12

Reinforcement learning

Dynamic programming and its approximations studied thus far all require knowledge of the probabilistic mechanisms underlying how data and rewards change over time. When these mechanisms are unknown, appropriate techniques to probe and learn about the underlying dynamics must be employed in order to find optimal actions. We shall refer to the solutions to sequential decision making problems when the dynamics are unknown as *reinforcement learning*. Depending on context, the term may refer to a body of work in artificial intelligence, the community of researchers and practitioners who apply a certain set of tools to sequential decision making, and data-driven dynamic programming. That said, it is a useful name to collect a set of problems of broad interest to machine learning, control theory, and robotics communities.

A particularly simple and effective strategy for reinforcement learning problems is to estimate a predictive model for the dynamical system and then to use the fit model as if it were the true model in the optimal control problem. This is an application of the *principle of certainty equivalence*, an idea tracing back to the dynamic programming literature of the 1950s.[1,2] Certainty equivalence is a general solution strategy for the following problem. Suppose you want to solve some optimization problem with a parameter ϑ that is unknown. However, suppose we can gather data to estimate ϑ. Then the certainty equivalent solution is to use a point estimate for ϑ as if it were the true value. That is, you act as if you were certain of the value of ϑ, even though you have only estimated ϑ from data. We will see throughout this chapter that such certainty equivalent solutions are powerfully simple and effective baseline for sequential decision making in the absence of well specified models.

Certainty equivalence is a very general principle. We can apply it to the output of a filtering scheme that predicts state, as we described in our discussion of partially observed Markov Decision Processes. We can also apply this principle in the study of MDPs with unknown parameters. For every problem in this chapter, our core baseline will always be the certainty equivalent solution. Surprisingly, we will see that certainty equivalent baselines are typically quite competitive and give a clear illustration of the best quality one can expect in many reinforcement learning problems.

Exploration-exploitation tradeoffs: Regret and PAC learning

In order to compare different approaches to reinforcement learning, we need to decide on some appropriate rules of comparison. Though there are a variety of important metrics for engineering practice that must be considered including ease of implementation and robustness, a first-order statistical comparison might ask how many samples are needed to achieve a policy with high reward.

For this question, there are two predominant conventions to compare methods: *PAC-error* and *regret*. PAC is a shorthand for *probably approximately correct*. It is a useful notion when we spend all of our time learning about a system, and then want to know how suboptimal our solution will be when built from the data gathered thus far. Regret is more geared towards online execution where we evaluate the reward accrued at all time steps, even if we are spending that time probing the system to learn about its dynamics. Our focus in this chapter will be showing that these two concepts are closely related.

Let us formalize the two notions. As in the previous chapter, we will be concerned with sequential decision making problems of the form

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

Let $\pi_\star$ denote the optimal policy of this problem.

For PAC, let's suppose we allocate N samples to probe the system and use them in some way to build a policy π_N. We can define the optimization error of this policy to be

$$\mathcal{E}(\pi_N) = \mathbb{E}\left[\sum_{t=1}^{T} R_t[X'_t, \pi_\star(X'_t), W_t]\right] - \mathbb{E}\left[\sum_{t=1}^{T} R_t[X_t, \pi_N(X_t), W_t]\right].$$

Our model has (δ, ϵ)-PAC error if $\mathcal{E}(\pi_N) \leq \epsilon$ with probability at least $1 - \delta$. The probability here is measured with respect to the sampling process and dynamics of the system.

Regret is defined similarly, but is subtly different. Suppose we are now only allowed T total actions and we want to understand the cumulative award achieved after applying these T actions. In this case, we have to balance the number of inputs we use to find a good policy (exploration) against the number of inputs used to achieve the best reward (exploitation).

Formally, suppose we use a policy π_t at each time step to choose our action. Suppose $\pi_\star$ is some other fixed policy. Let X_t denote the states induced by the policy sequence π_t and X'_t denote the states induced by $\pi_\star$. Then the *regret* of $\{\pi_t\}$ is defined to be

$$\mathcal{R}_T(\{\pi_t\}) = \mathbb{E}\left[\sum_{t=1}^{T} R_t[X'_t, \pi_\star(X'_t), W_t]\right] - \mathbb{E}\left[\sum_{t=1}^{T} R_t[X_t, \pi_t(X_t), W_t]\right].$$

It is simply the expected difference in the rewards generated under policy $\pi_\star$ as compared to those generated under policy sequence π_t. One major way that regret differs from PAC-error is the policy can change with each time step.

One note of caution for both of these metrics is that they are comparing to a policy $\pi_\star$. It's possible that the comparison policy $\pi_\star$ is not very good. So we can have small regret and still not have a particularly useful solution to the SDM problem. As a designer it's imperative to understand $\pi_\star$ to formalize the best possible outcome with perfect information. That said, regret and PAC-error are valuable ways to quantify how much exploration is necessary to find nearly optimal policies. Moreover, both notions have provided successful frameworks for algorithm development: many algorithms with low regret or PAC-error are indeed powerful in practice.

Multi-armed bandits

The multi-armed bandit is one of the simplest reinforcement learning problems, and studying this particular problem provides many insights into exploration-exploitation tradeoffs.

In the multi-armed bandit, we assume *no state whatsoever*. There are K total actions, and the reward is a random function of which action you choose. We can model this by saying there are i.i.d. random variables $W_{t1}, \ldots, W_{tk}$, and your reward is the dot product

$$R_t = [W_{t1}, \ldots, W_{tK}]e_{u_t}$$

where e_i is a standard basis vector. Here W_{ti} take values in the range $[0, 1]$. We assume that all of the W_{ti} are independent, and that W_{ti} and W_{si} are identically distributed. Let $\mu_i = \mathbb{E}[W_{ti}]$. Then the expected reward at time t is precisely μ_{u_t}.

The multi-armed bandit problem is inspired by gambling on slot machines. Indeed, a "bandit" is a colloquial name for a slot machine. Assume that you have K slot machines. Each machine has some probability of paying out when you play it. You want to find the machine that has the largest probability of paying out, and then play that machine for the rest of

time. The reader should take an opportunity to ponder the irony that much of our understanding of statistical decision making comes from gambling.

First let's understand what the optimal policy is if we know the model. The total reward is equal to

$$\mathop{\mathbb{E}}_{W_t}\left[\sum_{t=0}^{T} R_t(u_t, W_t)\right] = \sum_{t=1}^{T} \mu_{u_t}\,.$$

The optimal policy is hence to choose a constant action $u_t = k$ where $k = \arg\max_i \mu_i$.

When we don't know the model, it makes sense that our goal is to quickly find the action corresponding to the largest mean. Let's first do a simple PAC analysis, and then turn to the slightly more complicated regret analysis. Our simple baseline is one of certainty equivalence. We will try each action N/K times, and compute the empirical return. The empirical means are:

$$\widehat{\mu}_k = \frac{K}{N}\sum_{i=1}^{N/K} R_i^{(k)}$$

Our policy will be to take the action with the highest observed empirical return.

To estimate the value of this policy, let's assume that the best action is $u = 1$. Then define

$$\Delta_i = \mu_1 - \mu_i\,.$$

Then we have

$$\mathcal{E}(\pi_N) = \sum_{i=1}^{K} T\Delta_i\,\mathbb{P}[\forall i\colon \widehat{\mu}_k \geq \widehat{\mu}_i]\,.$$

We can bound the probability that action k is selected as follows. First, if action k has the largest empirical mean, it must have a larger empirical mean than the true best option, action 1:

$$\mathbb{P}[\forall i\colon \widehat{\mu}_k \geq \widehat{\mu}_i] \leq \mathbb{P}[\widehat{\mu}_k \geq \widehat{\mu}_1]\,.$$

We can bound this last term using Hoeffding's inequality. Let $m = N/K$. Since each reward corresponds to an independent draw of some random process, we have $\widehat{\mu}_k - \widehat{\mu}_1$ is the mean of $2m$ independent random variables in the range $[-1, 1]$:

$$\frac{1}{2}(\widehat{\mu}_k - \widehat{\mu}_1) = \frac{1}{2m}\left(\sum_{i=1}^{m} R_i^{(k)} + \sum_{i=1}^{m} -R_i^{(1)}\right)\,.$$

Now writing Hoeffding's inequality for this random variable gives the tail bound

$$\mathbb{P}[\widehat{\mu}_k \geq \widehat{\mu}_1] = \mathbb{P}[\tfrac{1}{2}(\widehat{\mu}_k - \widehat{\mu}_1) \geq 0] \leq \exp\left(-\frac{m\Delta_k^2}{4}\right)$$

which results in an optimization error

$$\mathcal{E}(\pi_N) \leq \sum_{i=1}^{K} T\Delta_i \exp\left(-\frac{N\Delta_i^2}{4K}\right).$$

with probability 1. This expression reveals that the multi-armed bandit problem is fairly simple. If Δ_i are all small, then any action will yield about the same reward. But if all of the Δ_i are large, then finding the optimal action only takes a few samples. Naively, without knowing anything about the gaps at all, we can use the fact that $xe^{-x^2/2} \leq \frac{1}{2}$ for nonnegative x to find

$$\mathcal{E}(\pi_N) \leq \frac{K^{3/2}T}{\sqrt{N}}.$$

This shows that no matter what the gaps are, as long as N is larger than K^3, we would expect to have a high quality solution.

