LQR, Inverse Reinforcement Learning, Learning from Expert Demonstrations

Hoang M. Le

April 20, 2016

1 Introduction

This set of notes aims to provide a road map to understand the theory behind the helicopter control application [1].

Typically a reinforcement learning problem can be described by a Markov Decision Process (MDP) consisting of a sextuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, H, I, \mathcal{R})$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, T is the dynamics model, H is the time horizon, $I \subset \mathcal{S}$ is the set of initial states; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward (cost) function.

As we deal with continuous state and action spaces, the focus is on (discrete-time) dynamical systems where the transition dynamics can be described by a (non-linear) function f: $x_{t+1} = f(x_t, u_t)$ for some action $u_t \in \mathcal{A}$.

A policy π is a mapping from states $x \in \mathcal{S}$ to actions $u \in \mathcal{A}$. Acting according to policy π yields expected sum of rewards $V(\pi) = \mathbb{E}\left[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t|\pi)\right]$ (also called value function). The goal is to find an optimal policy π^* that maximizes the expected sum of rewards: $\pi^* = \arg\max_\pi \left[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t|\pi)\right]$.

2 Linear Quadratic Regulator and Iterative LQR

Motivation. The key question from this section is: assume that we know the *dynamics* and *cost functions*, how do we derive optimal actions (design controller) to minimize cost or maximize rewards?

2.1 LQR for Linear Time Invariant Systems

We start with Linear Quadratic Regularization, which is a special case of the general MDP framework where the optimal policy can be computed exactly using dynamic programming.

Consider discrete-time system with linear dynamics:

$$x_{t+1} = Ax_t + Bu_t \tag{1}$$

where $x_t \in \mathcal{S} \subset \mathbb{R}^n, u_t \in \mathcal{A} \subset \mathbb{R}^m$ are continuous state and action (a.k.a control input), with initial condition $x_0 = x^{init}$. For now let's assume $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are independent of time t - this is called Linear Time Invariant (LTI) dynamical system in classical control. We will see later that our derivation of optimal policy extends naturally to the case where $A = A_t$ and $B = B_t$ are dependent on time (Linear Time Varying or LTV system).

The reward function is given by the quadratic form:

$$R(x_t, u_t) = -x_t^\intercal Q x_t - u_t^\intercal R u_t \tag{2}$$

where $Q = Q^\intercal \succeq 0$ and $R = R^\intercal \succeq 0$ are positive semi-definite matrices. For convenience, we use the terms reward and cost function (negative of reward) interchangeably.

Assume that A, B, Q, R **are known**. Optimal actions for linear dynamical systems with quadratic cost can be computed efficiently with standard linear algebra operations. This setting is the basic building-block of the Differential Dynamic Programming (DDP) method used for helicopter control (Section 3.2 of [1]).

Note the interpretation of the quadratic cost function: We want to drive the system from an initial condition $x_0 = x^{init}$ to the equilibrium $(x^*, u^*) = (0, 0)$ with "minimum actions" $u_0, \ldots, u_H$ (see figure 1). By reframing the state variable x and control variable u, we can easily extend this framework to the case of trajectory following.

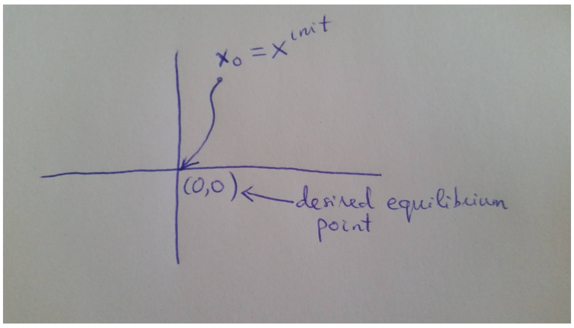

Figure 1: *Basic goal of LQR: stabilize a linear system around an equilibrium*

For $t = 0, \ldots, H$ define the "cost-to-go" $V_t : \mathbb{R}^n \to \mathbb{R}$ by the recursion:

$$\begin{aligned} V_{t+1}(x) &= \min_u x^\intercal Q x + u^\intercal u + \sum_{x' = Ax+Bu} V_t(x') && (3) \\ &= \min_u \left[x^\intercal Q x + u^\intercal R u + V_t(Ax + Bu) \right] && (4) \end{aligned}$$

LQR framework is convenient because all cost-to-go functions take the quadratic form $V_t(x) = x^\intercal P_t x$, for all $x \in \mathbb{R}^n$. This can be proved by backward induction on t.

