1. Introductions

PDF Version

Introduction

Hello everyone and welcome to CMPUT 605: Theoretical Foundations of Reinforcement Learning at the University of Alberta. We are very excited to be teaching this course and hope that you are excited to journey with us through reinforcement learning theory. The course will cover two sub-topics of RL theory: (1) Planning, and (2) Online RL:

- Planning refers to the problem of computing plans, or policies, or just action by interacting with some model.
- Online RL refers to the problem of coming up with actions that maximize total reward while interacting with an environment.

In all of these subproblems, we will use Markov Decision Processes, to describe how either the simulation models, or the environments work. Thus, we start by introducing the formal definition of a Markov Decision Process (MDP).

Markov Decision Process

A Markov Decision Process is a mathematical model for modelling sequential decision making in an environment that undergoes stochastic transitions. An MDP consists of the following elements: states, actions, rules of stochastic transitions between states, rewards, and an objective, which we take for now to be the discounted total expected reward, or return.

States are considered to be primitive thus we do not explicitly define what they are. The set of states will be denoted by $\mathcal{S}$. Actions are also primitive and their set is denoted by $\mathcal{A}$. For simplicity, we assume that both sets are finite. We will let the number of states be denoted by S, and similarly, we let the number of actions be denoted by A. Stochastic transitions between states s and s' are the result of choosing some action a in a given state. For a fixed state s and action a, the probabilities of landing in the various states s' is collected into a probability vector, which is denoted by $P_a(s)$. To minimize clutter, by slightly abusing notation, we will write $P_a(s, s')$ as the $s' \in \mathcal{S}$ component of this probability vector. This is the probability that the process will transition into state s', when in state s it takes action a.

Rewards are scalars and the reward incurred as a result of taking action a in state s is denoted by $r_a(s)$. Since the number of states and actions are finite, there is no loss in generality by assuming that all the rewards belong to the $[0, 1]$ interval. Taking action A_t at time step t gives rise to an infinitely long trajectory of state-action pairs $S_0, A_0, S_1, A_1, \dots$: here, S_{t+1} is the state that results from taking action A_t in time step $t \geq 0$ and the assumption is that as long as A_t is chosen based on the "past" only, the distribution of S_{t+1} given $S_0, A_0, \dots, S_t, A_t$ is solely determined by $P_{A_t}(S_t)$, and, in particular, $\mathbb{P}$-almost surely,

$$\mathbb{P}(S_{t+1} = s | S_0, A_0, \dots, S_t, A_t) = P_{A_t}(S_t, s)\,. \tag{1}$$

The objective is to find a way of choosing the actions that result in the largest possible return along the trajectories that arise. The return along a trajectory is defined as

$$R = r_{A_0}(S_0) + \gamma r_{A_1}(S_1) + \gamma^2 r_{A_2}(S_2) + \dots + \gamma^t r_{A_t}(S_t) + \dots$$

where $\gamma \in [0, 1)$ is the discount factor. Formally, a (discounted) MDP will thus be described by the 5-tuple $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $P = (P_a(s))_{s,a}$ and $r = (r_a(s))_{s,a}$ collect the transitions and the rewards, respectively.

On Discounting

Note that γ makes it so that the future reward does not matter as much as the present reward. Also, if we truncate the above sum after $H \geq 0$ terms, by our assumption on the rewards, the difference between the return and the truncated return is between zero and

$$\gamma^H \left[r_{A_H}(S_H) + \gamma r_{A_{H+1}}(S_{H+1}) + \dots \right] \leq \gamma^H \sum_{s \geq 0} \gamma^s = \frac{\gamma^H}{1-\gamma}$$

by using the summation rule for geometric series. Solving for the largest H under which the above upper bound on the difference is below ε, we get that this bound on the difference holds as long as H satisfies

$$H \geq \underbrace{\frac{\ln\left(\frac{1}{\varepsilon(1-\gamma)}\right)}{\ln(1/\gamma)}}_{H^*_{\gamma,\varepsilon}}\,.$$

For H satisfying this, the return is maximized already when considering only the first H time steps.

Notice that the critical value of H depends on not only ε but also γ. For a fixed ε, this critical value is called the *effective horizon*.

Oftentimes, for the sake of simplicity, we replace $H^*_{\gamma,\varepsilon}$ with the following quantity:

$$H_{\gamma,\varepsilon} := \frac{\ln\left(\frac{1}{\varepsilon(1-\gamma)}\right)}{1-\gamma} .$$

(In fact, the literature often calls the latter the effective horizon). This quantity is an upper bound on $H^*_{\gamma,\varepsilon}$. Furthermore, it is not hard to verify that the relative difference between these two quantities is of order $o(1-\gamma)$ as $\gamma \to 1$. Thus, $H^*_{\gamma,\varepsilon}$ behaves the same as $H_{\gamma,\varepsilon}$ up to a first-order approximation as $\gamma \to 1$. Since we are typically interested in this regime (large horizons), there is no loss in switching from $H^*_{\gamma,\varepsilon}$ to $H_{\gamma,\varepsilon}$.

The discounted setting may occasionally feel a bit cringey. Where is the discount factor coming from? One approach is to think about how many time steps in the future we think the optimization should look into for some level of desired accuracy and then work backwards to set γ so that the resulting effective horizon matches our expectation. However, it is more honest to admit that the discounted objective may not faithfully capture the nature of a decision problem. Indeed, there are other objectives that one can consider, such as the finite horizon, undiscounted (or discounted) setting, the infinite horizon setting with no discounting ("total reward"), or the infinite horizon with the average reward. All these have their own pros and cons and we will consider some of these objectives and their relationships in future lectures. For now, we will stick to the discounted objective for pedagogical reasons: the math underlying the discounted objective is simple and elegant. Also, many results transfer to the other settings mentioned, perhaps with some extra conditions, or a little change.

Policies

A policy is a rule that describes how the actions should be taken in light of the past. Here, the past at time step $t \ge 0$ is defined as

$$H_t = (S_0, A_0, \dots, S_{t-1}, A_{t-1}, S_t)$$

which is the sequence of state-action pairs leading up to the state of the process at the current time step t. We allow policies to randomize. As such, formally, a policy becomes an infinite sequence $\pi = (\pi_t)_{t\ge 0}$ of maps of histories to distributions over actions. For a (finite) set X let $\mathcal{M}_1(X)$ denote the set of probability distributions over X. These probability distributions are uniquely determined by what probability they assign to the individual elements of X. Hence, they will be identified with the probability vectors with $|X|$ components, each component giving the probability of some $x \in X$. If $p \in \mathcal{M}_1(X)$, we use both p_x and $p(x)$ to denote this probability (whichever is more convenient).

With this, we can write that the tth "rule" in π, which will be used in the tth time step to come up with the action for that time step, as

$$\pi_t : \mathcal{H}_t \to \mathcal{M}_1(\mathcal{A}) ,$$

where

$$\mathcal{H}_t = (\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S}\,.$$

Note that $\mathcal{H}_0 = \mathcal{S}$. Intuitively, following a policy π means that in time step $t \geq 0$, the distribution of the action A_t to be chosen for that timestep is $\pi_t(H_t)$: the probability that $A_t = a$ is $\pi_t(H_t)(a)$. Since writing $\pi_t(H_t)(a)$ is quite cumbersome, we abuse notation and will write $\pi_t(a|H_t)$ instead. Thus, when following a policy π, in time step $t \geq 0$ we get that, $\mathbb{P}$-almost surely,

$$\mathbb{P}(A_t = a|H_t) = \pi_t(a|H_t)\,. \qquad (2)$$

Initial State Distributions, Distributions over Trajectories

When a policy is *interconnected* with an MDP, the interconnection, together with an initial distribution $\mu \in \mathcal{M}_1(\mathcal{S})$ over the states, uniquely determines a distribution over the infinite-long trajectories

$$T = (\mathcal{S} \times \mathcal{A})^{\mathbb{N}}$$

such that for every time step $t \geq 0$, both (1) and (2) hold, in addition to that

$$\mathbb{P}(S_0 = s) = \mu(s)\,, \qquad s \in \mathcal{S}\,. \qquad (3)$$

In fact, this distribution could be over some potentially bigger probability space, in which case uniqueness does not hold. When we want to be specific and take the distribution that is defined over the infinite-long state-action trajectories, we will say that this is the distribution over the *canonical probability space* induced by the interconnection of the policy and the MDP.

To emphasize the dependence of the probability distribution $\mathbb{P}$ on μ and π, we will often use $\mathbb{P}_\mu^\pi$, but we will also take the liberty to drop any of these indices when its identity can be uniquely deduced from the context. When needed, the expectation operator corresponding to $\mathbb{P}$ (or $\mathbb{P}_\mu^\pi$) will be denoted by $\mathbb{E}$ (respectively, $\mathbb{E}_\mu^\pi$).

What is the probability assigned to a trajectory $\tau = (s_0, a_0, s_1, a_1, \dots) \in T$? Let $h_t = (s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)$. Recall that $H_t = (S_0, A_0, \dots, S_{t-1}, A_{t-1}, S_t)$. By a repeated application of the chain rule of probabilities, we get

$$
\begin{aligned}
&\mathbb{P}(H_t = h_t) \\
&\quad = \mathbb{P}(S_0 = s_0, A_0 = a_0, S_1 = s_1, \dots, S_t = s_t) \\
&\quad = \mathbb{P}(S_t = s_t | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}) \mathbb{P}(H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}) \\
&\quad = P_{a_{t-1}}(s_{t-1}, s_t) \mathbb{P}(H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}) && \text{(by (1))} \\
&\quad = P_{a_{t-1}}(s_{t-1}, s_t) \mathbb{P}(A_{t-1} = a_{t-1} | H_{t-1} = h_{t-1}) \mathbb{P}(H_{t-1} = h_{t-1}) \\
&\quad = P_{a_{t-1}}(s_{t-1}, s_t) \pi_{t-1}(a_{t-1} | h_{t-1}) \mathbb{P}(H_{t-1} = h_{t-1}) && \text{(by (2))} \\
&\quad \vdots \\
&\quad = P_{a_{t-1}}(s_{t-1}, s_t) \pi_{t-1}(a_{t-1} | h_{t-1}) \times \dots \times P_{a_0}(s_0, s_1) \pi_0(a_0 | s_0) \mathbb{P}(S_0 = s_0) \\
&\quad = P_{a_{t-1}}(s_{t-1}, s_t) \pi_{t-1}(a_{t-1} | h_{t-1}) \times \dots \times P_{a_0}(s_0, s_1) \pi_0(a_0 | s_0) \mu(s_0) \,. && \text{(by (3))}
\end{aligned}
$$

Collecting the terms,

$$
\mathbb{P}(H_t = h_t) = \mu_0(s_0) \left\{ \Pi_{i=0}^{t-1} \pi_i(a_i | h_i) \right\} \left\{ \Pi_{i=0}^{t-1} P_{a_i}(s_i, s_{i+1}) \right\} .
$$

Similarly,

$$
\mathbb{P}(H_t = h_t, A_t = a_t) = \mu_0(s_0) \left\{ \Pi_{i=0}^{t} \pi_i(a_i | h_i) \right\} \left\{ \Pi_{i=0}^{t-1} P_{a_i}(s_i, s_{i+1}) \right\} .
$$

Value Functions, the Optimal Value Function and the Objective

The total expected discounted reward, or the expected return of policy π in MDP M when the initial state is sampled from μ is

$$
v^\pi(\mu) = \mathbb{E}_\mu^\pi \left[R \right] .
$$

When $\mu = \delta_s$ where δ_s is the "Dirac" probability distribution that puts a point mass at s, we use $v^\pi(s)$ to denote the resulting value. Since this assigns a value to every state, v^π can be viewed as a function assigning a value to every state in $\mathcal{S}$. This function will be called the *value function* of policy π. When the dependence on the MDP is important, we may add "in MDP M" and denote the dependence by introducing an index: v^π_M.

The best possible value in state $s \in \mathcal{S}$ that can be obtained by optimizing over all possible policies is

$$
v^*(s) = \sup_\pi v^\pi(s) .
$$

Then, $v^* : \mathcal{S} \to \mathbb{R}$, viewed as a function, is called the *optimal value function*. A policy is *optimal* in state s if $v^\pi(s) = v^*(s)$. A policy is *uniformly optimal* if it is optimal in every state. In what follows, we will drop uniformly as we will usually be interested in finding uniformly optimal policies.

Given an MDP, we are interested in *efficiently computing* an optimal policy.

Planning=Computation

Computing an optimal policy can be seen as a planning problem: the optimal policy answers the question of how to take actions so that the expected return is maximized. This is also an *algorithmic* problem. The *input*, in the simplest case, is a big table (or a number of tables) that describes the transition probabilities and rewards. The interest is to develop algorithms that read in this table and then as *output* should return a description of an optimal policy. At this stage, it may seem unlikely that an efficient algorithm could do this: in the above unrestricted form, policies have an infinite description. As we shall find out soon though, we will be lucky with finite MDPs as in such MDPs one can always find optimal policies that have a short description. Then, the algorithmic question becomes interesting!

As for any algorithmic problem, the main question is how many elementary computational steps are necessary to solve an MDP? As can be suspected, the number of steps will need to scale with the number of states and actions. Indeed, even the size of the input scales with these. If computation indeed needs to scale with the number of state-action pairs, is there still any reason to consider this problem given that the number of states and actions in MDPs that one typically encounters in practical problems is astronomically large, if not infinite? Yes, there are:

- Not all MDPs are in fact large and it may be useful to know what it takes to "solve" a small MDP. Good solvers for "small" MDPs may serve as benchmarks for solvers developed for the "large MDP" case.
- Even if a problem is large (or infinite), one may be able to approximate it well with a small MDP. Then, a solver for a small MDP may be useful.
- Some ideas and tools developed for this problem also generalize (perhaps) with some twists to the "large" MDP setting.

At this stage, the reader may be wondering about what is meant by "small" and "large"? As a rough guideline, by "small" we mean problems where the tables describing the MDP (and/or policy) comfortably fit in the memory of whatever computer one has access to. Large is everything else.

Miscellaneous Remarks

Probabilities of infinite long trajectories?

Based on the above calculations, one expects that the probability of a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$ that never ends is

$$\begin{aligned}\mathbb{P}(S_0 = s_0, A_0 = a_0, S_1 = s_1, A_1 = a_1, \dots) = \mu(s_0) &\times \pi_0(a_0|h_0) \times P_{a_0}(s_0, s_1)\\ &\times \pi_1(a_1|h_1) \times P_{a_1}(s_1, s_2)\\ &\times \cdots\\ &\times \pi_t(a_t|h_t) \times P_{a_t}(s_t, s_{t+1})\\ &\times \cdots\end{aligned}$$

where $h_t = (s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)$ as before. However, this does not work: if in the trajectory, each action is taken with probability $1/2$ by the policy on the given history, the infinite product on the right-hand side is zero! This should make one pause at least for a moment: how is then $\mathbb{P}$ even *defined*? Does this distribution even exist? If yes, and it assigns zero probability to trajectories like above, could not it be that it assigns zero to all the trajectories of infinite length? In the world of infinite, one must tread carefully! The way out of this conundrum is that we must use *measure theoretic* probabilities, or we need to give up on objects like the return, $R = \sum_{t\ge 0} \gamma^t r_{A_t}(S_t)$, which is defined on trajectories of infinite length. The alternative to measure theoretical probability is to define everything through by taking limits (and always taking expectations over finite-length prefixes of the infinite long trajectories). As this would be quite cumbersome, we will take the measure-theoretic route, which will be explained in the next lecture.

