11

Sequential decision making and dynamic programming

As the previous chapters motivated we don't just make predictions for their own sake, but rather use data to inform decision making and action. This chapter examines sequential decisions and the interplay between predictions and actions in settings where our repeated actions are directed towards a concrete end-goal. It will force us to understand statistical models that evolve over time and the nature of dependencies in data that is temporally correlated. We will also have to understand feedback and its impact on statistical decision-making problems.

In machine learning, the subfield of using statistical tools to direct actions in dynamic environments is commonly called "reinforcement learning" (RL). However, this blanket term tends to lead people towards specific solution techniques. So we are going to try to maintain a broader view of the area of *sequential decision making*, including perspectives from related fields of predictive analytics and optimal control. These multiple perspectives will allow us to highlight how RL is different from the machine learning we are most familiar with.

This chapter will follow a similar flow to our study of prediction. We will formalize a temporal mathematical model of sequential decision making involving notions from *dynamical systems*. We will then present a common optimization framework for making sequential decisions when models are known: *dynamic programming*. Dynamic programming will enable algorithms for finding or approximating optimal decisions under a variety of scenarios. In the sequel, we will turn to the learning problem of how to best make sequential decisions when the mechanisms underlying dynamics and costs are not known in advance.

From predictions to actions

Let's first begin with a discussion of how sequential decision making differs from static prediction. In our study of decision theory, we laid out a framework for making optimal predictions of a binary covariate Y when we had access to data X, and probabilistic models of how X and Y were related.

Supervised learning was the resulting problem of making such decisions from data rather than probabilistic models.

In sequential decision making, we add two new variables. First, we incorporate *actions* denoted U that we aim to take throughout a procedure. We also introduce rewards R that we aim to maximize. In sequential decision making, the goal is to analyze the data X and then subsequently choose U so that R is large. We have explicit agency in choosing U and are evaluated based on some quality scoring of U and X. There are an endless number of problems where this formulation is applied from supply chain optimization to robotic planning to online engagement maximization. Reinforcement learning is the resulting problem of taking actions so as to maximize rewards where our actions are only a function of previously observed data rather than probabilistic models. Not surprisingly, the optimization problem associated with sequential decision making is more challenging than the one that arises in decision theory.

Dynamical systems

In addition to the action variable in sequential decision making, another key feature of sequential decision making problems is the notion of time and sequence. We assume data is collected in an evolving process, and our current actions influence our future rewards.

We begin by bringing all of these elements together in the general definition of a discrete time dynamical system. The definitions both simple and broad. We will illustrate with several examples shortly.

A *dynamical system model* has a *state* X_t, *exogenous input* U_t modeling our *control action*, and *reward* R_t. The state evolves in discrete time steps according to the equation

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

where W_t is a random variable and f_t is a function. The reward is assumed to be a function of these variables as well:

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

for some function g_t. To simplify our notation throughout, we will commonly write R explicitly as a function of (X, U, W), that is, $R_t[X_t, U_t, W_t]$.

Formally, we can think of this definition as a structural equation model that we also used to define causal models. After all, the equations above give us a way to incrementally build up a data-generating process from noise variables. Whether or not the dynamical system is intended to capture

any causal relationships in the real world is a matter of choice. Practitioners might pursue this formalism for reward maximization without modeling causal relationships. A good example is the use of sequential decision making tools for revenue maximization in targeted advertising. Rather than modeling causal relationships between, say, preferences and clicks, targeted advertising heavily relies on all sorts of signals, be they causal or not.

Concrete examples

Grocery shopping. Bob really likes to eat cheerios for breakfast every morning. Let the state X_t denotes the amount of Cheerios in Bob's kitchen on day t. The action U_t denotes the amount of Cheerios Bob buys on day t and W_t denotes the amount of Cheerios he eats that day. The random variable W_t varies with Bob's hunger. This yields the dynamical system

$$X_{t+1} = X_t + U_t - W_t \,.$$

While this example is a bit cartoonish, it turns out that such simple models are commonly used in managing large production supply chains. Any system where resources are stochastically depleted and must be replenished can be modeled comparably. If Bob had a rough model for how much he eats in a given day, he could forecast when his supply would be depleted. And he could then minimize the number of trips he'd need to make to the grocer using optimal control.

Moving objects. Consider a physical model of a flying object. The simplest model of the dynamics of this system are given by Newton's laws of mechanics. Let Z_t denote the position of the vehicle (this is a three-dimensional vector). The derivative of position is velocity

$$V_t = \frac{\partial Z_t}{\partial t} \,,$$

and the derivative of velocity is acceleration

$$A_t = \frac{\partial V_t}{\partial t} \,.$$

Now, we can approximate the rules in discrete time with the simple Taylor approximations

$$\begin{aligned} Z_{t+1} &= Z_t + \Delta V_t \\ V_{t+1} &= V_t + \Delta A_t \end{aligned}$$

We also know by Newton's second law, that acceleration is equal to the total applied force divided by the mass of the object: $F = mA$. The flying object will be subject to external forces such as gravity and wind W_t and it

will also receive forces from its propellers U_t. Then we can add this to the equations to yield a model

$$Z_{t+1} = Z_t + \Delta V_t$$
$$V_{t+1} = V_t + \frac{\Delta}{m}(W_t + U_t)\,.$$

Oftentimes, we only observe the acceleration through an accelerometer. Then estimating the position and velocity becomes a filtering problem. Optimal control problems for this model include flying the object along a given trajectory or flying to a desired location in a minimal amount of time.