First note that $V_T(x) = x^\intercal Q x$ is quadratic in x, by definition of the cost function. Then assume $V_t(x) = x^\intercal P_t x$ for all x, we have:

$$\begin{aligned} V_{t-1}(x) &= \min_u \left[x^\intercal Q x + u^\intercal R u + V_t(Ax + Bu) \right] && (5) \\ &= \min_u \left[x^\intercal Q x + u^\intercal R u + (Ax + Bu)^\intercal P_t (Ax + Bu) \right] && (6) \\ &= x^\intercal \underbrace{\left(A^\intercal P_t A + Q - (A^\intercal P_t B)(B^\intercal P_t B)^{-1}(B^\intercal P_t A) \right)}_{P_{t-1}} x && (7) \end{aligned}$$

where equation (7) is obtained by setting the gradient of expression from (6) w.r.t. x to 0. This enables exact dynamic programming solution to the optimal policy via the Ricatti recursion:

1. initialize $P_H \coloneqq Q$
2. for each $t = H, H-1, \ldots, 1$ set:
$$P_{t-1} = A^\intercal P_t A + Q - (A^\intercal P_t B)(B^\intercal P_t B)^{-1}(B^\intercal P_t A) \tag{8}$$
3. at each step, the optimal action (a.k.a control input) is given by
$$u_t^* = \underbrace{-(B^\intercal P_t B + R)^{-1} B^\intercal P_t A}_{K_t} x_t \tag{9}$$

This is a well-known result in control theory: optimal policy for the LQR problem is a linear feedback controller, which can be efficiently computed using dynamic programming. The value function and cost-to-go for any time step t is given by $V_t(x) = x^\intercal P_t x$.

2.2 LQR for Time Varying Systems

The derivation above carries analogously to time varying systems setting, where the dynamics is governed by:

$$x_{t+1} = A_t x_t + B_t u_t \tag{10}$$

with the same quadratic cost function $x_t^\intercal Q x_t + u_t^\intercal R u_t$.

The Ricatti recursion for optimal policy looks almost the same as before, except with time-dependent A_t, B_t:

$$P_{t-1} = A_t^\intercal P_t A_t + Q - (A_t^\intercal P_t B_t)(B_t^\intercal P_t B_t)^{-1}(B_t^\intercal P_t A_t) \tag{11}$$
$$u_t^* = K_t x_t = -(B_t^\intercal P_t B_t + R)^{-1} B_t^\intercal P_t A_t x_t \tag{12}$$

Similar solutions easily extend to time-varying quadratic cost with $Q = Q_t$ and $R = R_t$, but it is not necessary to consider this for the helicopter application.

2.3 LQR Around of Trajectory with Non-Linear Dynamics

Suppose we have a (discrete time) non-linear dynamical system:

$$x_{t+1} = f(x_t, u_t), \; x_0 = x^{init} \tag{13}$$

and a method to generate a trajectory $\{x_t^*, u_t^*\}_{t=0}^H$ that satisfies the dynamics. We want to stabilize the system around this trajectory (because running u_t^* on a real system may not necessarily result in x_t^* due to modeling errors and noises). This objective is an extension of keeping the system around a fixed equilibrium point in the basic LTI LQR setting.

The goal of this trajectory following problem can be formalized as:

$$\min_{u_0,\dots,u_H} \sum_{t=0}^{H} (x_t - x_t^*)^\intercal Q (x_t - x_t^*) + (u_t - u_t^*)^\intercal R (u_t - u_t^*) \tag{14}$$

$$\text{s.t. } x_{t+1} = f(x_t, u_t) \tag{15}$$

We linearize the non-linear system around the desired trajectory by first-order Taylor expansion as follows:

$$x_{t+1} \approx f(x_t^*, u_t^*) + \underbrace{\frac{\partial f}{\partial x}(x_t^*, u_t^*)}_{A_t}(x_t - x_t^*) + \underbrace{\frac{\partial f}{\partial u}(x_t^*, u_t^*)}_{B_t}(u_t - u_t^*) \tag{16}$$

where A_t and B_t are the Jacobian matrices of f w.r.t. x and u, evaluated at points along the trajectory. This linearization transforms the original problem into a linear time varyting (LTV) LQR problem with the approximately linear dynamics:

$$x_{t+1} - x_{t+1}^* \approx A_t(x_t - x_t^*) + B_t(u_t - u_t^*) \tag{17}$$

2.4 Following Expert Trajectory with Iterative LQR

Imagine we have an expert trajectory $\{x_t^*, u_t^*\}_{t=0}^H$ that we wish to follow.