Let's now turn to analyzing regret of a simple certainty equivalence baseline. Given a time horizon T, we can spend the first m time steps searching for the best return. Then we can choose this action for the remaining $T - m$ time steps. This strategy is called *explore-then-commit*.

The analysis of the explore-then-commit strategy for the multi-armed bandit is a straightforward extension of the PAC analysis. If at round t, we apply action k, the expected gap between our policy and the optimal policy is Δ_k. So if we let T_k denote the number of times action k is chosen by our policy then we must have

$$\mathcal{R}_T = \sum_{k=1}^{K} \mathbb{E}[T_k]\Delta_k.$$

T_k are necessarily random variables: what the policy learns about the different means will depend on the observed sequence x_k which are all random variables.

Suppose that for exploration, we mimic our offline procedure, trying each action m times and record the observed rewards for that action $r_i^{(k)}$ for $i = 1, \ldots, m$. At the end of these mk actions, we compute the empirical mean associated with each action as before. Then we must have that

$$\mathbb{E}[T_k] = m + (T - mK)\,\mathbb{P}[\forall i\colon \widehat{\mu}_k \geq \widehat{\mu}_i].$$

The first term just states that each action is performed m times. The second term states that action k is chosen for the commit phase only if its empirical mean is larger than all of the other empirical means.

Using Hoeffding's inequality again to bound these probabilities, we can put everything together bound the expected regret as

$$\mathcal{R}_T \leq \sum_{k=1}^{K} m\Delta_k + (T - mK)\Delta_k \exp\left(-\frac{m\Delta_k^2}{4}\right) .$$

Let's specialize to the case of *two* actions to see what we can take away from this decomposition:

1. **Gap dependent regret.** First, assume we know the gap between the means, Δ_2, but we don't know which action leads to the higher mean. Suppose that

$$m_0 = \left\lceil \frac{4}{\Delta_2^2} \log\left(\frac{T\Delta_2^2}{4}\right) \right\rceil \geq 1 .$$

Then using $m = m_0$, we have

$$\begin{aligned} \mathcal{R}_T &\leq m\Delta_2 + T\Delta_2 \exp\left(-\frac{m\Delta_2^2}{4}\right) \\ &\leq \Delta_2 + \frac{4}{\Delta_2}\left(\log\left(\frac{T\Delta_2^2}{4}\right) + 1\right) . \end{aligned}$$

If $m_0 < 1$, then $\Delta_2 < \frac{2}{\sqrt{T}}$, then choosing a random arm yields total expected regret at most

$$\mathcal{R}_T = \frac{T}{2}\Delta_2 \leq \sqrt{T} .$$

If Δ_2 is very small, then we might also just favor the bound

$$\mathcal{R}_T \leq \tfrac{1}{2}\Delta_2 T .$$

Each of these bounds applies in different regimes and tells us different properties of this algorithm. The first bound shows that with appropriate choice of m, explore-then-commit incurs regret asymptotically bounded by $\log(T)$. This is effectively the smallest asymptotic growth achievable and is the gold standard for regret algorithms. However, this logarithmic regret bound depends on the gap Δ_2. For small Δ_2, the second bound shows the regret is never worse than $\sqrt{T}$ for any value of the gap. $\sqrt{T}$ is one of the more common values for regret,

and though it is technically asymptotically worse than logarithmic, algorithms with $\sqrt{T}$ regret tend to be more stable and robust than their logarithmic counterparts. Finally, we note that a very naive algorithm will incur regret that grows linearly with the horizon T. Though linear regret is not typically an ideal situation, there are many applications where it's acceptable. If Δ_2 is tiny to the point where it is hard to observe the difference between μ_1 and μ_2, then linear regret might be completely satisfactory for an application.

2. **Gap independent regret.** We can get a gap independent, $\sqrt{T}$ regret for explore then commit for any value of Δ_2. This just requires a bit of calculus:

$$\begin{aligned}\frac{4}{\Delta_2}\left(\log\left(\frac{T\Delta^2}{4}\right)+1\right) &= 2\sqrt{T}\left(\frac{2}{\Delta_2\sqrt{T}}\left(\log\left(\frac{T\Delta^2}{4}\right)+1\right)\right)\\ &= 2\sqrt{T}\sup_{x\geq 0}\frac{2\log(x)+1}{x} \leq 4e^{-1/2}\sqrt{T} \leq 2.5\sqrt{T}\,.\end{aligned}$$

   Hence,

$$\mathcal{R}_T \leq \Delta_2 + 2.5\sqrt{T}$$

   no matter the size of Δ_2 the gap is. Often times this unconditional bound is smaller than the logarithmic bound we derived above.

3. **Gap independent policy.** The stopping rule we described thus far requires knowing the value of Δ_2. However, if we set $m = T^{2/3}$ then we can achieve sublinear regret no matter what the value of Δ_2 is. To see this again just requires some calculus:

$$\begin{aligned}\mathcal{R}_T &\leq T^{2/3}\Delta_2 + T\Delta_2\exp\left(-\frac{T^{2/3}\Delta_2^2}{4}\right)\\ &= T^{2/3}\left(\Delta_2 + T^{1/3}\Delta_2\exp\left(-\frac{T^{2/3}\Delta_2^2}{4}\right)\right)\\ &\leq T^{2/3}\left(\Delta_2 + 2\sup_{x\geq 0} xe^{-x^2}\right) \leq 2T^{2/3}\,.\end{aligned}$$

   $O(T^{2/3})$ regret is technically "worse" than an asymptotic regret of $O(T^{1/2})$, but often times such algorithms perform well in practice. This is because there is a difference between worst case and average case behavior, and hence these worst-case bounds on regret themselves do not tell the whole story. A practitioner has to weigh the circumstances of their application to decide what sorts of worst-case scenarios are acceptable.

Interleaving exploration and exploitation

Explore-then-commit is remarkably simple, and illustrates most of the phenomena associated with regret minimization. There are essentially two main shortcomings in the case of the multi-armed bandit:

1. For a variety of practical concerns, it would be preferable to interleave exploration with exploitation.
2. If you don't know the gap, you only get a $T^{2/3}$ rate.

A way to fix this is called *successive elimination*. As in explore-then-commit, we try all actions m times. Then, we drop the actions that are clearly performing poorly. We then try the remaining actions $4m$ times, and drop the poorly performing actions. We run repeated cycles of this pruning procedure, yielding a collection of better actions on average, aiming at convergence to the best return.

Successive Elimination Algorithm:

- Given number of rounds B and an increasing sequence of positive integers $\{m_\ell\}$.
- Initialize the active set of options $\mathcal{A} = \{1, \dots, K\}$.
- For $\ell = 1, \dots, B$:
  1. Try every action in $\mathcal{A}$ for m_ℓ times.
  2. Compute the empirical means $\widehat{\mu}_k$ from this iteration only.
  3. Remove from $\mathcal{A}$ any action j with $\mu_j + 2^{-\ell} < \max_{k \in \mathcal{A}} \mu_k$.

The following theorem bounds the regret of successive elimination, and was proven by Auer and Ortner.[3]

Theorem 1. *With* $B = \lfloor \frac{1}{2} \log_2 \frac{T}{e} \rfloor$ *and* $m_\ell = \lceil 2^{2\ell+1} \log \frac{T}{4^\ell} \rceil$, *the successive elimination algorithm accrues expected regret*

$$\mathcal{R}_T \leq \sum_{i \colon \Delta_i > \lambda} \left(\Delta_i + \frac{32 \log(T\Delta_i^2) + 96}{\Delta_i} \right) + \max_{i \colon \Delta_i \leq \lambda} \Delta_i T$$

for any $\lambda > \sqrt{e/T}$.

Another popular strategy is known as *optimism in the face of uncertainty*. This strategy is also often called "bet on the best." At iteration t, take all of the observations seen so far and form a set up upper confidence bounds B_i such that

$$\mathbb{P}[\forall i \colon \mu_i \leq B_i(t)] \leq 1 - \delta$$

This leads to the Upper Confidence Bound (UCB) algorithm.

UCB Algorithm

- For $t = 1, \ldots, T$:
  1. Choose action $k = \arg\max_i B_i(t-1)$.
  2. Play action k and observe reward r_t.
  3. Update the confidence bounds.

For the simple case of the multi-armed bandit, we can use the bound that would directly come from Hoeffding's inequality:

$$B_i(t) = \widehat{\mu}_i(t) + \sqrt{\frac{2\log(1/\delta)}{T_i(t)}}$$

where we remind the reader that $T_i(t)$ denotes the number of times we have tried action i up to round t. Though $T_i(t)$ is a random variable, one can still prove that this choice yields an algorithm with nearly optimal regret.