Markov decision processes

Our definition in terms of structural equations is not the only dynamical model used in machine learning. Some people prefer to work directly with probabilistic transition models and conditional probabilities. In a *Markov Decision Process*, we again have a state X_t and input U_t, and they are linked by a probabilistic model

$$\mathbb{P}[X_{t+1} \mid X_t, U_t]\,.$$

This is effectively the same as the structural equation model above except we hide the randomness in this probabilistic notation.

Example: machine repair. The following example illustrates the elegance of the conditional probability models for dynamical systems. This example comes from Bertsekas.[1] Suppose we have a machine with ten states of repair. State 10 denotes excellent condition and 1 denotes the inability to function. Every time one uses the machine in state j, it has a probability of falling into disrepair, given by the probabilities $\mathbb{P}[X_{t+1} = i \mid X_t = j]$, where $\mathbb{P}[X_{t+1} = i \mid X_t = j] = 0$ if $i > j$. The action a one can take at any time is to repair the machine, resetting the system state to 10. Hence

$$\mathbb{P}[X_{t+1} = i \mid X_t = j, U_t = 0] = \mathbb{P}[X_{t+1} = i \mid X_t = j]$$

and

$$\mathbb{P}[X_{t+1} = i \mid X_t = j, U_t = 1] = \mathbb{1}\{i = 10\}\,.$$

While we could write this dynamical system as a structural equation model, it is more conveniently expressed by these probability tables.

Optimal sequential decision making

Just as risk minimization was the main optimization problem we studied in static decision theory, there is an abstract class of optimization problems

that underlie most sequential decision making (SDM) problems. The main problem is to find a sequence of decision *policies* that maximize a cumulative reward subject to the uncertain, stochastic system dynamics. At each time, we assign a reward $R_t(X_t, U_t, W_t)$ to the current state-action pair. The goal is to find a sequence of actions to make the summed reward as large as possible:

$$\begin{array}{ll} \text{maximize}_{\{u_t\}} & \mathbb{E}_{W_t}\left[\sum_{t=0}^{T} R_t(X_t, u_t, W_t)\right] \\ \text{subject to} & X_{t+1} = f_t(X_t, u_t, W_t) \\ & (x_0 \text{ given}) \end{array}$$

Here, the expected value is over the sequence of stochastic disturbance variables W_t. Note here, W_t is a random variable and X_t are is hence also a random variable. The sequence of actions $\{u_t\}$ is our decision variable. It could be chosen via a random or deterministic procedure as a matter of design. But it is important to understand what information is allowed to be used in order to select u_t.

Since the dynamics are stochastic, the optimal SDM problem typically allows a policy to observe the state before deciding upon the next action. This allows a decision strategy to continually mitigate uncertainty through feedback. This is why we optimize over policies rather than over a deterministic sequence of actions. That is, our goal is to find functions of the current state π_t such that $U_t = \pi_t(X_t, X_{t-1}, \ldots)$ is optimal in expected value. By a *control policy* (or simply "a policy") we mean a function that takes a trajectory from a dynamical system and outputs a new control action. In order for π_t to be implementable, it must only have access only to previous states and actions.

The policies π_t are the decision variables of the problem:

$$\begin{array}{ll} \text{maximize}_{\pi_t} & \mathbb{E}_{W_t}\left[\sum_{t=0}^{T} R_t(X_t, U_t, W_t)\right] \\ \text{subject to} & X_{t+1} = f(X_t, U_t, W_t) \\ & U_t = \pi_t(X_t, X_{t-1}, \ldots) \\ & (x_0 \text{ given}) \end{array}$$

Now, U_t is explicitly a random variable as it is a function of the state X_t.

This SDM problem will be the core of what we study in this chapter. And our study will follow a similar path to the one we took with decision theory. We will first study how to solve these SDM problems when we know the model. There is a general purpose solution for these problems known as *dynamic programming*.

Dynamic programming

The dynamic programming solution to the SDM problem is based on the *principle of optimality*: if you've found an optimal control policy for a time horizon of length T, $\pi_1, \ldots, \pi_T$, and you want to know the optimal strategy starting at state x at time t, then you just have to take the optimal policy starting at time t, $\pi_t, \ldots, \pi_T$. The best analogy for this is based on driving directions: if you have mapped out an optimal route from Seattle to Los Angeles, and this path goes through San Francisco, then you must also have the optimal route from San Francisco to Los Angeles as the tail end of your trip. Dynamic programming is built on this principle, allowing us to recursively find an optimal policy by starting at the final time and going backwards in time to solve for the earlier stages.