Why Markov?

Equation (1) tells us that the only thing that matters from the history of the process as far as the prediction of the next state is concerned is the last action and the last state. This is known as the Markov property. More generally, Markov chains, which are specific stochastic processes, have a similar property.

Bellman's curse of dimensionality

Richard Bellman, who has made many foundational contributions to the early theory, coined the term the "curse of dimensionality". By this, Bellman meant the following: oftentimes when MDPs are used to model a practical decision making problem, the state space oftentimes takes the product form $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_d$ with some $d > 0$. If each set $\mathcal{S}_i$ here has at only two(!) elements, the state space will have at least 2^d elements. This is an exponential growth as a function of d, which is taken as the fundamental scaling quantity. Thus, any algorithm that needs to even just *enumerate* the states in the state space is "cursed" to perform a very lengthy calculation. While we start with considering the case when both the state and the action space are small (as described above), the main focus will be on the case when this is not true anymore. In this way, the problem will be to figure out ways of *breaking the curse*. But just to make things clear, in the worst-case, there is no cure to this curse, as we shall see it soon in a rigorous fashion. Any cure will come by changing the problem, either by changing the objective, or by changing the inputs available, or both.

Actions shared across states?

We described MDPs as if the same set of actions was available in all the states. This may create the (false) impression that action a_1 in state s_1 has something to do with action a_1 in state s_2 (i.e., their rewards, or next state distributions are shared or are similar). Given the MDP definition though, clearly, no such assumptions are made.

In a way, a better way of describing an MDP is using a set Z and an equivalence relation over Z, or, equivalently, the partition induced by it over Z. We should think of Z as the set of possible state-action pairs: The equivalence relation over Z then gives which of these share a common state. Alternatively, if z_1 and z_2 are in the same partition, they share a state, which we can identify with the partition. Then, for every $z \in Z$, the MDP would specify a distribution over the parts of the partition (the "next states") and one should specify a reward. While this description is appealing from a mathematical perspective, it is nonstandard and would make it harder to relate everything to the literature. Furthermore, the description chosen, apart from the inconvenience that one need to forcefully remember that actions do not keep their identity across states, is quite intuitive and compact.

A common variation in the literature, which avoids the "sharing issue" is to assume that every state is equipped with a set $\mathcal{A}(s)$ of actions admissible to the state and these sets are disjoint across the states. This description allows the number of actions to be varied across the states. While this has a minor advantage, our notation is simpler and tends not to lose much in comparison to these more sophisticated alternatives.

Are states observed?

In many practical problems it is not a priori clear whether the problem has a good approximate description as an MDP. One critical aspect that is missing from the MDP description is that the states of the MDP may not be available for measurement and thus the control (the choice of the action) cannot use state information. For now, we push this problem aside, but we shall return to it time-to-time. The reason is that it is best to start with the simpler questions and, at least intuitively, the problem of finding a policy that can use state information feels easier than finding one that cannot even access the state information. First, at least, we should find out what can be done in this case (and how efficiently), hoping that the more complex cases will either be reducible to this case, or will share some common patterns.

On the notation

Why use $r_a(s)$ rather than, say, $r(s,a)$? Or $P_a(s)$, or $P_a(s,s')$ rather than $P(s'|s,a)$? All these notations have pros and cons. None of them is ideal for all purposes. One explanation for using this notation is that later we will replace a with π, where π will be a special policy (a memoryless, or stationary Markov policy). When doing so, the notation of r_π (suppressing s) and P_π (a stochastic matrix!) will be tremendously useful.

A bigger question is why use s for states and a for actions. Is not the answer in the words? Well, people working in control would disagree. They would prefer to use x for state and u for actions, and I am told by Maxim Raginsky, that these come from Russian abbreviations, so they make at least as much sense as the notation used here. That is, if one speaks Russian (and if not, why not learn it?). Dimitri Bertsekas likes using i, j etc. for states, which seems fine if one has discrete (countable) state spaces.

Stochastic rewards

Some authors (e.g., this author in some of their papers or even in his book) considers rewards which are stochastic. This may matter when the problem is to learn a good policy, or to find a good plan while interacting with a stochastic simulator. However, when it comes to defining the object of computation, we can safely ignore (well-behaved) stochastic rewards. Here, the well-behaved stochastic rewards are those whose conditional expectation given an arbitrary history up to a state s and an action a taken in that state depends only on (s, a). Which is what we start here from.

References

"The" book about MDPs is:

Puterman, Martin L. 2005. Markov Decision Processes (Discrete Stochastic Dynamic Programming). Wiley-Interscience.

2. The Fundamental Theorem

PDF Version

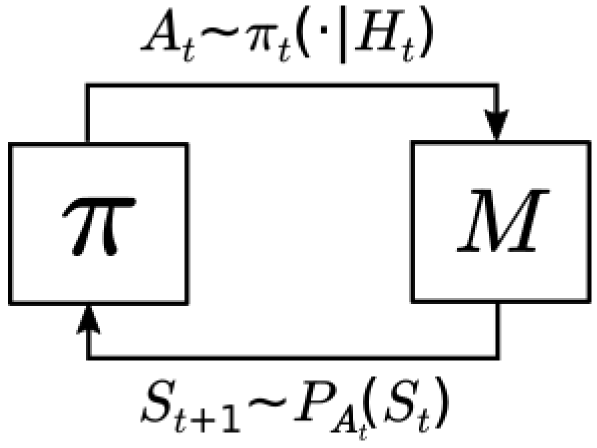
We start by recapping the definition of MDPs and then firm up the loose ends from the previous lecture: why do the probability distributions $\mathbb{P}_\mu^\pi$ exist and how are they defined? We then continue with the introduction of what we call the Fundamental Theorem of Dynamic Programming and end with the discussion of value iteration.

Introduction

A **Markov decision Process (MDP)** is a 5-tuple $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $P = (P_a(s))_{s,a}$ collects the next state distributions for each state-action pair (to represent the transition dynamics), $r = (r_a(s))_{s,a}$ gives the immediate rewards incurred for taking a given action in a given state, and $0 \le \gamma < 1$ is the discount factor. As said before, for simplicity, the state set $\mathcal{S}$ and the action set $\mathcal{A}$ are finite.

A **policy** $\pi = (\pi_t)_{t \ge 0}$ is an infinite long sequence where for each $t \ge 0$, $\pi_t : (\mathcal{S} \times \mathcal{A})^{t-1} \times \mathcal{S} \to \mathcal{M}_1(\mathcal{A})$ assigns a probability distribution to histories of length t. (For $\rho \ge 0$ we use $\mathcal{M}_\rho(X)$ to denote the set of nonnegative measures μ over X that satisfy $\mu(X) = \rho$.) Following a policy in an MDP means that the distribution of the actions in each time step $t \ge 0$ will follow what is prescribed by the policy for whatever the history is at that time.

When a policy is used in an MDP, the **interconnection** of the policy and the MDP, together with a start-state distribution, results in a distribution $\mathbb{P}$ such that for the infinite long sequence of state-action pairs $S_0, A_0, S_1, A_1, \dots, S_0 \sim \mu(\cdot)$, $A_t \sim \pi_t(\cdot|H_t)$, and $S_{t+1} \sim P_{A_t}(S_t)$ for all $t \ge 0$ where $H_t = (S_0, A_0, \dots, S_{t-1}, A_{t-1}, S_t)$ is the history at time step t. This closed loop interaction (or interconnection) of the policy and the MDP is shown in the figure below.



Probabilities over Trajectories

One loose end from the previous lecture was the existence of the probability measures $\mathbb{P}_\mu^\pi$. For this, we have the following result:

Theorem (existence theorem): Fix a finite MDP M with state space $\mathcal{S}$ and action space $\mathcal{A}$. Then there exists a measurable space $(\Omega, \mathcal{F})$ and a sequence of random elements $S_0, A_0, S_1, A_1, \dots$ over this space, $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$ for $t \geq 0$, such that for any policy $\pi = (\pi_t)_{t \geq 0}$ of the MDP M and any probability measure $\mu \in \mathcal{M}_1(\mathcal{S})$ over $\mathcal{S}$, there exists a probability measure $\mathbb{P}(= \mathbb{P}_\mu^\pi)$ over $(\Omega, \mathcal{F})$ satisfying the following properties:

1. $\mathbb{P}(S_0 = s) = \mu(s)$ for all $s \in \mathcal{S}$,
2. $\mathbb{P}(A_t = a|H_t) = \pi_t(a|H_t)$ for all $a \in \mathcal{A}, t \geq 0$, and
3. $\mathbb{P}(S_{t+1} = s'|H_t, A_t) = P_{A_t}(S_t, s')$ for all $s' \in \mathcal{S}$.

Furthermore, **uniqueness** holds in the following sense: if $(\tilde{\Omega}, \tilde{\mathcal{F}})$ together with $\tilde{S}_0, \tilde{A}_0, \tilde{S}_1, \tilde{A}_1, \dots$ also satisfy the conditions of the definition with $\tilde{\mathbb{P}}_\mu^\pi$ denoting the associated probability measures for specific choices of (π, μ) then for any π, μ, the joint distribution of $S_0, A_0, S_1, A_1, \dots$ under $\mathbb{P}_\mu^\pi$ and that of $\tilde{S}_0, \tilde{A}_0, \tilde{S}_1, \tilde{A}_1, \dots$ under $\tilde{\mathbb{P}}_\mu^\pi$ are identical.

Proof: Use the Ionescu-Tulcea theorem (Theorem 3.3 in the "bandit book", though the theorem statement there is weaker in that the uniqueness property is left out). ■

Property 3 above is known as the **Markov property** and is how MDPs derive their name. Note that implicit in the statement of this result is that $\mathcal{S}$ and $\mathcal{A}$ are endowed with the discrete σ-algebra. This is because we want both $S_t = s$ and $A_t = a$ to be events for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$ (these appear in the conditions underlying properties 1-3).

Note that the result does not point to any singular measurable space. Indeed, there are many ways to choose $(\Omega, \mathcal{F})$. However, as long as we are only concerned with properties of the distributions of state-action trajectories, thanks to the uniqueness part of the theorem, no ambiguity will arise from this. As a result, in general, we will not care about the choice of $(\Omega, \mathcal{F})$: Any choice as given in the theorem will work. However, for some proofs, it will be convenient to choose $(\mathcal{S} \times \mathcal{A})^{\mathbb{N}}$, the set of infinite long trajectories as Ω, while setting $S_t((s_0, a_0, s_1, a_1, \dots)) = s_t$, $A_t((s_0, a_0, s_1, a_1, \dots)) = a_t$ $(t \geq 0)$ and choosing $\mathcal{F} = (2^{\mathcal{S}\times\mathcal{A}})^{\otimes \mathbb{N}}$, which the smallest σ algebra that makes (S_t, A_t) measurable for any $t \geq 0$. We will call the resulting probability space the **canonical probability space** underlying the MDP.

Optimality and Some Notation

As usual, we use $\mathbb{E}$ to denote the expectation operator underlying a probability measure $\mathbb{P}$. When the dependence on μ or π is important, we use $\mathbb{E}_\mu^\pi$. We may drop any of these, when the dropped quantity is clear from the context. We will pay special attention to start state distributions concentrated on a single state. When this is state s, the distribution is denoted by δ_s: this is the well-known Dirac distribution with an atom at s. The reason we pay special attention to these is because these in a way form the basis of all start state distributions (and in fact quantities that depend linearly on start state distributions). We will use the shorthand $\mathbb{P}_s^\pi$ for $\mathbb{P}_{\delta_s}^\pi$. Similarly, we use $\mathbb{E}_s^\pi$ for $\mathbb{E}_{\delta_s}^\pi$.

Define the return over a trajectory $\tau = (S_0, A_0, S_1, A_1, \dots)$ as

$$R = \sum_{t=0}^{\infty} \gamma^t r_{A_t}(S_t).$$

The value function v^π of policy π maps states to values and in particular for a state $s \in \mathcal{S}$, $v^\pi(s)$ is defined via $v^\pi(s) = \mathbb{E}_s^\pi[R]$: This is the expected return under the distribution induced by the interconnection of policy π and the MDP when the start state is s. Note that $v^\pi(s)$ is well-defined. This is because it is the expectation of a quantity that is a function of the trajectory τ; for an explanation see the end-notes.

The **standard goal** in an MDP is to identify a policy that maximizes this value **in every state**. A policy achieving this is known as an **optimal policy**. Whether an optimal policy exists at all is not clear at this stage. In any case, if it exist, an optimal policy must satisfy $v^\pi = v^*$ where $v^* : \mathcal{S} \to \mathbb{R}$ is defined by

$$v^*(s) = \sup_\pi v^\pi(s)\,, \qquad s \in \mathcal{S}\,.$$

By the definition of the optimal value function, we have $v^\pi(s) \le v^*(s)$ for all $s \in \mathcal{S}$ and any policy π. We also use $v^\pi \le v^*$ to express this. In general, $f \le g$ for two functions f, g that are defined over the same domain and take values (say) in the reals, if $f(z) \le g(z)$ holds for all the possible elements z of their common domain. We similarly define $f \ge g$.

We will also identify functions with vectors and allow vector-space operations on them. All vectors, unless otherwise stated, are column vectors. The symbol $\boldsymbol{1}$ is defined as a vector of ones. The length of this vector can change depending on the context. In this lecture, it will be S-dimensional. This symbol will be very useful in a number of calculations. We start with a definition that uses it.

Approximately optimal policies

Let $\varepsilon > 0$. A policy π is said to be ε-optimal if

$$v^\pi \ge v^* - \varepsilon \boldsymbol{1}\,.$$

Finding an ε-optimal policy with a positive ε should intuitively be easier than finding an optimal policy.

Memoryless Policies (ML)

If optimal policies would need to remember the past of arbitrary length, it would be hopeless to search for efficient algorithms that can compute them as even describing them could take infinite time. Luckily, this is not the case. In finite MDPs, it will turn out to be sufficient to consider policies that use only the most recent state without losing optimality: this is the subject of the **fundamental theorem of MDPs**, which we will give shortly. We call the policies that take only the most recent state into account **memoryless**.

Formally, a memoryless policy can be identified with a map from the states to probability distributions over the actions: $m : \mathcal{S} \to \mathcal{M}_1(\mathcal{A})$. Given m, the memoryless policy, using our previous policy notation, is $\pi_t(a|s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t) = m(a|s_t)$, where we abuse notation by using $m(a|s_t)$ in place of $m(s_t)(a)$. Thus, as expected, the policy itself "forgets" the past and just uses the most recent state in assigning probabilities to the individual actions. Under a distribution induced by interconnecting a memoryless policy with an MDP, the sequence of state-action pairs forms a **Markov chain**.

In what follows, by abusing notation further, when it comes to a memoryless policy, we will identify π with m and will just write $\pi : \mathcal{S} \to \mathcal{M}_1(\mathcal{A})$.

For building up to the proof of the fundamental theorem, we start with the concept of discounted occupancy measures.

