorithm 1 is performed either over a finite-horizon (which yields an approximate solution), or until convergence to a stationary (i.e. time-invariant) optimal value function/policy⁵.

³Notice the similarity of the value function to the expected cost in the stochastic decision making problem from the previous chapter.

⁴The term “discount factor” is used because it “discounts” future rewards. The intuition behind this can be explained by the phrase “a dollar today is better than a dollar tomorrow”.

⁵In the infinite horizon case, the optimal value function is unique and the optimal policy is stationary and deterministic, but not necessarily unique.

$V_0^*(\mathbf{x}) = 0$, for all $\mathbf{x} \in X$
for $k = 0$ **to** $T - 1$ **do**
 $V_{k+1}^*(\mathbf{x}) = \max_{\mathbf{u} \in U} R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) V_k^*(\mathbf{x}')$, for all $\mathbf{x} \in X$
 $\pi_{T-1-k}^*(\mathbf{x}) = \arg \max_{\mathbf{u} \in U} R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) V_k^*(\mathbf{x}')$, for all $\mathbf{x} \in X$
end
return $V_0^*(\cdot), \dots, V_T^*(\cdot), \pi_0^*(\cdot), \dots, \pi_{T-1}^*(\cdot)$
Algorithm 1: Dynamic Programming (Reinforcement Learning)

To solidify the relationship between value iteration in the context of RL and dynamic programming in the context of stochastic decision making from the previous chapter, the inventory control example from the previous chapter is revisited:

Example 22.1 (Inventory Control). Consider again the inventory control problem from the previous chapter, where the available stock of a particular item is the state $x_t \in \mathbb{N}$, the control $u_t \in \mathbb{N}$ adds items to the inventory, the demand w_t is uncertain, and the dynamics and constraints are:

$$\begin{aligned}
x_t &= \max\{0, x_{t-1} + u_{t-1} - w_{t-1}\}, \\
p(w = 0) &= 0.1, \quad p(w = 1) = 0.7, \quad p(w = 2) = 0.2.
\end{aligned}$$

and

$$x_t + u_t \leq 2.$$

Based on the dynamics, the probabilistic model (1) is given by:

$$\begin{aligned}
p(x_t = \{0, 1, 2\} | x_{t-1} = 0, u_{t-1} = 0) &= \{1, 0, 0\}, \\
p(x_t = \{0, 1, 2\} | x_{t-1} = 0, u_{t-1} = 1) &= \{0.9, 0.1, 0\}, \\
p(x_t = \{0, 1, 2\} | x_{t-1} = 0, u_{t-1} = 2) &= \{0.2, 0.7, 0.1\}, \\
p(x_t = \{0, 1, 2\} | x_{t-1} = 1, u_{t-1} = 0) &= \{0.9, 0.1, 0\}, \\
p(x_t = \{0, 1, 2\} | x_{t-1} = 1, u_{t-1} = 1) &= \{0.2, 0.7, 0.1\}, \\
p(x_t = \{0, 1, 2\} | x_{t-1} = 2, u_{t-1} = 0) &= \{0.2, 0.7, 0.1\},
\end{aligned}$$

where some transition values are not explicitly written due to the control constraints. Next, the reward function is defined as:

$$\begin{aligned}
R(x_t, u_t) &= -E[u_t + (x_t + u_t - w_t)^2], \\
&= -(u_t + (x_t + u_t - E[w_t])^2 + Var(w_t)),
\end{aligned}$$

and a discount factor of $\gamma = 1$ is used. As in the previous chapter, this reward penalizes (a negative reward is a penalty) ordering new stock and having available stock at the next time step (i.e. having to store stock).

