Introduction to Reinforcement Learning
A Short Course

Scott Moura, Saehong Park

July 2020

1 Optimal Control

The canonical formulation is given by:

$$\begin{aligned}
\text{minimize}_{x_k,u_k} \quad & J = \sum_{k=0}^{N-1} c_k(x_k,u_k) + c_N(x_N) && (1)\\
\text{subject to:} \quad & x_{k+1} = f(x_k,u_k), \quad k = 0,\cdots,N-1 && (2)\\
& x_0 = x_{\text{init}} && (3)\\
& x_k \in \mathcal{X}_k, \quad u_k \in \mathcal{U}_k, \quad k=0,\cdots,N && (4)
\end{aligned}$$

Notation: The most fundamental elements

- $x_k \in \mathbb{R}^n$: State variable
- $u_k \in \mathbb{R}^p$: Control variable (a.k.a. "action")

What is a state? Given a trajectory of inputs $u_0,\cdots,u_N$, the (initial) state x_0 is sufficient to completely predict the evolution of a dynamic system.

Equation (1) is the objective function

- $c_k(x_k,u_k)$: instantaneous or stage cost
- c_N : terminal cost

Infinite time horizon:

- discounted cost: $J = \lim_{N\to\infty} \sum_{k=0}^{N} \gamma^k \cdot c(x_k,u_k)$, $\gamma \in [0,1]$
- average cost: $J = \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N} c(x_k,u_k)$

Why these formulations?

- Mathematical reason: Ensure cost remains finite.
- Conceptual reason: Stage costs are more uncertain in the future, so we discount their impact relative to immediate stage costs.

Examples:

1. "State regulation": $J = \sum_{k=0}^{\infty} \underbrace{x_k^T Q x_k}_{\text{send } x_k \text{ to } 0} + \underbrace{u_k^T R u_k}_{\text{don't use excessive control effort}}$
2. "State tracking": $J = \sum_{k=0}^{\infty} \underbrace{(x_k - x_k^{\text{ref}})^T Q (x_k - x_k^{\text{ref}})}_{\text{send } x_k \text{ close to } x^{ref}} + u_k^T R u_k$

3. "Minimum-time": $\text{minimize}_{x_k, u_k, N} \quad J = \sum_{k=0}^{N-1} 1$

Equation (2) are the dynamics paired with an initial condition:

$$x_{k+1} = f(x_k, u_k), \quad x_0 = x_{\text{init}} \tag{5}$$

The dynamics are a mathematical model of physical laws (differential equations), e.g.

- Newtonian mechanics
- Maxwell's equations
- Navier-Stokes
- Traffic conservation laws
- Laws of Thermodynamics
- (Electro-)Chemical Processes
- Infectious disease dynamics

Can be known/unknown, deterministic/stochastic

Equation (4) are "admissible" state and control/action sets

- $x_k \in \mathcal{X}_k, \qquad e.g. \underline{x}_k \leq x_k \leq \overline{x}_k$
- $u_k \in \mathcal{U}_k, \qquad \underline{u}_k \leq u_k \leq \overline{u}_k$

Objective: Find a control law (a.k.a. control "policy") of the form

$$u_k = \pi_k(x_k) \tag{6}$$

that solves the optimal control problem. Note that the control law/policy takes a "state feedback" form. That is, it maps states to actions. Examples:

1. Linear state feedback: $u_k = -K \cdot x_k, \qquad K \in \mathbb{R}^{p \times n}$
2. Neural network state feedback: $u_k = f_{NN}(x_k), \qquad f : \mathbb{R}^n \to \mathbb{R}^p$

2 Dynamic Programming

Dynamic programming is an algorithmic technique for solving optimal control problems by breaking it up into a recursion of sub-problems.

Consider the discounted cost optimal control formulation

$$\text{minimize} \quad J = \sum_{k=0}^{\infty} \gamma^k \cdot c(x_k, u_k), \qquad \gamma \in [0,1] \tag{7}$$

$$\text{subject to:} \quad x_{k+1} = f(x_k, u_k), \quad k = 0, 1, \cdots \tag{8}$$

Definition 1. (Value Function) Define $V(x_k)$ as the "value function," which represents the cumulative cost from time step k onward towards infinity, given the current state is x_k. Furthermore, let $V_\pi(x_k)$ represent the value function corresponding to control policy $u_k\pi(x_k)$, which may or may not be optimal.

Note $V_\pi(x_k) = c(x_k, u_k) + \gamma \cdot \underbrace{\sum_{\tau=k+1}^{\infty} \gamma^{\tau-(k+1)} \cdot c(x_\tau, u_\tau)}_{=V_\pi(x_{k+1})}$

$$V_\pi(x_k) = c(x_k, u_k) + \gamma \cdot V_\pi(x_{k+1}) \tag{9}$$

which can be used to recursively compute the value function corresponding to some policy $u_k = \pi(x_k)$.

We are now positioned to state Bellman's Principle of Optimality Equation:

$$\boxed{\begin{aligned} V(x_k) &= \min_{\pi(\cdot)} \{c(x_k, \pi(x_k)) + \gamma \cdot V(x_{k+1})\} && (10) \\ \text{where} \quad & x_{k+1} = f(x_k, \pi(x_k)) && (11) \end{aligned}}$$

The optimal policy is

$$\pi^\star(x_k) = \arg\min_{\pi(\cdot)} \{c(x_k, \pi(x_k)) + \gamma \cdot V(x_{k+1})\} \tag{12}$$

Remark 1. Bellman's Principle of Optimality Equation is also known as the discrete-time Hamilton-Jacobi-Bellman (HJMB) equation.