(Discounted) Occupancy Measure

Given a start state distribution $\mu \in \mathcal{M}_1(\mathcal{S})$ and a policy π, the (discounted) occupancy measure $\nu_\mu^\pi \in \mathcal{M}_{1/(1-\gamma)}(\mathcal{S} \times \mathcal{A})$ induced by μ and π and the underlying MDP M is defined as

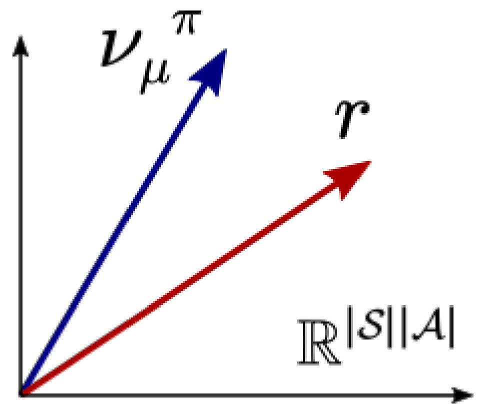
$$\nu_\mu^\pi(s,a) = \sum_{t=0}^\infty \gamma^t \mathbb{P}_\mu^\pi(S_t = s, A_t = a).$$

Interestingly, the value function can be represented as an inner product between the immediate reward function r and the occupancy measure ν_μ^π:

$$\begin{aligned} v^\pi(\mu) &= \mathbb{E}_\mu^\pi \left[\sum_{t=0}^\infty \gamma^t r_{A_t}(S_t) \right] \\ &= \sum_{s,a} \sum_{t=0}^\infty \gamma^t \mathbb{E}_\mu^\pi \left[r_{A_t}(S_t) \mathbb{I}(S_t = s, A_t = a) \right] \\ &= \sum_{s,a} r_a(s) \sum_{t=0}^\infty \gamma^t \mathbb{E}_\mu^\pi \left[\mathbb{I}(S_t = s, A_t = a) \right] \\ &= \sum_{s,a} r_a(s) \sum_{t=0}^\infty \gamma^t \mathbb{P}_\mu^\pi(S_t = s, A_t = a) \\ &= \sum_{s,a} r_a(s) \nu_\mu^\pi(s,a) \\ &=: \langle \nu_\mu^\pi, r \rangle, \end{aligned}$$

where $\mathbb{I}(S_t = s, A_t = a)$ is the indicator of the event $S_t = s, A_t = a$, which gives the value of one when the event holds (i.e., $S_t = s$ and $A_t = a$), and gives zero otherwise. That the summation over (s,a) can be moved outside of the expectation in the first equality follows because expectations are linear. That the infinite sum can be moved outside is more subtle: this follows from Lebesgue's dominated convergence theorem. See, for example, Chapter 2 of Lattimore & Szepesvári (2020).

With the above equation, we see that the problem of maximizing the expected reward for a given initial distribution is the same as choosing a policy that "stirs" the occupancy measure to maximally align with the reward vector r. A better alignment will result in a higher value for the policy. This is depicted in the figure below.



A key step in proving the sufficiency of memoryless policies for optimal control is the following result:

Theorem: For any policy π and a start state distribution $\mu \in \mathcal{M}_1(\mathcal{S})$, there exists a memoryless policy π' such that

$$\nu_\mu^{\pi'} = \nu_\mu^\pi.$$

Proof (hint): First define the occupancy measure over the state space $\tilde{\nu}_\mu^\pi(s) := \sum_a \nu_\mu^\pi(s,a)$. Then show that the theorem statement holds for the policy π' defined as follows:

$$\pi'(a|s) = \begin{cases} \frac{\nu_\mu^\pi(s,a)}{\tilde{\nu}_\mu^\pi(s)} & \text{if } \tilde{\nu}_\mu^\pi(s) \neq 0 \\ \pi_0(a) & \text{otherwise,} \end{cases}$$

where $\pi_0(a) \in \mathcal{M}_1(\mathcal{A})$ is an arbitrary distribution. To do this, expand $\tilde{\nu}_\mu^\pi$ using the definition of discounted occupancy measures and use algebra.

■

Note that it is crucial that the memoryless policy obtained depends on the start state distribution: The reader should try to convince themselves that there are non-memoryless policies whose value function cannot be reproduced by a memoryless policy at every state.

Bellman Operators, Contractions

The last definitions and results that we need before stating the fundamental theorem concern what are known as **Bellman operators**.

Fix a memoryless policy π. Recall that S is the cardinality (size) of $\mathcal{S}$. First, define $r_\pi(s) = \sum_a \pi(a|s) r_a(s)$ to be the expected reward under policy π for a given state s. Again, we overload the notation and let $r_\pi \in \mathbb{R}^{\mathrm{S}}$ denote a vector whose sth element $(r_\pi)_s = r_\pi(s)$. Similarly, we define $P_\pi(s,s') := \sum_a \pi(a|s) P_a(s,s')$ and let $P_\pi \in [0,1]^{\mathrm{S}\times\mathrm{S}}$ denote the stochastic transition matrix where the element in the sth row and s'th column $(P_\pi)_{s,s'} = P_\pi(s,s')$. Note that each row of P_π sums to one:

$$P_\pi \mathbf{1} = \mathbf{1}\,.$$

The **Bellman/policy evaluation operator** underlying π, $T_\pi : \mathbb{R}^{\mathrm{S}} \to \mathbb{R}^{\mathrm{S}}$, is defined as

$$\begin{aligned} T_\pi v(s) &= \sum_a \pi(a|s) \left\{ r_a(s) + \gamma \sum_{s'} P_a(s,s') v(s') \right\} \\ &= \sum_a \pi(a|s) \left\{ r_a(s) + \gamma \langle P_a(s), v \rangle \right\} \end{aligned}$$

or, in short,

$$T_\pi v = r_\pi + \gamma P_\pi v,$$

where $v \in \mathbb{R}^{\mathrm{S}}$. The Bellman operator performs a one-step lookahead (also called a Bellman lookahead) on the value function. We will use the notations $(T_\pi(v))(s)$, $T_\pi v(s)$, and $(T_\pi v)_s$ interchangeably. T_π is also known as the policy evaluation operator for the policy π.

The **Bellman optimality operator** $T : \mathbb{R}^{\mathrm{S}} \to \mathbb{R}^{\mathrm{S}}$ is defined as

$$Tv(s) = \max_a \{ r_a(s) + \gamma \langle P_a(s), v \rangle \}.$$

We use $\|\cdot\|_\infty$ to denote the **maximum-norm**: $\|v\|_\infty = \max_i |v_i|$. The maximum-norm is a “good friend” of the operators we just defined. This is because stochastic matrices, viewed as operators and "maximizing" are "good friends" of this norm. All this results in the following proposition:

Proposition (γ-contraction of the Bellman Operators): Given any two vectors $u, v \in \mathbb{R}^{\mathrm{S}}$ and any memoryless policy π,

1. $\|T_\pi u - T_\pi v\|_\infty \le \gamma \|u - v\|_\infty$, and
2. $\|Tu - Tv\|_\infty \le \gamma \|u - v\|_\infty$.

The proposition can be proved by elementary algebra and the complete proof can be found in Appendix A.2 of Szepesvári (2010).

For action $a \in \mathcal{A}$, we will find it useful to also define the operator $T_a : \mathbb{R}^{\mathrm{S}} \to \mathbb{R}^{\mathrm{S}}$ which matches T_π with the memoryless policy which in *every* state chooses action a. Of course, this operator, being a special case, satisfies the above contraction property as well. This can be seen as performing a one-step lookahead with a fixed action.

From Banach's fixed point theorem, we get the following corollary:

Proposition (Fixed-point iteration): Given any $u \in \mathbb{R}^{\mathrm{S}}$ and any memoryless policy π,

1. $v^\pi = \lim_{k\to\infty} T_\pi^k u$ and in particular for any $k \ge 0$, $\|v^\pi - T_\pi^k u\|_\infty \le \gamma^k \|u - v^\pi\|_\infty$ where v^π is the unique vector/function that satisfies $T_\pi v^\pi = v^\pi$;
2. $v_\infty = \lim_{k\to\infty} T^k u$ is well-defined and in particular for any $k \ge 0$, $\|v_\infty - T^k u\|_\infty \le \gamma^k \|u - v_\infty\|_\infty$. Furthermore, v_∞ is the unique vector/function that satisfies $Tv_\infty = v_\infty$.

The Fundamental Theorem

Definition: A memoryless policy π is greedy w.r.t. to a value function $v : \mathcal{S} \to \mathbb{R}$ if in every state $s \in \mathcal{S}$, with probability one π chooses actions that maximize $(T_a v)(s) = r_a(s) + \gamma \langle P_a(s), v \rangle$.

Note that there can be more than one action that maximizes the (one-step) *Bellman lookahead* $(T_a v)(s)$ at any given state (in case there are ties). In fact, ties can be extremely common: Just imagine "duplicating an action" in every state (i.e., the new action has the same associated transitions and rewards as the copied one). If the copied one was maximizing the Bellman lookahead at some state, the new action will do the same. Because we have finitely many actions, a maximizing action always exist. Thus, we can always "take" a greedy policy w.r.t. any $v \in \mathbb{R}^{\mathrm{S}}$.

Proposition (Characterizing greedyness): A memoryless policy π is greedy w.r.t. $v \in \mathbb{R}^{\mathrm{S}}$ if and only if

$$T_\pi v = T v\,.$$

With this, we are ready to state what I call the Fundamental Theorem of MDPs:

Theorem (Fundamental Theorem of MDPs): The following hold true in any finite MDP:

1. Any policy π that is greedy with respect to v^* is optimal: $v^\pi = v^*$;
2. It holds that $v^* = T v^*$.

The equation $v = Tv$ is known as the **Bellman optimality equation** and the second part of the result can be stated in words by saying that the optimal value function satisfies the Bellman optimality equation. Also, our previous proposition on fixed-point iteration, where we already came across the Bellman optimality equation, foreshadows a way of approximately computing v^* that we will get back to after the proof.

Proof: The proof would be easy if we only considered memoryless policies when defining v^*. In particular, letting ML stand for the set of memoryless policies of the given MDP, define

$$\tilde{v}^*(s) = \sup_{\pi \in \mathrm{ML}} v^\pi(s) \quad \text{for all } s \in \mathcal{S}\,.$$

As we shall see soon, it is not hard to show the theorem just with v^* replaced everywhere with $\tilde{v}^*$. That is:

1. Any policy π that is greedy with respect to $\tilde{v}^*$ satisfies $v^\pi = \tilde{v}^*$;
2. It holds that $\tilde{v}^* = T\tilde{v}^*$.

This is what we will show in Part 1 of the proof, while in Part 2 we will show that $\tilde{v}^* = v^*$. Clearly, the two parts together establish the desired result.

Part 1: The idea of the proof is to first show that

$$\tilde{v}^* \le T\tilde{v}^* \tag{1}$$

and then show that for any greedy policy π, $v^\pi \ge \tilde{v}^*$.

The displayed equation follows by noticing that $v^\pi \le \tilde{v}^*$ holds for all memoryless policies π by definition. Applying T_π on both sides, using $v^\pi = T_\pi v^\pi$, we get $v^\pi \le T_\pi \tilde{v}^*$. Taking the supremum of both sides over π and noticing that $Tv = \sup_{\pi \in \mathrm{ML}} T_\pi v$ for any v, together with the definition of $\tilde{v}^*$ gives (1).

Now, take any memoryless policy π that is greedy w.r.t. $\tilde{v}^*$. Thus, $T_\pi \tilde{v}^* = T\tilde{v}^*$.

Combined with (1), we get

$$T_\pi \tilde{v}^* \ge \tilde{v}^* \,. \tag{2}$$

Applying T_π on both sides and noticing that T_π keeps the inequality intact (i.e., for any u, v such that $u \le v$ we get $T_\pi u \le T_\pi v$), we get

$$T_\pi^2 \tilde{v}^* \ge T_\pi \tilde{v}^* \ge \tilde{v}^* \,,$$

where the last inequality follows from (2). With the same reasoning we get that for any $k \ge 0$,

$$T_\pi^k \tilde{v}^* \ge T_\pi^{k-1} \tilde{v}^* \ge \dots \ge \tilde{v}^* \,,$$

Now, by our proposition, the fixed-point iteration $T_\pi^k \tilde{v}^*$ converges to v^π. Hence, taking the limit above, we get

$$v^\pi \ge \tilde{v}^*.$$

This, together with $v^\pi \le \tilde{v}^*$ shows that $v^\pi = \tilde{v}^*$.

Finally, $T\tilde{v}^* = T_\pi \tilde{v}^* = T_\pi v^\pi = v^\pi = \tilde{v}^*$.

Part 2: It remains to be shown that $\tilde{v}^* = v^*$. Let Π be the set of all policies. Because $\mathrm{ML} \subset \Pi$, $\tilde{v}^* \le v^*$. Thus, it remains to show that

$$v^* \le \tilde{v}^* \,. \tag{3}$$

To show this, we will use the theorem that guaranteed that for any state-distribution μ and policy π (memoryless or not) we can find a memoryless policy, which we will call for now $\mathrm{ML}(\pi)$, such that $\nu_\mu^\pi = \nu_\mu^{\mathrm{ML}}$. Fix a state $s \in \mathcal{S}$. Applying this result with $\mu = \delta_s$, we get

$$\begin{aligned} v^\pi(s) &= \langle \nu_s^\pi, r \rangle \\ &= \langle \nu_s^{\mathrm{ML}(\pi)}, r \rangle \\ &\le \sup_{\pi' \in \mathrm{ML}} \langle \nu_s^{\pi'}, r \rangle \\ &= \sup_{\pi' \in \mathrm{ML}} v^{\pi'}(s) = \tilde{v}^*(s) \,. \end{aligned}$$

Taking the supremum of both sides over π, we get $v^*(s) = \sup_{\pi \in \Pi} v^\pi(s) \le \tilde{v}^*(s)$. Since $s \in \mathcal{S}$ was arbitrary, we get $v^* \le \tilde{v}^*$, finishing the proof. $\blacksquare$

A property that came up during the proof that we will repeatedly use is that T_π is monotone as an operator. The same holds for T. For the record, we state these as a proposition:

Proposition (monotonicity of Bellman operators): For any memoryless policy π, $T_\pi u \le T_\pi v$ holds for any $u, v \in \mathbb{R}^{\mathrm{S}}$ such that $u \le v$. The same also holds for T, the Bellman optimality operator.

According to the Fundamental Theorem of MDPs, if we have access to the optimal value function v^*, then we can find the optimal policy in an efficient and effective way. We just have to greedify it w.r.t. to the value function: (abusing the policy notation) $\pi(s) = \arg\max_{a \in \mathcal{A}} \{ r_a(s) + \gamma \langle P_a(s), v^* \rangle \} \quad \forall s \in \mathcal{S}$. Such a greedy policy can be found in $O(\mathrm{S}^2 \mathrm{A})$ time.

Hence, if we can efficiently find the optimal value function, we will get an efficient way of computing an optimal policy. This is to be contrasted with the *naive* approach to finding an optimal policy, which is to enlist all the policies and compare their value functions to find a policy whose value function dominates the value functions of all the other policies.

However, even if we restrict ourselves to just the set of deterministic policies, there are $\Theta(\mathrm{A}^{\mathrm{S}})$ such policies and thus this can be a costly procedure.

As it turns out, for finite MDPs, there is a way to calculate optimal policies in time that is polynomial in S, A, and $1/(1-\gamma)$, avoiding the exponential growth of the naive approach with the size of the state space. Algorithms that can do this belong to the family of **dynamic programming** algorithms. For our purposes, we call any algorithm a dynamic programming algorithm that uses the idea of keeping track of value of states (that is, uses value functions) while doing its calculations.