More generally, optimistic algorithms work by maintaining an uncertainty set about the dynamics model underlying the SDM problem. The idea is to maintain a set S where we have confidence our true model lies. The algorithm then proceeds by choosing the model in S which gives the highest expected reward. The idea here is that either we get the right model in which case we get a large reward, or we learn quickly that we have a suboptimal model and we remove it from our set S.

Contextual bandits

Contextual bandits provide a transition from multi-armed bandits to full-fledged reinforcement learning, introducing *state* or *context* into the decision problem. Our goal in contextual bandits is to iteratively update a policy to maximize the total reward:

$$\text{maximize}_{u_t} \; \mathop{\mathbb{E}}_{W_t}\left[\sum_{t=1}^{T} R(X_t, u_t, W_t)\right]$$

Here, we choose actions u_t according to some policy that is a function of the observations of the random variables X_t, which are called *contexts* or *states*. We make no assumptions about how contexts evolve over time. We assume that the reward function is unknown and, at every time step, the received reward is given by

$$R(X_t, u_t, W_t) = R(X_t, u_t) + W_t$$

where W_t is a random variable with zero mean and independent from all other variables in the problem.

Contextual bandits are a convenient way to abstractly model engagement problems on the internet. In this example, contexts correspond to information about a person. Every interaction a person has with the website can be scored in term of some sort of reward function that encodes outcomes such as whether the person clicked on an ad, liked an article, or purchased an item. Whatever the reward function is, the goal will be to maximize the total reward accumulated over all time. The X_t will be features describing the person's interaction history, and the action will be related to the content served.

As was the case with the multi-armed bandit, the key idea in solving contextual bandits is to reduce the problem to a prediction problem. In fact we can upper bound our regret by our errors in prediction. The regret accrued by a policy π is

$$\mathbb{E}\left\{\sum_{t=1}^{T} \max_u R(X_t, u) - R(X_t, \pi(X_t))\right\}.$$

This is because if we know the reward function, then the optimal strategy is to choose the action that maximizes R. This is equivalent to the dynamic programming solution when the dynamics are trivial.

Let's reduce this problem to one of prediction. Suppose that at time t we have built an approximation of the reward function R that we denote $\widehat{R}_t(x, u)$. Let's suppose that our algorithm uses the policy

$$\pi(X_t) = \arg\max_u \widehat{R}_t(X_t, u).$$

That is, we take our current estimate as if it was the true reward function, and pick the action that maximizes reward given the context X_t.

To bound the regret for such an algorithm, note that we have for any action u

$$\begin{aligned} 0 &\leq \widehat{R}_t(X_t, \pi(X_t)) - \widehat{R}_t(X_t, u) \\ &\leq R(X_t, \pi(X_t)) - R(X_t, u) \\ &\quad + \left[\widehat{R}_t(X_t, \pi(X_t)) - R(X_t, \pi(X_t))\right] + \left[\widehat{R}_t(X_t, u) - R(X_t, u)\right]. \end{aligned}$$

Hence,

$$\sum_{t=1}^{T} \max_u R(X_t, u) - R(X_t, \pi(X_t)) \leq 2 \sum_{t=1}^{T} \max_u |\widehat{R}_t(X_t, u) - R(X_t, u)|.$$

This final inequality shows that if the prediction error goes to zero, the associated algorithm accrues sublinear regret.

While there are a variety of algorithms for contextual bandits, we focus our attention on two simple solutions that leverage the above reduction to prediction. These algorithms work well in practice and are by far the most commonly implemented. Indeed, they are so common that most applications don't even call these implementations of contextual bandit problems, as they take the bandit nature completely for granted.

Our regret bound naturally suggests the following explore-then-commit procedure.

Explore-then-commit for contextual bandits

- For $t = 1, 2, \ldots, m$:
  1. Receive new context x_t.
  2. Choose a random action u_t.
  3. Receive reward r_t.
- Find a function $\widehat{R}_m$ to minimize prediction error: $\widehat{R}_m := \arg\min_f \sum_{s=1}^{m} loss(f(x_s, u_s), r_s)$.
- Define the policy $\pi(x) = \arg\max_u \widehat{R}_m(x_t, u)$.
- For $t = m+1, m+2, \ldots$:
  1. Receive new context x_t.
  2. Choose the action given by $\pi(x_t)$.
  3. Receive reward r_t.

Second, an even more popular method is the following greedy algorithm. The greedy algorithm avoids the initial random exploration stage and instead picks whatever is optimal for the data seen so far.

Greedy algorithm for contextual bandits

- For $t = 1, 2, \ldots$
  1. Find a function $\widehat{R}_t$ to minimize prediction error:
  $$\widehat{R}_t := \arg\min_f \sum_{s=1}^{t-1} loss(f(x_s, u_s), r_s).$$
  2. Receive new context x_t.
  3. Choose the action given by the policy
  $$\pi_t(x_t) := \arg\max_u \widehat{R}_t(x_t, u).$$
  4. Receive reward r_t.

In worst-case settings, the greedy algorithm may accrue linear regret. However, worst-case contexts appear to be rare. In the linear contextual bandits problem, where rewards are an unknown linear function of the context, even slight random permutations of a worst-case instance lead to sublinear regret.[4]

The success of the greedy algorithm shows that it is not always desirable to be exploring random actions to see what happens. This is especially true for industrial applications where random exploration is often costly and the value of adding exploration seems limited.[5,6] This context is useful to keep in mind as we move to the more complex problem of reinforcement learning and approximate dynamic programming.

When the model is unknown: Approximate dynamic programming

We now bring dynamics back into the picture and attempt to formalize how to solve general SDM problems when we don't know the dynamics model or even the reward function. We turn to exploring the three main approaches in this space: certainty equivalence fits a model from some collected data and then uses this model as if it were true in the SDM problem. Approximate Dynamic Programming uses Bellman's principle of optimality and stochastic approximation to learn Q-functions from data. Direct Policy Search directly searches for policies by using data from previous episodes in order to improve the reward. Each of these has their advantages and disadvantages as we now explore in depth.

Certainty equivalence for sequential decision making

One of the simplest, and perhaps most obvious strategies to solve an SDM problem when the dynamics are unknown is to estimate the dynamics from some data and then to use this estimated model as if it were the true model in the SDM problem.

Estimating a model from data is commonly called "system identification" in the dynamical systems and control literature. System identification differs from conventional estimation because one needs to carefully choose the right inputs to excite various degrees of freedom and because dynamical outputs are correlated over time with the parameters we hope to estimate, the inputs we feed to the system, and the stochastic disturbances. Once data is collected, however, conventional prediction tools can be used to find the system that best agrees with the data and can be applied to analyze the number of samples required to yield accurate models.

Let's suppose we want to build a predictor of the state x_{t+1} from the trajectory history of past observed states and actions. A simple, classic strategy is simply to inject a random probing sequence u_t for control and then measure how the state responds. Up to stochastic noise, we should have that

$$x_{t+1} \approx \varphi(x_t, u_t)\,,$$

where φ is some model aiming to approximate the true dynamics. φ might arise from a first-principles physical model or might be a non-parametric approximation by a neural network. The state-transition function can then be fit using supervised learning. For instance, a model can be fit by solving the least-squares problem

$$\text{minimize}_{\varphi} \quad \sum_{t=0}^{N-1} ||x_{t+1} - \varphi(x_t, u_t)||^2\,.$$

Let $\widehat{\varphi}$ denote the function fit to the collected data to model the dynamics. Let ω_t denote a random variable that we will use as a model for the noise process. With such a point estimate for the model, we might solve the optimal control problem

$$\begin{array}{ll} \text{maximize} & \mathbb{E}_{\omega_t}[\sum_{t=0}^{N} R(x_t, u_t)] \\ \text{subject to} & x_{t+1} = \widehat{\varphi}(x_t, u_t) + \omega_t,\ u_t = \pi_t(\tau_t)\,. \end{array}$$

In this case, we are solving the wrong problem to get our control policies π_t. Not only is the model incorrect, but this formulation requires some plausible model of the noise process. But we emphasize that this is standard engineering practice. Though more sophisticated techniques can be used to account for the errors in modeling, feedback often can compensate for these modeling errors.

Approximate dynamic programming

Approximate dynamic programming approaches the RL problem by directly approximating the optimal control cost and then solving this with techniques from dynamic programming. Approximate Dynamic Programming methods typically try to infer Q-functions directly from data. The standard assumption in most practical implementations of Q-learning is that the Q-functions are static, as would be the case in the infinite horizon, discounted optimal control problem.

Probably the best known approximate dynamic programming method is *Q-learning*.[7] Q-learning simply attempts to solve value iteration using *stochastic approximation*. If we draw a sample trajectory using the policy given by the optimal policy, then we should have (approximately and in

expectation)

$$\mathcal{Q}_\gamma(x_t, u_t) \approx R(x_t, u_t) + \gamma \max_{u'} \mathcal{Q}_\gamma(x_{t+1}, u') \,.$$

Thus, beginning with some initial guess $\mathcal{Q}_\gamma^{(\text{old})}$ for the Q-function, we can update

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

where η is a *step-size* or *learning rate*.