Algorithm 1 can now be applied, starting with the value function with no steps to go:

$$V_0^*(x) = 0,$$

and then recursively computing:

$$\begin{aligned} V_1^*(0) &= \max_{u \in \{0,1,2\}} - (u + (u - 1.1)^2 + 0.29) = -1.3, \\ V_1^*(1) &= \max_{u_2 \in \{0,1\}} - (u + (1 + u - 1.1)^2 + 0.29) = -0.3, \\ V_1^*(2) &= -((2 - 1.1)^2 + 0.29) = -1.1, \end{aligned}$$

where $E[w] = 1.1$ and $Var(w) = 0.29$. The optimal stage policies associated with this step are:

$$\pi_{T-1}^*(0) = 1, \quad \pi_{T-1}^*(1) = 0, \quad \pi_{T-1}^*(2) = 0.$$

In the next step:

$$\begin{aligned} V_2^*(0) &= \max_{u \in \{0,1,2\}} - (u + (u - 1.1)^2 + 0.29) + \sum_{x'} p(x' | x = 0, u) V_1^*(x') = -2.5, \\ V_2^*(1) &= \max_{u \in \{0,1\}} - (u + (1 + u - 1.1)^2 + 0.29) + \sum_{x'} p(x' | x = 1, u) V_1^*(x') = -1.5, \\ V_2^*(2) &= -((2 - 1.1)^2 + 0.29) + \sum_{x'} p(x' | x = 2, u = 0) V_1^*(x') = -1.68, \end{aligned}$$

with optimal stage policies:

$$\pi_{T-2}^*(0) = 1, \quad \pi_{T-2}^*(1) = 0, \quad \pi_{T-2}^*(2) = 0.$$

Finally, in the last step:

$$\begin{aligned} V_3^*(0) &= \max_{u \in \{0,1,2\}} - (u + (u - 1.1)^2 + 0.29) + \sum_{x'} p(x' | x = 0, u) V_2^*(x') = -3.7, \\ V_3^*(1) &= \max_{u \in \{0,1\}} - (u + (1 + u - 1.1)^2 + 0.29) + \sum_{x'} p(x' | x = 1, u) V_2^*(x') = -2.7, \\ V_3^*(2) &= -((2 - 1.1)^2 + 0.29) + \sum_{x'} p(x' | x = 2, u = 0) V_2^*(x') = -2.818, \end{aligned}$$

with optimal stage policies:

$$\pi_{T-3}^*(0) = 1, \quad \pi_{T-3}^*(1) = 0, \quad \pi_{T-3}^*(2) = 0.$$

These results are, in fact, identical to the results from the example in the previous chapter! The only difference is the formulation of the problem in the RL framework instead of the stochastic decision making problem framework.

22.3 Policy Iteration

Another common algorithm that can be used to solve the reinforcement learning problem (5) is *policy iteration*. The main idea of policy iteration is that if the value function can be computed for any arbitrary policy π , then the policy can be incrementally improved to yield a better policy π' .

22.3.1 Policy Evaluation

The first key element of the policy iteration algorithm is *policy evaluation*, which is used to compute the value function $V_k^\pi(\mathbf{x})$ for a given policy π . Policy evaluation is based on the recursion:

$$V_{k+1}^\pi(\mathbf{x}) = R(\mathbf{x}, \pi(\mathbf{x})) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \pi(\mathbf{x})) V_k^*(\mathbf{x}'), \quad (7)$$

which is very similar to the Bellman equation (6) except that there is no optimization over the control (since it is fixed). The policy evaluation algorithm is given in Algorithm 2.

```

Data:  $\pi$ 
Result:  $V_0^\pi(\cdot), \dots, V_T^\pi(\cdot)$ 
 $V_0^\pi(\mathbf{x}) = 0$ , for all  $\mathbf{x} \in X$ 
for  $k = 0$  to  $T - 1$  do
  |  $V_{k+1}^\pi(\mathbf{x}) = R(\mathbf{x}, \pi(\mathbf{x})) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \pi(\mathbf{x})) V_k^\pi(\mathbf{x}')$ , for all  $\mathbf{x} \in X$ 
end
return  $V_0^\pi(\cdot), \dots, V_T^\pi(\cdot)$ 

```

Algorithm 2: Policy Evaluation

In the infinite-horizon case where a stationary policy is used, the iteration in Algorithm 2 stops when the value function has converged to its stationary value. However, in some special cases it is also possible to solve for the value function directly, for example by formulating the problem as a linear system of equations.