The Fundamental Theorem is somewhat surprising: how come that we can find policies whose value function dominates that of all other policies? In a way, the Fundamental Theorem tells us that the set of value functions of all policies in some MDP (as a set in $\mathbb{R}^{\mathrm{S}}$) is very special: It has a "vertex" which dominates all the other value functions. This is quite fascinating. Of course, the key was the Markov property as this gave us the tool to show the result that allowed us to switch from arbitrary policies to memoryless ones.

Value Iteration

By the Fundamental Theorem, v^* is the fixed point of T. By our earlier proposition, which built on the Banach's fixed point theorem, the sequence $\{T^k v\}_{k\ge 0}$ converges to v^* at a geometric rate. In the context of MDPs, the process of repeatedly applying T to some function is called **value iteration**. The initial function is usually taken to be the all-zero function, which we denote by $\mathbf{0}$, but, of course, if there is a better initial guess on v^*, that guess can also be used at initialization. The next result gives a bound on the number of iterations required to reach an ε-neighborhood (in the max-norm sense) of v^*:

Theorem (Value Iteration): Consider an MDP with immediate rewards in the $[0,1]$ interval. Pick an arbitrary positive number $\varepsilon > 0$. Let $v_0 = \mathbf{0}$ and set

$$v_{k+1} = T v_k \quad \text{for } k = 0, 1, 2, \ldots$$

Then, for $k \ge \ln(1/(\varepsilon(1-\gamma))/\ln(1/\gamma)$, $\|v_k - v^*\|_\infty \le \varepsilon$.

Before the proof recall that

$$H_{\gamma,\varepsilon} := \frac{\ln(1/(\varepsilon(1-\gamma)))}{1-\gamma} \geq \frac{\ln(1/(\varepsilon(1-\gamma)))}{\ln(1/\gamma)} \,.$$

Thus, the effective horizon, $H_{\gamma,\varepsilon}$, whom we met in the first lecture, appeared again. Of course, this is no coincidence.

Proof: By our assumptions on the rewards, $\mathbf{0} \leq v^\pi \leq \frac{1}{1-\gamma}\mathbf{1}$ holds for any policy π. Hence, $\|v^*\|_\infty \leq \frac{1}{1-\gamma}$ also holds. By our fixed-point iteration proposition, we get

$$\|v_k - v^*\|_\infty \leq \gamma^k \|v^* - \mathbf{0}\|_\infty = \gamma^k \|v^*\|_\infty \leq \frac{\gamma^k}{1-\gamma} \,.$$

Solving for the smallest k such that $\gamma^k/(1-\gamma) \leq \varepsilon$ gives the result.

■

For fixed $\gamma < 1$, note the mild dependence of the iteration complexity on the target accuracy ε: we can expect with only a handful iterations to get in a small vicinity of v^*. Note also that the total computation cost is $O(\mathrm{S}^2\mathrm{A}k)$ and the space required is at most $O(\mathrm{S})$, all assuming each value takes up $O(1)$ memory and arithmetic and logic operations also require $O(1)$ time.

Note that accuracy requirement was set up in the form of additive errors. If the value function v^* is of order $1/(1-\gamma)$ (the maximum possible order), a relative accuracy of order 2 means setting $\epsilon = 0.5/(1-\gamma)$, making the iteration complexity to be $\ln(2)/(1-\gamma)$. However, for controlling the relative error, the more interesting case is when v^* takes on small values. Here, we see that the complexity may grow unbounded. Later, we will see that in a way this lack of fine-grained error control of value iteration will mean that value iteration is not ideal for calculating exactly optimal policies.

Notes

Value functions are well-defined

As noted in the text, value functions are well-defined despite that the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is not uniquely defined. In fact, for any $f : (\mathcal{S} \times \mathcal{A})^{\mathbb{N}} \to \mathbb{R}$ (measurable) function and for any $(\Omega, \mathcal{F}, \mathbb{P})$ and $(\Omega', \mathcal{F}', \mathbb{P}')$ probability spaces, as long as both $\mathbb{P}$ and $\mathbb{P}'$ satisfy the requirements postulated in the existence theorem,

$\int f(\tau(\omega))\mathbb{P}(d\omega) = \int f(\tau(\omega))\mathbb{P}'(d\omega)$, or, introducing $\mathbb{E}$ ($\mathbb{E}'$) to denote the expectation operator underlying $\mathbb{P}$ (respectively, $\mathbb{P}'$), $\mathbb{E}[f(\tau)] = \mathbb{E}'[f(\tau)]$. It also follows that if we only need probabilities and expectations over trajectories, it suffices to choose $(\Omega, \mathcal{F}, \mathbb{P})$ as the canonical probability space induced by the state-action space of the MDP at hand.

Other types of MDPs

The obvious question is what survives of all this in other types of MDPs, such as finite-horizon homogenous or inhomogeneous, with or without discounting, total cost (i.e. negative rewards only), or of course the average cost setting? The story is that the arguments can be usually made to work, but this is not entirely automatic. The subject is well-studied and we will give some references and hints later, perhaps even answer some of these questions.

Infinite spaces anyone?

The first thing that changes when we switch to infinite spaces is that we cannot take the assumption that the immediate rewards are bounded for granted. This can cause quite a bit of trouble: v^π for some policies can be unbounded, and the same holds for v^*. Negative infinite values could be especially “hurtful”. (LQR control is the simplest example where this comes up.)

Another issue is that we cannot take the existence of greedy policies for granted. This happens already when the number of actions is infinite (what is the action that maximizes the reward $r_a(s) = 1 - 1/a$ where $a > 0$?). Oftentimes compactness of the action space and continuity assumptions help with this, though, as much of what we will do will be approximate, approximate greedification should be sufficient for most of the time. From this perspective, that greedy actions may not exist is just annoyance.

Finally, when either the state or action space is uncountably infinite, one has to be careful even with the definition of policies. Using a technical term from probability theory, a choice that makes thing work is to restrict policies to be probability kernels. Using this definition means that we need to put measurability structures over both the state and action spaces (this is only crucial when either respective set has a larger than countable cardinality). The main change here is that with policies defined this way, for any U measurable subset of $\mathcal{A}$, $h_t \mapsto \pi_t(U|h_t)$ must be measurable. This allows us then the use of the Ionescu-Tulcea theorem and at least the definitions can be made to work. The next difficulty in this case is that “greedification” may lead to outside of the set of these “measurable policies”, which could prevent the existence of optimal policies (again, if we are contend with approximate optimality, this difficulty disappears). There is a large literature concerned with these issues.

From infinite trajectories to their finite prefixes

Since trajectories are allowed to be infinitely long, we have a nonconstructive result only for the existence of the probability measures induced by the interconnection of policies and MDPs. Oftentimes we need to check whether two probability measures over these infinitely long trajectories coincide. How can this be done? A general result from measure theory says that two measures agree, if they agree of a generator of the underlying σ-algebra. A convenient generator system for the σ-algebra over the trajectories (for the canonical probability space) is the system whose elements take the form

$$\{s_0\} \times \{a_0\} \times \cdots \times \{s_t\} \times \mathcal{A} \times (\mathcal{S} \times \mathcal{A})^{\mathbb{N}}$$

and

$$\{s_0\} \times \{a_0\} \times \cdots \times \{s_t\} \times \{a_t\} \times (\mathcal{S} \times \mathcal{A})^{\mathbb{N}}$$

for some $s_0, a_0, \dots, s_t, a_t, \dots$. That is, if $\mathbb{P}$ and $\mathbb{P}'$ agree on the probabilities assigned to these sets, they agree everywehere. This makes things a full circle: what this result says is that we only need to check the probabilities assigned to finite prefixes of the infinitely long trajectories. Phew. Since the probabilities assigned to these finite prefixes are a function of μ, P and π alone, it follows that there is a **unique probability measure over the trajectory space** $(\mathcal{S} \times \mathcal{A})^{\mathbb{N}}$ that satisfies the requirements postulated in the existence theorem. That is, the canonical probability space is uniquely defined.

Optimization with (Discounted) Occupancy Measures

We learned that the value function can be represented as $v^\pi(\mu) = \sum_{s,a} r_a(s) \nu_\mu^\pi(s,a) = \langle \nu_\mu^\pi, r \rangle$. Thus, maximizing the value function for a given initial distribution μ is equivalent to maximizing the dot product between ν_μ^π and r. Next, we present a concrete example and point out some interesting results.

To keep this example as simple as possible, we introduce some new notation. Let $\mathcal{A}(s)$ represent the set of actions admissable to the state $s \in \mathcal{S}$. We now define the MDP. Let $\mathcal{S} = \{s_1, s_2\}$, $\mathcal{A}(s_1) = \{a_1, a_2\}$ and $\mathcal{A}(s_2) = \{a_3\}$. Also, let

$$\begin{aligned} P_{a_1}(s_1, s_1) &= 1, \quad r_{a_1}(s_1) = 1 \\ P_{a_2}(s_1, s_2) &= 1, \quad r_{a_2}(s_1) = 1/2 \\ P_{a_3}(s_2, s_2) &= 1, \quad r_{a_3}(s_2) = 1/2. \end{aligned}$$

Our policy π can be parametrized by one parameter p as

$$\begin{aligned}\pi(a_1|s_1) &= p\\ \pi(a_2|s_1) &= 1-p\\ \pi(a_3|s_2) &= 1.\end{aligned}$$

Finally, we assume $\mu(s_1)=1$.

We explicitly write out $\nu_\mu^\pi(s,a)=\sum_{t=0}^\infty \gamma^t \mathbb{P}_\mu^\pi(S_t=s,A_t=a)$ for all state-action pairs.

$$\begin{aligned}\nu_\mu^\pi(s_1,a_1) &= \sum_{t=0}^\infty \gamma^t p^{t+1}\\ &= p\sum_{t=0}^\infty (\gamma p)^t\\ &= \frac{p}{1-\gamma p}\\ \nu_\mu^\pi(s_1,a_2) &= \sum_{t=0}^\infty \gamma^t p^t(1-p)\\ &= (1-p)\sum_{t=0}^\infty(\gamma p)^t\\ &= \frac{1-p}{1-\gamma p}\\ \nu_\mu^\pi(s_2,a_3) &= \frac{1}{1-\gamma}-\frac{p}{1-\gamma p}-\frac{1-p}{1-\gamma p}\end{aligned}$$

Recall, our goal is to maximize $\sum_{s,a} r_a(s)\nu_\mu^\pi(s,a)$. To do this we plug in the above quantities for $r_a(s)$ and $\nu_\mu^\pi(s,a)$

$$\begin{aligned}\sum_{s,a} r_a(s)\nu_\mu^\pi(s,a) &= \frac{1-p}{1-\gamma p}+\frac{1}{2}\left(\frac{p}{1-\gamma p}\right)+\frac{1}{2}\left(\frac{1}{1-\gamma}-\frac{p}{1-\gamma p}-\frac{1-p}{1-\gamma p}\right)\\ &= \frac{1}{2}\left(\frac{p}{1-\gamma p}\right)+\frac{1}{2}\left(\frac{1}{1-\gamma}\right).\end{aligned}$$

Noting that the function on the right hand side is monotone increasing for $p\in[0,1]$, so we get that the above quantity is maximized for $p=1$.

Thus, the optimal policy is

$$\begin{aligned}\pi(a_1|s_1) &= 1\\ \pi(a_2|s_1) &= 0\\ \pi(a_3|s_2) &= 1.\end{aligned}$$

Which, aligns with our intuition that action a_1 should always be selected in state s_1 since it produces larger reward. Notice how the set of occupancy measures

$$\{(t, (1-\gamma t - t), 1/(1-\gamma) - t - (1-\gamma t - t)) : t \in [0, 1/(1-\gamma)]\}$$

is a convex set. This examples shows that optimizing in the space of occupancy measures could be a linear optimization while optimizing with a policy parametrization could be a non-linear optimization.

Fundamental Theorem

I think I have seen Bertsekas and Shreve call the theorem I call fundamental also by the same name. However, this is not quite a standard name. Nevertheless, the result is important and many other things follow from it. In a way, this is the result that is at the heart of all the theory. I think it deserves this name. I have probably read the proof presented here somewhere, but this was a while ago and the source escapes me. In the RL literature people often start with memoryless policies and work with $\tilde{v}^*$ rather than with v^*. The question whether $\tilde{v}^* = v^*$ is well-studied and understood, mostly in the control and operations research literature.

The geometry of the space of value functions

An alternative way of seeing the fundamental theorem is as a result concerning the geometry of the space of value functions. Indeed, fix an MDP M and let $\mathcal{V} = \{v^\pi \,:\, \pi \text{ is a policy of } M\}$, while let $\mathcal{V}^{\text{DET}} = \{v^\pi \,:\, \pi \text{ is a deterministic memoryless policy of } M\}$. The set $\mathcal{V}$ is the set of all value functions of M. Both sets are subsets of $\mathbb{R}^{\mathcal{S}}$. Using terminology from multicriteria optimization, the optimal value function, v^*, is the **ideal point** of $\mathcal{V}$: $v^*(s) = \sup\{v(s) \,:\, v \in \mathcal{V}\}$ for all $s \in \mathcal{S}$. Then, the fundamental theorem states that the ideal point of $\mathcal{V}$ belongs to $\mathcal{V}$: $v^* \in \mathcal{V}$ and in fact $v^* \in \mathcal{V}^{\text{DET}}$. However, more is known about $\mathcal{V}$:

Theorem (existence theorem): Fix a finite MDP M. Then $\mathcal{V} \subset \mathbb{R}^{\mathcal{S}}$ is convex. Furthermore, any extreme point of $\mathcal{V}$ belongs to $\mathcal{V}^{\text{DET}}$.

This result is due to Dadashi et al. (2019).

Banach's fixed point theorem

This theorem can be found in Appendix A.1 of my short RL book (Szepesvári, 2010). However, of course, it can be found in many places (the Wikipedia article is also OK). It is worthwhile to spend some time with this theorem to understand its conditions, going back to concepts like Cauchy-sequences (which should perhaps be called sequences with vanishing oscillations) and completeness of the set of real numbers.

References

The references mentioned before:

- Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.
- Szepesvári, C. (2010). Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning, 4(1), 1-103.

The next work (a book chpater) gives a concise yet relatively thorough introduction. The chapter also gives a proof of the fundamental theorem; through the sufficiency of Markov policies. This is done for the discounted and also for a number of alternate criteria.

- Garcia, Frédérick, and Emmanuel Rachelson. 2013. "Markov Decision Processes." In Markov Decision Processes in Artificial Intelligence, 1–38. Hoboken, NJ USA: John Wiley & Sons, Inc.

A summary of basic results for countable and Borel state-space, and Borel action spaces, with potentially unbounded (from below) reward functions can be found in the next (excellent) paper, which also gives a concise overview of the history of these results:

- Feinberg, Eugene A. 2011. Total Expected Discounted Reward MDPS: Existence of Optimal Policies. In Wiley Encyclopedia of Operations Research and Management Science. Hoboken, NJ, USA: John Wiley & Sons, Inc.

An argument showing the fundamental theorem for the finite-horizon case derived from a general result of David Blackwell can be found in a blog-post of Maxim Raginsky, who gives further pointers, most notable this. David Blackwell has contributed in numerous ways to the foundations of statistics, decision theory, probability theory, and many many other subjects and the importance of his work cannot be overstated.

