Reinforcement Learning I

Tengyuan Liang[1]

[1] The University of Chicago Booth School of Business

DLA Lecture 5: Explore vs. Exploit

Dynamic Programming and Bellman Equations. An overview of reinforcement learning. Readings: Hardt and Recht [2] Chapters 11 and 12, Lattimore and Szepesvari [3].

[2] Moritz Hardt and Benjamin Recht. *Patterns, Predictions, and Actions: A Story about Machine Learning*. Princeton University Press, 2022

[3] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020

Contents

1 Sequential Decision Making Framework 1

2 Dynamic Programming 2

2.1 Infinite horizons and stationary policies 3

2.2 Tabular Markov Decision Process (MDP) 4

2.3 Linear Quadratic Regulator 4

3 Policy and Value Iteration 5

4 Unknown Dynamics and/or Rewards: A Brief Overview 6

4.1 Principle of Certainty Equivalence 6

4.2 Approximate Dynamic Programming 7

4.3 Policy Gradient 8

1 Sequential Decision Making Framework

First, let's consider the generic formulation of sequential decision making when the model is known.

- State (or data): $X = \{X_t\}_{t=1,2,...}$
- Action (or decision): $U = \{U_t\}_{t=1,2,...}$
- Reward (or feedback): $R = \{R_t\}_{t=1,2,...}$

The goal is to analyze data X sequentially and then subsequently choose decision U so that the reward R is large.

Definition 1 (Stochastic Dynamical Systems)**.** *A dynamical system model has a rule for how states evolve over time. A (discrete time) stochastic dynamical system has a state X_t, exogenous input U_t modeling our control action, and reward R_t. The state evolves according to the equation*

$$X_{t+1} = f_t(X_t, U_t, W_t)$$

and the reward

$$R_t = g_t(X_t, U_t, W_t)$$

where f_t, g_t are known functions, W_t is a random variable modeling the stochasticity of the system.

Definition 2 (Control Policy)**.** *We call $\pi_t, t = 1, 2 \ldots$ a control policy that takes the trajectory of previous states $\{X_s\}_{s=0}^t$ and maps to a control action*

$$U_t = \pi_t(X_t, X_{t-1}, \ldots) \,.$$

TL: Note here we are vague about where π_t is a randomized policy or a deterministic policy, as in many problems, one may need to use randomized policy to balance the exploration vs. exploitation.

Definition 3. *Sequential Decision Making (SDM) Problem The sequential decision-making problem solves the following policy optimization*

$$\begin{aligned} \max_{\pi_t} \; & \mathop{\mathbb{E}}_{W_t} \big[\sum_{t=0}^{T} R_t(X_t, U_t, W_t) \big] \\ \text{s.t. } & X_{t+1} = f_t(X_t, U_t, W_t) \\ & U_t = \pi_t(X_t, X_{t-1}, \ldots) \\ & (X_0 = x_0 \; \textit{given}) \end{aligned}$$

TL: For MDPs, we encode the stochastic nature of the states into W_t.

2 *Dynamic Programming*

Dynamic programming (DP) provides a general-purpose solution to SDM problems. The DP solution to the SDM problem is based on the *principle of optimality*: an optimal route from A to C that goes through B, must induce an optimal route from B to C. DP is built on this principle, allowing us to recursively find an optimal policy by starting at the final time and going backward in time to solve for the earlier stages.

Definition 4 (Q-function)**.** *Define the Q-function to be the mapping:*

$$\begin{aligned} Q_{s \to e}(x, u) \; := & \max_{u_t} \mathop{\mathbb{E}}_{W_t} \big[\sum_{t=s}^{e} R_t(X_t, u_t, W_t) \big] \\ & \text{s.t. } X_{t+1} = f_t(X_t, u_t, W_t), \; t = s, \ldots, e-1 \\ & (X_s, u_s) = (x, u) \end{aligned}$$

The Q-function determines the best achievable value of the SDM problem from start time s to end e.

Definition 5 (Bellman's Equation)**.** *The DP solves the following problem by iteratively calculating the Q function backward in time*

$$Q_{T \to T}(x, u) = \mathop{\mathbb{E}}_{W_T} [R_T(x, u, W_t)] \,,$$

then compute recursively for $t = T-1, T-2, \dots, 0$

$$Q_{t\to T}(x,u) = \mathop{\mathbb{E}}_{W_t}\left[R_t(x,u,W_t) + \max_{u'} Q_{t+1\to T}\big(f_t(x,u,W_t),u'\big)\right] .$$

The above expression is known as Bellman's equation.