The update here only requires data generated by the policy Q_γ^{old} and does not need to know the explicit form of the dynamics. Moreover, we don't even need to know the reward function if this is provided online when we generate trajectories. Hence, Q-learning is often called "model free." We strongly dislike this terminology and do not wish to dwell on it. Unfortunately, distinguishing between what is "model-free" and what is "model-based" tends to just lead to confusion. All reinforcement learning is inherently based on models, as it implicitly assumes data is generated by some Markov Decision Process. In order to run Q-learning we need to know the form of the Q-function itself, and except for the tabular case, how to represent this function requires *some* knowledge of the underlying dynamics. Moreover, assuming that value iteration is the proper solution of the problem is a modeling assumption: we are assuming a discount factor and time invariant dynamics. But the reader should be advised that when they read "model-free," this almost always means "no model of the state transition function was used when running this algorithm."

For continuous control problems methods like Q-learning appear to make an inefficient use of samples. Suppose the internal state of the system is of dimension d. When modeling the state-transition function, each sample provides d pieces of information about the dynamics. By contrast, Q-learning only uses 1 piece of information per time step. Such inefficiency is often seen in practice. Also troubling is the fact that we had to introduce the discount factor in order to get a simple form of the Bellman equation. One can avoid discount factors, but this requires either considerably more sophisticated analysis. Large discount factors do in practice lead to brittle methods, and the discount factor becomes a hyperparameter that must be tuned to stabilize performance.

We close this section by noting that for many problems with high dimensional states or other structure, we might be interested in not representing Q-functions as a look up table. Instead, we might approximate the Q-functions with a parametric family: $\mathcal{Q}(x, u; \vartheta)$. Though we'd like to update

the parameter ϑ using something like gradient descent, it's not immediately obvious how to do so. The simplest attempt, following the guide of stochastic approximation is to run the iterations:

$$\begin{aligned}\delta_t &= R(x_t, u_t) + \gamma \mathcal{Q}(x_{t+1}, u_{t+1}; \vartheta_t) - \mathcal{Q}(x_t, u_t; \vartheta_t) \\ \vartheta_{t+1} &= \vartheta_t + \eta \delta_t \nabla \mathcal{Q}(x_t, u_t, \vartheta_t)\end{aligned}$$

This algorithm is called *Q-learning with function approximation*. A typically more stable version uses momentum to average out noise in Q-learning. With δ_t as above, we add the modification

$$\begin{aligned}e_t &= \lambda e_{t-1} + \nabla \mathcal{Q}(x_t, u_t, \vartheta_t) \\ \vartheta_{t+1} &= \vartheta_t + \eta \delta_t e_t\end{aligned}$$

for $\lambda \in [0, 1]$. This method is known as SARSA(λ).[8]

Direct policy search

The most ambitious form of control without models attempts to directly learn a policy function from episodic experiences without ever building a model or appealing to the Bellman equation. From the oracle perspective, these policy driven methods turn the problem of RL into derivative-free optimization.

In turn, let's first begin with a review of a general paradigm for leveraging random sampling to solve optimization problems. Consider the general unconstrained optimization problem

$$\text{maximize}_{z \in \mathbb{R}^d} \quad R(z) \,.$$

Any optimization problem like this is equivalent to an optimization over probability densities on z:

$$\text{maximize}_{p(z)} \quad \mathbb{E}_p[R(z)] \,.$$

If $z_\star$ is the optimal solution, then we'll get the same value if we put a δ-function around $z_\star$. Moreover, if p is a density, it is clear that the *expected value of the reward function* can never be larger than the maximal reward achievable by a fixed z. So we can either optimize over z or we can optimize over *densities* over z.

Since optimizing over the space of all probability densities is intractable, we must restrict the class of densities over which we optimize. For example, we can consider a family parameterized by a parameter vector ϑ: $p(z; \vartheta)$ and attempt to optimize

$$\text{maximize}_{\vartheta} \quad \mathbb{E}_{p(z;\vartheta)}[R(z)] \,.$$

If this family of densities contains all of the Delta functions, then the optimal value will coincide with the non-random optimization problem. But if the family does not contain the Delta functions, the resulting optimization problem only provides a lower bound on the optimal value no matter how good of a probability distribution we find.

That said, this reparameterization provides a powerful and general algorithmic framework for optimization. In particular, we can compute the derivative of $J(\vartheta) := \mathbb{E}_{p(z;\vartheta)}[R(z)]$ using the following calculation (called "the log-likelihood trick"):

$$\begin{aligned}
\nabla_\vartheta J(\vartheta) &= \int R(z) \nabla_\vartheta p(z;\vartheta) dz \\
&= \int R(z) \left(\frac{\nabla_\vartheta p(z;\vartheta)}{p(z;\vartheta)} \right) p(z;\vartheta) dz \\
&= \int \left(R(z) \nabla_\vartheta \log p(z;\vartheta) \right) p(z;\vartheta) dz \\
&= \mathbb{E}_{p(z;\vartheta)} \left[R(z) \nabla_\vartheta \log p(z;\vartheta) \right] .
\end{aligned}$$

This derivation reveals that the gradient of J with respect to ϑ is the expected value of the function

$$G(z,\vartheta) = R(z) \nabla_\vartheta \log p(z;\vartheta)$$

Hence, if we sample z from the distribution defined by $p(z;\vartheta)$, we can compute $G(z,\vartheta)$ and will have an unbiased estimate of the gradient of J. We can follow this direction and will be running stochastic gradient descent on J, defining the following algorithm:

REINFORCE algorithm:

- *Input Hyperparameters:* step-sizes $\alpha_j > 0$.
- *Initialize:* ϑ_0 and $k = 0$.
- Until the heat death of the universe, do:
  1. Sample $z_k \sim p(z;\vartheta_k)$.
  2. Set $\vartheta_{k+1} = \vartheta_k + \alpha_k R(z_k) \nabla_\vartheta \log p(z_k;\vartheta_k)$.
  3. $k \leftarrow k+1$.

The main appeal of the REINFORCE Algorithm is that it is not hard to implement. If you can efficiently sample from $p(z;\vartheta)$ and can easily compute $\nabla \log p$, you can run this algorithm on essentially any problem. But such generality must and does come with a significant cost. The algorithm operates on stochastic gradients of the sampling distribution, but the function we cared about optimizing—R—is only accessed through function evaluations. Direct search methods that use the log-likelihood

trick are necessarily derivative free optimization methods, and, in turn, are necessarily less effective than methods that compute actual gradients, especially when the function evaluations are noisy. Another significant concern is that the choice of distribution can lead to very high variance in the stochastic gradients. Such high variance in turn implies that many samples need to be drawn to find a stationary point.

That said, the ease of implementation should not be readily discounted. Direct search methods are easy to implement, and oftentimes reasonable results can be achieved with considerably less effort than custom solvers tailored to the structure of the optimization problem. There are two primary ways that this sort of stochastic search arises in reinforcement learning: Policy gradient and pure random search.

Policy gradient

Though we have seen that the optimal solutions of Bellman's equations are deterministic, probabilistic policies can add an element of exploration to a control strategy, hopefully enabling an algorithm to simultaneously achieve reasonable awards and learn more about the underlying dynamics and reward functions. Such policies are the starting point for *policy gradient* methods.[9]

Consider a *parametric, randomized policy* such that u_t is sampled from a distribution $p(u|\tau_t;\vartheta)$ that is only a function of the currently observed trajectory and a parameter vector ϑ. A probabilistic policy induces a probability distribution over trajectories:

$$p(\tau;\vartheta) = \prod_{t=0}^{L-1} p(x_{t+1}|x_t,u_t)p(u_t|\tau_t;\vartheta)\,.$$

Moreover, we can overload notation and define the reward of a trajectory to be

$$R(\tau) = \sum_{t=0}^{N} R_t(x_t,u_t)$$

Then our optimization problem for reinforcement learning takes the form of stochastic search. Policy gradient thus proceeds by sampling a trajectory using the probabilistic policy with parameters ϑ_k, and then updating using REINFORCE.

Using the log-likelihood trick and the factored form of the probability distribution $p(\tau;\vartheta)$, we can see that the gradient of J with respect to ϑ *is not a function of the underlying dynamics*. However, at this point this should not be surprising: by shifting to distributions over policies, we push the burden of optimization onto the sampling procedure.