To proceed, define the *Q-function* to be the mapping:

$$\begin{aligned} \mathcal{Q}_{a\to b}(x,u) = \max_{\{u_t\}} \mathop{\mathbb{E}}_{W_t} \left[\sum_{t=a}^{b} R_t(X_t, u_t, W_t) \right] \\ \text{s.t.} \quad X_{t+1} = f_t(X_t, u_t, W_t), \quad (X_a, u_a) = (x,u) \end{aligned}$$

The Q-function determines the best achievable value of the SDM problem over times a to b when the action at time a is set to be u and the initial condition is x. It then follows that the optimal value of the SDM problem is $\max_u \mathcal{Q}_{0\to T}(x_0, u)$, and the optimal policy is $\pi(x_0) = \arg\max_u \mathcal{Q}_{0\to T}(x_0, u)$. If we had access to the Q-function for the horizon $0, T$, then we'd have everything we'd need to know to take the first step in the SDM problem. Moreover, the optimal policy is only a function of the current state of the system. Once we see the current state, we have all the information we need to predict future states, and hence we can discard the previous observations.

We can use dynamic programming to compute this Q-function and the Q-function associated with every subsequent action. That is, clearly we have that the terminal Q-function is

$$\mathcal{Q}_{T\to T}(x,u) = \mathop{\mathbb{E}}_{W_T} \left[R_T(x,u,W_T) \right],$$

and then compute recursively

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

This expression is known as Bellman's equation. We also have that for all times t, the optimal policy is $u_t = \arg\max_u \mathcal{Q}_{t\to T}(x_t, u)$ and the policy depends only on the current state.

To derive this form of the Q-function, we assume inductively that this form is true for all times beyond $t+1$ and then have the chain of identities

$$\begin{aligned}
\mathcal{Q}_{t\to T}(x,u) &= \max_{\pi_{t+1},\ldots,\pi_T} \mathop{\mathbb{E}}_{w}\left[R_t(x,u,W_t) + \sum_{s=t+1}^{T} R_s(X_s,\pi_s(X_s),W_s)\right] \\
&= \mathop{\mathbb{E}}_{W_t}\left[R_t(x,u,W_t) + \max_{\pi_{t+1},\ldots,\pi_T} \mathop{\mathbb{E}}_{W_{t+1},\ldots,W_T}\left\{\sum_{s=t+1}^{T} R_s(X_s,\pi_s(X_s),W_s)\right\}\right] \\
&= \mathop{\mathbb{E}}_{W_t}\left[R_t(x,u,W_t) + \max_{\pi_{t+1}} Q\left\{f(x,u,W_t),\pi_{t+1}(f(x,u,W_t))\right\}\right] \\
&= \mathop{\mathbb{E}}_{W_t}\left[R_t(x,u,W_t) + \max_{u'} Q(f(x,u,W_t),u')\right].
\end{aligned}$$

Here, the most important point is that the maximum can be exchanged with the expectation with respect to the first W_t. This is because the policies are allowed to make decisions based on the history of observed states, and these states are deterministic functions of the noise process.

Infinite time horizons and stationary policies

The Q-functions we derived for these finite time horizons are *time varying*. One applies a different policy for each step in time. However, on long horizons with time invariant dynamics and costs, we can get a simpler formula. First, for example, consider the limit:

$$\begin{array}{ll}
\text{maximize} & \lim_{N\to\infty} \mathbb{E}_{W_t}[\frac{1}{N}\sum_{t=0}^{N} R(X_t,U_t,W_t)] \\
\text{subject to} & X_{t+1} = f(X_t,U_t,W_t),\ U_t = \pi_t(X_t) \\
& (x_0 \text{ given}).
\end{array}$$

Such infinite time horizon problems are referred to as *average cost* dynamic programs. Note that there are no subscripts on the rewards or transition functions in this model.

Average cost dynamic programming is deceptively difficult. These formulations are not directly amenable to standard dynamic programming techniques except in cases with special structure. A considerably simpler infinite time formulation is known as *discounted* dynamic programming, and this is the most popular studied formulation. Discounting is a mathematical convenience that dramatically simplifies algorithms and analysis. Consider the SDM problem

$$\begin{array}{ll}
\text{maximize} & (1-\gamma)\,\mathbb{E}_{W_t}[\sum_{t=0}^{\infty} \gamma^t R(X_t,U_t,W_t)] \\
\text{subject to} & X_{t+1} = f(X_t,U_t,W_t),\ U_t = \pi_t(X_t) \\
& (x_0 \text{ given}).
\end{array}$$

where γ is a scalar in $(0,1)$ called the *discount factor*. For γ close to 1, the discounted reward is approximately equal to the average reward. However, unlike the average cost model, the discounted cost has particularly clean optimality conditions. If we define $\mathcal{Q}_\gamma(x,u)$ to be the Q-function obtained from solving the discounted problem with initial condition x, then we have a discounted version of dynamic programming, now with the same Q-functions on the left and right hand sides:

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

The optimal policy is now for *all times* to let

$$u_t = \arg\max_u \mathcal{Q}_\gamma(x_t,u).$$

The policy is time invariant and one can execute it without any knowledge of the reward or dynamics functions. At every stage, one simply has to maximize a function to find the optimal action. Foreshadowing to the next chapter, the formula additionally suggest that the amount that needs to be "learned" in order to "control" is not very large for these infinite time horizon problems.

Computation

Though dynamic programming is a beautiful universal solution to the very general SDM problem, the generality also suggests computational barriers. Dynamic programming is only efficiently solvable for special cases, and we now describe a few important examples.

Tabular MDPs

Tabular MDPs refer to Markov Decision Processes with small number of states and actions. Say that there are S states and A actions. Then the transition rules are given by tables of conditional probabilities $\mathbb{P}[X_{t+1}|X_t,U_t]$, and the size of such tables are S^2A. The Q-functions for the tabular case are also tables, each of size SA, enumerating the cost-to-go for all possible state action pairs. In this case, the maximization

$$\max_{u'} \mathcal{Q}_{a\to b}(x,u')$$

corresponds to looking through all of the actions and choosing the largest entry in the table. Hence, in the case that the rewards are deterministic

functions of x and u, Bellman's equation simplifies to

$$\mathcal{Q}_{t\to T}(x,u) = R_t(x,u) + \sum_{x'} \mathbb{P}[X_{t+1} = x' | X_t = x, U_t = u] \max_{u'} \mathcal{Q}_{t+1\to T}(x',u') \,.$$

This function can be computed by elementary matrix-vector operations: the Q-functions are $S \times A$ arrays of numbers. The "max" operation can be performed by operating over each row in such an array. The summation with respect to x' can be implemented by multiplying a $SA \times S$ array by an S-dimensional vector. We complete the calculation by summing the resulting expression with the $S \times A$ array of rewards. Hence, the total time to compute $\mathcal{Q}_{t\to T}$ is $O(S^2A)$.

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*. In control design, this class of problems is generally referred to as the problem of the Linear Quadratic Regulator (LQR):

$$\begin{array}{ll} \text{minimize} & \mathbb{E}_{W_t}\left[\frac{1}{2}\sum_{t=0}^{T} X_t^T \Phi_t X_t + U_t^T \Psi_t U_t\right], \\ \text{subject to} & X_{t+1} = A_t X_t + B_t U_t + W_t,\ U_t = \pi_t(X_t) \\ & (x_0 \text{ given}). \end{array}$$

Here, Φ_t and Ψ_t are most commonly positive semidefinite matrices. w_t is noise with zero mean and bounded variance, and we assume W_t and $W_{t'}$ are independent when $t \neq t'$. The state transitions are governed by a linear update rule with A_t and B_t appropriately sized matrices. We also abide by the common convention in control textbooks to pose the problem as a minimization—not maximization—problem.

As we have seen above, many systems can be modeled by linear dynamics in the real world. However, we haven't yet discussed cost functions. It's important to emphasize here that cost functions are *designed* not given. Recall back to supervised learning: though we wanted to minimize the number of errors made on out-of-sample data, on in-sample data we minimized convex surrogate problems. The situation is exactly the same in this more complex world of dynamical decision making. Cost functions are designed by the engineer so that the SDM problems are tractable but also so that the desired outcomes are achieved. Cost function design is part of the toolkit for online decision making, and quadratic costs can often yield surprisingly good performance for complex problems.

Quadratic costs are also attractive for computational reasons. They are convex as long as Φ_t and Ψ_t are positive definite. Quadratic functions are

closed under minimization, maximization, addition. And for zero mean noise W_t with covariance Σ, we know that the noise interacts nicely with the cost function. That is, we have

$$\mathbb{E}_W[(x+W)^T M(x+W)] = x^T M x + \mathrm{Tr}(M\Sigma)$$

for any vector x and matrix M. Hence, when we run dynamic programming, every Q-function is necessarily quadratic. Moreover, since the Q-functions are quadratic, the optimal action is a *linear function* of the state

$$U_t = -K_t X_t$$

for some matrix K_t.