22.3.2 Q-function

The second key component in the policy iteration algorithm is the state-action value function, which is also referred to as the *Q-function*. Unlike the value function, the Q-function is a function of both state and control:

$$Q_{k+1}^\pi(\mathbf{x}, \mathbf{u}) = R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) V_k^\pi(\mathbf{x}'), \quad (8)$$

and can be interpreted as the “value” of starting in state \mathbf{x} , applying control \mathbf{u} , and then following the policy π for the remaining time. In other words, it specifies the value associated with deviating from the given policy π for a single time-step.

Similarly, the *optimal* Q-function is:

$$Q_{k+1}^*(\mathbf{x}, \mathbf{u}) = R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) V_k^*(\mathbf{x}'),$$

where the shorthand notation $Q_k^*(\mathbf{x}, \mathbf{u}) = Q_k^{\pi^*}(\mathbf{x}, \mathbf{u})$ is used. Note that from the Bellman equation (6) the optimal value function can be written as an optimization over the optimal Q-function:

$$V_{k+1}^*(\mathbf{x}) = \max_{\mathbf{u} \in U} Q_{k+1}^*(\mathbf{x}, \mathbf{u}).$$

Therefore, instead of computing the optimal value function using value iteration it is possible to deal directly with the Q-function!

22.3.3 Policy Iteration Algorithm

The policy iteration algorithm incrementally updates the policy by performing local optimizations of the Q-function. In particular, a single iteration of the policy update is shown in Algorithm 3. It can be proven theoretically that under the policy iteration algorithm

Data: π
Result: π'
 $V_0^\pi(\cdot), \dots, V_T^\pi(\cdot) \leftarrow \text{PolicyEvaluation}(\pi)$
for $k = 0$ **to** $T - 1$ **do**
 $Q_{k+1}^\pi(\mathbf{x}, \mathbf{u}) = R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) V_k^\pi(\mathbf{x}')$ for all $\mathbf{x} \in X$
 $\pi'_{T-1-k}(\mathbf{x}) = \arg \max_{\mathbf{u} \in U} Q_{k+1}^\pi(\mathbf{x}, \mathbf{u})$, for all $\mathbf{x} \in X$
end
return $\pi' = \{\pi'_0, \dots, \pi'_{T-1}\}$

Algorithm 3: Policy Iteration Step

the value function is monotonically increasing with each new policy, and the procedure is run until convergence. While policy iteration and value iteration are quite similar, policy iteration can end up converging faster in some cases.

22.4 Q-Learning

The value and policy iteration algorithms are applicable only to problems where the probabilistic system model $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is *known*. In many real-world robotics applications this may not be the case!

In general, many reinforcement learning algorithms address problems where it is assumed that the agent only has access to the rewards r_t and states \mathbf{x}_t , but does not know how they are generated. Therefore the agent must “learn” policies from experience, or in other words from trajectory data:

$$\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_T\}, \quad \{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1}\}, \quad \{r_0, r_1, \dots, r_{T-1}\}.$$

One algorithm for solving this more challenging problem is *Q-learning*. The core idea behind Q-learning is that it is possible to collect data samples $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$, and over time learn what the optimal Q-function $Q^*(\mathbf{x}, \mathbf{u})$ is. For simplicity an infinite-horizon ($T = \infty$) problem will be considered, such that the optimal value and Q-functions will be stationary, and in particular the optimal Q-function will satisfy:

$$Q^*(\mathbf{x}, \mathbf{u}) = R(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}'} p(\mathbf{x}' | \mathbf{x}, \mathbf{u}) \max_{\mathbf{u}' \in U} Q^*(\mathbf{x}', \mathbf{u}').$$

Importantly, even if the model and reward function are unknown, for a given sample $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$ that is collected by interacting with the environment, the *expectation*:

$$E\left[\left(r_t + \gamma \max_{\mathbf{u}' \in U} Q^*(\mathbf{x}_{t+1}, \mathbf{u}')\right) - Q^*(\mathbf{x}_t, \mathbf{u}_t)\right] = 0,$$

will hold for the optimal Q-function. Therefore, the idea of Q-learning is that an approximation of the optimal Q-function can be improved over time by collecting data and trying to ensure that the above conditions holds. This leads to the Q-learning algorithm described in Algorithm 4.