Pure random search

An older and more widely applied method to solve the generic stochastic search problem is to directly perturb the current decision variable z by random noise and then update the model based on the received reward at this perturbed value. That is, we apply the REINFORCE Algorithm with sampling distribution $p(z;\vartheta) = p_0(z-\vartheta)$ for some distribution p_0. Simplest examples for p_0 would be the uniform distribution on a sphere or a normal distribution. Perhaps less surprisingly here, REINFORCE can again be run without any knowledge of the underlying dynamics. The REINFORCE algorithm has a simple interpretation in terms of gradient approximation. Indeed, REINFORCE is equivalent to approximate gradient ascent of R

$$\vartheta_{t+1} = \vartheta_t + \alpha g_\sigma(\vartheta_k)$$

with the gradient approximation

$$g_\sigma(\vartheta) = \frac{R(\vartheta+\sigma\epsilon) - R(\vartheta-\sigma\epsilon)}{2\sigma}\epsilon\,.$$

This update says to compute a finite difference approximation to the gradient along the direction ϵ and move along the gradient. One can reduce the variance of such a finite-difference estimate by sampling along multiple random directions and averaging:

$$g_\sigma^{(m)}(\vartheta) = \frac{1}{m}\sum_{i=1}^{m}\frac{R(\vartheta+\sigma\epsilon_i) - R(\vartheta-\sigma\epsilon_i)}{2\sigma}\epsilon_i\,.$$

This is akin to approximating the gradient in the random subspace spanned by the ϵ_i

This particular algorithm and its generalizations goes by many different names. Probably the earliest proposal for this method is by Rastrigin.[10] Somewhat surprisingly, Rastrigin initially developed this method to solve reinforcement learning problems. His main motivating example was an inverted pendulum. A rigorous analysis using contemporary techniques was provided by Nesterov and Spokoiny.[11] Random search was also discovered by the evolutionary algorithms community and is called (μ,λ)-Evolution Strategies.[12,13] Random search has also been studied in the context of stochastic approximation[14] and bandits.[15,16] Algorithms that get invented by four different communities probably have something good going for them.

Deep reinforcement learning

We have thus far spent no time discussing *deep* reinforcement learning. That is because there is nothing conceptually different other than using

neural networks for function approximation. That is, if one wants to take any of the described methods and make them deep, they simply need to add a neural net. In model-based RL, φ is parameterized as a neural net, in ADP, the Q-functions or Value Functions are assumed to be well-approximated by neural nets, and in policy search, the policies are set to be neural nets. The algorithmic concepts themselves don't change. However, convergence analysis certainly will change, and algorithms like Q-learning might not even converge. The classic text Neurodynamic Programming by Bertsekas and Tsitisklis discusses the adaptations needed to admit function approximation.[17]

Certainty equivalence is often optimal for reinforcement learning

In this section, we give a survey of the power of certainty equivalence in sequential decision making problems. We focus on the simple cases of tabular MDPs and LQR as they are illustrative of more general problems while still being manageable enough to analyze with relatively simple mathematics. However, these analyses are less than a decade old. Though the principle of certainty equivalence dates back over 60 years, our formal understanding of certainty equivalence and its robustness is just now solidifying.

Certainty equivalence for LQR

Consider the linear quadratic regulator problem

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

We have shown that the solution to this problem is static state feedback $U_t = -K_\star X_t$ where

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

and M is the unique stabilizing solution to the Discrete Algebraic Riccati Equation

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

Suppose that instead of knowing (A, B, Φ, Ψ) exactly, we only have estimates $(\widehat{A}, \widehat{B}, \widehat{\Phi}, \widehat{\Psi})$. Certainty equivalence would then yield a control

policy $U_t = -\widehat{K}X_t$ where $\widehat{K}$ can be found using $(\widehat{A}, \widehat{B}, \widehat{\Phi}, \widehat{\Psi})$ in place of (A, B, Φ, Ψ) in the formulae above. What is the cost of this model?

The following discussion follows arguments due to Mania et al.[18] Let $J(K)$ denote that cost of using the policy K. Note that this cost may be infinite, but it will also be differentiable in K. If we unroll the dynamics and compute expected values, one can see that the cost is the limit of polynomials in K, and hence is differentiable.

Suppose that

$$\epsilon := \max\left\{\|\widehat{A} - A\|, \|\widehat{B} - B\|, \|\widehat{\Phi} - \Phi\|, \|\widehat{\Psi} - \Psi\|\right\}$$

If we Taylor expand the cost we find that for some $t \in [0, 1]$,

$$J(\widehat{K}) - J_\star = \langle \nabla J(K_\star), \widehat{K} - K_\star \rangle + \frac{1}{2}(\widehat{K} - K_\star)^T \nabla^2 J(\tilde{K})(\widehat{K} - K_\star)\,.$$

where $\tilde{K} = (1-t)K_\star + t\widehat{K}$. The first term is equal to zero because $K_\star$ is optimal. Since the map from (A, B, Φ, Ψ) to $K_\star$ is differentiable, there must be constants L and ϵ_0 such that $\|\widehat{K} - K_\star\| \leq L\epsilon$ whenever $\epsilon \leq \epsilon_0$. This means that as long as the estimates for (A, B, Φ, Ψ) are close enough to the true values, we must have

$$J(\widehat{K}) - J_\star = O(\epsilon^2)\,.$$

Just how good should the estimates for these quantities be? Let's focus on the dynamics (A, B) as the cost matrices Φ and Ψ are typically design parameters, not unknown properties of the system. Suppose A is $d \times d$ and B is $d \times p$. Basic parameter counting suggests that if we observe T sequential states from the dynamical system, we observe a total of dT numbers, one for each dimension of the state per time step. Hence, a naive statistical guess would suggest that

$$\max\left\{\|\widehat{A} - A\|, \|\widehat{B} - B\|\right\} \leq O\left(\sqrt{\frac{d+p}{T}}\right)\,.$$

Combining this with our Taylor series argument implies that

$$J(\widehat{K}) - J_\star = O\left(\frac{d+p}{T}\right)\,.$$

As we already described, this also suggests that certainty equivalent control accrues a regret of

$$\mathcal{R}_T = O(\sqrt{T})\,.$$

This argument can be made completely rigorous.[18] The regret accrued also turns out to be the optimal.[19] Moreover, the Taylor series argument here works for any model where the cost function is twice differentiable. Thus, we'd expect to see similar behavior in more general SDM problems with continuous state spaces.

Certainty equivalence for tabular MDPs

For discounted, tabular MDPs, certainty equivalence also yields an optimal sample complexity. This result is elementary enough to be proven in a few pages. We first state an approximation theorem that shows that if you build a policy with the wrong model, the value of that policy can be bounded in terms of the inaccuracy of your model. Then, using Hoeffding's inequality, we can construct a sample complexity bound that is nearly optimal. The actual optimal rate follows using our main approximation theorem coupled with slightly more refined concentration inequalities. We refer readers interested in this more refined analysis to the excellent reinforcement learning text by Agarwal et al.[20]

Let $V_\star(x)$ denote the optimal expected reward attainable by some policy on the discounted problem

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

Note that $V_\star$ is a function of the initial state. This mapping from initial state to expected rewards is called the *value function* of the SDM problem. Let $V^\pi(x)$ denote the expected reward attained when using some fixed, static policy π. Our aim is to evaluate the reward of particular policies that arise from certainty equivalence.

To proceed, first let $\widehat{Q}$ be any function mapping state-action pairs to a real value. We can always define a policy $\pi_{\widehat{Q}}(x) = \arg\max_u \widehat{Q}(x, u)$. The following theorem quantifies the value of $\pi_{\widehat{Q}}$ when $\widehat{Q}$ is derived by solving the Bellman equation with an approximate model of the MDP dynamics. This theorem has been derived in numerous places in the RL literature, and yet it does not appear to be particularly well known. As pointed out by 'Avila Pires and Szepesvari,[21] it appears as a Corollary to Theorem 3.1 in Whitt[22] (1978), as Corollary 2 in Singh and Yee[23] (1994), as a corollary of Proposition 3.1 in Bertsekas[24] (2012). We emphasize it here as it demonstrates immediately why certainty equivalence is such a powerful tool in sequential decision making problems.