Given the sequence of $Q_{t\to T}, t = 0,1,2,\dots$, the optimal decision at time t given state x_t is

$$u_t = \arg\max_u Q_{t\to T}(x_t,u) \tag{2.1}$$

and the policy only depends on the current state.

TL: this is no surprise due to the fact that the current and future rewards (taking optimal decision onwards) only depend on the current state and action, and the past states are the past and only affect x_t.

Now we consider some special cases to give concrete examples, where specific instances of Bellman equations are either efficiently solvable or take a clean form to approximate.

2.1 *Infinite horizons and stationary policies*

The Q-functions we derived so far are time-varying. The computation scales up when the horizon times increase. With a bit more structure—namely time-invariant dynamics, and costs, together with a discount factor—the Bellman equation can be reduced to a fixed point equation.

Let $\gamma \in (0,1)$ be a discount factor and consider the infinite horizon SDM problem

$$\begin{aligned}\max_\pi \ & \mathop{\mathbb{E}}_{W_t}\big[\sum_{t=0}^{\infty}\gamma^t R(X_t,U_t,W_t)\big]\\ \text{s.t. } & X_{t+1} = f(X_t,U_t,W_t)\\ & U_t = \pi(X_t)\\ & (X_0 = x_0 \text{ given})\end{aligned}$$

Then the Q_γ function can be verified to be time-invariant and satisfies the following fixed point equations

$$Q_\gamma(x,u) = \mathop{\mathbb{E}}_{W}\left[R(x,u,W) + \gamma\max_{u'} Q_\gamma\big(f(x,u,W),u'\big)\right] . \tag{2.2}$$

The optimal policy is now time-invariant as well

$$u_t = \arg\max_u Q_\gamma(x_t,u) . \tag{2.3}$$

In a second, we will show that Policy and Value iterations are approximation schemes to solve the fixed point equation.

2.2 Tabular Markov Decision Process (MDP)

Consider the following finite state and action spaces Markov decision processes, which the next state X_{t+1} is given by the current state X_t and action U_t given by a transition kernel

$$\mathbb{P}\big[X_{t+1} = x' | X_t = x, U_t = u\big] \tag{2.4}$$

which is a probability transition matrix/table $[0,1]^{|\mathcal{X}|\times|\mathcal{X}|}$ indexed by $u \in \mathcal{A}$.

The Q function for the tabular case is also tables of size $|\mathcal{X}| \times |\mathcal{A}|$.

$$Q_{t\to T}(x,u) = R_t(x,u) + \sum_{x'\in\mathcal{X}} \mathbb{P}\big[X_{t+1} = x' | X_t = x, U_t = u\big] \max_{u'} Q_{t+1\to}(x',u') \tag{2.5}$$

This function can be computed by simple matrix-vector operations. The max operations can be performed by operating each row of the table $\max_{u'} Q_{t+1\to}(x',u')$ to get an array of size $|\mathcal{X}|$. For each column of $Q_{t\to T}(\cdot,u)$, one can compute with a matrix-vector product with time $|\mathcal{X}|^2$.

2.3 Linear Quadratic Regulator

The other important problem where dynamic programming is efficiently solvable is the case when the dynamics are linear and the rewards are quadratic.

$$\begin{aligned}
\max_{\pi_t} \ & \mathop{\mathbb{E}}_{W_t}\Big[\sum_{t=0}^{T} X_t^\top \Phi_t X_t + U_t^\top \Psi_t U_t\Big] \\
\text{s.t. } & X_{t+1} = A_t X_t + B_t U_t + W_t \\
& U_t = \pi_t(X_t) \\
& (X_0 = x_0 \text{ given})
\end{aligned}$$

Here Φ_t, Ψ_t are most commonly PSD matrices, W_t is noise with zero mean and bounded variances that are independent across t.