- Robert Dadashi, Adrien Ali Taïga, Nicolas Le Roux, Dale Schuurmans, Marc G. Bellemare. 2019. The Value Function Polytope in Reinforcement Learning. ICML. arXiv

3. Value Iteration and Our First Lower Bound

PDF Version

Last time, we discussed the Fundamental Theorem of Dynamic Programming, which then led to the efficient "value iteration" algorithm for finding the optimal value function. And then we could find the optimal policy by greedifying w.r.t. the optimal value function. In this lecture we will do two things:

1. Elaborate more on the the properties of value iteration as a way of obtaining near-optimal policies;
2. Discuss the computational complexity of planning in finite MDPs.

Finding a Near-Optimal Policy using Value Iteration

In the previous lecture we found that the iterative computation that starts with some and then obtains using the "Bellman update"

leads to a sequence whose th term approaches , the optimal value function, at a geometric rate:

While this is reassuring, our primary goal is to obtain an optimal, or at least a near-optimal policy. Since any policy that is greedy with respect to (w.r.t) is optimal, a natural idea is to stop the value iteration after some finite number of iteration steps and return a policy that is greedy w.r.t. the approximation of that was just obtained. If we stop the process after the th step, this defines a policy such that is greedy w.r.t. : . The hope is that as approaches , the policies will also get better in the sense that decreases.

The next theorem guarantees that this will indeed be the case.

Theorem (Policy Error Bound): Let be arbitrary and be the greedy policy w.r.t. : . Then,

In words, the theorem states that the **policy error** ( ) of a policy that is greedy with respect to a function is controlled by the distance of to . This can also be seen as stating that the "greedy operator" , which maps functions to a policy that is greedy w.r.t. , is continuous at when the "distance" between policies is defined as the maximum norm distance between their value functions: . Indeed, with the help of this notation, an alternative form of the theorem statement is that for any ,

In words, this can be described as that is is " -smooth" at when the input space is equipped with the maximum norm distance and the output space is equipped with . One can also show that this result is sharp in that the constant cannot be improved.

The proof is an archetypical example of proofs of using contraction and monotonicity arguments to prove error bounds. We will see variations of this proof many times. Before the proof, let us introduce the notation for a vector to mean the componentwise absolute value of the vector: , .

As a way of using this notation, note that for any memoryless policy ,

and hence

In Eq. the first inequality follows because is monotone and . For the proof it will also be useful to recall that we also have for any , and memoryless policy . These two identities follow just by the definitions of and , as the reader can easily verify them.

Proof: Let be as in the theorem statement and let . Let . The result follows by algebra once we prove that . Hence, we only need to prove this inequality.

By our assumptions on and , . Now,

Taking the (pointwise) absolute value of both sides and using the triangle inequality, and then Eq. we find that The proof is finished by taking the maximum over the components, noting that .

An alternative way of finishing the proof is to note that from , by reordering and using that is a monotone operator, . Taking the max-norm of both sides, we get .

Value Iteration as an Approximate Planning Algorithm

From Eq. we see for ——————, started with , value iteration yields such that and consequently, for a policy that is greedy w.r.t. , —— . Now, for a fixed setting so that —— holds, we see that after ____ iterations, we get a -optimal policy : . Computing using takes elementary arithmetic (and logic) operations. Putting things together we get the following result:

Theorem (Runtime of Approximate Planning with Value Iteration): Fix a finite discounted MDP and a target accuracy . Then, after

elementary arithmetic operations, value iteration produces a policy that is -optimal: , where the result holds when is fixed and hides a term.

Note that the number of operations needed depends very mildly on the target accuracy. However, accuracy here means an additive error. While the optimal value could be as high as , it can easily happen that the best value that can be achieved, , is significantly smaller than . It may be for example that , in which case a guarantee with is vacuous.

By a careful inspection of we can improve the previous result so that this problem is avoided:

Theorem (Runtime when Controlling for the Relative Error): Fix a finite discounted MDP and a target accuracy . Then, stopping value iteration after iterations, the policy produced satisfies the relative error bound

while the total number of elementary arithmetic operations is

where hides .

Notice that the runtime required to achieve a fixed relative accuracy appears to be the same as the runtime required to achieve the same level of absolute accuracy. In fact, the runtime slightly decreases. This should make sense: The worst-case for the fixed absolute accuracy is when , and in this case the relative accuracy is significantly less demanding: With , value iteration can stop after guaranteeing values of , which, as a value, is much smaller than , the target with the absolute accuracy level of .

Note that the relative error bound is not without problems either: It is possible that for some states , is negative, a vacuous guarantee. A reasonable stopping criteria would be to stop when the policy that we read out satisfies

Since is not available, to arrive at a stopping condition that can be verified and which implies the above inequality, one can replace above with an upper bound on it, such as . In this imagined procedure, in each iteration, one also needs to compute the value function of policy to verify whether the stopping condition is met. If we do this much computation, we may as well replace with in the update equation hoping that this will further speed up convergence. This results in what is known as **policy iteration**, which is the subject of the next lecture.

The Computational Complexity of Planning in MDPs

Now that we have our first results for the computation of approximately optimal policies, it is time to ask whether the algorithm we discovered is doing unnecessary work. That is, what is the minimax computational cost of calculating an optimal, or approximately optimal policy?

To precisely formulate this problem, we need to specify the inputs and the outputs of the algorithms considered. The simplest setting is when the inputs to the algorithms are arrays, describing the transition probabilities and the rewards for each state action pair with some ordering of state-action pairs (and next states in the case of transition probabilities). The output, by the Fundamental Theorem, can be a memoryless policy, either deterministic or stochastic. To describe such a policy, the algorithm could write a table. Clearly, the runtime of the algorithm will be at least the size of the table that needs to be written, so the shorter the output, the better the runtime can be. To be nice with the algorithms, we should allow them to output deterministic policies. After all, the Fundamental Theorem also guarantees that we can always find a deterministic memoryless policy which is optimal. Further, greedy policies can also be chosen to be deterministic, so the value-iteration algorithm would also satisfy this requirement. The shortest specification for a deterministic policy is an array of the size of the state space that has entries.

Thus, the runtime of any algorithm that needs to "produce" a fully specified policy is at least .

This is quite bad! As was noted before, , the number of states, in typical problems is expected to be gigantic. But by this easy argument we see that if we demand algorithms to produce fully specified policies then without any further help, they have to do as much work as the number of states. However, things are a bit even worse.

In Homework 0, we have seen that no algorithm can find a given value in an array without looking at all entries of the array (curiously, we saw that if we allow randomized computation, that on expectation it is enough to check half of the entries).

Based on this, it is not hard to show the following result:

Theorem (Computation Complexity of Planning in MDPs):

Let . Any algorithm that is guaranteed to produce -optimal policies in any finite MDP described with tables, with a fixed discount factor and rewards in the interval needs at least elementary arithmetic operations on some MDP with the above properties and whose state space is of size and action space is of size .

Proof sketch: We construct a family of MDPs such that no matter the algorithm, the algorithm will need to perform the said number of operations in at least one of the MDPs.

One-third of the states is reserved for "heaven", one-third is reserved for "hell" states. The remaining one-third set of states, call them , is where the algorithms will need to make some nontrivial amount of work. The MDPs are going to be deterministic. In the tables given to the algorithms as input, we (conveniently for the algorithms) order the states so that the "hell" states come first, followed by the "heaven" states, followed by the states in .

In the "heaven" class, all states self-loop under all actions and give a reward of one. The optimal value of any of these states is . In the "hell" class, states also self-loops under all actions but give a reward of zero. The optimal value of these states is . For the remaining states, all actions except one lead to some hell state, while the chosen special action leads to some state in the heaven class.

The optimal value of all states in set have a value of and the value of a policy that in a state in does not choose the special optimal action gets the value of in that state. It follows that any algorithm that is guaranteed to be optimal needs to identify the unique optimal action at every state in .

In particular, for every state and action , the algorithm needs to read entries of the transition probability vector or it can't find out whether leads to a state in the heaven class or the hell class: The probability vector will have a single one at such an entry, either among the entries representing the hell, or the entries representing the heaven states. By the aforementioned homework problem, any algorithm that needs to find this "needle" requires to check entries. Since the number of states in is also , we get that the algorithm needs to do work.

We immediately see two differences between the lower bound and our previous upper bound(s): In the lower bound there is no dependence on (the effective horizon at a constant precision). Furthermore, there is no dependence on , the inverse accuracy.

As it turns out, the dependence on of value-iteration is superfluous and can be removed. The algorithm that achieves this is policy iteration, which was mentioned earlier. However, this result is saved for the next lecture. After this, the only remaining gap will be the order of the polynomials and the dependence on , which is closely related to the said polynomial order.

And of course, we save for later the most pressing issue that we need to somehow be able to avoid the situation when the runtime depends on the size of the state space (forgetting about the action space for a moment). By the lower bound just presented we already know that this will require changing the problem setting. Just how to do this will be the core question that we will keep returning to in the class.

Notes

Value iteration

The idea of value iteration is probably due to Richard Bellman.

Error bound for greedification

This theorem is due to Singh & Yee, 1994.

The example that shows that the result stated in the theorem is **tight**. Consider an MDP with two states, call them and , two actions, and deterministic dynamics. Call the two actions and . Regardless the state where it is used, action makes the next state transit to state , while giving a reward of . Analogously, action makes the next state transit to state , while giving a reward of . The optimal values in both states are . Let be so that , while . Thus, underestimates the value of , while it overestimates the value of state . It is not hard to see that the policy that uses action regardless the state is greedy with respect to (actually, the action-values of the two actions tie at both states). The value function of this policy assigns the value of to both states, showing that the result stated in the theorem is indeed tight.

Computational complexity lower bound

The last theorem is due to Chen and Wang (2017), but the construction is also (unsurprisingly) similar to one that appeared in an earlier paper that studied query complexity in the setting when the access to the MDP is provided by a simulation model. In fact, we will present this lower bound later in a lecture where we study batch RL. According to this result, the **query-complexity** (also known as sample-complexity) of finding a -optimal policy with constant probability in discounted MDPs accessible through a random access simulator, apart from logarithmic factors, is , where .

Representations matter

We already saw that in order to just clearly define the computational problems (which is necessary for being able to talk about lower bounds), we need to be clear about the inputs (and the outputs). The table representation of MDPs is far from being the only possibility. We just mentioned the "simulation model". Here the algorithm "learns" about the MDP by issuing next state and reward queries to the simulator at some state-action pair of its choice to which the simulator responds with a random next state (drawn fresh) and the . Interestingly, this can provably reduce the number of queries compared to the table representation.

Another alternative, which still keeps tables, is to give the algorithm a cumulative probability representation. In this representation, the states are identified with as before but instead of giving the algorithm the tables for fixed , the algorithm is given (the last entry could be saved, because it is always equal to one, but in the grand scheme of things, of course, this does not matter). Now, it is not hard to see that if the original probability vector had a single one and zeroes everywhere else, the "needle in the haystack problem" used in the lower bound, with the integral representation above, a clever algorithm can find the entry with the one with at most queries. As it turns out, with this representation, the **query complexity** (number of queries required) of producing a good policy can indeed be reduced from the quadratic dependence on the size of the state-space to a log-linear dependence. Hence, we see that the input representation crucially matters. Chen and Wang (2017) also make this point and they discuss yet another, "tree" representation, which leads to a similar speedup.

MDPs with short descriptions

The simulator model assumption addresses the problem that just reading the input may be the bottleneck. This is not the only possibility. One can imagine various classes of MDPs that have a short description, which may raise the hope that one can find out a good policy in them without touching each state-action pair. There are many examples of classes of MDPs that belong to this category. These include

- factored MDPs: The transition dynamics have a short, structured (factored) representation, and the same applies to the reward
- parametric MDPs: The transition dynamics and the rewards have a short, parametric representation. Examples include linear-quadratic regulation (linear dynamics, quadratic reward, Euclidean state and action spaces, Gaussian noise in the transition dynamics), robotic systems, various operations research problems.

For factored MDPs one is out of luck: In these, planning is provably "very hard" (computationally). For linear-quadratic regulation, on the other hand, planning is "easy"; once the data is read, all one has to do is to solve some algebraic equations, for which efficient solution methods have been worked out.

Query vs. computational complexity

The key idea of the lower bound crucially hinges upon that good algorithms need to "learn" about their inputs: The number of arithmetic and logic operations of any algorithm is at least as large as the number of "read" operations it issues. The minimum number of required read operations to produce an input of some desired property is often called the problems **query complexity** and by the above reasoning we see that the computational complexity is lower bounded by the query complexity. As it happens, query complexity is much easier to bound than computational complexity in the sense that it is rare to see computational complexity lower bounds strictly larger than the query complexity (the exceptions to this come when a “compact” representation of the MDP is available, such as in the case of factored MDPs). At the heart of query complexity lower bounds is often the needle in the haystack problem. This seems to be generally true when the inputs are “deterministic”. When querying results in stochastic (random) outcomes, multiple queries may be necessary to “reject”, “reduce”, or “filter out” the noise and then new considerations appear.

In any case, query complexity is a question about quickly determining the information crucial to arrive at a good decision early and is in a way about “learning”: Before a table is read, the algorithm does not *know* which MDP it faces. Hence, query complexity is essentially an “information” question and is also sometimes called **information complexity** and we can think of query complexity as the most basic information theory question. This is a bit different though than mainstream information theory, which is somehow tied up in dealing with reducing the effect of random responses (random “corruptions” of the clean information).

Query complexity everywhere

Query complexity is widely studied in a number of communities which, sadly, are almost entirely disjoint. Information-theory, mentioned above is one of them, though as was noted, here the problems are often tied to studying the speed of gaining information in the presence of noise. Besides information theory, there is the whole field of information-based complexity, which has its own journal, multiple books and more. Also notable is the theory community that studies the complexity of evolutionary algorithms. Besides these, of course, query complexity made appearances in the optimization literature (with or without noise), operations research, and of course in the machine learning and statistics community. In particular, in the machine learning and statistics community, when the algorithm is just handed over noisy data, “the sample”, one can ask how large this sample needs to be to achieve some good outcome (e.g., good predictions on unseen data). This leads to the notion of **sample complexity**, which is the same as our query complexity except that the queries are of the “dull”, “passive” nature of “give me the next datapoint”. As opposed to this, “active learning” refers to the case when the algorithms themselves control some aspects of how the data is collected.

Free lunches, needles and a bit of philosophy

Everyone after going to a few machine learning conferences or reading their first book, or blog posts would have heard about David Wolpert’s “no-free lunch theorems”. Yet, I find that to most people the exact nature (or significance) of these theorems remain elusive. Everyone heard that these theorem essentially state that “in the lack of bias, all algorithms are equal” (and therefore there is no free lunch), from which we should conclude that the only way to choose between algorithms is by introducing bias.

But what does bias means? If one reads these results carefully (and the theory community of evolutionary computation made a good job of making them accessible) one finds that the results are nothing more that describing some corollaries that to find a needle in a haystack (the special entry in a long array), one needs to search the whole haystack (query almost all entries of the array).

Believers of the power of data like to dismiss the significance of the no-free lunch result by claiming that it is ridiculous in that it assumes no structure at all. I find these arguments weak. The main problem is that they are evasive. The evasiveness comes from the reluctance to be clear about what we expect the algorithms to achieve. The claim is that once we are clear about this, that is, clear about the goals, or just the problem specification, we can always hunt for the “needle in the haystack” subproblems within the problem class. This is about figuring out the symmetries (as symmetry equals no structure) that sneakily appear in pretty much any reasonable problem we think of worth studying. The only problems that do not have “needle in the haystack” situations embedded into them are the ones that are not specified at all.