As before, assume we know the dynamics of the non-linear system: $x_{t+1} = f(x_t, u_t), x_0 = x^{init}$, and the objective is to minimize:

$$V(u_0, \dots, u_H) = \sum_{t=0}^{H} (x_t - x_t^*)^\intercal Q (x_t - x_t^*) + (u_t - u_t^*)^\intercal R (u_t - u_t^*) \tag{18}$$

For a given sequence of action u, we could simulate the system to find x using $x_{t+1} = f(x_t, u_t)$. **The key problem** here is that current trajectory $\{x_t, u_t\}_{t=0}^H$ might be far away from the desired expert trajectory $\{x_t^*, u_t^*\}_{t=0}^H$, and thus local approximation by linearizing around the current trajectory $\{x_t, u_t\}$ as described in Section 2.3 does not guarantee to converge to the expert trajectory. See Figure 2 for an example. To address this problem, Differential Dynamic Programming

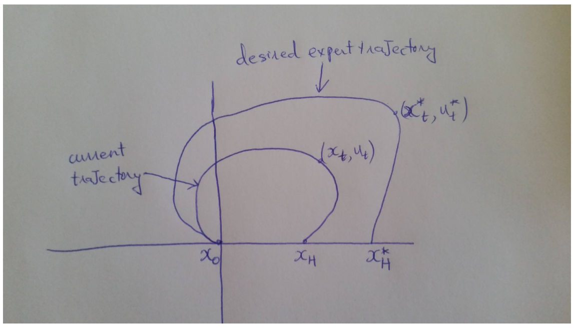

Figure 2: *Linearization around current trajectory may not converge to desired expert trajectory. Thus iterative LQR is used to update the action (control law)*

(Section 3.2 of [1]) is used to iteratively update the actions (or control inputs) as follows:

- Initialize with some guess of $u = \{u_0, \dots, u_H\}$
- With each iteration, given the current u:
  1. simulate the system to find x, using $x_{t+1} = f(x_t, u_t)$
  2. linearize around the current trajectory (as desribed in subsection 2.3) to obtain a linear system

$$\bar{x}_{t+1} = A_t \bar{x}_t + B_t \bar{u}_t \tag{19}$$

$$\text{where } A_t = \frac{\partial f}{\partial x}(x_t, u_t), B_t = \frac{\partial f}{\partial u}(x_t, u_t) \tag{20}$$

  3. solve time-varying LQR problem with cost

$$V = \sum_{t=0}^{H} (x_t + \bar{x}_t - x_t^*)^\intercal Q (x_t + \bar{x}_t - x_t^*) + (u_t + \bar{u}_t - u_t^*)^\intercal R (u_t + \bar{u}_t - u_t^*) \tag{21}$$

  4. update $u_t \coloneqq u_t + \bar{u}_t$ and repeat

Remark: Note that the full version of Differential Dynamic Programming involves not only linear approximation of dynamics but also quadratic approximation of the cost function. The technique used in [1] is essentially iterative LQR, not full-blown DDP.

3 Inverse Reinforcement Learning for Learning the Cost Function

Motivation. So far we have derived the optimal policy for discrete-time, continuous space and action dynamical systems with **known** dynamics and quadratic cost functions:

$$x_{t+1} = f(x_t, u_t) \tag{22}$$

$$V(u_0, \dots, u_H) = \sum_{t=0}^{H} x_t^\intercal Q x_t + u_t^\intercal R u_t \tag{23}$$

It is common to assume that the cost / reward function is given (AlphaGo example: most of the time the cost is 0, cost is -1 or +1 when the game ends). However, in many cases a natural specification of the cost function is difficult, and may need to be learned from data. The problem of learning a cost / reward function from observed behavior is referred to as inverse reinforcement learning [2].

The key assumption from [2] is that the true reward function can be expressed as a linear combination of known “features”.