Note the following about Bellman's principle of optimality equation:

- The equation is recursive in $V(x_k)$
- Offline "planning" method

2.1 Case Study: Infinite-time Linear Quadratic Regulator (LQR)

Consider the classic and widely used linear quadratic regulator (LQR) optimal control problem

$$\text{minimize} \quad J = \sum_{k=0}^{\infty} \left[x_k^T Q x_k + u_k^T R u_k\right] \qquad \text{Note } \gamma = 1 \tag{13}$$

$$\text{subject to:} \quad x_{k+1} = A x_k + B u_k, \quad k = 0, 1, \cdots \tag{14}$$

$$x_0 = x_{\text{init}} \tag{15}$$

$$\text{where} \quad Q = Q^T \underbrace{\succeq 0}_{\text{p.s.d.}}, \quad R = R^T \underbrace{\succ 0}_{\text{p.d.}}, \quad Q_f = Q_f^T \succeq 0 \tag{16}$$

where "p.s.d." is a positive semi-definite matrix and "p.d." is a positive definite matrix. We will solve the LQR problem via dynamic programming to arrive at the so-called discrete-time algebraic Riccati equation (DARE).

We will discover that

- $V(x_k) = x_k^T P x_k$ (quadratic), $P = P^T \succeq 0$, and $P \in \mathbb{R}^{n \times n}$
- $\pi^\star(x_k) = K \cdot x_k$ (linear), $K \in \mathbb{R}^{p \times n}$

Bellman's (Principle of Optimality) equation for $V(x_k) = x_k^T P x_k$ is ...

$$V(x_k) = c(x_k, u_k) + \gamma \cdot V(x_{k+1}) \tag{17}$$

$$x_k^T P x_k = \left(x_k^T Q x_k + u_k^T R u_k\right) + 1 \cdot x_{k+1}^T P x_{k+1} \tag{18}$$

and substituting $u_k = K x_k$ gives

$$x_k^T P x_k = x_k^T \left[Q + K^T R K + (A + BK)^T P (A + BK)\right] x_k \tag{19}$$

Since this equation must hold for all x_k, we have the following matrix equation:

$$(A + BK)^T P (A + BK) - P + Q + K^T R K = 0 \tag{20}$$

This equation is linear in P, and is known as the "Lypanunov equation" to find P when K is fixed. It yields $P = P^T \succeq 0$ such that $V(x_k) = x_k^T P x_k$. That is:

$$V(x_k) = \sum_{\tau=k}^{\infty} x_\tau^T Q x_\tau + u_k^T R u_k \tag{21}$$

$$= \sum_{\tau=k}^{\infty} x_\tau^T \left[Q + K^T R K\right] x_\tau = x_k^T P x_k \tag{22}$$

To find an expression for K, write Bellman's optimality equation as

$$x_k^T P x_k = \min_w \left\{x_k^T Q x_k + w^T R w + (A x_k + B w)^T P (A x_k + B w)\right\} \tag{23}$$

differentiating with respect to (w.r.t.) w and setting to zero gives

$$2Rw + 2B^T P (A x_k + B w) = 0 \tag{24}$$

$$\Rightarrow w^\star = \underbrace{-(R + B^T P B)^{-1} B^T P A}_{=K} \cdot x_k \tag{25}$$

and thsu we have $u_k^\star = K \cdot x_k$ where $K = -(R + B^T P B)^{-1} B^T P A$. Substitute back into the Bellman equation and simplify to yield

$$A^T P A - P + Q - A^T P B (R + B^T P B)^{-1} B^T P A = 0 \tag{26}$$

which is quadratic in P and is known as the "Discrete-time Algebraic Riccati Equation" (DARE)

Summary of Infinite-time LQR:

$$u_k^{lqr} = K \cdot x_k \tag{27}$$

where $\quad K = -(R + B^T P B)^{-1} B^T P A \qquad (28)$

and P solves:

$$A^T P A - P + Q - A^T P B (R + B^T P B)^{-1} B^T P A = 0 \tag{29}$$

Value Function: $\quad V(x_k) = x_k^T P x_k \qquad (30)$

3 Policy Iteration & Value Iteration

So far, we have discussed offline designs via dynamic programming. We are also interested in online learning algorithms. Next we show that the Bellman equations are fixed point equations that enable forward-in-time methods for online learning.

Consider the discounted cost optimal control formulation over infinite time

$$\text{minimize} \quad \sum_{k=0}^{\infty} \gamma^k c(x_k, u_k) \qquad \gamma \in [0,1] \tag{31}$$

$$\text{subject to:} \quad x_{k+1} = f(x_k, u_k) \tag{32}$$

Define $V_\pi(x)$ as the value function corresponding to policy π (which may or may not be optimal). Note:

$$V_\pi(x_k) = \sum_{\tau=k}^{\infty} \gamma^{\tau-k} \cdot c(x_\tau, u_\tau) \tag{33}$$

$$= c(x_k, u_k) + \gamma \cdot \underbrace{\sum_{\tau=k+1}^{\infty} \gamma^{\tau-(k+1)} c(x_\tau, u_\tau)}_{=V_\pi(x_{k+1})} \tag{34}$$

$$V_\pi(x_k) = c(x_k, u_k) + \gamma \cdot V_\pi(x_{k+1}) \tag{35}$$

all where $u_k = \pi(x_k)$.

Observation & Question: This equation is implicit in $V_\pi(\cdot)$ and suggests the iterative scheme:

$$V_\pi^{j+1}(x_k) = c(x_k, u_k) + \gamma \cdot V_\pi^j(x_{k+1}) \qquad j = 0, 1, \cdots \tag{36}$$

$$V_\pi^0(x_k) = 0 \qquad \forall\ x_k \in \mathcal{X} \tag{37}$$

Q: Does V_π^j converge as $j \to \infty$? A: YES!