Now consider the case where there are static costs $\Phi_t = \Phi$ and $\Psi_t = \Psi$, and time invariant dynamics such that $A_t = A$ and $B_t = B$ for all t. One can check that the Q-function on a finite time horizon satisfies a recursion

$$\mathcal{Q}_{t\to T}(x,u) = x^T \Phi x + u^T \Psi u + (Ax+Bu)^T M_{t+1}(Ax+Bu) + c_t\,.$$

for some positive definite matrix M_{t+1}. In the limit as the time horizon tends to infinity, the optimal control *policy* is *static, linear state feedback*:

$$u_t = -K x_t\,.$$

Here the matrix K is defined by

$$K = (\Psi + B^T M B)^{-1} B^T M A$$

and M is a solution to the *Discrete Algebraic Riccati Equation*

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

Here, M is the unique solution of the Riccati equation where all of the eigenvalues of $A - BK$ have magnitude less than 1. Finding this specific solution is relatively easy using standard linear algebraic techniques. It is also the limit of the Q-functions computed above.

Policy and value iteration

Two of the most well studied methods for solving such discounted infinite time horizon problems are *value iteration* and *policy iteration*. Value iteration proceeds by the steps

$$\mathcal{Q}_{k+1}(x,u) = \mathop{\mathbb{E}}_{W}\left[R(x,u,W) + \gamma \max_{u'} \mathcal{Q}_k(f(x,u,W),u')\right]\,.$$

That is, it simply tries to solve the Bellman equation by running a fixed point operation. This method succeeds when the iteration is a contraction mapping, and this occurs in many contexts.

On the other hand, Policy Iteration is a two step procedure: *policy evaluation* followed by *policy improvement*. Given a policy π_k, the policy evaluation step is given by

$$\mathcal{Q}_{k+1}(x,u) = \mathbb{E}\left[R(x,u,W) + \gamma \mathcal{Q}_k(f(x,\pi_k(x),W),\pi_k(x))\right] .$$

And then the policy is updated by the rule

$$\pi_{k+1}(x) = \arg\max_u \mathcal{Q}_{k+1}(x,u) .$$

Often times, several steps of policy evaluation are performed before updating the policy.

For both policy and value iteration, we need to be able to compute expectations efficiently and must be able to update *all values* of x and u in the associated Q functions. This is certainly doable for tabular MDPs. For general low dimensional problems, policy iteration and value iteration can be approximated by gridding state space, and then treating the problem as a tabular one. Then, the resulting Q function can be extended to other (x,u) pairs by interpolation. There are also special cases where the maxima and minima yield closed form solutions and hence these iterations reduce to simpler forms. LQR is a canonical example of such a situation.

Model predictive control

If the Q-functions in value or policy iteration converge quickly, long-term planning might not be necessary, and we can effectively solve infinite horizon problem with short-term planning. This is the key idea behind one of the most powerful techniques for efficiently and effectively finding quality policies for SDM problems called *model predictive control*.

Suppose that we aim to solve the infinite horizon average reward problem:

$$\begin{array}{ll} \text{maximize} & \lim_{T\to\infty} \mathbb{E}_{W_t}[\frac{1}{T}\sum_{t=0}^{T} R_t(W_t,U_t)] \\ \text{subject to} & X_{t+1} = f_t(X_t,U_t,W_t) \\ & U_t = \pi_t(X_t) \\ & (x_0 \text{ given}). \end{array}$$

Model Predictive Control computes an *open loop* policy on a finite horizon H

$$\begin{array}{ll} \text{maximize}_{u_t} & \mathbb{E}_{W_t}[\sum_{t=0}^{H} R_t(X_t,u_t)] \\ \text{subject to} & X_{t+1} = f_t(X_t,u_t,W_t) \\ & (X_0 = \text{x}). \end{array}$$

This gives a sequence $u_0(x), \ldots, u_H(x)$. The policy is then set to be $\pi(x) = u_0(x)$. After this policy is executed, we observe a new state, x', based on the dynamics. We then recompute the optimization, now using $x_0 = x'$ and setting the action to be $\pi(x') = u_0(x')$.

MPC is a rather intuitive decision strategy. The main idea is to plan out a sequence of actions for a given horizon, taking into account as much uncertainty as possible. But rather than executing the entire sequence, we play the first action and then gain information from the environment about the noise. This direct feedback influences the next planning stage. For this reason, model predictive control is often a successful control policy even when implemented with inaccurate or approximate models. Model Predictive Control also allows us to easily add a variety of constraints to our plan at little cost, such as bounds on the magnitude of the actions. We just append these to the optimization formulation and then lean on the computational solver to make us a plan with these constraints.

To concretely see how Model Predictive Control can be effective, it's helpful to work through an example. Let's suppose the dynamics and rewards are time invariant. Let's suppose further that the reward function is bounded above, and there is some state-action pair $(x_\star, u_\star)$ which achieves this maximal reward $R_{\max}$.

Suppose we solve the finite time horizon problem where we enforce that (x, u) must be at $(x_\star, u_\star)$ at the end of the time horizon:

$$\begin{array}{ll} \text{maximize} & \mathbb{E}_{W_t}[\sum_{t=0}^{H} R(X_t, u_t)] \\ \text{subject to} & X_{t+1} = f(X_t, u_t, W_t) \\ & (X_H, U_H) = (x_\star, u_\star) \\ & (X_0 = \mathrm{x}). \end{array}$$

We replan at every time step by solving this optimization problem and taking the first action.