What is the upshot of all this? In a way, the real problem is to be clear about what the problem we want to solve is. This is the problem that most theoreticians in my field struggle with every day. Just because this is hard, we cannot give up on this before even starting, or this will just lead to chaos.

As we shall see in this class, how to specify the problem is also at the very heart of reinforcement learning theory research. We constantly experiment with various problem definitions, tweaking them in various ways, trying to separate hopelessly hard problems from the easy, but reasonably general ones. Theoreticians like to build a library of various problem settings that they can classify in various ways, including relating the problem settings to each other. While algorithm design is the constructive side of RL (and computer science, more generally), understanding the relationship between the various problem settings is just as equally important.

References

- Chen, Y., & Wang, M. (2017). Lower bound on the computational complexity of discounted markov decision problems. arXiv preprint arXiv:1705.07312. [link]
- Singh, S. P., & Yee, R. C. (1994). An upper bound on the loss from approximate optimal-value functions. Machine Learning, 16(3), 227-233. [link]

4. Policy Iteration

PDF Version

In this lecture we

1. formally define policy iteration and
2. show that with $\tilde{O}(\text{poly}(\mathrm{S}, \mathrm{A}, \frac{1}{1-\gamma}))$ elementary arithmetic operations, it produces an **optimal** policy

This latter bound is to be contrasted with what we found out about the runtime of value-iteration in the previous lecture. In particular, value-iteration's runtime bound that we discovered previously grew linearly with $\log(1/\delta))$ where δ was the targeted suboptimality level. This may appear as a big difference in the limit of $\delta \to 0$. Is this difference real? Is value-iteration truly inferior to policy-iteration? We will discuss these at the end of the lecture.

Policy Iteration

Policy iteration starts with an arbitrary deterministic (memoryless) policy π_0. Then, in step $k = 0, 1, 2, \ldots$, the following computations are done:

1. calculate v^{π_k}, and
2. obtain π_{k+1}, another deterministic memoryless policy, by "greedifying" w.r.t. v^{π_k}.

How do we calculate v^{π_k}? Recall that v^π, for an arbitrary memoryless policy π, is the fixed-point of the operator T_π: $v^\pi = T_\pi v^\pi$. Also, recall that $T_\pi v = r_\pi + \gamma P_\pi v$ for any $v \in \mathbb{R}^{\mathcal{S}}$. Thus, $v^\pi = T_\pi v^\pi$ is just a linear equation in v^π, which we can solve explicitly. In the context of policy iteration from this we get

$$v^{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k} \,. \qquad (1)$$

The careful reader will think of why the inverse of the matrix $I - \gamma P_{\pi_k}$ exist. There are many tools we have at this stage to argue that the above is well-defined. One approach is to note that $(I - A)^{-1} = \sum_{i \ge 0} A^i$ holds whenever all eigenvalues of the square matrix A lie strictly within the unit circle on the complex plain (see homework 0). This is known as the von Neumann series expansion of $I - A$, but these big words just hide that at the heart of this is the elementary geometric series formula, $1/(1-x) = \sum_{i \ge 0} x^i$, which holds for all $|x| < 1$, as we have all learned in high school.

Based on Eq. (1) we see that v^{π_k} can be obtained with at most $O(\mathrm{S}^3)$ (and in fact with at most $O(\mathrm{S}^{2.373...})$) arithmetic and logic operations. In particular, the cost of computing r_{π_k} is $O(\mathrm{S})$ (since π_k is deterministic), the cost of computing P_{π_k}, with the table representation of the MDP and "random access" to the tables, is $O(\mathrm{S}^2)$. Note that all these are independent of the number of actions.

Computationally, the "greedification step" above just means to compute for each state $s \in \mathcal{S}$ an action that maximizes the one-step Bellman lookahead values w.r.t. v^{π_k}. Writing this out, we see that we need to solve the maximization problem

$$\max_{a \in \mathcal{A}} r_a(s) + \gamma \langle P_a(s), v^{\pi_k} \rangle$$

and store the result as the action that will be selected by π_{k+1}. Since we agreed that all these policies will be deterministic, we may remove a bit of the storage redundancy, if we allow the algorithm just to store the action chosen by π_{k+1} at every state (and eventually produce the output in this form), rather than requiring it to produce a probability vector for each state, which would have a lot of redundant zero entries in it. Correspondingly, we will further abuse notation and will allow deterministic memoryless policies to be identified with $\mathcal{S} \to \mathcal{A}$ maps. Thus, $\pi_{k+1} : \mathcal{S} \to \mathcal{A}$.

Given v^{π_k}, a vector of length S, the cost of evaluating the argument of the maximum is $O(\mathrm{S})$. Thus, the cost of computing the maximum is $O(\mathrm{SA})$: This is where the number of actions appears (in these steps) in the runtime.

Our main result will be a theorem that states that after $\tilde{O}(\mathrm{SA}/(1-\gamma))$ iterations, the policy computed by policy iteration is necessarily optimal (and not only approximately optimal!). The proof of this result hinges up on two key observations:

1. Policy iteration converges geometrically
2. After every $H_{\gamma,1}$ iterations, it eliminates at least one suboptimal action at some state.

The first result follows from comparing policy iteration with value iteration. We know that value iteration converges at a geometric rate regardless of its initialization. Hence, if we can prove that $\|v^{\pi_k} - v^*\|_\infty \le \|T^k v^{\pi_0} - v^*\|_\infty$ then we will be done. In the so-called "policy improvement lemma", we will in fact prove a result that implies

$$T^k v^{\pi_0} \le v^{\pi_k}, \qquad k = 0, 1, 2, \ldots \tag{2}$$

which is stronger than the geometric convergence result.

Lemma (Geometric Progress Lemma): Let π, π' be memoryless policies such that π' is greedy w.r.t. v^π. Then,

$$v^\pi \le T v^\pi \le v^{\pi'} \,.$$

Proof: By definition, $Tv^\pi = T_{\pi'} v^\pi$. We also have $v^\pi = T_\pi v^\pi \le T v^\pi$. Chaining these, we get

$$v^\pi \le T v^\pi = T_{\pi'} v^\pi \,. \tag{3}$$

We prove by induction on $i \ge 1$ that

$$v^\pi \le T v^\pi \le T_{\pi'}^i v^\pi \,. \tag{4}$$

From this, the result will follow by taking $i \to \infty$ of both sides.

The base case of induction $i = 1$ has just been established. For the general case, assume that the required inequality holds for $i \ge 1$. We show that it also holds for $i+1$. For this, apply $T_{\pi'}$ on both sides of Eq. (4). Since $T_{\pi'}$ is monotone, we get

$$T_{\pi'} v^\pi \le T_{\pi'}^{i+1} v^\pi \,.$$

Chaining this with Eq. (3), we get

$$v^\pi \le T v^\pi = T_{\pi'} v^\pi \le T_{\pi'}^{i+1} v^\pi \,,$$

finishing the inductive step, and hence the proof. ■

The lemma shows that the value functions are monotonically increasing. Applying this lemma k times starting with $\pi = \pi_0$ gives Eq. (2) and this implies the promised result:

Corollary (Geometric convergence): Let $\{\pi_k\}_{k \ge 0}$ be the sequence of policies produced by policy iteration. Then, for any $k \ge 0$,

$$\|v^{\pi_k} - v^*\|_\infty \le \gamma^k \|v^{\pi_0} - v^*\|_\infty \,. \tag{5}$$

Proof: By (2),

$$T^k v^{\pi_0} \le v^{\pi_k} \le v^* \,, \qquad k = 0, 1, 2, \dots \,.$$

Hence,

$$v^* - v^{\pi_k} \le v^* - T^k v^{\pi_0}\,, \qquad k = 0, 1, 2, \dots\,.$$

Taking componentwise absolute values and then the maximum over the states, we get that

$$\|v^* - v^{\pi_k}\|_\infty \le \|v^* - T^k v^{\pi_0}\|_\infty = \|T^k v^* - T^k v^{\pi_0}\|_\infty \le \gamma^k \|v^* - v^{\pi_0}\|_\infty\,,$$

which is the desired statement. In the equality above we used the Fundamental Theorem and in the last inequality we used that T is a γ-contraction. ■

We now set out to finish by showing the “strict progress lemma”. The lemma uses the corollary we just obtained, but it will also require some truly novel ideas.

Lemma (Strict progress lemma): Fix an arbitrary suboptimal memoryless policy π_0 and let $\{\pi_k\}_{k\ge 0}$ be the sequence of policies produced by policy iteration. Then, there exists a state $s_0 \in \mathcal{S}$ such that for any $k \ge k^* := \lceil H_{\gamma,1} \rceil + 1$,

$$\pi_k(s_0) \ne \pi_0(s_0)\,.$$

The lemma shows that after every $k^* = \tilde O\left(\frac{1}{1-\gamma}\right)$ iterations, policy iteration eliminates one action-choice at one state until there remains no suboptimal action to be eliminated. This can only be continued for at most $SA - S$ times: In every state, at least one action must be optimal. As an immediate corollary of the progress lemma, we get the main result of this lecture:

Theorem (Runtime Bound for Policy Iteration): Consider a finite, discounted MDP with rewards in $[0, 1]$. Let k^* be as in the progress lemma, $\{\pi_k\}_{k\ge 0}$ the sequence of policies obtained by policy iteration starting from an arbitrary initial policy π_0. Then, after at most $k = k^*(\mathrm{SA} - \mathrm{S}) = \tilde O\left(\frac{\mathrm{SA}-\mathrm{S}}{1-\gamma}\right)$ iterations, the policy π_k produced by policy iteration is optimal: $v^{\pi_k} = v^*$. In particular, policy iteration computes an optimal policy with at most $\tilde O\left(\frac{\mathrm{S}^4\mathrm{A}+\mathrm{S}^3\mathrm{A}^2}{1-\gamma}\right)$ arithmetic and logic operations.

It remains to prove the progress lemma. We start with an identity which will be useful beyond the proof of this lemma. The identity is called the value difference identity and it gives us an alternate form of the difference of values functions of two memoryless policies. Let π, π' be two memoryless policies. Recalling that $v^{\pi'} = (I - \gamma P_{\pi'})^{-1} r_{\pi'}$, by algebra, we find that

$$\begin{aligned} v^{\pi'} - v^{\pi} &= (I - \gamma P_{\pi'})^{-1}[r_{\pi'} - (I - \gamma P_{\pi'}) v^{\pi}] \\ &= (I - \gamma P_{\pi'})^{-1}[T_{\pi'} v^{\pi} - v^{\pi}] \,. \end{aligned}$$

Introducing

$$g(\pi', \pi) = T_{\pi'} v^{\pi} - v^{\pi} \,,$$

which we can think of the "advantage" of π' relative to π, we get the following lemma:

Lemma (Value Difference Identity): For all memoryless policies π, π',

$$v^{\pi'} - v^{\pi} = (I - \gamma P_{\pi'})^{-1} g(\pi', \pi) \,.$$

Of course, a symmetric relationship also holds.

With this, we are now ready to prove the progress lemma. Note that if π^* is an optimal memoryless policy then for any other memoryless policy π, $g(\pi, \pi^*) \le 0$. In fact, the reverse statement also holds: if the above holds for any π, π^* must be optimal. This makes it $-g(\pi_k, \pi^*)$ an ideal target to track the progress that policy iteration makes. We expect this to start at a high value and decrease as k increases. Note, in particular, that if

$$-g(\pi_k, \pi^*)(s_0) < -g(\pi_0, \pi^*)(s_0) \tag{6}$$

for some state $s_0 \in \mathcal{S}$ then, by algebra,

$$r_{\pi_k(s_0)}(s_0) + \gamma \langle P_{\pi_k(s_0)}, v^* \rangle > r_{\pi_0(s_0)}(s_0) + \gamma \langle P_{\pi_0(s_0)}, v^* \rangle$$

which means that $\pi_k(s_0) \ne \pi_0(s_0)$. Hence, the idea of the proof is to show that Eq. (6) holds for *any* $k \ge k^*$.

Proof (of the progress lemma): Fix $k \ge 0$ and π_0 such that π_0 is not optimal. Let π^* be an arbitrary memoryless optimal policy. Then, for policy π_k, by the value difference identity and since π^* is optimal,

$$-g(\pi_k, \pi^*) = (I - \gamma P_{\pi_k})(v^* - v^{\pi_k}) = (v^* - v^{\pi_k}) - \gamma P_{\pi_k}(v^* - v^{\pi_k}) \le v^* - v^{\pi_k} \,,$$

where the last inequality follows because P_{π_k} is stochastic and hence monotone and because $v^* - v^{\pi_k} \ge 0$. Our goal is to relate the right-hand side to $-g(\pi_0, \pi^*)$. Since Eq. (5) allows us to relate the right-hand side to $v^* - v^{\pi_0}$, and the value difference identity then lets us bring in $-g(\pi_0, \pi^*)$, preparing to use Eq. (5), we first take the max-norm of both sides of the above inequality, noting that this keeps the inequality by the definition of the max-norm. Then, as planned, we use Eq. (5) and the value difference identity to get

$$\begin{aligned} \|g(\pi_k, \pi^*)\|_\infty \le \|v^* - v^{\pi_k}\|_\infty \le \gamma^k \|v^* - v^{\pi_0}\|_\infty = \gamma^k \|(I - \gamma P_{\pi_0})^{-1}(-g(\pi_0, \pi^*))\|_\infty \\ \le \frac{\gamma^k}{1-\gamma} \|g(\pi_0, \pi^*)\|_\infty , \end{aligned} \tag{7}$$

where the last inequality follows by noting that $(I - \gamma P_{\pi_0})^{-1} = \sum_{i \ge 0} \gamma^i P_{\pi_0}^i$ and thus from the triangle inequality and because P_{π_0} is a max-norm non-expansion, $\|(I - \gamma P_{\pi_0})^{-1} x\|_\infty \le \frac{1}{1-\gamma} \|x\|_\infty$ holds for any $x \in \mathbb{R}^{\mathrm{S}}$.

Now, define $s_0 \in \mathcal{S}$ to be the state that satisfies $-g(\pi_0, \pi^*)(s_0) = \|g(\pi_0, \pi^*)(s_0)\|_\infty$. Since $\mathcal{S}$ is finite, this exists. Noting that $0 \le -g(\pi_k, \pi^*)(s_0) \le \|g(\pi_k, \pi^*)\|_\infty$, we get from Eq. (7) that

$$-g(\pi_k, \pi^*)(s_0) \le \|g(\pi_k, \pi^*)\|_\infty \le \frac{\gamma^k}{1-\gamma}(-g(\pi_0, \pi^*)(s_0)).$$

Now when $k \ge k^*$, $\frac{\gamma^k}{1-\gamma} < 1$. Since $\pi_0 \ne \pi^*$, $0 < \|g(\pi_0, \pi^*)\|_\infty = -g(\pi_0, \pi^*)(s_0)$ and thus,

$$-g(\pi_k, \pi^*)(s_0) \le \frac{\gamma^k}{1-\gamma}(-g(\pi_0, \pi^*)(s_0)) < -g(\pi_0, \pi^*)(s_0) ,$$

which is Eq. (6), and thus, by our earlier discussion, $\pi_k(s_0) \ne \pi_0(s_0)$. The proof is done because this holds for any $k \ge k^*$. ■

Is Value Iteration Inferior?

Our earlier result on the runtime of value iteration involves a $\log(1/\delta)$ term which grows without bounds as δ, the required precision level, decreases towards zero. However, at this stage it is not clear whether this extra term is the result of a loose analysis or whether it is a property of value-iteration.