Algorithm 1. (Iterative Policy Evaluation Algorithm) To compute the value function corresponding to some arbitrary policy:

For $j = 0, 1, \cdots$,

$$V_\pi^{j+1}(x_k) = c(x_k, u_k) + \gamma \cdot V_\pi^j(x_{k+1}) \qquad \forall\ x_k \in \mathcal{X} \tag{38}$$

where $u_k = \pi(x_k)$,

$$V_\pi^0(x_k) = 0 \quad \forall\ x_k \in \mathcal{X} \tag{39}$$

Sutton & Barto refer to $V_\pi^j(x_k)$ as $j \to \infty$ as a "full backup".

Now that we have a method to evaluate a given policy, we wish to improve it. An intuitive idea is

$$\pi^{\text{NEW}} = \arg\min_{\pi(\cdot)} \{c(x_k, \pi(x_k)) + \gamma \cdot V_{\pi^{\text{OLD}}}(x_{k+1})\} \tag{40}$$

$$\text{where} \quad x_{k+1} = f(x_k, \pi(x_k)) \tag{41}$$

Bertsekas [1996] has proven that π^{NEW} is improved from π^{OLD} in the sense that $V_{\pi^{\text{NEW}}}(x_k) \leq V_{\pi^{\text{OLD}}}(x_k)$ $\forall$ $x_k \in \mathcal{X}$. We call this step "policy improvement".

SUMMARY

Policy Evaluation
Given an arbitrary policy π, find V_π
For $j = 0, 1, \cdots$
$V_\pi^{j+1} = c(x_k, u_k) + V_\pi^j(x_{k+1})$
$V_\pi^0(x_k) = 0\ \forall\ x_k \in \mathcal{X}$
where $u_k = \pi(x_k)$, $x_{k+1} = f(x_k, u_k)$

Policy Improvement
Given $V_{\pi^{\text{OLD}}}$ for some arbitrary policy π^{OLD}, find improved policy π^{NEW} such that $V_{\pi^{\text{NEW}}}(x_k) \leq V_{\pi^{\text{OLD}}}(x_k)\ \forall\ x_k \in \mathcal{X}$
$\pi^{\text{NEW}} = \arg\min_{\pi(\cdot)} \{c(x_k, \pi(x_k)) + \gamma \cdot V_{\pi^{\text{OLD}}}(x_{k+1})\}$
where $x_{k+1} = f(x_k, \pi(x_k))$

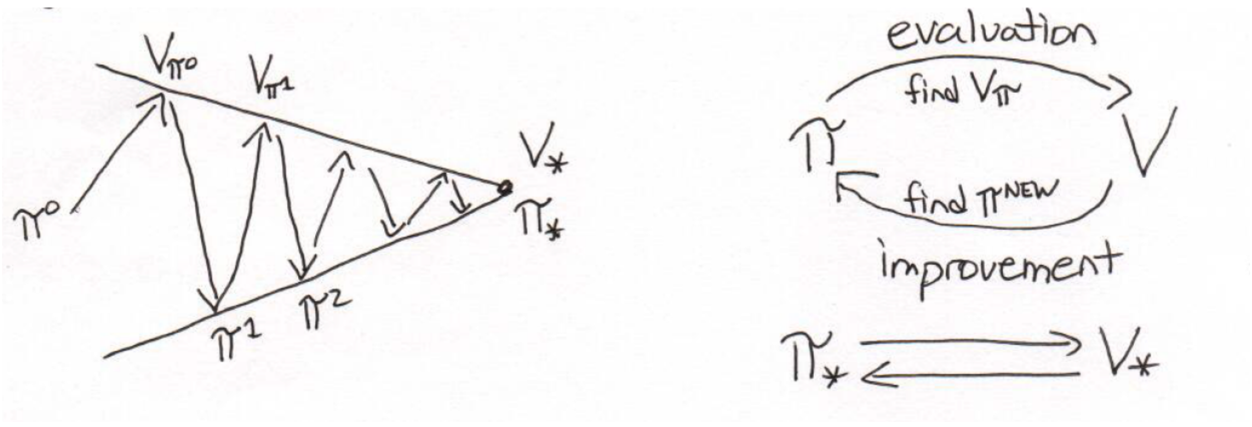

Figure 1: Schematic of Policy Iteration Algorithm

This motivated a class of algorithms called Policy Iteration (PI) and Generalized Policy Iteration (GPI), and Value Iteration (VI).

Algorithm 2. (Policy Iteration (PI) Algorithm [Sutton & Barto, Bertsekas])

1. Initialize admissible policy π^0. Set $m = 0$.
2. Policy Evaluation: Set $V_\pi(x_k) = 0 \; \forall \; x_k \in \mathcal{X}$. $\pi \leftarrow \pi^m$.
   for $j = 0, 1, \cdots$

$$V_\pi^{j+1} = c(x_k, u_k) + V_\pi^j(x_{k+1}), \quad \forall \; x_k \in \mathcal{X} \tag{42}$$

$$\text{where } u_k = \pi(x_k), \quad x_{k+1} = f(x_k, u_k) \tag{43}$$

3. Policy Iteration: Set $V_{\pi^{\text{OLD}}} \leftarrow V_\pi^{j+1}$

$$\pi^{\text{NEW}} = \arg\min_{\pi(\cdot)} \{c(x_k, \pi(x_k)) + \gamma \cdot V_{\pi^{\text{OLD}}}(x_{k+1})\} \tag{44}$$

$$\text{where } x_{k+1} = f(x_k, \pi(x_k)) \tag{45}$$

   Set $\pi^{m+1} = \pi^{\text{NEW}}$, $m \leftarrow m + 1$.
   Go to Step 2.

Remark 2. The policy evaluation step can be computationally onerous if $j \to \infty$, and we must perform each iteration for all $x_k \in \mathcal{X}$.