When we run DP, every Q-function can be verified to be quadratic, and thus the optimal action is a linear function of the state

$$U_t = -K_t X_t. \tag{2.6}$$

To be specific, consider only the time-invariant quadratic costs $\Phi_t = \Phi, \Psi_t = \Psi$, and that the underlying dynamics are time-invariant

as well $A_t = A$ and $B_t = B$. Define

$$\begin{aligned} Q_{T\to T}(x,u) &= x^\top \Phi x + u^\top \Psi u \\ Q_{T-1\to T}(x,u) &= x^\top \Phi x + u^\top \Psi u + \mathop{\mathbb{E}}_{W}[\min_{u'} Q_{T\to T}(Ax+Bu+W,u')] \\ &= x^\top \Phi x + u^\top \Psi u + (Ax+Bu)^\top \underbrace{\Phi}_{:=M_1}(Ax+Bu) + \text{const.} \end{aligned}$$

The optimal decision $U_{T-1} = -(\Psi + B^\top M_1 B)^{-1} B^\top M_1 A \cdot X_{T-1}$. At time $T-2$,

$$\begin{aligned} Q_{T-2\to T}(x,u) &= x^\top \Phi x + u^\top \Psi u + \mathop{\mathbb{E}}_{W}[Q_{T-1\to T}(\tilde{x}, -(\Psi + B^\top M_1 B)^{-1} B^\top M_1 A \cdot \tilde{x})] \quad \text{where } \tilde{x} = Ax+Bu+W \\ &= x^\top \Phi x + u^\top \Psi u + \mathop{\mathbb{E}}_{W}[\tilde{x}^\top \underbrace{(\Phi + A^\top M_1 A - (A^\top M_1 B)(\Psi + B^\top M_1 B)^{-1}(B^\top M_1 A))}_{:=M_2} \tilde{x}] + \text{const.} \\ &= x^\top \Phi x + u^\top \Psi u + (Ax+Bu)^\top M_2 (Ax+Bu) + \text{const.} \end{aligned}$$

By induction, one can show that the Q function at time t satisfies

$$Q_{t\to T}(x,u) = x^\top \Phi x + u^\top \Psi u + (Ax+Bu)^\top M_{T-t}(Ax+Bu) + \text{const.} \tag{2.7}$$

with the optimal control decision

$$u_t = -(\Psi + B^\top M_{T-t} B)^{-1} B^\top M_{T-t} A \cdot x_t \tag{2.8}$$

Here the PSD matrix M satisfies the recursion

$$\begin{aligned} M_0 &= 0 \\ M_{t+1} &= \Phi + A^\top M_t A - (A^\top M_t B)(\Psi + B^\top M_t B)^{-1}(B^\top M_t A)\,. \end{aligned}$$

In the large T limit, the optimal control policy is static, linear state feedback with

$$\begin{aligned} u_t &= -Kx_t \\ K &:= (\Psi + B^\top M B)^{-1} B^\top M A \end{aligned}$$

where M is the fixed point to the Discrete Algebraic Riccati Equation

$$M = \Phi + A^\top M A - (A^\top M B)(\Psi + B^\top M B)^{-1}(B^\top M A)$$

When all of the eigenvalues of $A - BK$ lie strictly inside the unit circle of the complex plane, the M is the unique solution to the Riccati Equation.

3 *Policy and Value Iteration*

In this section, we describe two celebrated methods for solving discounted infinite time horizon SDM.

Value Iteration considers the following updating scheme, for $k = 1, 2, \ldots$

$$Q_{k+1}(x,u) = \mathop{\mathbb{E}}_{W}\left[R(x,u,W) + \gamma \max_{u'} Q_k\big(f(x,u,W),u'\big)\right] \quad \forall x,u \tag{3.1}$$

In words, this simply solves the Bellman equation by running a fixed point iteration. The method works if the iteration is indeed a contractive mapping, namely $\|Q_{k+1} - Q'_{k+1}\|_\infty < \|Q_k - Q'_k\|_\infty$

TL: Insert a proof for the MDP case.

Policy Iteration is a two-step procedure: *policy evaluation* followed by *policy improvement*. For $k = 1, 2, \ldots$, and for the current policy π_k

$$\text{policy evaluation} \quad Q^{\text{eval}}_{k+1}(x,u) = \mathop{\mathbb{E}}_{W}\left[R(x,u,W) + \gamma Q^{\text{eval}}_{k}\big(x' = f(x,u,W), \pi_k(x')\big)\right] \tag{3.2}$$

$$\text{policy improvement} \quad \pi_{k+1}(x) = \arg\max_{u} Q^{\text{eval}}_{k+1}(x,u) \tag{3.3}$$

Oftentimes, several steps of policy evaluation are performed (to reach a fixed point) before optimizing the policy.

For tabular MDPs, LQR, the above iterations can be carried out exactly in an efficient way.