Theorem 2. ***Model-error for MDPs.*** *Consider a γ-discounted MDP with dynamics governed by a model p and a reward function r. Let $\widehat{Q}$ denote the Q-function for the MDP with the same rewards but dynamics $\widehat{\mathbb{P}}$. Then we have*

$$V_\star(x) - V^{\pi_{\widehat{Q}}}(x) \leq \frac{2\gamma}{(1-\gamma)^2} \sup_{x,u} \left| \mathop{\mathbb{E}}_{\widehat{\mathbb{P}}(\cdot|x,u)}[V_\star] - \mathop{\mathbb{E}}_{\mathbb{P}(\cdot|x,u)}[V_\star] \right|.$$

This theorem states that the values associated with the policy that we derive using the wrong dynamics will be close to optimal if $\mathbb{E}_{\widehat{\mathbb{P}}(\cdot|x,u)}[V_\star]$ and $\mathbb{E}_{\mathbb{P}(\cdot|x,u)}[V_\star]$ are close for all state-action pairs (x,u). This is a remarkable result as it shows that we can control our regret by our prediction errors. But the only prediction that matters is our predictions of the optimal value vectors $V_\star$. Note further that this theorem makes no assumptions about the size of the state spaces: it holds for discrete, tabular MDPs, and more general discounted MDPs. A discussion of more general problems is covered by Bertsekas.[25]

Focusing on the case of finite-state tabular MDPs, suppose the rewards are in the range $[0,1]$ and there are S states and A actions. Then the values are in the range $V_\star(x) \in [0, (1-\gamma)^{-1}]$. Immediately from this result, we can derive a sample-complexity bound. Let's suppose that for each pair (x,u), we collect n samples to estimate the conditional probabilities $\mathbb{P}[X' = x'|x,u]$, and define $\widehat{\mathbb{P}}(X' = x'|x,u)$ to be the number of times we observe x' divided by n. Then by Hoeffding's inequality

$$\mathbb{P}\left[\left| \mathop{\mathbb{E}}_{\widehat{\mathbb{P}}[\cdot|x,u]}[V_\star] - \mathop{\mathbb{E}}_{\mathbb{P}[\cdot|x,u]}[V_\star] \right| \geq \epsilon \right] \leq 2\exp\left(-2n\epsilon^2(1-\gamma)^2\right)$$

and therefore, by the union bound

$$\sup_{x,u} \left| \mathop{\mathbb{E}}_{\widehat{\mathbb{P}}[\cdot|x,u]}[V_\star] - \mathop{\mathbb{E}}_{\mathbb{P}[\cdot|x,u]}[V_\star] \right| \leq \sqrt{\frac{\log\left(\frac{2SA}{\delta}\right)}{n(1-\gamma)^2}}.$$

with probability $1-\delta$. If we let $N = SAn$ denote the total number of samples collected, we see that

$$V_\star(x) - V^{\pi_{\widehat{Q}}}(x) \leq \frac{2\gamma}{(1-\gamma)^3} \sqrt{\frac{SA\log\left(\frac{2SA}{\delta}\right)}{N}}.$$

Our naive bound here is nearly optimal: the dependence on γ can be reduced from $(1-\gamma)^{-3}$ to $(1-\gamma)^{-3/2}$ using more refined deviation inequalities, but the dependence on S, A, and N is optimal.[20,26] That is, certainty equivalence achieves an optimal sample complexity for the discounted tabular MDP problem.

Proof of the model-error theorem

The proof here combines arguments by Bertsekas[24] and Agarwal.[20] Let us first introduce notation that makes the proof a bit more elegant. Let Q be any function mapping state-action pairs to real numbers. Given a policy π and a state-transition model $\mathbb{P}$, denote $\mathcal{T}_{\mathbb{P}}$ to be the map from functions to functions where

$$[\mathcal{T}_{\mathbb{P}} Q](x,u) = r(x,u) + \gamma \sum_{x'} \max_{u'} Q(x',u')\, \mathbb{P}[X' = x' | x, u]\,.$$

With this notation, the Bellman equation for the discounted MDP simply becomes

$$Q_\star = \mathcal{T}_{\mathbb{P}} Q_\star\,.$$

If we were to use $\widehat{\mathbb{P}}$ instead of $\mathbb{P}$, this would yield an alternative Q-function, $\widehat{Q}$, that satisfies the Bellman equation $\widehat{Q} = \mathcal{T}_{\widehat{\mathbb{P}}} \widehat{Q}$.

The operator $\mathcal{T}_{\mathbb{P}}$ is a *contraction mapping* in the ℓ_∞ norm. To see this, note that for any functions Q_1 and Q_2,

$$\begin{aligned}
|[\mathcal{T}_{\mathbb{P}} Q_1 - \mathcal{T}_{\mathbb{P}} Q_2](x,u)| &= \left| \gamma \sum_{x'} (\max_{u_1} Q_1(x',u_1) - \max_{u_2} Q_2(x',u_2))\, \mathbb{P}[X' = x' | x, u] \right| \\
&\leq \gamma \sum_{x'} \mathbb{P}[X' = x' | x, u] \left| \max_{u_1} Q_1(x',u_1) - \max_{x_2} Q_2(x',u_2) \right| \\
&\leq \gamma \| Q_1 - Q_2 \|_\infty\,.
\end{aligned}$$

Since $\mathcal{T}_{\mathbb{P}}$ is a contraction, the solution of the discounted Bellman equations are unique and $Q_\star = \lim_{k\to\infty} {\mathcal{T}_{\mathbb{P}}}^k Q$ for any function Q. Similarly, $\widehat{Q} = \lim_{k\to\infty} {\mathcal{T}_{\widehat{\mathbb{P}}}}^k Q$.

Now we can bound

$$\left\| {\mathcal{T}_{\widehat{\mathbb{P}}}}^k Q_\star - Q_\star \right\|_\infty \leq \sum_{i=1}^{k} \| {\mathcal{T}_{\widehat{\mathbb{P}}}}^i Q_\star - {\mathcal{T}_{\widehat{\mathbb{P}}}}^{i-1} Q_\star \|_\infty \leq \sum_{i=1}^{k} \gamma^{i-1} \| \mathcal{T}_{\widehat{\mathbb{P}}} Q_\star - Q_\star \|_\infty\,.$$

Taking limits of both sides as $k \to \infty$, we find

$$\left\| \widehat{Q} - Q_\star \right\|_\infty \leq \frac{1}{1-\gamma} \| \mathcal{T}_{\widehat{\mathbb{P}}} Q_\star - Q_\star \|_\infty\,.$$

But since $Q_\star = \mathcal{T}_{\mathbb{P}} Q_\star$, and

$$\begin{aligned}
&[\mathcal{T}_{\widehat{\mathbb{P}}} Q_\star - \mathcal{T}_{\mathbb{P}} Q_\star](x,u) \\
=&\gamma \sum_{x'} \max_{u'} Q_\star(x',u') \left(\widehat{\mathbb{P}}[X' = x' | x, u] - \mathbb{P}[X' = x' | x, u] \right)\,,
\end{aligned}$$

we have the bound

$$\left\|\widehat{Q} - Q_\star\right\|_\infty \leq \frac{\gamma}{1-\gamma} \sup_{x,u} \left| \mathop{\mathbb{E}}_{\widehat{\mathbb{P}}[\cdot|x,u]}[V_\star] - \mathop{\mathbb{E}}_{\mathbb{P}[\cdot|x,u]}[V_\star] \right| .$$

To complete the proof, it suffices to use $\left\|\widehat{Q} - Q_\star\right\|_\infty$ to upper bound the difference between the values of the two policies. Let $\pi_\star(x) = \arg\max_u Q_\star(x,u)$ and $\widehat{\pi}(x) = \arg\max_u \widehat{Q}_\star(x,u)$ denote the optimal policies for the models $\mathbb{P}$ and $\widehat{\mathbb{P}}$ respectively. For any policy π, we have

$$V^\pi(x) = r(x,\pi(x)) + \gamma \sum_{x'} \mathbb{P}[X' = x'|x,\pi(x)] V^\pi(x') ,$$

and hence we can bound the optimality gap as

$$\begin{aligned}
V_\star(x) - V^{\widehat{\pi}}(x) &= Q_\star(x,\pi_\star(x)) - V^{\widehat{\pi}}(x) \\
&= Q_\star(x,\pi_\star(x)) - Q_\star(x,\widehat{\pi}(x)) + Q_\star(x,\widehat{\pi}(x)) - V^{\widehat{\pi}}(x) \\
&= Q_\star(x,\pi_\star(x)) - Q_\star(x,\widehat{\pi}(x)) \\
&\qquad + \gamma \sum_{x'} \mathbb{P}[X' = x'|x,\widehat{\pi}(x)] \left(V_\star(x') - V^{\widehat{\pi}}(x') \right) \\
&\leq Q_\star(x,\pi_\star(x)) - \widehat{Q}(x,\pi_\star(x)) + \widehat{Q}(x,\widehat{\pi}(x)) - Q_\star(x,\widehat{\pi}(x)) \\
&\qquad + \gamma \sum_{x'} \mathbb{P}[X' = x'|x,\widehat{\pi}(x)] \left(V_\star(x') - V^{\widehat{\pi}}(x') \right) \\
&\leq 2\|Q_\star - \widehat{Q}\|_\infty + \gamma \|V_\star - V^{\widehat{\pi}}\|_\infty .
\end{aligned}$$

Here, the first inequality holds because $\widehat{Q}(x,\pi_\star(x)) \leq \max_u \widehat{Q}(x,u) = \widehat{Q}(x,\widehat{\pi}(x))$. Rearranging terms shows that

$$V_\star(x) - V^{\widehat{\pi}}(x) \leq \frac{2}{1-\gamma} \|Q_\star - \widehat{Q}\|_\infty ,$$

which, when combined with our previous bound on $\|Q_\star - \widehat{Q}\|_\infty$, completes the proof.