Data: Set \mathcal{S} of trajectory samples $\{\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1}\}$, learning rate α
Result: $Q(\mathbf{x}, \mathbf{u})$
Initialize $Q(\mathbf{x}, \mathbf{u})$ for all $\mathbf{x} \in X$ and $\mathbf{u} \in U$
for $\{\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1}\} \in \mathcal{S}$ **do**
 $Q(\mathbf{x}_t, \mathbf{u}_t) \leftarrow Q(\mathbf{x}_t, \mathbf{u}_t) + \alpha \left(r_t + \gamma \max_{\mathbf{u} \in U} Q(\mathbf{x}_{t+1}, \mathbf{u}) - Q(\mathbf{x}_t, \mathbf{u}_t) \right)$
end
return $Q(\mathbf{x}, \mathbf{u})$

Algorithm 4: Q-learning

Q-learning is referred to as a *model-free* method, however this is not because it doesn't require knowledge of the true model, but rather because it does not rely on explicitly *learning* an model. It is also called a *value-based* model-free method since it does not directly build the policy, but rather estimates the optimal Q-function to implicitly define the policy.

Q-learning can be guaranteed to converge to the optimal Q-function under certain conditions, but has some practical disadvantages. In particular, unless the number of possible states and controls are finite and relatively small, it can be intractable to store the Q-value associated with each state-value pair. One variation of the Q-learning algorithm to handle large or continuous control spaces is to parameterize the Q-function as $Q_\theta(\mathbf{x}, \mathbf{u})$ and to simply update the parameters θ . Another disadvantage of Q-learning is that sometimes the Q-function can be complex and therefore potentially hard to learn.

22.5 Policy Gradient

The *policy gradient* method for reinforcement learning problems can also be applied to situations where the model is *unknown*. This approach can be particularly useful for scenarios where the optimal policy may be relatively simple compared to the Q-function, in which case Q-learning may be challenging.

In the policy gradient approach, a class of candidate policies π_θ is defined based on a set of parameters θ , and the goal is to *directly* modify the parameters θ to improve performance. This is accomplished by using trajectory data to estimate a gradient of the performance with respect to the policy parameters θ , and then using the gradient to update θ . Because this method works directly on a policy (and does not learn a model or value function), it is referred to as a *model-free policy-based* approach.

In general, policy-based RL methods such as policy gradient can converge more easily than value-based methods, can be effective in high-dimensional or continuous action spaces, can learn stochastic policies. However, one challenge with directly learning policies is that it is easy to converge to local optima, and it requires data from the *current* policy to compute the gradients (in contrast to Q-learning which is agnostic to the policy used), which makes it data-inefficient.

22.6 Actor-Critic

Another popular reinforcement learning algorithm is the *actor-critic* algorithm, which blends the concepts of value-based and policy-based model-free RL. In particular, a parameterized policy π_{θ} is learned (actor) along side an estimated Q-function for the policy (critic). The addition of the critic helps to reduce the variance in the gradient estimates for the actor policy, which makes the overall learning process more data-efficient.

22.7 Deep Reinforcement Learning

Neural networks are a powerful function approximator that can be utilized in reinforcement learning algorithms. For example, in Q-learning the Q-function can be approximated by a neural network to extend the approach to nonlinear, continuous state space domains. In policy gradient methods, the policy π_{θ} can be parameterized as a neural network, enabling the policy to operate on high-dimensional states including images (i.e. visual feedback)!

22.8 Exploration vs Exploitation

When learning from experience (e.g. using Q-learning, policy gradient, actor-critic, deep RL, etc.) it is important to ensure that the experienced trajectories (i.e. the collected data points) are meaningful! For example, an abundance of data related to a particular set of actions/states will not necessarily be sufficient to learn good policies for *all* possible situations. Therefore an important part of reinforcement learning is *exploring* different combinations of states and actions. One simple approach to exploration is referred to as ϵ -greedy exploration, where a random control is applied instead of the current (best) policy with probability ϵ .

However, exploration can lead to suboptimal performance since any knowledge accumulated about the optimal policy is ignored. This leads to the *exploration vs exploitation* trade-off: a fundamental challenge in reinforcement learning.

References

- [Ber19] D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.
- [SB18] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.