- Q: Can we truncate to M iterations? A: YES! Generalized Policy Iteration (GPI) algorithm.
- Q: Can we truncate to 1 iteration? A: YES! Value Iteration (VI) algorithm.
- Ref: [Howard 1960], [Puterman 1978] for theory. [Sutton & Barto, Bertsekas] for textbooks.

3.1 Case Study: LQR

Consider the linear quadratic regulator (LQR) problem:

$$\text{minimize} \quad \sum_{k=0}^{\infty} \left[x_k^T Q x_k + u_k^T R u_k\right] \tag{46}$$

$$\text{subject to:} \quad x_{k+1} = A x_k + B u_k, \quad k = 0, 1, \cdots \tag{47}$$

Recall that

- $V(x_k) = x_k^T P x_k$
- $u_k = K x_k$

Let's apply the policy evaluation and policy improvement steps!

1. Policy Evaulation: substitute into Bellman equation

$$x_k^T P^{j+1} x_k = x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P^j (Ax_k + Bu_k) \qquad (V = x^T P x) \tag{48}$$

$$x_k^T P^{j+1} x_k = x_k^T \left[Q + K^T R K + (A+BK)^T P^j (A+BK)\right] x_k \quad \forall\ x_k \qquad (u = Kx) \tag{49}$$

$$\Rightarrow \qquad P^{j+1} = Q + K^T R K + (A+BK)^T P^j (A+BK) \qquad \text{for some } K \tag{50}$$

   which provides an iterative solution to the Lyapunov equation.

2. Policy Improvement: Recall from before:

$$\min_w \left\{x_k^T Q x_k + w^T R w + (Ax_k + Bw)^T P^{\text{OLD}} (Ax_k + Bw)\right\} \tag{51}$$

   differentiating w.r.t. w, setting to zero, and re-arranging gives

$$w^\star = \underbrace{-\left(R + B^T P^{\text{OLD}} B\right)^{-1} B^T P^{\text{OLD}} A}_{=K^{\text{NEW}}} \cdot x_k \tag{52}$$

   We now have everything to state an iterative method to solve the infinite-time LQR optimal control problem, using policy evaluation and policy improvement.

3. Summary

$$\boxed{\begin{array}{rl} & \text{Summary of Iterative Method to solve Infinite-time LQR} \\ u_k &= K^{j+1} \cdot x_k \\ K^{j+1} &= -\left(R + B^T P^{j+1} B\right)^{-1} B^T P^{j+1} A \\ P^{j+1} &= Q + (K^j)^T R K^j + (A + BK^j)^T P^j (A + BK^j) \end{array}} \qquad \begin{array}{r} (53) \\ (54) \\ (55) \\ (56) \end{array}$$

   In the control systems community, this is called Hewer's method [Hewer 1971].

4 Approximate Dynamic Programming (ADP)

In this section, we finally arrive at methods to perform online adaptive optimal control (i.e. reinforcement learning) using data measured along the system trajectories. You will see that these methods incorporate supervised learning (regression, in particular) and can be model-based or model-free. These methods are broadly called "approximate dynamic programming" [Werbos 1991, 1992] or "neruo-dynamic programming" [Bertsekas 1996].

First, we require two concepts:

1. the temporal difference error, and
2. value function approximation

4.1 Temporal Difference (TD) Error

The Bellman equation used for policy evaluation (shown below) can be thought of as a consistency equation for the value function of a given policy π.

$$V_\pi(x_k) = c(x_k, \pi(x_k)) + \gamma \cdot V_\pi(x_{k+1}) \tag{57}$$

To turn this into an online adaptive method, consider time-varying residual:

$$e_k = c(x_k, \pi(x_k)) + \gamma \cdot V_\pi(x_{k+1}) - V_\pi(x_k) \tag{58}$$

The symbol e_k is known as the "temporal different (TD) error," which is simply RL jargon for the residual in Bellman's equation.

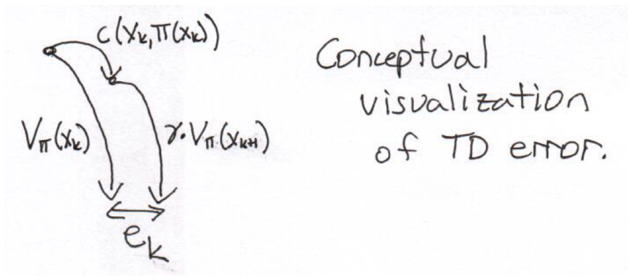
$c(x_k, \pi(x_k))$
$V_\pi(x_k)$
$\gamma \cdot V_\pi(x_{k+1})$
e_k
Conceptual visualization of TD error.

Figure 2: Conceptual visualization of TD error

Suppose that at each time step k we collect data $(x_k, x_{k+1}, c(x_k, \pi(x_k)))$. We can use this data for supervised learning. For example, we can fit a regression model for V_π such that it minimizes the sum of squared residuals, e_k, i.e. perform least squares on the TD errors. Next we describe the regression models to approximate the value function.

4.2 Value Function Approximation

To perform supervised learning on V_π using the TD errors (i.e. Bellman equation residuals), we must parameterize $V_\pi(\cdot)$ is some way.

Consider the Weierstrass higher order approximation theorem: There exists a (dense) basis set $\{\phi_i(x)\}$ such that

$$V_\pi(x) = \sum_{i=1}^{\infty} w_i \phi_i(x) = \sum_{i=1}^{L} w_i \phi_i(x) + \underbrace{\sum_{i=L+1}^{\infty} w_i \phi_i(x)}_{\varepsilon_L} \tag{59}$$

$$= W^T \phi(x) + \varepsilon_L \tag{60}$$

$$\text{where} \quad \phi(x) = [\phi_1(x), \phi_2(x), \cdots, \phi_L(x)]^T \tag{61}$$

$$W = [w_1, w_2, \cdots, w_L]^T \tag{62}$$

and $\varepsilon_L \to 0$ uniformly in x as $L \to \infty$.