In the helicopter example, we can think of the cost matrices Q and R as diagonal matrices:

$$Q = \begin{bmatrix} q_1 & 0 & 0 & \dots & 0 \\ 0 & q_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & q_n \end{bmatrix}, R = \begin{bmatrix} r_1 & 0 & 0 & \dots & 0 \\ 0 & r_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & r_m \end{bmatrix} \tag{24}$$

Designing cost matrices Q and R then becomes the problem of choosing the coefficients that trade-off different state and action variables. The cost function in this case becomes:

$$x^\intercal Qx + u^\intercal Ru = w^* \cdot \phi(x, u) \text{ where } w^* = \begin{bmatrix} q_1 \\ \vdots \\ q_n \\ r_1 \\ \vdots \\ r_m \end{bmatrix} \in \mathbb{R}^{m+n} \tag{25}$$

The unknown vector w^* specifies the relative weighting between different desiderata (see section 3.3 of [1]).

Learning goal. The expected value of a policy π can be written as:

$$\mathbb{E}_{x_0 \sim I}\left[V^\pi(x_0)\right] = \mathbb{E}\left[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t)\right] = \mathbb{E}\left[\sum_{t=0}^{H} w \cdot \phi(x_t, u_t)\right] \tag{26}$$

$$= w \cdot \mathbb{E}\left[\sum_{t=0}^{H} \phi(x_t, u_t)\right] = w \cdot \mu(\pi) \tag{27}$$

where $\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{H} \phi(x_t, u_t)\right]$ is called the *feature expectation.* Thus the value function can be compactly expressed as an inner product of w and feature expectation of the policy.

To learn such a set of coefficients w, we use demonstrations by some expert policy π_E, with the corresponding feature expectation $\mu_E = \mu(\pi_E)$. Assume a bounded set of coefficients $w \in \mathbb{R}^k$ with $\|w\|_1 \leq 1$, **the goal is to find a policy** $\tilde{\pi}$ such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon$, since such a policy $\tilde{\pi}$ will yield:

$$\left|\mathbb{E}\left[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t)|\pi_E\right] - \mathbb{E}\left[\sum_{t=0}^{H} \mathcal{R}(x_t, u_t)|\pi_E\right]\right| = |w^\intercal \mu(\tilde{\pi}) - w^\intercal \mu_E| \tag{28}$$

$$\leq \|w\|_2 \|\mu(\tilde{\pi}) - \mu_E\|_2 \tag{29}$$

$$\leq 1 \cdot \epsilon = \epsilon \tag{30}$$

Inverse RL Algorithm. The main algorithm proceeds as follows: (section 3 of [2])

- Initialize: randomly pick some policy π_0, compute (or approximate via Monte Carlo) $\mu_0 = \mu(\pi_0)$
- Main Loop:
  1. for each $i \geq 1$, compute

     $$t_i = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0,\ldots,i-1\}} w^\intercal(\mu_E - \mu_j) \tag{31}$$

     and let w_i be the value of w that attains this maximum.
  2. if $t_i \leq \epsilon$, terminate

3. otherwise, using some RL algorithm to compute the optimal policy π_i for the MDP using rewards $\mathcal{R} = w_i^\intercal \phi$
4. Compute (or estimate) $\mu_i = \mu(\pi_i)$
5. set $i = i + 1$, and go back to Step 1

Note that the optimization in Step 1 of the main loop is equivalent to an SVM optimization problem:

$$\max_{t,w} \quad t \tag{32}$$

$$\text{s.t. } w^\intercal \mu_E \geq w^\intercal \mu_j + t, j = 0, \ldots, i-1 \tag{33}$$

$$\|w\|_2 \leq 1 \tag{34}$$

Suppose the algorithm terminates after n steps with $t_{n+1} \leq \epsilon$, then as a consequence of the optimization problem above, we have:

$$\forall w \text{ with } \|w\|_2 \leq 1 \quad \exists i \text{ s.t. } w^\intercal \mu_i \geq w^\intercal \mu_E - \epsilon \tag{35}$$

In particular, since $\|w^*\|_2 \leq \|w^*\|_1 \leq 1$, this means there is at least one policy among $\pi_0, \ldots, \pi_n$ whose performance under $\mathcal{R}^*$ is at least as good as the expert's performance minus ϵ.

Analysis. The algorithm is guaranteed to terminate after $n = O(\frac{k}{\epsilon^2} \log \frac{k}{\epsilon})$ iterations, where k is the dimension of feature mapping ϕ (Theorem 1 of [2]). Finally, although the algorithm optimizes over expert feature expectation π_E, this quantity is often unknown and thus m different Monte Carlo samples of expert trajectories are obtained to provide an estimate $\hat{\pi}_E$ of π_E (empirical mean of m estimates). Theorem 2 of [2] provides a sufficient number of expert trajectories needed to guarantee the correctness of the algorithm with high probability. See [2] for detailed theorem statement and proof.