The following proposition summarizes how this policy performs

Proposition 1. *Assume that all rewards are bounded above by $R_{\max}$. Then with the above MPC policy, we have for all T that*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T} R(x_t, u_t)\right] \geq \frac{Q_{0\to H}(x_0, u_0) - HR_{\max}}{T} + \mathbb{E}_W[R(f(x_\star, u_\star, W), 0)]\,.$$

The proposition asserts that there is a "burn in" cost associated with the initial horizon length. This term goes to zero with T, but will have different values for different H. The policy converges to a residual average cost due to the stochasticity of the problem and the fact that we try to force the system to the state $(x_\star, u_\star)$.

Proof. To analyze how the policy performs, we turn to Bellman's equation. For any time t, the MPC policy is

$$u_t = \arg\max_u \mathcal{Q}_{0\to H}(x_t, u)$$

Now,

$$\mathcal{Q}_{0\to H}(x_t, u) = R(x_t, u) + \mathbb{E}[\max_{u'} \mathcal{Q}_{1\to H}(X_{t+1}, u')] \,.$$

Now consider what to do at the time $t+1$. A *suboptimal* strategy at this time is to try to play the optimal strategy on the horizon $1 \to H$, and then do nothing on the last step. That is,

$$\max_u \mathcal{Q}_{0\to H}(x_{t+1}, u) \geq \max_{u'} \mathcal{Q}_{1\to H}(x_{t+1}, u') + \mathbb{E}[R(f(x_\star, u_\star, W_{t+H}), 0)] \,.$$

The last expression follows because the action sequence from $1 \to H$ enforces $(x_{t+H}, u_{t+H}) = (x_\star, u_\star)$. The first term on the right hand side was computed in expectation above, hence we have

$$\mathbb{E}[\max_u \mathcal{Q}_{0\to H}(X_{t+1}, u)] \geq \mathbb{E}[\mathcal{Q}_{0\to H}(x_t, u_t)] - \mathbb{E}[R(x_t, u_t, W)] + \mathbb{E}[R(f(x_\star, u_\star, W), 0)] \,.$$

Unwinding this recursion, we find

$$\begin{aligned}\mathbb{E}[\max_u \mathcal{Q}_{0\to H}(X_{T+1}, u)] \geq \mathcal{Q}_{0\to H}(x_0, u_0) - \mathbb{E}\left[\sum_{t=0}^{T} R(x_t, u_t, W_t)\right] \\ + T\mathbb{E}[R(f(x_\star, u_\star, W), 0)] \,.\end{aligned}$$

Since the rewards are bounded above, we can upper bound the left hand side by $R_{\max} H$. Rearranging terms then proves the theorem.

□

The main caveat with this argument is that there may not *exist* a policy that drives x to $x_\star$ from an arbitrary initial condition and any realization of the disturbance signal. Much of the analysis of MPC schemes is devoted to guaranteeing that the problems are *recursively feasible*, meaning that such constraints can be met for all time.

This example also shows how it is often helpful to have some sort of recourse at the end of the planning horizon to mitigate the possibility of being too greedy and driving the system into a bad state. The terminal condition of forcing $x_H = 0$ adds an element of safety to the planning, and ensures stable execution for all time. More general, adding some terminal condition to the planning horizon $\mathcal{C}(x_H)$ is part of good Model Predictive Control design and is often a powerful way to balance performance and robustness.

Partial observation and the separation heuristic

Let's now move to the situation where instead of observing the state directly, we observe an output Y_t:

$$Y_t = h_t(X_t, U_t, W_t)\,.$$

All of the policies we derived from optimization formulations above required feeding back a function of the state. When we can only act on outputs, SDM problems are considerably more difficult.

1. *Static Output Feedback is NP-hard.* Consider the case of just building a static policy from the output Y_t. Let's suppose our model is just the simple linear model:

$$X_{t+1} = AX_t + BU_t$$
$$Y_t = CX_t$$

   Here, A, B and C are matrices. Suppose we want to find a feedback policy $U_t = KY_t$ (where K is a matrix) and all we want to guarantee is that for any initial x_0, the system state converges to zero. This problem is called *static state feedback* and is surprisingly *NP-Hard*. It turns out that the problem is equivalent to finding a matrix K such that $A + BKC$ has all of its eigenvalues inside the unit circle in the complex plane.[2] Though in the MDP case, static state feedback was not only optimal, but computable for tabular MDPs and certain other SDM problems, static output feedback is computationally intractable.