One of the main contributions of Werbos and Bertsekas was to use neural networks for the regressor vector $\phi(x)$.

4.3 Example: LQR

In LQR problems, we established:

- $V(x_k) = x_k^T P x_k$ is quardratic
- $u_k = K x_k$ is linear

Then the TD error for LQR is:

$$e_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P x_{k+1} - x_k^T P x_k \tag{63}$$

which is linear in the parameter matrix P. Let's re-write $V(x_k) = x_k^T P x_k$ to enable linear (least squares) regression:

$$V(x_k) = x_k^T P x_k = W^T \phi(x) \tag{64}$$

where $W = \text{vec}(P)$ stacks the columns of P matrix into vector W, and $\phi(x) = x_k \otimes x_k$ is a vector of monomials of x_k. This is best understood by example. Consider:

$$x_k = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad P = \begin{bmatrix} p_{11} & p_{12} \\ * & p_{22} \end{bmatrix} \tag{65}$$

then

$$x_k^T P x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ * & p_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} p_{11}x_1 + p_{12}x_2 \\ p_{12}x_1 + p_{22}x_2 \end{bmatrix} \tag{66}$$

$$= p_{11}x_1^2 + p_{12}x_2x_1 + p_{12}x_1x_2 + p_{22}x_2^2 \tag{67}$$

$$= \underbrace{\begin{bmatrix} p_{11} & 2p_{12} & p_{22} \end{bmatrix}}_{=W^T} \underbrace{\begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}}_{=\phi(x)} \tag{68}$$

$$= W^T \phi(x) \tag{69}$$

Note that because P is symmetric we have $\phi(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n(n+1)/2}$. Using this parameterization, we can re-write the LQR TD error as:

$$e_k = \underbrace{x_k^T Q x_k + u_k^T R u_k}_{=c(x_k,u_k)} + W^T \phi(x_{k+1}) - W^T \phi(x_k) \tag{70}$$

$$= c(x_k, u_k) + W^T \left[\phi(x_{k+1}) - \phi(x_k)\right] \tag{71}$$

The TD error can be computed for supervised learning by, at each time step k, collecting data $(x_k, x_{k+1}, c(x_k, u_k))$.

Remark 3. Previously, DP algorithms required evaluation of Bellman's equation at all $x_k \in \mathcal{X}$. To achieve this computationally, we considered a discrete-valued state space $\mathcal{X}$. This results in an exponential increase in calculations as the state vector size increases, known as the "curse of dimensionality". Value function approximation (and policy approximation) bypasses this challenge!

4.4 Online ADP Algorithm

We are now positioned to write our first online RL algorithms. At each time step k, collect data $(x_k, x_{k+1}, c(x_k, \pi(x_k)))$. Consider value function approximation $V_\pi(x) = W^T \phi(x)$. Then the TD error is:

$$e_k = c(x_k, \pi(x_k)) + \gamma \cdot W^T \phi(x_{k+1}) - W^T \phi(x_k) \tag{72}$$

corresponding to linear regression model (a.k.a. the Bellman equation):

$$c(x_k, \pi(x_k)) = [\phi(x_k) - \gamma \cdot \phi(x_{k+1})]^T W \tag{73}$$

We now provide an online policy iteration algorithm, in which policy evaluation is performed via regression.

Online Policy Iteration

0. **Initialization:** Select any admissible control policy π^0. Set $m = 0$.
1. **Policy Evaluation:** Run control policy π^m on the environment/system for one episode. Collect L measured data tuples $(x_k, x_{k+1}, c(x_k, \pi(x_k)))$. Find the least squares solution w.r.t. W_m for regression model (a.k.a. Bellman equation)

$$\underbrace{\begin{bmatrix} \vdots \\ c(x_k, \pi^m(x_k)) \\ \vdots \end{bmatrix}}_{=C, L\times 1} = \underbrace{\begin{bmatrix} \vdots \\ [\phi(x_k) - \gamma \cdot \phi(x_{k+1})]^T \\ \vdots \end{bmatrix}}_{=\Phi, L\times n_w} \underbrace{W}_{n_w \times 1} \tag{74}$$

written compactly as $C = \Phi W$. For example, the ordinary least squares solution is

$$W_m \leftarrow W^\star = (\Phi^T \Phi)^{-1} \Phi^T C \tag{75}$$

2. **Policy Improvement:** Find an improved policy via

$$\pi^{m+1} = \arg\min_{\pi(\cdot)} \left\{ c(x_k, \pi(x_k)) + \gamma \cdot W_m^T \phi(x_{k+1}) \right\}, \qquad \forall\ x_k \in \mathcal{X} \tag{76}$$

Set $m \leftarrow m + 1$. Go to Step 1.

Remark 4. Besides batch least squares, one may alternatively perform the regression in Step 1 by recursive least squares, gradient method, ridge regression, LASSO regression, etc.

Remark 5. In online policy iteration, the regressor $[\phi(x_k) - \gamma \cdot \phi(x_{k+1})]$ must be "persistently excited" for least squares to converge to a unique solution. This guarantees that $(\Phi^T \Phi)$ in (75) is invertible.

Remark 6. Observe that Step 1 doesn't require knowledge of the dynamics, only data $(x_k, x_{k+1}, c(x_k, \pi(x_k)))$. However, the minimization in Step 2 requires us to solve:

$$\frac{\partial c}{\partial u}(x_k, \pi(x_k)) + \gamma \cdot W^T \frac{\partial \phi}{\partial x}(x_{k+1}) \cdot \frac{\partial f}{\partial u}(x_k, \pi(x_k)) = 0 \tag{77}$$

which requires explicit knowledge of cost function $c(\cdot, \cdot)$ and dynamics function $f(\cdot, \cdot)$.