Sample complexity of other RL algorithms

The sample complexity of reinforcement learning remains an active field, with many papers honing in on algorithms with optimal complexity. Researchers have now shown a variety of methods achieve the optimal complexity of certainty equivalence including those based on ideas from approximate dynamic programming and Q-learning. For LQR on the other hand, no other methods are currently competitive.

While sample complexity is important, there are not significant gains to be made over simple baselines that echo decades of engineering practice. And, unfortunately, though sample complexity is a well posed problem which excites many researchers, it does not address many of the impediments preventing reinforcement learning form being deployed in more applications. As we will see in a moment, the optimization framework itself has inherent weaknesses that cannot be fixed by better sample efficiency, and these weaknesses must be addressed head-on when designing an SDM system.

The limits of learning in feedback loops

Though we have shown the power of certainty equivalence, it is also a useful example to guide how reinforcement learning—and optimal sequential decision making more generally—can go wrong. First, we will show how optimal decision making problems themselves can be set up to be very sensitive to model-error. So treating a model as true in these cases can lead to misguided optimism about performance. Second, we will adapt this example to the case where the state is partially observed and demonstrate a more subtle pathology. As we discussed in the last chapter, when state is not perfectly observed, decision making is decidedly more difficult. Here we will show an example where improving your prediction paradoxically increases your sensitivity to model error.

Fragile instances of the linear quadratic regulator

Consider the following innocuous dynamics:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

This system is a simple, two-state shift register. Write the state out with indexed components $x = [x^{(1)}, x^{(2)}]^\top$. New states enter through the control B into the second state. The first state $x^{(1)}$ is simply whatever was in the second register at the previous time step. The open loop dynamics of this system are as stable as you could imagine. Both eigenvalues of A are zero.

Let's say our control objective aims to try to keep the two components of the state equal to each other. We can model this with the quadratic cost matrices

$$\Phi = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad \Psi = 0\,.$$

Here, $\Psi = 0$ for simplicity, as the formulae are particularly nice for this case. But, as we will discuss in a moment, the situation is not improved simply by having R be positive. For the disturbance, assume that W_t is zero mean, has bounded second moment, $\Sigma_t = \mathbb{E}[W_t W_t^\top]$, and is uncorrelated with X_t and U_t.

The cost is asking to minimize

$$\mathbb{E}\left[\sum_{t=1}^{N} (X_t^{(1)} - X_t^{(2)})^2\right]$$

When $W_t = 0$, $X_t^{(1)} + X_t^{(2)} = X_{t-1}^{(2)} + U_{t-1}$, so the intuitive best action would be to set $U_t = X_t^{(2)}$. This turns out to be the optimal action, and one can prove this directly using standard dynamic programming computations or a Discrete Algebraic Riccati Equation. With this identification, we can write *closed loop* dynamics by eliminating the control signal:

$$X_{t+1} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} X_t + W_t\,.$$

This closed-loop system is *marginally stable*, meaning that while signals don't blow up, some states will persist forever and not converge to 0. The second component of the state simply exhibits a random walk on the real line. We can analytically see that the system is not stable by computing the eigenvalues of the state-transition matrix, which are here 0 and 1. The 1 corresponds the state where the two components are equal, and such a state can persist forever.

If we learned an incorrect model of the dynamics, how would that influence the closed loop behavior? The simplest scenario is that we identified B from some preliminary experiments. If the true $B_\star = \alpha B$, then the closed loop dynamics are

$$X_{t+1} = \begin{bmatrix} 0 & 1 \\ 0 & \alpha \end{bmatrix} X_t + W_t\,.$$

This system is unstable for any $\alpha > 1$. That is, the system is arbitrarily sensitive to misidentification of the dynamics. This lack of robustness has nothing to do with the noise sequence. The structure of the cost is what drives the system to fragility.

If $\Psi > 0$, we would get a slightly different policy. Again, using elementary dynamic programming shows that the optimal control is $u_t = \beta_t(\Psi) x_t^{(2)}$

for some $\beta_t(\Psi) \in (1/2, 1)$. The closed loop system will be a bit more stable, but this comes at the price of reduced performance. You can also check that if you add ϵ times the identity to Φ, we again get a control policy proportional to the second component of the state, $x_t^{(2)}$.

Similar examples are fairly straightforward to construct. The state-transition matrix of the closed loop dynamics will always be of the form $A - BK$, and we can first find a K such that $A - BK$ has an eigenvalue of magnitude 1. Once this is constructed, it suffices to find a vector v such that $(v'B)^{-1}v'A = K$. Then the cost $\Phi = vv'$ yields the desired pathological example.

One such example that we will use in our discussion of partially observed systems is the model:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

$$\Phi = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1/4 \end{bmatrix}, \quad \Psi = 0.$$

The dynamics here are our "Newton's Law" dynamics studied in our dynamic programming examples. One can check that the closed loop dynamics of this system are

$$X_{t+1} = \begin{bmatrix} 1 & 1 \\ -2 & -2 \end{bmatrix} X_t + W_t.$$

The transition matrix here has eigenvalues 0 and -1, and the state $x = [1/2, -1]$ will oscillate in sign and persist forever.

Partially observed example

Recall the generalization of LQR to the case with imperfect state observation is called "Linear Quadratic Gaussian" control (LQG). This is the simplest, special case of a POMDP. We again assume linear dynamics:

$$X_{t+1} = AX_t + BU_t + W_t.$$

where the state is now corrupted by zero-mean Gaussian noise, W_t. Instead of measuring the state X_t directly, we instead measure a signal Y_t of the form

$$Y_t = CX_t + V_t.$$

Here, V_t is also zero-mean Gaussian noise. Suppose we'd still like to minimize a quadratic cost function

$$\lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T} X_t^\top \Phi X_t + U_t^\top \Psi U_t\right].$$

This problem is very similar to our LQR problem except for the fact that we get an indirect measurement of the state and need to apply some sort of *filtering* of the Y_t signal to estimate X_t.

The optimal solution for LQG is strikingly elegant. Since the observation of X_t is through a Gaussian process, the maximum likelihood estimation algorithm has a clean, closed form solution. As we saw in the previous chapter, our best estimate for X_t, denoted $\widehat{x}_t$, given all of the data observed up to time t is given by a Kalman Filter. The estimate obeys a difference equation

$$\widehat{x}_{t+1} = A\widehat{x}_t + Bu_t + L(y_t - C\widehat{x}_t)\,.$$

The matrix L that can be found by solving an discrete algebraic Riccati equation that depends on the variance of v_t and w_t and on the matrices A and C. In particular, it's the DARE with data $(A^\top, C^\top, \Sigma_w, \Sigma_v)$.

The optimal LQG solution takes the estimate of the Kalman Filter, $\widehat{x}_t$, and sets the control signal to be

$$u_t = -K\widehat{x}_t\,.$$

Here, K is gain matrix that would be used to solve the LQR problem with data (A, B, Φ, Ψ). That is, LQG performs optimal filtering to compute the best state estimate, and then computes a feedback policy as if this estimate was a noiseless measurement of the state. That this turns out to be optimal is one of the more amazing results in control theory. It decouples the process of designing an optimal filter from designing an optimal controller, enabling simplicity and modularity in control design. This decoupling where we treat the output of our state estimator as the true state is yet another example of certainty equivalence, and yet another example of where certainty equivalence turns out to be optimal. However, as we will now see, LQG highlights a particular scenario where certainty equivalent control leads to misplaced optimism about robustness.

Before presenting the example, let's first dive into *why* LQG is likely less robust than LQR. Let's assume that the true dynamics are generated as:

$$X_{t+1} = AX_t + B_\star U_t + W_t\,,$$

though we computed the optimal controller with the matrix B. Define an error signal, $E_t = X_t - \widehat{x}_t$, that measures the current deviation between the actual state and the estimate. Then, using the fact that $u_t = -K\widehat{x}_t$, we get the closed loop dynamics

$$\begin{bmatrix} \widehat{X}_{t+1} \\ E_{t+1} \end{bmatrix} = \begin{bmatrix} A - BK & LC \\ (B - B_\star)K & A - LC \end{bmatrix} \begin{bmatrix} \widehat{X}_t \\ E_t \end{bmatrix} + \begin{bmatrix} LV_t \\ W_t - LV_t \end{bmatrix}.$$

When $B = B_\star$, the bottom left block is equal to zero. The system is then stable provided $A - BK$ and $A - LC$ are both stable matrices (i.e., have

eigenvalues with magnitude less than one). However, small perturbations in the off-diagonal block can make the matrix unstable. For intuition, consider the matrix

$$\begin{bmatrix} 0.9 & 1 \\ 0 & 0.8 \end{bmatrix}.$$

The eigenvalues of this matrix are 0.9 and 0.8, so the matrix is clearly stable. But the matrix

$$\begin{bmatrix} 0.9 & 1 \\ t & 0.8 \end{bmatrix}$$

has an eigenvalue greater than one if $t > 0.02$. So a tiny perturbation significantly shifts the eigenvalues and makes the matrix unstable.