2. POMDPs are PSPACE hard. Papadimitriou and Tsitsiklis showed that optimization of general POMDPs, even on small state spaces, was in all likelihood completely intractable.[3] They reduced the problem of *quantifier elimination* in logical satisfiability problems (QSAT) to POMDPs. QSAT seeks to determine the validity of statements like "there exists x such that for all y there exists z such that for all w this logical formula is true." Optimal action in POMDPs essentially have to keep track of all of the possible true states that might have been visited given the partial observation and make actions accordingly. Hence, the policies have a similar flavor to quantifier elimination as they seek actions that are beneficial to all possible occurrences of the unobserved variables. Since these policies act over long time horizons, the number of counterfactuals that must be maintained grows exponentially large.

Despite these challenges, engineers solve POMDP problems all of the time. Just because the problems are hard in general, doesn't mean they are intractable on average. It only means that we cannot expect to have general

purpose optimal algorithms for these problems. Fortunately, suboptimal solutions are oftentimes quite good for practice, and there are many useful heuristics for decision making with partial information. The most common approach to the output feedback problem is the following two-stage strategy:

1. Filtering. Using all of your past data $\{y_s\}$ for $s = 0, \ldots, t$, build an estimate, $\widehat{x}_t$, of your state.

2. Action based on certainty equivalence. Solve the desired SDM problem as if you had perfect observation of the state X_t, using $\widehat{x}_t$ wherever you would have used an observation $X_t = x_t$. At run time, plug in the estimator $\widehat{x}_t$ as if it were a perfect measurement of the state.

This strategy uses a *separation principle* between prediction and action. For certain problems, this two-staged approach is actually optimal. Notably, if the SDM problem has quadratic rewards/costs, if the dynamics are linear, and if the noise process is Gaussian, then the separation between prediction and action is optimal. More commonly, the separation heuristic is suboptimal, but this abstraction also enables a simple heuristic that is easy to debug and simple to design.

While we have already covered algorithms for optimal control, we have not yet discussed state estimation. Estimating the state of a dynamical system receives the special name *filtering*. However, at the end of the day, filtering is a prediction problem. Define the observed data up to time t as

$$\tau_t := (y_t, \ldots, y_1, u_{t-1}, \ldots, u_1) .$$

The goal of filtering is to estimate a a function $h(\tau_t)$ that predicts X_t. We now describe two approaches to filtering.

Optimal filtering

Given a model of the state transition function and the observation function, we can attempt to compute the maximum *a posteriori* estimate of the state from the data. That is, we could compute $p(x_t|\tau_t)$ and then estimate the mode of this density. Here, we show that such an estimator has a relatively simple recursive formula, though it is not always computationally tractable to compute this formula.

To proceed, we first need a calculation that takes advantage of the conditional independence structure of our dynamical system model. Note

that

$$\begin{aligned} p(y_t, x_t, x_{t-1}, u_{t-1}|\tau_{t-1}) = \\ p(y_t|x_t, x_{t-1}, u_{t-1}, \tau_{t-1}) p(x_t|x_{t-1}, u_{t-1}, \tau_{t-1}) \\ \times p(x_{t-1}|\tau_{t-1}) p(u_{t-1}|\tau_{t-1}) \\ = p(y_t|x_t) p(x_t|x_{t-1}, u_{t-1}) p(x_{t-1}|\tau_{t-1}) p(u_{t-1}|\tau_{t-1}) \,. \end{aligned}$$

This decomposition is into terms we now recognize. $p(x_t|x_{t-1}, u_{t-1})$ and $p(y_t|x_t)$ define the POMDP model and are known. $p(u_t|\tau_t)$ is our policy and it's what we're trying to design. The only unknown here is $p(x_{t-1}|\tau_{t-1})$, but this expression gives us a recursive formula to $p(x_t|\tau_t)$ for all t.

To derive this formula, we apply Bayes rule and then use the above calculation:

$$\begin{aligned} p(x_t|\tau_t) &= \frac{\int_{x_{t-1}} p(x_t, y_t, x_{t-1}, u_{t-1}|\tau_{t-1})}{\int_{x_t, x_{t-1}} p(x_t, y_t, x_{t-1}, u_{t-1}|\tau_{t-1})} \\ &= \frac{\int_{x_{t-1}} p(y_t|x_t) p(x_t|x_{t-1}, u_{t-1}) p(x_{t-1}|\tau_{t-1}) p(u_{t-1}|\tau_{t-1})}{\int_{x_t, x_{t-1}} p(y_t|x_t) p(x_t|x_{t-1}, u_{t-1}) p(x_{t-1}|\tau_{t-1}) p(u_{t-1}|\tau_{t-1})} \\ &= \frac{\int_{x_{t-1}} p(y_t|x_t) p(x_t|x_{t-1}, u_{t-1}) p(x_{t-1}|\tau_{t-1})}{\int_{x_t, x_{t-1}} p(y_t|x_t) p(x_t|x_{t-1}, u_{t-1}) p(x_{t-1}|\tau_{t-1})} \,. \end{aligned} \tag{1}$$

Given a prior for x_0, this now gives us a formula to compute a MAP estimate of x_t for all t, incorporating data in a streaming fashion. For tabular POMDP models with small state spaces, this formula can be computed simply by summing up the conditional probabilities. In POMDPs without inputs—also known as *hidden Markov models*—this formula gives the forward pass of Viterbi's decoding algorithm. For models where the dynamics are linear and the noise is Gaussian, these formulas reduce into an elegant closed form solution known as *Kalman filtering*. In general, this optimal filtering algorithm is called *belief propagation* and is the basis of a variety of algorithmic techniques in the field of graphical models.