Can value iteration be guaranteed to find an optimal policy with computation which is polynomial in S, A *and the planning horizon* $1/(1-\gamma)$, *assuming all value functions takes values in* $[0, 1/(1-\gamma)]$?

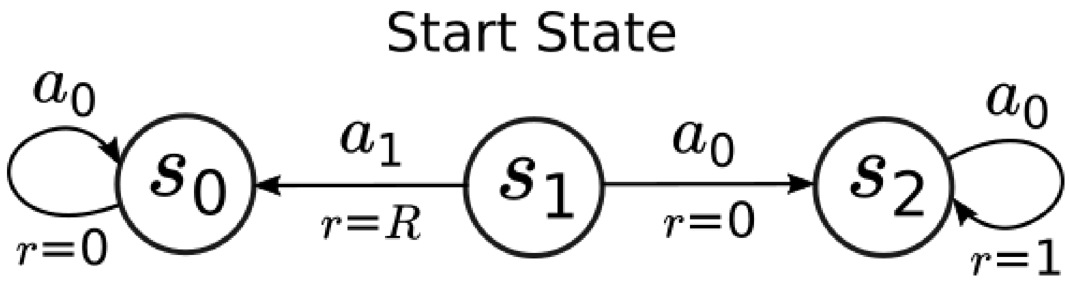
Calling any algorithm that achieves the above **strongly polynomial**, we see that with this terminology we can say that policy iteration is strongly polynomial. Note that in the above definition rather than assuming that the rewards lie in $[0, 1]$, we use the assumption that the value functions for all policies take values in $[0, 1/(1-\gamma)]$. This is a weaker assumption, but checking our proof for the runtime on policy iteration we see that it only needed this assumption.

However, as it turns out, value-iteration is not strongly polynomial:

Proposition: There exists a family of MDPs with deterministic transitions, three states, two actions and value functions for all policies taking values in $[0, 1/(1-\gamma)]$ such that the worst-case iteration complexity of value iteration over this set of MDPs to find an optimal policy is infinite.

Here, iteration complexity means the smallest number of iterations k after which π_k, as computed by value iteration, is optimal, for any of the MDPs in the family. Of course, an infinite iteration complexity also implies an infinite runtime complexity.

Proof: The MDP is depicted in the following figure:



The circles show the states with their names in the circles, the arrows with labels a_0 and a_1 show the transitions between the states as a result of using the actions. The label $r = \cdot$ shows how much reward is incurred along a transition. On the figure, R is not a return, but a free parameter, which is chosen in the interval $[0, \gamma/(1-\gamma)]$ and which will govern the iteration complexity of value iteration.

We consider value iteration initialized at $v_0 = \mathbf{0}$. It is easy to see that the unique optimal action at s_1 is a_0, incurring a value of $\gamma/(1-\gamma)$ at this state. It is also easy to see that $\pi_0(s_1) = a_1 \neq a_0$. We will show that value iteration can "hug" action a_1 at state s_0 indefinitely as R approaches $\gamma/(1-\gamma)$ from below. For this, just note that $v_k(s_0) = 0$ and that $v_k(s_2) = \frac{\gamma}{1-\gamma}(1-\gamma^k)$ for any $k \geq 0$. Then, a little calculation shows that $\pi_k(s_1) = a_1$ as long as $R > v_k(s_2)$. If we want value iteration to spend more than k_0 iterations, all we have to do is to choose $R = \frac{v^*(s_2)+v_{k_0}(s_2)}{2} < \gamma/(1-\gamma)$. ■

It is instructive to note how policy iteration avoids the blow-up of the iteration-counts. This result shows that value-iteration, as far as we are concerned with calculating an optimal policy, exactly, is clearly inferior to policy iteration. However, we also had our earlier positive result for value iteration that showed that the cost of achieving δ-suboptimal policies is at most $\log(1/\delta)$ (and polynomial in the remaining quantities).

What does this all mean? Should we really care about that value-iteration is not finite for exact computation? We have many reasons to not to care much about exact calculations. In the end, we will do sampling, learning, all of which make exact calculations impossible. Also, recall that our models are just models: The models themselves introduce errors. Why would we want to care about exact optimality? In summary:

Exact optimality is nice to have, but approximate computations with runtime growing mildly with the required precision should be almost equally acceptable.

Yet, it remains intriguing to think of how policy iteration can just "snap" into the right solution and how by changing just a few lines of code, a drastic improvement in runtime may be possible. We will keep returning to the question of whether an algorithm has some provable advantage over some others. When this can be shown, it is a true win: We do not need to bother with the inferior algorithm anymore. While this is great, remember that all this depends on how the problems are defined. As we have seen before, and we will see many more times, changing the problem definition can drastically change the landscape of what works and what does not work. And who knows, some algorithm may be inferior in some context, and be superior in some other.

Notes

The runtime bound on policy iteration

The first result that showed that after $\mathrm{poly}(\mathrm{S}, \mathrm{A}, \frac{1}{1-\gamma})$ arithmetic and logic operations one can compute an optimal policy is due to Yinyu Ye (2011). This was a real breakthrough of the time. The theorem we proved is by Bruno Scherrer (2016) and we followed closely his proof. This proof is much simpler than the first one by Yinyu Ye, though the main ideas can be traced back to the proof of Yinyu Ye.

Runtime of value iteration

The example that shows that value iteration is not strongly polynomial is due to Eugene A. Feinberg, Jefferson Huang and Bruno Scherrer (2014).

Ties and stopping

More often than one may imagine, two actions may tie for the maximum in the above problem. Which one to use in this case? As it turns out, it matters only if we want to build a stopping condition for the algorithm that stops the first time it detects that $\pi_{k+1} = \pi_k$. This stopping condition takes $O(\mathrm{S})$ operations, so is quite cheap. If we use this stopping condition, we better make sure that when there are ties, the algorithm resolves them in a systematic fashion, meaning that it has a fixed preference relation over the actions that it respects in case of ties. Otherwise, in the case when there are two optimal actions at some state s, π_k is an optimal policy, π_{k+1} may choose the optimal action that π_k did not choose, and then π_{k+2} could choose the same action as π_k at the same state, etc. and the stopping condition would fail to detect that all these policies are optimal.

Alternatively to resolving ties systematically one may simply change the stopping condition to checking whether $v^{\pi_k} = v^{\pi_{k+1}}$. The reader is invited to check that this would work. "In practice", though, this may be problematic if v^{π_k} and $v^{\pi_{k+1}}$ are computed with finite precision and somehow the approximation errors that arise in this calculation lead to different answers. Can this happen at all? It can! We may have $v^{\pi_k} = v^{\pi_{k+1}}$ (with infinite precision), while $r_{\pi_k} \neq r_{\pi_{k+1}}$ and $I - \gamma P_{\pi_k} \neq I - \gamma P_{\pi_{k+1}}$. And so with finite precision calculations, there is no guarantee that we get the same outcomes in the two cases! The only guarantee that we get with finite precision calculations is that with identical inputs, the outputs are identical.

An easy way out, of course, is just to use the theorem above and stop after the number of iterations is sufficiently large. However, this may be, needlessly, wasteful.

References

- Feinberg, E. A., Huang, J., & Scherrer, B. (2014). Modified policy iteration algorithms are not strongly polynomial for discounted dynamic programming. Operations Research Letters, 42(6-7), 429-431. [link]
- Scherrer, B. (2016). Improved and generalized upper bounds on the complexity of policy iteration. Mathematics of Operations Research, 41(3), 758-774. [link]
- Ye, Y. (2011). The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. Mathematics of Operations Research, 36(4), 593-603. [link]

5. Online Planning - Part I.

PDF Version

In this lecture we

1. introduce online planning;
2. show that for deterministic MDPs there is an online planner whose runtime per call is independent of the size of the state space;
3. show that this online planner has in fact a near-optimal runtime in a worst-case sense.

What is Online Planning?

In a previous lecture we have seen that in discounted MDP with S states and A actions, no algorithm can output a $\delta \le \gamma/(1-\gamma)$ optimal or better policy with a computation cost less than $\Omega(S^2 A)$ provided that the MDP is given with a table representation. One of the SA factors here comes from that to specify a policy one needs to compute (and output) what action to take in every state. The additional S factor comes from because to figure out whether an action is any good, one needs to read almost all entries of the next-state distribution vector.

An unpleasant tendency of the world is that if a problem is modelled as an MDP (that is, the Markov assumption is faithfully observed), the size of the state space tends to blow up. Bellman's curse of dimensionality is one reason why this happens. To be able to deal with such large MDPs, we expect our algorithm's **runtime to be independent of the size of the state space**. However, our lower bound tells us that this is a pipe dream.

But why did we require the planner to output a full policy? And why did we assume that the only way to get information about the MDP is to read big tables of transition probabilities? In fact, if the planner is used inside an "agent" that is embedded in an environment, there is no need for the planner to output a full policy: In every moment, the planner just needs to calculate the action to be taken in the state corresponding to the current circumstances of the environment. In particular, there is no need to specify what action to take under any other circumstances than the current one!

As we usually do in these lectures, assume that the environment is an MDP and the agent gets access to the state in every step when it needs to make a decision. Further, assume that the agent is lucky to also have access to a simulator of the MDP that describes its environment. Just think of the simulator as a black box that can be, fed with a state-action pair and responds with the immediate reward and a random next state from the correct next-state distribution. One can then perhaps build a planner that uses this black box with a "few" queries and quickly returns an action, to be taken by the agent, moving the environment to a random next state, from where the process continues.

Now, the planner does not need to output actions at all states and it does not need to spend time on reading long probability vectors. Hence, in theory, the obstacles that led to the lower bound are removed. The question still remains whether in this new situation planner's can indeed get away with runtime independent of the size of the state space. To break the suspense, the answer is yes and it comes very easily for deterministic environments. For stochastic environments a little more work will be necessary.

In the remainder of this lecture we give a formal problem definition for the **online planning problem** that was described informally above. Next, the result is explained for deterministic environments. This result will be matched with a lower bound.

Online Planning: Formal Definitions

We start with the definition of MDP simulators. We use a language similar to that used to describe optimization problems where one talks about optimization in the presence of various oracles (zeroth-order, first order, noisy, etc.). Because we assume that all MDPs are finite, we identify the state and action spaces with subsets of the natural numbers and for the action set we also require that the action set is $[\mathrm{A}]$ where A is the number of actions. This simplifies the description quite a bit.

Definition (MDP simulator): A simulator implementing an MDP $M = (\mathcal{S}, \mathcal{A}, P, r)$ is a "black-box oracle" that when **queried** with a state action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ returns the reward $r_a(s)$ and a random state $S' \sim P_a(s)$, where $r = (r_a(s))_{s,a}$ and $P = (P_a(s))_{s,a}$.

Users of the black-box must pay attention avoid querying it for state-action pairs outside of $\mathcal{S} \times \mathcal{A}$. Our next notion is that of an online planner:

Definition (Online Planner): An online planner takes as input the number of actions A, a state $s \in \mathbb{N}$, an MDP simulator "access point". After querying this simulator finitely many times, the planner needs to return an action from $[\mathrm{A}]$.

(Online) planners may randomize their calculation. Even if they do not randomize, the action returned by a planner is in general random due to the randomness of the simulator that the planner uses. A planner is **well-formed** if no matter what MDP it interfaces with through a simulator, it returns an action after querying the simulator finitely many times. This also means that the planner can never feed the simulator with state-action pair outside of the set of such pairs.

If an online planner is given access to a simulator of M, the planner and the MDP M together induce a policy of the MDP. We will just refer to this policy as the planner-induced policy π when the MDP is clear from the context. Yet, this policy depends on the MDP implemented by the simulator. If an online planner is well-formed, this policy is well-defined no matter the MDP that is implemented by the simulator.

Online planners are expected to produce good policies:

Definition (δ-sound Online Planner): We say that an online planner is δ-sound if it is well-formed and for any MDP M, the policy π induced by it and a simulator implementing M is δ-optimal in M. In particular,

$$v^\pi \geq v^* - \delta \mathbf{1}$$

must hold where v^* is the optimal value function in M.

The (per-state, worst-case) **query-cost** of an online planner is the maximum number of queries it submits to the simulator where the maximum is over both the MDPs and the initial states.

The following vignette summarizes the problem of online planning:

Model:	Any finite MDP M
Oracle:	Black-box simulator of M
Local input:	State s
Local output:	Action A
Outcome:	Policy π
Postcondition:	$v_M^\pi \ge v_M^* - \delta \boldsymbol{1}$

As an optimization, we let online planners also take as input δ, the target suboptimality level.

Online Planning through Value Iteration and Action-value Functions

Recall value iteration:

```
Let $v_0 = \boldsymbol{0}$
For $k = 1, 2, \dots$ let $v_{k+1} = T v_k$
```

As we have seen, if the iteration is stopped so that $k \ge H_{\gamma,\delta(1-\gamma)/(2\gamma)}$, the policy π_k defined via

$$\pi_k(s) = \arg\max_a r_a(s) + \gamma \langle P_a(s), v_k \rangle$$

is guaranteed to be δ-optimal. Can this be used for online planning? As we shall see, in a way, yes. But before showing this, it will be worthwhile to introduce some additional notation that, in the short term, will save us some writing. More importantly, the new notation will also be seen to influence algorithm design.

The observation is that to decide about what action to take, we need to calculate the one-step lookahead value of the various actions. Rather than doing this in a separate step as shown above, we could have as well chosen to keep track of these lookahead values throughout the whole procedure. Indeed, define $\tilde{T} : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ as

$$\tilde{T} q = r + \gamma P M q, \qquad (q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}) ,$$

where $r \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ and the operators $P : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ and $M : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}}$ are defined via

$$r(s,a) = r_a(s)\,, \quad (Pv)(s,a) = \langle P_a(s), v\rangle\,, \quad (Mq)(s) = \max_{a\in\mathcal{A}} q(s,a)$$

with $s \in \mathcal{S}, a \in \mathcal{A}, v \in \mathbb{R}^{\mathcal{S}}, q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$.

Then the definition of π_k can be shortened to

$$\pi_k(s) = \arg\max_a (\tilde{T}^{k+1}\mathbf{0})(s,a)\,.$$

It is instructive to write the above computation in a recursive, algorithmic form. Let

$$q_k = \tilde{T}^k\mathbf{0}.$$

Using a Python-like pseudocode, our function to calculate the values $q_k(s,\cdot)$ looks as follows:

```
1. define q(k,s):
2.  if k = 0 return [0 for a in A] # base case
3.  return [ r(s,a) + gamma * sum( [P(s,a,s') * max(q(k-1,s')) for s' in S] ) for a in A ]
4. end
```

Line 3, which is where the recursive call happens uses Python's list comprehensions: the brackets create lists and the function itself returns a list. This is a recursive function (since it calls itself in line 3. The runtime is easily seen to be $(\mathrm{AS})^k$, which is not very hopeful until we notice that if the MDP was deterministic, that is, $P(s,a,\cdot)$ has a single one entry, and we have a way of looking up which entry is this without going through all the states, say, $g : \mathcal{S}\times\mathcal{A} \to \mathcal{S}$ is a function that gives the next states, we can rewrite the above as

```
1. define q(k,s):
2.  if k = 0 return [0 for a in A] # base case
3.  return [ r(s,a) + gamma * max(q(k-1,g(s,a))) for a in A ]
4. end
```

As in line 3 there is no loop over the next states (no summing up over these), the runtime becomes

$$O(A^k)$$

which is the first time we see that a good action can be calculated with effort regardless of the size of the state space! And of course, if one is given a simulator of the underlying MDP, which is deterministic, calling g is the same as calling the simulator (once). But will this idea extend to the stochastic case? The answer is yes, but the details will be given in the next lecture. Instead, in this lecture we take a brief look at whether there is any possibility to do better than the above recursive procedure.