4 Learning Dynamics from Expert Demonstrations

Motivation. Techniques from section 2 and section 3 assume that somehow the dynamics of the system is known. In fact, this is frequently the most difficult part of the reinforcement learning problem. Many existing methods (E^3,tree search, model-based RL with ϵ-greedy) require extensive exploration to accurately learn the dynamics, which could be computationally intractable, or could lead the systems to unsafe trajectories (e.g. the helicopter may crash while trying to aggressively explore poorly modeled parts of the state space). The key idea from [3] leverages expert demonstrations to lessen this burden on exploration and focus the learning on repeatedly executing the exploitation policies.

For the helicopter control application, we assume *linearly parameterized dynamics* given by:

$$x_{t+1} = A\phi(x_t) + Bu_t + w_t \tag{36}$$

where ϕ is a feature mapping of the state space. Note that this is **not a linear dynamical system**, only a linearly parameterized one, since ϕ may be non-linear. The process noise w_t is assumed to be IID multivariate Gaussian with known variance. The key question from this section is: how do we estimate matrices A and B from expert data collected from expert policy π_E?

Algorithm. The rephrased version of the **main algorithm** proceeds as follows (section 4 of [3]):

- given $\alpha > 0$, parameters N_E and k_1
- Initialize: run N_E trials from expert π_E. Collect the state-action trajectories during these trials. Estimate the value function $\widehat{V}(\pi_E)$ of expert π_E by averaging the sum of rewards in each of N_E trials. Set i = 1
- Main Loop:
  1. *aggregate the state-action trajectories from (unsuccessful) test to the training set* so far, and use the combined data set to estimate A and B using regularized linear regression. Call the estimated dynamics $\widehat{T}_i$
  2. derive the optimal policy of the system with dynamics $\widehat{T}_i$ using (iterative) LQR. Call this policy π_i
  3. evaluate the value function of π_i by running k_1 trials on the real system. Let $\widehat{V}(\pi_i)$ be the average sum of rewards of the k_1 trials
  4. If $\widehat{V}(\pi_i) \geq \widehat{V}(\pi_E) - \alpha/2$, return π_i and exit. Otherwise set $i = i + 1$ and return to step 1.

For the regularized linear regression step from step 1 of the main loop, the k^{th} rows of A and B are estimated by:

$$\arg\min_{A_{k,:},B_{k,:}} \sum_j \left(x_{next}^{(j)} - (A_{k,:}\phi(x_{curr}^{(j)}) + B_{k,:}u_{curr}^{(j)}) \right)^2 + \frac{1}{\lambda^2}(\|A_{k,:}\|_2^2 + \|B_{k,:}\|_2^2) \quad (37)$$

where j indexes over all state-action-state triples $\{(x_{curr}^{(j)}, u_{curr}^{(j)}, x_{next}^{(j)})\}_j$ occuring after each other in the trajectories observed for the system.

Analysis. The key takeaway from theorem 3 (main theorem of [3]) is that using a polynomial number of trials N_E, k_1 and after a polynomial number of iterations, the returned policy will be approximately optimal, in the sense that the *true* value function $V(\pi)$ of the returned policy π will be no less than α within the true value function of the expert policy, with high probability.

The proof rests on showing the following two facts:

1. After we have collected sufficient data from the expert, the estimated model is accurate for evaluating the value function of the expert's policy in every iteration of the algorithm. (Note this does not merely require that the model has to be accurate after the N_E trials under the expert's policy, but also has to stay accurate after extra data is collected from testing the policies $\{\pi_i\}$)
2. One can visit inaccurately modeled state-action pairs only a "small" number of times until all state-action pairs are accurately modeled

Below is a high-level intuition behind how these two facts are proved. After we have collected sufficient data from the expert, the state-action pairs that are visited often under the expert's policy are modeled well. From the Simulation Lemma (Lemma 1 from [3]), we know that an accurate model of the state-action pairs visited often under the expert's policy is sufficient for accurate evaluation of the value function of the expert's policy. This establishes (1). Every time an inaccurate state-action pair is visited, the data collected for that state-action pair can be used to improve the model. However the model can be improved only a "small" number of times until it is accurate for all state-action pairs. This establishes (2).

Acknowledgement

These notes, especially the optimal control sections, were prepared in consultation with Pieter Abbeel's lecture slides at UC Berkeley CS287 class and lecture slides from Stephen Boyd's EE363 class at Stanford University.

References

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

[2] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.

[3] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2005.