Kalman filtering

For the case of linear dynamical systems, the above calculation has a simple closed form solution that looks similar to the solution of LQR. This estimator is called a *Kalman filter*, and is one of the most important tools in signal processing and estimation. The Kalman filter assumes a linear dynamical system driven by Gaussian noise with observations corrupted by Gaussian noise

$$\begin{aligned} X_{t+1} &= AX_t + BU_t + W_t \\ Y_t &= CX_t + V_t \,. \end{aligned}$$

Here, assume W_t and V_t are independent for all time and Gaussian with means zero and covariances Σ_W and Σ_V respectively. Because of the joint Gaussianity, we can compute a closed form formula for the density of X_t conditioned on the past observations and actions, $p(x_t|\tau_t)$. Indeed, X_t is a Gaussian itself.

On an infinite time horizon, the Kalman filter takes a simple and elucidating form:

$$\begin{aligned}\widehat{x}_{t+1} &= A\widehat{x}_t + Bu_t - L(y_t - \widehat{y}_t)\\ \widehat{y}_t &= C\widehat{x}_t\end{aligned}$$

where

$$L = APC^T(CPC^T + \Sigma_V)^{-1}$$

and P is the positive semidefinite solution to the discrete algebraic Riccati equation

$$P = APA^T + \Sigma_W - (APC^T)(CPC^T + \Sigma_V)^{-1}(C\Sigma A^T)\,.$$

The derivation of this form follows from the calculation in Equation 1. But the explanation of the formulas tend to be more insightful than the derivation. Imagine the case where $L = 0$. Then our estimate $\widehat{x}_t$ simulates the same dynamics as our model with no noise corruption. The matrix L computes a correction for this simulation based on the observed y_t. This feedback correction is chosen in such a way such that P is the steady state covariance of the error $X_t - \widehat{x}_t$. P ends up being the minimum variance possible with an estimator that is *unbiased* in the sense that $\mathbb{E}[\widehat{x}_t - X_t] = 0$.

Another interesting property of this calculation is that the L matrix is the LQR gain associated with the LQR problem

$$\begin{array}{ll}\text{minimize} & \lim_{T\to\infty} \mathbb{E}_{W_t}\left[\frac{1}{2}\sum_{t=0}^{T} X_t^T\Sigma_W X_t + U_t^T\Sigma_V U_t\right],\\ \text{subject to} & X_{t+1} = A^T X_t + B^T U_t + W_t,\ U_t = \pi_t(X_t)\\ & (x_0 \text{ given}).\end{array}$$

Control theorists often refer to this pairing as the *duality between estimation and control*.

Feedforward prediction

While the optimal filter can be computed in simple cases, we often do not have simple computational means to compute the optimal state estimate. That said, the problem of state estimation is necessarily one of prediction, and the first half of this course gave us a general strategy for building

such estimators from data. Given many simulations or experimental measurements of our system, we can try to estimate a function h such that $X_t \approx h(\tau_t)$. To make this concrete, we can look at *time lags* of the history

$$\tau_{t-s\to t} := (y_t, \ldots, y_{t-s}, u_{t-1}, \ldots, u_{t-s}) .$$

Such time lags are necessarily all of the same length. Then estimating

$$\text{minimize}_h \quad \sum_t \text{loss}(h(\tau_{t-s\to t}), x_t)$$

is a supervised learning problem, and standard tools can be applied to design architectures for and estimate h.

Chapter notes

This chapter and the following chapter overlap significantly with a survey of reinforcement learning by Recht,[4] which contains additional connections to continuous control. Those interested in learning more about continuous control from an optimization viewpoint should consult the book by Borrelli et al.[5] This book also provides an excellent introduction to model predictive control. Another excellent introduction to continuous optimal control and filtering is Boyd's lecture notes.[6]

An invaluable introduction to the subject of dynamic programming is by Bertsekas, who has done pioneering research in this space and has written some of the most widely read texts.[1] For readers interested in a mathematical introduction to dynamic programming on discrete processes, we recommend Puterman's text.[7] Puterman also explains the linear programming formulation of dynamic programming.

Bibliography

[1] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 4th edition, 2017.

[2] Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

[3] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[4] Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2, 2019.

[5] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[6] Stephen Boyd. EE363: Linear dynamical systems. Notes available at `https://stanford.edu/class/ee363/`, 2009.

[7] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.