Similar things happen in LQG. Let's return to our simple dynamics inspired by Newton's Laws of Motion

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

And let's use *any* cost matrices Φ and Ψ. We assume that the noise variances are

$$\mathbb{E}\left[W_t W_t^\top\right] = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad \mathbb{E}\left[V_t^2\right] = \sigma^2$$

The open loop system here is unstable, having two eigenvalues at 1. We can stabilize the system only by modifying the second state. The state disturbance is aligned along the direction of the vector $[1/2; 1]$, and the state cost only penalizes states aligned with this disturbance. The SDM goal is simply to remove as much signal as possible in the $[1; 1]$ direction without using large inputs. We only are able to measure the first component of the state, and this measurement is corrupted by Gaussian noise.

What does the optimal policy look like? Perhaps unsurprisingly, it focuses all of its energy on ensuring that there is little state signal along the disturbance direction. The optimal L matrix is

$$L = \begin{bmatrix} 3 - d_1 \\ 2 - d_2 \end{bmatrix}.$$

where d_1 and d_2 are small positive numbers that go to zero as σ goes to zero. The optimal K will have positive coefficients whenever we choose for Φ and Ψ to be positive semidefinite: if K has a negative entry, it will necessarily not stabilize (A, B).

Now what happens when we have model mismatch? Let's assume for simplicity that $\sigma = 0$. If we set $B_\star = tB$ and use the formula for the closed

loop above, we see that closed loop state transition matrix is

$$A_{cl} = \begin{bmatrix} 1 & 1 & 3 & 0 \\ -k_1 & 1-k_2 & 2 & 0 \\ 0 & 0 & -2 & 1 \\ k_1(1-t) & k_2(1-t) & -2 & 1 \end{bmatrix}.$$

It's straight forward to check that when $t = 1$ (i.e., no model mismatch), the eigenvalues of $A - BK$ and $A - LC$ all have real parts with magnitude less than or equal to 1. For the full closed loop matrix, analytically computing the eigenvalues themselves is a pain, but we can prove instability by looking at a characteristic polynomial. For a matrix to have all of its eigenvalues in the left half plane, its characteristic polynomial necessarily must have all positive coefficients. If we look at the linear term in the characteristic polynomial, of $-I - A_{cl}$ we see that if $t > 1$, A_{cl} must have an eigenvalue with real part less than -1, and hence the closed loop is unstable. This is a very conservative condition, and we could get a tighter bound if we'd like, but it's good enough to reveal some paradoxical properties of LQG. The most striking is that if we build a sensor that gives us a better and better measurement, our system becomes more and more fragile to perturbation and model mismatch. For machine learning scientists, this seems to go against all of our training. How can a system become *less* robust if we improve our sensing and estimation?

Let's look at the example in more detail to get some intuition for what's happening. When the sensor noise gets small, the optimal Kalman Filter is more aggressive. The filter rapidly damps any errors in the disturbance direction $[1; 1/2]$ and, as σ decreases, it damps the $[1; 1]$ direction less. When $t \neq 1$, $B - B_\star$ is aligned in the $[0; 1]$ and can be treated as a disturbance signal. This undamped component of the error is fed errors from the state estimate, and these errors compound each other. Since we spend so much time focusing on our control along the direction of the injected state noise, we become highly susceptible to errors in a different direction and these are the exact errors that occur when there is a gain mismatch between the model and reality.

The fragility of LQG has many takeaways. It highlights that noiseless state measurement can be a dangerous modeling assumption, because it is then optimal to trust our model too much. Model mismatch must be explicitly accounted for when designing the decision making policies.

This should be a cautionary tale for modern AI systems. Most papers in reinforcement learning consider MDPs where we perfectly measure the system state. Building an entire field around optimal actions with perfect state observation builds too much optimism. Any realistic scenario is going to have partial state observation, and such problems are much thornier.

A second lesson is that it is not enough to just improve the prediction components in feedback systems that are powered by machine learning. Improving prediction will increase sensitivity to a modeling errors in some other part of the engineering pipeline, and these must all be accounted for together to ensure safe and successful decision making.

Chapter notes

This chapter and the previous chapter overlap significantly with a survey of reinforcement learning by Recht, which contains additional connections to continuous control.[27]

Bertsekas has written several valuable texts on reinforcement learning from different perspectives. His seminal book with Tsitsiklis established the mathematical formalisms of Neurodynamic Programming that most resemble contemporary reinforcement learning.[17] The second volume of his Dynamic Programming Book covers many of the advanced topics in approximate dynamic programming and infinite horizon dynamic programming.[28] And his recent book on reinforcement learning builds ties with his earlier work and recent advances in reinforcement learning post AlphaGo.[29]

For more on bandits from a theoretical perspective, the reader is invited to consult the comprehensive book by Lattimore and Szepesvari.[30] Agarwal et al. provide a thorough introduction to the theoretical aspects of reinforcement learning from the perspective of learning theory.[20]

The control theoretic perspective on reinforcement learning is called *dual control*. Its originated at a similar time to reinforcement learning, and many attribute Feldbaum's work as the origin point.[31] Wittenmark surveys the history of this topic, its limitations, and its comparison to certainty equivalence methods.[32] For further exploration of the limits of classical optimal control and how to think about robustness, Gunter Stein's "Respect the Unstable" remains a classic lecture on the subject.[33]

Bibliography

[1] Herbert A. Simon. Dynamic programming under uncertainty with a quadratic criterion function. *Econometrica*, 24(1):74–81, 1956.

[2] Henri Theil. A note on certainty equivalence in dynamic planning. *Econometrica*, 25(2):346–349, 1957.

[3] Peter Auer and Ronald Ortner. UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

[4] Sampath Kannan, Jamie H. Morgenstern, Aaron Roth, Bo Waggoner, and Zhiwei Steven Wu. A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. In *Advances in Neural Information Processing Systems*, 2018.

[5] Patrick Hummel and R. Preston McAfee. Machine learning in an auction environment. *Journal of Machine Learning Research*, 17(1):6915–6951, 2016.

[6] Alberto Bietti, Alekh Agarwal, and John Langford. A contextual bandit bake-off. *arXiv:1802.04064*, 2018.

[7] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[8] Gavin Adrian Rummery and Mahesan Niranjan. Online Q-learning using connectionist systems. Technical report, CUED/F-INFENG/TR 166, Cambridge University Engineering Dept., 1994.

[9] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[10] Leonard A. Rastrigin. About convergence of random search method in extremal control of multi-parameter systems. *Avtomat. i Telemekh.*, 24(11):1467—1473, 1963.

[11] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[12] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution Strategies—a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[13] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, TU Berlin, 1975.

[14] James C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *Transactions on Automatic Control*, 37(3):332–341, 1992.

[15] Abraham D. Flaxman, Adam T. Kalai, and H. Brendan McMahan. Online convex optimization in the bandit setting: Gradient descent without a gradient. In *Symposium on Discrete Algorithms*, pages 385–394, 2005.

[16] Alekh Agarwal, Ofer Dekel, and Lin Xiao. Optimal algorithms for online convex optimization with multi-point bandit feedback. In *Conference on Learning Theory*, 2010.

[17] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[18] Horia Mania, Stephen Tu, and Benjamin Recht. Certainty equivalence is efficient for linear quadratic control. In *Advances in Neural Information Processing Systems*, 2019.

[19] Max Simchowitz and Dylan Foster. Naive exploration is optimal for online LQR. In *International Conference on Machine Learning*, 2020.

[20] Alekh Agarwal, Nan Jiang, Sham M. Kakade, and Wen Sun. *Reinforcement Learning: Theory and Algorithms*. 2020. Preprint Available at `rltheorybook.github.io`.

[21] Bernardo Ávila Pires and Csaba Szepesvári. Policy error bounds for model-based reinforcement learning with factored linear models. In *Conference on Learning Theory*, 2016.

[22] Ward Whitt. Approximations of dynamic programs, I. *Mathematics of Operations Research*, 3(3):231–243, 1978.

[23] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[24] Dimitri P. Bertsekas. Weighted sup-norm contractions in dynamic programming: A review and some new applications. LIDS Tech Report LIDS-P-2884, Department of Electrical Engineering and Computer Science, Massachusetts Institute Technology, 2012.

[25] Dimitri P. Bertsekas. *Abstract Dynamic Programming*. Athena Scientific, 2nd edition, 2018.

[26] Alekh Agarwal, Sham M. Kakade, and Lin F. Yang. Model-based reinforcement learning with a generative model is minimax optimal. In *Conference on Learning Theory*, 2020.

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

[28] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4th edition, 2012.

[29] Dimitri P. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[30] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[31] Aleksandr Aronovich Feldbaum. Dual control theory. *Avtomatika i Telemekhanika*, 21(9):1240–1249, 1960.

[32] Björn Wittenmark. Adaptive dual control methods: An overview. In *Adaptive Systems in Control and Signal Processing*, pages 67–72. Elsevier, 1995.

[33] Gunter Stein. Respect the unstable. *IEEE Control Systems Magazine*, 23(4):12–25, 2003.