4 *Unknown Dynamics and/or Rewards: A Brief Overview*

So far, we know the underlying dynamical system which models how the state evolves after taking actions. What if the dynamics model or even the reward function is unknown?

There are three mainstream approaches (1) *Principle of Certainty Equivalence,* namely one fits a model from some collected data and then uses this model as if it were true in the SDM problem. (2) *Approximate Dynamic Programming*, which uses Bellman's principle of optimality and stochastic approximation to learn Q-functions from data. (3) *Direct Policy Search*, where one directly searches for policies by using data from previous episodes in order to improve the reward. Each of these has its advantages and disadvantages, as we now explore.

TL: Explore then Commit

TL: Q-Learning

TL: Policy Gradient

4.1 *Principle of Certainty Equivalence*

The principle of certainty equivalence [4] refers to a simple yet effective strategy for reinforcement learning, that is to estimate a predictive model for the dynamical system, and then use the fit model as if it were the true model in the subsequent optimal control problem.

Estimating the model from data is also referred to as system identification in the dynamical system and control literature. System identification differs from conventional estimation because one needs

[4] Herbert A Simon. Dynamic programming under uncertainty with a quadratic criterion function. *Econometrica, Journal of the Econometric Society*, pages 74–81, 1956; and Henri Theil. A note on certainty equivalence in dynamic planning. *Econometrica: Journal of the Econometric Society*, pages 346–349, 1957

to choose the right inputs u to stimulate the system, and the outputs are correlated over time.

Suppose we want to build a predictor of the state evolution from a trajectory history of the past observed states and actions. A simple, classic strategy in system identification is to inject a random probing sequence u_t for control and then measure how the state responds, conceptually

$$x_{t+1} \approx \psi(x_t, u_t) \tag{4.1}$$

One can fit the state transition function using supervised learning techniques, for instance,

$$\widehat{\psi} = \arg\min_{\psi} \frac{1}{T} \sum_{t=0}^{T-1} \|x_{t+1} - \psi(x, u_t)\|^2_{\text{norm}} \tag{4.2}$$

After the exploration phase, with the estimated transition model $\widehat{\psi}$ that approximates the dynamic system, one solves the optimal control "conditioned on" the estimated $\widehat{\psi}$,

$$\begin{aligned} \max_{\pi_t} \ & \mathop{\mathbb{E}}_{W_t} \big[\sum_{t=0}^{T} R(X_t, U_t, W_t) \big] \\ \text{s.t. } & X_{t+1} = \widehat{\psi}(X_t, U_t) + W_t \\ & U_t = \pi_t(X_t) \\ & (X_0 = x_0 \text{ given}) \end{aligned}$$

Albeit naive in hindsight, we emphasize that this is standard engineering practice.

4.2 *Approximate Dynamic Programming*

Approximate Dynamic Programming methods typically try to infer Q-functions directly from data. The standard assumption in most practical implementations of Q-learning is that the Q-functions are time-invariant, as would be the case in the infinite horizon, discounted optimal control problem.

One appealing thing for Q-learning is that one operates directly on the Q-function using stochastic approximations, without requiring a priori model for the underlying state evolution, the intuition is based on the fact that based on a trajectory of data $(x_t, u_t)_{t=1,2,\dots}$ based on the **optimal policy** that solves the Bellmann equation

$$Q_\gamma(x_t, u_t) \approx R(x_t, u_t) + \gamma \max_{u'} Q_\gamma(x_{t+1}, u') \ . \tag{4.3}$$

Taking to data, Q-learning simply attempts to solve value iteration using **stochastic approximation**. Namely, draw a trajectory based on

policy Q_γ^{old}, then update

$$Q_\gamma^{\text{new}}(x_t, u_t) = (1-\eta)Q_\gamma^{\text{old}} + \eta\left(R(x_t, u_t) + \gamma \max_{u'} Q_\gamma^{\text{old}}(x_{t+1}, u')\right) \tag{4.4}$$

Another way is to use **function approximation** techniques: suppose the Q function can be approximated by a parametric family $\mathcal{Q} := \{Q(x, u; \theta), \theta \in \Theta\}$, then one can solve gradient descent based on the following criteria

$$f_t(\theta) := \big(\overbrace{R(x_t, u_t) + \gamma Q(x_{t+1}, u_{t+1}; \theta)}^{\text{label}} - Q(x_t, u_t; \theta)\big)^2 \tag{4.5}$$

a stochastic gradient update to solve $\min_\theta \frac{1}{T}\sum_{t=0}^{T-1} f_t(\theta)$ is to update