Remark 7. Let us examine Step 2 again. Here, we must still perform a minimization for all $x_k \in \mathcal{X}$. Thus, we have only partially avoided the curse of dimensionality. This motivates a second regression model (or "actor neural net" in the words of Werbos and Bertsekas) to approximate the policy.

4.5 Actor-Critic Method

The previous remark motivates a second function approximator – for the control policy:

$$u_k = \pi(x_k) = U^T \sigma(x_k) \tag{78}$$

Similar to $\phi(x)$, we define $\sigma(x_k) = [\sigma_1(x_k), \sigma_2(x_k), \cdots, \sigma_M(x_k)]^T$ is a truncated (dense) basis set for $\pi(\cdot)$, and $U^T \in \mathbb{R}^{p \times M}$ are the policy's weights to be learned online.

Now let us return to Step 2: Policy Improvement. After evaluating a given policy in Step 1: Policy Evaluation, we obtain W_m. Now we execute the minimization in Step 2, given measured $(x_k, x_{k+1}, c(x_k, \pi(x_k)))$.

$$\text{minimize}_U \qquad \underbrace{c(x_k, U^T\sigma(x_k)) + \gamma \cdot W_m^T \phi(x_{k+1})}_{=T(U),\text{given } x_k, x_{k+1}} \tag{79}$$

$$\text{where} \quad x_{k+1} = f(x_k, U^T\sigma(x_k)) \tag{80}$$

For convenience, we define $T(U)$ to focus our attention on minimizing the expression w.r.t. policy parameter vector U. A classic approach is gradient descent, i.e.

$$U_{j+1} = U_j - \beta \cdot \frac{\partial T}{\partial U}(U_j), \qquad \text{for } \beta > 0 \tag{81}$$

It is instructive to derive the gradient: $\partial T/\partial U \in \mathbb{R}^{M\times 1}$, where for simplicity we assume a scalar control input.

$$\frac{\partial T}{\partial U}(U_j) = \left[\frac{\partial c}{\partial u}(x_k, U_j^T\sigma(x_k)) \cdot \sigma(x_k) + \gamma \cdot W_m^T \cdot \nabla\phi(x_{k+1}) \cdot \frac{\partial f}{\partial u}\left(x_k, U_j^T\sigma(x_k)\right) \cdot \sigma(x_k)\right] \tag{82}$$

$$= \left[\frac{\partial c}{\partial u}(x_k, U_j^T\sigma(x_k)) + \gamma \cdot W_m^T \cdot \nabla\phi(x_{k+1}) \cdot \frac{\partial f}{\partial u}\left(x_k, U_j^T\sigma(x_k)\right)\right] \cdot \sigma(x_k) \tag{83}$$

The gradient expression is a bit complex in the general case. Let us examine the LQR case:

$$\text{minimize}_U \quad T(U) = x_k^T Q x_k + R u_k^2 + W^T \nabla\phi(x_{k+1}) \tag{84}$$

$$= x_k^T Q x_k + R(U^T\sigma(x_k))^2 + W^T \nabla\phi(Ax_k + BU^T\sigma(x_k)) \tag{85}$$

where the gradient is:

$$\frac{\partial T}{\partial U} = \left[2R(U^T\sigma(x_k)) + W^T\nabla\phi(x_{k+1}) \cdot B\right] \cdot \sigma(x_k) \tag{86}$$

Remark 8. A relevant question is does gradient descent converge to a global minimum? Examine $T(U)$ for the LQR case. $T(U)$ is convex in U if and only if the basis function $\phi(\cdot)$ is convex in its argument. Since we know the value function is quadratic in LQR with positive semi-definite kernal matrix $P \succeq 0$, $\phi(\cdot)$ is convex and gradient descent converges to the global minimum.

Remark 9. Amazingly, in LQR this actor-critic scheme requires no knowledge of system matrix A! Only matrix B appears. More generally, policy evaluation requires zero knowledge of the dynamics. Policy improvement only requires knowledge of the Jacobian $\frac{\partial f}{\partial u}$! Consequently, we have a partial model-based scheme. The next section will introduce the Q-function, our key object for enabling completely model-free schemes.

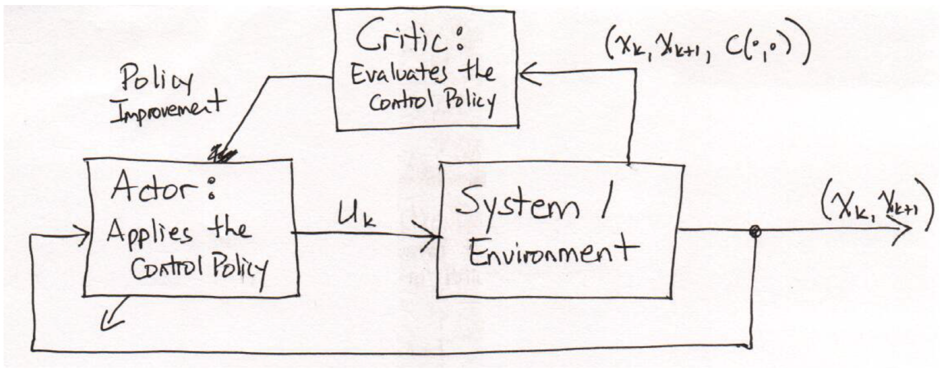
Critic:
Evaluates the Control Policy
$(x_k, x_{k+1}, c(\cdot,\cdot))$
Policy Improvement
Actor:
Applies the Control Policy
u_k
System / Environment
(x_k, x_{k+1})

Figure 3: Schematic of actor-critic RL algorithm.

5 Q-learning

From this section, we are going to study Reinforcement Learning in a stochastic sense. So far, the system is deterministic, which enables to compute the derivative of the system and deal with continuous states and action. The system can be stochastic, which is main difference from previous sections. First of all, we define Markov Decision Process (MDP) to describe the stochastic system.

5.1 Markov Decision Process

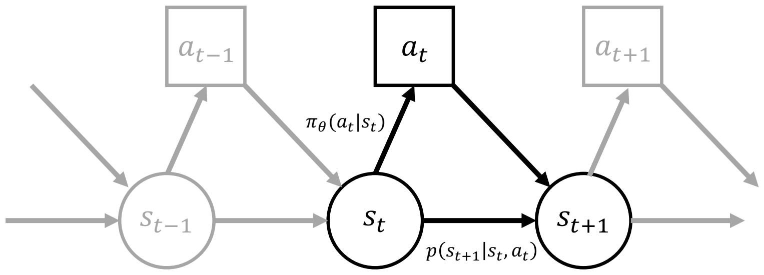

Figure 4: Markov Decision Process

At each time step, the agent performs an action which leads to two things: the state evolves, and then the agent receives a reward from the environment. The goal of the agent is to discover an optimal policy (a.k.a. state feedback controller in the language of controls scientists) such that it maximizes the total value of rewards received from the environment in response to its actions. MDP consists of a tuple of 5 elements:

1 $\mathcal{S}$: Set of states. At each time step the state of the environment is an element, $\boldsymbol{s} \in \mathcal{S}$.

2 $\mathcal{A}$: Set of actions. At each time step the agent chooses an action $\boldsymbol{a} \in \mathcal{A}$ to perform.

3 $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$: Probabilistic state transition model that describes how the environment state changes when the user performs an action $\boldsymbol{a}$.

4 $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$: Reward model that describes the real-valued reward an agent receives from the environment after performing an action. In MDP, the reward value depends on the current state and the action performed, i.e, $r(\boldsymbol{s}, \boldsymbol{a})$.

5 γ: A discount factor that controls the importance of future rewards.

The control policy (i.e. control law) is denoted by the symbol π. We consider randomized policies, so π yields the probability of applying action $\boldsymbol{a}$ conditioned on being in state $\boldsymbol{s}$:

$$\pi(\boldsymbol{a}|\boldsymbol{s}) : \mathcal{A} \times \mathcal{S} \to [0, 1]. \tag{87}$$

The goal of the agent is to pick the best policy that will maximize the total rewards received from the environment. Assume that environment has current states, $\boldsymbol{s}_t$, the agent observes the states $\boldsymbol{s}_t$ and picks an action $\boldsymbol{a}_t$, then upon performing its action the environment state becomes $\boldsymbol{s}_{t+1}$ and the agent receives a reward $\boldsymbol{r}_{t+1}$. Then, the total discounted reward from time t onward can be expressed as:

$$\begin{aligned} R_t &= r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma r(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) + \gamma^2 r(\boldsymbol{s}_{t+2}, \boldsymbol{a}_{t+2}) + \dots \\ &= \sum_{k=0}^{T-1} \gamma^k r\left(s_{t+k}, a_{t+k}\right) \end{aligned} \tag{88}$$

The *state value function*, $V^\pi(\boldsymbol{s})$ is the expected total reward for an agent starting from state $\boldsymbol{s}$. In the controls community, this is sometimes called the cost-to-go, or reward-to-go. Importantly, note the value function depends on the control policy. If the agent uses a given policy π to select actions, the corresponding value function is given by:

$$
\begin{aligned}
V^{\pi}(\boldsymbol{s}_t) &\doteq \mathbb{E}_{\pi}\Big[R_t \mid \boldsymbol{s}_t\Big] \\
&= \mathbb{E}_{\pi}\sum_{t'=t}^{T-1}\left[r(s_{t'}, a_{t'})|s_t\right] \\
&= \sum_{t'=t}^{T-1}\mathbb{E}_{\pi}\left[r(s_{t'}, a_{t'})|s_t\right]
\end{aligned} \tag{89}
$$

Then, the optimal policy π^* is the policy that corresponds to the optimal value function.

$$
\pi^* = \arg\max_{\pi} V^{\pi}(s) \tag{90}
$$

The next definition, known as the "Q-function," is critical for model-free reinforcement learning. Consider the *state-action value function*, $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$, which is a function of the state-action pair and returns a real value. This is the value of taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$, and then following policy π. Mathematically,

$$
\begin{aligned}
Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) &\doteq \mathbb{E}_{\pi}\Big[R_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t\Big] \\
&= r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + \mathbb{E}_{\boldsymbol{s}_{t+1}\sim p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)}\Big[\gamma V^{\pi}(\boldsymbol{s}_{t+1})\Big]
\end{aligned} \tag{91}
$$

The optimal Q-function $Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t)$ gives the expected total reward received by an agent that starts in $\boldsymbol{s}_t$, picks (possibly non-optimal) action ($\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$), and then behaves optimally afterwards. $Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t)$ indicates how good it is for an agent to pick action $\boldsymbol{a}_t$ while being in state $\boldsymbol{s}_t$. Since $V^*(\boldsymbol{s}_t)$ is the maximum expected total reward starting from state $\boldsymbol{s}_t$, it will also be the maximum of $Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t)$ over all possible actions. The relationship between $Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and $V^*(\boldsymbol{s}_t)$ is written as :

$$
V^*(\boldsymbol{s}_t) = \max_{\boldsymbol{a}_t} Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t),\ \forall \boldsymbol{s} \in \mathcal{S} \tag{92}
$$

If we know the optimal Q-function, then the optimal policy can be extracted by choosing the action $\boldsymbol{a}$ that maximizes $Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t)$ for state $\boldsymbol{s}_t$,

$$
\pi^*(s) = \arg\max_{a} Q^*(s, a),\ \forall s \in \mathcal{S} \tag{93}
$$

5.2 Iterated Q-learning

Iterated Q-learning is to update the Q-function iteratively by using the following relationship:

$$
Q_i(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \sum_{s_{t+1}\in\mathcal{S}} p(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{94}
$$

where i is iteration index for specific state and action. This is guaranteed to converge to the optimal one, however, it is required to know the state transition function.