$\theta_{t+1} = \theta_t - \eta_t \cdot \nabla f_t(\theta_t)$
where $\nabla f_t(\theta) := -\big(R(x_t, u_t) + \gamma \cdot Q(x_{t+1}, u_{t+1}; \theta) - Q(x_t, u_t; \theta)\big) \cdot \nabla Q(x_t, u_t; \theta)$

Other acceleration techniques with momentum (Nesterov) can also be incorporated: with $\lambda \geq 0$,

$$\begin{aligned} m_t &= \lambda m_{t-1} - \eta_t \cdot \nabla f_t(\theta_t + \lambda m_{t-1}) \\ \theta_{t+1} &= \theta_t + m_t \,. \end{aligned}$$

4.3 *Policy Gradient*

TL: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html leave as HW

The most ambitious form of control without models attempts to directly learn a policy function from episodic experiences without ever building a model or appealing to the Bellman equation. From the oracle perspective, these policy-driven methods turn the problem of RL into a pure optimization problem: either a derivative-free method, or a gradient based method using the "log density trick".

Let us setup the generic problem: consider a finite horizen SDM with trajectory $\tau := (x_0, u_0, \ldots, x_T, u_T)$, and a parametric family of distributions that generate τ, denoted as π_θ. How a certain policy π_θ performs can be written as

$$J(\pi_\theta) := \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] \,. \tag{4.6}$$

Ideally, one wishes to search over the space of policies and do optimzation in that realm, namely, either calculating or approximating

$$\theta_{k+1} = \theta_k + \alpha \cdot \nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{4.7}$$

The cucial quantity is on how to get the *policy gradient* $\nabla_\theta J(\pi_\theta)$.

The first method is based on *log density trick*: define $p_\theta(\tau)$ as the density of trajectory induced by π_θ. Note that

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau)\,\mathrm{d}\tau \\ &= \int p_\theta(\tau) R(\tau) \tfrac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}\,\mathrm{d}\tau \\ &= \int p_\theta(\tau) R(\tau) \nabla_\theta \log p_\theta(\tau)\,\mathrm{d}\tau \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta \log p_\theta(\tau)]\end{aligned}$$

The above expression makes it ammendable to optimize without the numeric differentiation, as long as we know analytically the expression of $\log p_\theta(\tau)$: note $p_\theta(\tau) = \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\pi_\theta(u_t|x_t)$

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(u_t|x_t)$$

Put things together, we know

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[R(\tau)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(u_t|x_t)\right] \tag{4.8}$$

and the optimization is now amendable to stochastic approximation: sample m trajectories $\tau^{(j)}$ with policy π_{θ_k}, and then update the policy

$$\theta_{k+1} = \theta_k + \alpha \cdot \frac{1}{m}\sum_{j=1}^{m} R(\tau^{(j)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(u_t^{(j)}|x_t^{(j)}) \tag{4.9}$$

The second method is based on zeroth order approximation to gradients, namely

$$\lim_{\epsilon\to 0} \mathop{\mathbb{E}}_{g\sim\mathcal{N}(0,I)}\left[g\cdot\frac{f(\theta+\epsilon\cdot g)-f(\theta)}{\epsilon}\right] = \nabla_\theta f(\theta) \tag{4.10}$$

Implementing such stochasic zeroth order search is similar: sample m trajectories $\tau^{(j)}$ with policy $\pi_{\theta_k+\epsilon g_j}$ (let $\tau^{(0)}$ denote the trajectory with policy π_{θ_k}), and then update the policy

$$\theta_{k+1} = \theta_k + \alpha \cdot \frac{1}{m}\sum_{j=1}^{m} g_j \cdot \frac{R(\tau^{(j)}) - R(\tau^{(0)})}{\epsilon}\,. \tag{4.11}$$

References

Moritz Hardt and Benjamin Recht. *Patterns, Predictions, and Actions: A Story about Machine Learning*. Princeton University Press, 2022.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Herbert A Simon. Dynamic programming under uncertainty with a quadratic criterion function. *Econometrica, Journal of the Econometric Society*, pages 74–81, 1956.

Henri Theil. A note on certainty equivalence in dynamic planning. *Econometrica: Journal of the Econometric Society*, pages 346–349, 1957.