5.3 Q-learning

The key idea of Q-learning (model-free) approach is to approximate

$$
\mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t, a_t)}\left[V(s_{t+1})\right] \approx \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})
$$

by using samples of the next state s_{t+1} in place of expectation. Q-function is iteratively updated by following:

$$
Q(s_t, a_t) \leftarrow r(s_t, a_t) + \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{95}
$$

the algorithm involves computing (95) iteratively, $\forall\ \boldsymbol{s}_t \in \mathcal{S}$. What about $\boldsymbol{a_t}$? A natural idea is to use $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}_t} Q_k(\boldsymbol{s}_t, \boldsymbol{a}_t)$, often referred to as "greedy". However, provable convergence of Q-learning requires visiting all state-action pairs infinitely many times as $k \to \infty$ [1]. The intuitive reason is that a greedy approach allows the state to only evolve within a subset of the entire state-space, to maximize the reward. By only exploring a subset, the algorithm may get stuck within a local maximum for Q_k. This is referred to as the "exploration problem" in reinforcement learning. A common method to alleviate this problem and ensure convergence to the optimum is to use the so-called "ϵ-greedy" approach, where $\boldsymbol{a_t} \sim \pi_\epsilon(\boldsymbol{a}_t|\boldsymbol{s}_t)$:

$$\pi_\epsilon(\boldsymbol{a}_t|\boldsymbol{s}_t) = \begin{cases} 1-\epsilon, & \text{if } \bar{\boldsymbol{a}}_t = \arg\max_{\boldsymbol{a}_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ \epsilon/(|\mathcal{A}|-1), & \boldsymbol{a}_t \in \mathcal{A} \setminus \{\bar{\boldsymbol{a}}_t\} \end{cases} \tag{96}$$

In words, ϵ-greedy does the following: With a probability $1-\epsilon$, the policy takes the current best action, denoted $\bar{\boldsymbol{a}}_t$. All other admissible actions are chosen with equal probability $\epsilon/(|\mathcal{A}|-1)$.

After the Q-learning algorithm has converged, then a deterministic state-feedback control policy can be recovered for implementation by setting $\epsilon = 0$:

$$\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \begin{cases} 1, & \text{if } \boldsymbol{a}_t = \arg\max_{\boldsymbol{a}_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ 0, & \text{otherwise} \end{cases} \tag{97}$$

5.4 Q-learning with function approximator

For a large number of states, tabular case is intractable to iterate Q-function update. A function approximator, i.e., deep neural network is used to represent Q-function. This function approximator is denoted by Q_θ. Gradient descent algorithm is used to train the parameterized Q-function,

$$\begin{aligned} y_i &\leftarrow r(s_{(i,t)}, a_{(i,t)}) + \max_{a_{(i,t+1)}} Q_\theta(s_{(i,t+1)}, a_{(i,t+1)}) \\ \theta &\leftarrow \arg\min_\theta \sum_i \|y^i - Q_\theta(s_{i,t}, a_{i,t})\|^2 \end{aligned}$$

5.5 Deep Q Network (DQN)

Deep-Q-Network (DQN) is proposed by Google Deepmind [2] showing that deep Q-network outperforms human behavior in Atari games. Compared to Q-learning, the main novelty of their work is as follows:

- **Experience replay**: the idea is that by storing an agent's experiences, and randomly drawing batches of them to train the network, agent can learn to perform well in the task. By keeping the experiences agent draws random, we are able to prevent the network from only learning about what it is immediately doing in the environment.
- **Separate target network**: the another major idea is that the utilization of a second network during the training procedure. This second network is used to generate the target-Q values that will be used to compute the loss for every action during training. Target network separation prevents Q-network's values from diverging.

The full algorithm for training deep Q-networks is presented in Algorithm 1. The agent selects and executes actions according to an ε-greedy policy based on Q. Because using histories of arbitrary length as inputs to a neural network can be difficult, Q-function works on a fixed length representation of histories produced by the function ϕ. The DQN-algorithm modifies standard online Q-learning in two ways to make it suitable for training.

First, we use a technique known as experience replay in which we store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data set $D_t = e_1, \ldots, e_t$, pooled over many episodes (where the end of an episode occurs when a terminal state is reached) into a replay memory. During the inner loop of the algorithm, we apply Q-learning updates, or minibatch updates, to samples of experience, $(s, a, r, s^{'})\ U(D)$, drawn at random from the pool of stored samples.

The second modification to online Q-learning aimed at further improving the stability of our method with neural networks is to use a separate network for generating the targets y_j in the Q-learning update. More precisely, every m-updates we clone the network Q to obtain a target network $\hat{Q}$ and use $\hat{Q}$ for generating the Q-learning targets y_j for the following m updates to Q. This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases the target y_j, possibly leading to oscillations or divergence of the policy. Generating the targets using an older set of parameters adds a delay between the time an update to Q is made and the time the update affects the targets y_j, making divergence or oscillations much more unlikely.

Algorithm 1: Deep Q-learning with experience replay

```
Initialize replay memory D to capacity N
Initialize action-value function $Q$ with random weight $\theta$
Initialize target-action value function $\hat{Q}$ with weight $\theta^- = \theta$
for episode = 1, $M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for $t$=1,$T$ do
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network
          parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    end
end
```

References

[1] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Systems*, vol. 32, no. 6, pp. 76–105, 2012.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.