Lower Bound

Theorem (online planning lower bound): Take any online planner p that is δ-sound with $\delta < 1$ for discounted MDPs with rewards in $[0, 1]$. Then there exist some MDPs on which p uses at least $\Omega(\mathrm{A}^k)$ queries at some state with

$$k = \left\lceil \frac{\ln(1/(\delta(1-\gamma)))}{\ln(1/\gamma)} \right\rceil, \tag{1}$$

where A is the number of actions in the MDP.

Denote by k_γ the value defined in (1). Then, for $\gamma \to 1$, $k_\gamma = \Omega(H_{\gamma,\delta})$.

Proof: This is a typical needle-in-the-haystack argument. We saw in Question 5 on Homework 0 that no algorithm can find out which element of a binary array of length m is one with less than $\Omega(m)$ queries. Take a rooted regular A-ary tree of depth k. The tree has exactly A^k leafs. Consider an MDP with states corresponding to the nodes of this tree. Call the root s_0. Let the dynamics be deterministic: Taking an action at a node (of the tree) makes the next state the child of that node, unless the node is a leaf node, which are absorbing states: The next state under any action at any leaf state s is s itself. Let all the rewards be zero except at exactly one of the leaf nodes, where the reward under any action is set to one.

If a planner is δ-sound, we claim that it must find the optimal action at s_0. This holds because the value of this action is $\sum_{i=k}^{\infty} \gamma^i = \gamma^k/(1-\gamma)$ and, by our choice of k, $\gamma^k/(1-\gamma) \geq \delta$, while the value of any other action at s_0 is zero. It follows that the planner needs to be able to identify the unique action at the unique leaf node whose reward is one, which, by Question 5 on Homework 0, needs at least $\Omega(\mathrm{A}^k)$ queries. ■

Notes

Dealing with larger state spaces

For a fully formal specification the reader may worry about how a state is described to an online planner, especially, if we allowed uncountably many states. Because the online planner will only have access to the state that it receives as its input and the other states that are returned from the simulator, for the purpose of communication between the online planner and its environment and the simulator, all these states can just be assigned unique numbers to identify them.

Gap between the lower and upper bound

There is an obvious gap between the lower and the upper bound that should be closed.

Local planning vs. online planning

Last year's lecture notes used the expression **local planning** in place of **online planning**. There are pros and cons for both expressions, but perhaps online planning better expresses that the planner will be used in an online fashion, that is, every time after a transition happens.

On simulators and access modes

Simulators come in many shapes and forms. A general planner needs to be prepared to be used in an interconnection with any simulator. But this is too much: Every simulator provides an interface to the planners and planners need to be designed around these interfaces. Therefore, planners will be specialized to the specific interface used. Here, we distinguish three types of interfaces based on what access the interface allows to generating data. The access can be **global**, **local** or **online**.

Global access means that the simulator provides a function that returns a description of the full state space. For finite MDPs this would just mean returning the number of states S. Then, the simulator can be called for any (s, a) pair where $s \in [S]$ and $a \in [A]$ (the simulator should also have a function that returns the number of actions, A). Internally, the simulator then needs to translate the integer indices s and a into appropriate data for which the simulation can be done. Then, the simulator would generate the next state, and translate it back to an integer in $[S]$, which is the data returned from the call. The simulator should also return the associated reward. Often, the reward would also be random (in the lecture, we are concerned with deterministic rewards, but this is just done for the sake of simplicity: random rewards at this stage would not create further difficulties).

Local access means that the simulator allows the planner to generate transitions starting only from states that were passed to the planner previously. To implement a local access simulator, one can just introduce an array that is used to remember all the states that have been returned to the planner. For the sake of interfacing with the planner, one can then use the indexing into this array. This way, the planner does not need to know the details of how states are internally represented and it also becomes possible to interface with simulators where the number of states is infinite, or when it is finite, but calculating this number would be impractical or intractable. Of course, the simulator needs the ability to "go back" to a previously visited state and generate new transition data from there. This can be usually implemented on the top of existing simulators without much trouble (the ability to do this is known as "checkpointing").

Online access simulators have an "internal state", which the planners can manipulate in two ways: they can reset this internal state to the initial state (which is provided to the planner when the planner is called), or they can ask for a transition from the current internal state, by providing an action. As a result of this, the simulator's internal state would move to a random next state, which is what would be returned to the planner (along with the associated reward).

Clearly, any planner prepared to work with online access, can also be used with simulator that provide either local access or global access, and any planner prepared to work with local access can be used with simulators providing global access. In this way, online access is the most general of the access modes, local access is least general, and global access is the most restrictive.

Note that even with online access there is the issue that state information about the state of the environment has to be communicated to the planner in a way that is consistent with how state information can be passed from the planner to the simulator. To keep planners general, the environment and the simulator need to work on an appropriate consistent way of serializing information about the state, which is a pure engineering issue and can usually be done without much trouble.

"**Planning with a generative models**" is an alternative, early terminology that is still used in the literature today. Most commonly, this is means online planning with a global access simulator. However, as the expression itself is not as easy to adopt to different situations as described here, we will refrain from using it.

6. online planning - Part II.

PDF Version

In the previous lecture online planning was introduced. The main idea is to amortize the cost of planning by asking a planner to produce an action to be taken at a particular state so that the policy induced by repeatedly calling the planner at the states just visited and then using the action returned by the planner is near-optimal. We have seen that with this, the cost of planning can be made independent of the size of the state space – at least for deterministic MDPs. For this, one can use just a recursive implementation of value iteration, which, for convenience, we wrote using **action-value functions** and the corresponding Bellman optimality operator, T, defined by

$$Tq(s,a) = r_a(s) + \gamma \langle P_a(s), Mq \rangle \,.$$

(in the previous lecture we used $\tilde{T}$ to denote this operator, but to reduce clutter from now on, we will drop the tilde).

We have also seen that no procedure can do significantly better in terms of its runtime (or query cost) than this simple recursive procedure. In this lecture we show that these ideas also extend to the stochastic case.

Sampling May Save the Day?

Assume now that the MDP is stochastic. Recall the pseudocode of the recursive form of value iteration from the last lecture that computes $(T^k\mathbf{0})(s,\cdot)$:

```
1. define q(k,s):
2.  if k = 0 return [0 for a in A] # base case
3.  return [ r(s,a) + gamma * sum( [P(s,a,s') * max(q(k-1,s')) for s' in S] ) for a in A ]
4. end
```

Obviously, the size of the state space creeps in because in line 3 we need to calculate an expected value over the next state distribution at (s,a). As noted beforehand, in deterministic systems when a simulator is available, the sum over the next-states can be replaced with a single simulator call. But the reader may remember from Probability 101 that sampling allows one to approximate expected values, where the **error of approximation is independent of the cardinality of the set over which we average the values**. Here, this set is $\mathcal{S}$, the state space. This is extremely lucky!

To quantify the size of these errors, we recall Hoeffding's inequality:

Lemma (Hoeffding's Inequality): Given m independent, identically distributed (i.i.d.) random variables that take values in the $[0,1]$ interval, for any $0 \le \zeta < 1$, with probability at least $1-\zeta$ it holds that

$$\left| \frac{1}{m} \sum_{i=1}^m X_i - \mathbb{E}[X_1] \right| \le \sqrt{\frac{\log \frac{2}{\zeta}}{2m}}\,.$$

Letting $S'_1, \dots, S'_m \overset{\text{i.i.d.}}{\sim} P_a(s)$ for some state-action pair (s,a) and $v : S \to [0, v_{\max}]$, by this result, for any $0 \le \zeta < 1$, with probability $1-\zeta$,

$$\left| \frac{1}{m} \sum_{i=1}^m v(S'_i) - \langle P_a(s), v \rangle \right| \le v_{\max} \sqrt{\frac{\log \frac{2}{\zeta}}{2m}}. \qquad (1)$$

This suggests the following approach: For **each** state action pair (s,a) draw $S'_1, \dots, S'_m \overset{\text{i.i.d.}}{\sim} P_a(s)$ and store it in a list $C(s,a)$. Then, whenever for some function v we need the value of $\langle P_a(s), v \rangle$, just use the sample average

$$\frac{1}{m} \sum_{s' \in C(s,a)} v(s')\,.$$

Plugging this approximation into our previous pseudocode gives the following new code:

```
define q(k,s):
 if k = 0 return [0 for a in A] # base case
 return [ r(s,a) + gamma/m * sum( [max(q(k-1,s')) for s' in C(s,a)] ) for a in A ]
end
```

The total runtime of this function is now $O((m\mathrm{A})^{k+1})$. What is important is that this will give us a compute time independent of the size of the state space as long as we can show that m can be set independently of S while meeting our target for the suboptimality of the induced policy.

This pseudocode sweeps under the rug on who creates the lists $C(s,a)$ and when? A simple and effective approach is to use "lazy evaluation" (or memoization): Create $C(s,a)$ at the first time it is needed (and do not create it otherwise). An alternative to the approach we follow here is to avoid storing these lists and just create them on demand. Both procedures are valid, but we will stick to the procedure that creates the lists only once and will comment on the other approach at the end in the notes.

Good Action-Value Approximations Suffice

As a first step towards understanding the strength and weaknesses of this approach, let us define $\hat{T} : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ by

$$(\hat{T}q)(s,a) = r_a(s) + \frac{\gamma}{m} \sum_{s' \in C(s,a)} \max_{a' \in \mathcal{A}} q(s',a')\,.$$

With the help of this definition, when called with state $s = s_0$, the planner computes

$$A = \arg\max_{a\in\mathcal{A}} \underbrace{(\hat{T}^H \mathbf{0})(s_0,a)}_{Q_H(s_0,a)}\,,$$

The conciseness of this formulae, if anything, must please everyone!

Let us now turn to the question of whether the policy $\hat{\pi}$ induced by this planners is a good one. We start with a lemma that parallels our earlier result that bounded the suboptimality of a policy that is greedy w.r.t. a function over the states as a function of how well the function approximates the optimal value function. To state the lemma, we need the analog of optimal value functions but with action values.

Suboptimality of ϵ-optimizing policies

Define

$$q^*(s,a) = r_a(s) + \gamma \langle P_a(s), v^* \rangle\,.$$

We call this function q^* the **optimal action-value function** (in our MDP). The function q^* is easily seen to satisfy $Mq^* = v^*$ and thus also $q^* = Tq^*$. The promised lemma is as follows:

Lemma (Policy error bound - I.): Let π be a memoryless policy and choose a function $q : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ and $\epsilon \ge 0$. Then, the following hold:

1. If π is ϵ**-optimizing** in the sense that $\sum_a \pi(a|s) q^*(s,a) \ge v^*(s) - \epsilon$ holds for every state $s \in \mathcal{S}$ then π is $\epsilon/(1-\gamma)$ suboptimal: $v^\pi \ge v^* - \frac{\epsilon}{1-\gamma}\mathbf{1}\,.$
2. If π is greedy with respect to q then π is 2ϵ-optimizing with $\epsilon = \|q - q^*\|_\infty$ and thus

$$v^\pi \ge v^* - \frac{2\|q-q^*\|_\infty}{1-\gamma}\mathbf{1}\,.$$

For the proof, which is partially left to the reader, we need to introduce a bit more notation. In particular, for a memoryless policy, define the operator $M_\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}}$:

$$(M_\pi q)(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q(s,a)\,, \qquad (q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}},\ s \in \mathcal{S}).$$

With the help of this operator the condition that π is greedy with respect to q can be written as

$$M_\pi q = Mq\,.$$

Further, the second claim of the lemma can be stated in the more concise form $M_\pi q^* \geq v^* - 2\epsilon\boldsymbol{1}$.

For future reference, we will also find it useful to define $P_\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$:

$$P_\pi = PM_\pi\,.$$

Note that here we abused notation as P_π has already been used to denote the operator that maps functions of the states to functions of the state. From the context, the meaning of P_π will always be clear.

Proof: The first part of the proof is standard and is left to the reader. For the second part note that

$$M_\pi q^* \geq M_\pi(q-\epsilon\boldsymbol{1}) = M_\pi q - \epsilon\boldsymbol{1} = Mq - \epsilon\boldsymbol{1} \geq M(q^*-\epsilon\boldsymbol{1}) - \epsilon\boldsymbol{1} = Mq^* - 2\epsilon\boldsymbol{1} = v^* - 2\epsilon\boldsymbol{1}\,.$$

Then use the first part. ■

Suboptimality of almost ϵ-optimizing policies

There are two issues that need to be taken care of. One is that the planner is randomizing when computing the values $Q_H(s_0,\cdot)$. What happens when the random next states obtained from the simulator are not "representative"? We cannot expect the outcome of this randomized computation to be precise! Indeed, the best we can expect is that the outcome is "accurate" with some probability, hopefully close to one. In fact, from Hoeffding's inequality, we see that if we want to achieve small errors in the computation for some target probability, we need to increase the sample size. But Hoeffding's inequality, in all cases, allows errors which are uncontrolled on some **failure event**.

All in all, the best we can hope for is that with each call, $Q_H(s_0,\cdot)$ is a good approximation to $q^*(s_0,\cdot)$ outside of some "failure event" $\mathcal{F}$ whose probability we will control separately. Let us say the probability of $\mathcal{F}$ is at most ζ:

$$\mathbb{P}_{s_0}(\mathcal{F}) \leq \zeta\,.$$

Here, $\mathbb{P}_{s_0}$ denotes the probability measure induced by the interaction of the planner and the MDP simulator on an appropriate probability space. We will choose $\mathcal{F}$ so that on $\mathcal{F}^c$, the complementer of $\mathcal{F}$ (a "good" event), it holds that

$$\delta_H = \|Q_H(s_0,\cdot) - q^*(s_0,\cdot)\|_\infty \leq \epsilon\,. \tag{2}$$

Then, on $\mathcal{F}^c$,

$$q^*(s_0,A) \geq Q_H(s_0,A) - \epsilon = \max_a Q_H(s_0,a) - \epsilon \geq \max_a (q^*(s_0,a)-\epsilon) - \epsilon = v^*(s_0) - 2\epsilon\,.$$

That is, on the good event $\mathcal{F}^c$, the action A returned by the planner is 2ϵ optimizing at state s_0.

Let $\hat\pi(a|s_0)$ denote the probability that action A returned by the planner is a: $\hat\pi(a|s_0) = \mathbb{P}_{s_0}(A = a)$. Then,

$$
\begin{aligned}
&\sum_a \hat\pi(a|s_0)\mathbb{I}(q^*(s_0,a) \ge v^*(s_0) - 2\epsilon) \\
&\quad = \mathbb{P}_{s_0}(q^*(s_0,A) \ge v^*(s_0) - 2\epsilon) \\
&\quad = \mathbb{P}_{s_0}(q^*(s_0,A) \ge v^*(s_0) - 2\epsilon, \mathcal{F}^c) + \mathbb{P}_{s_0}(q^*(s_0,A) \ge v^*(s_0) - 2\epsilon, \mathcal{F}) \\
&\quad \ge \mathbb{P}_{s_0}(q^*(s_0,A) \ge v^*(s_0) - 2\epsilon, \mathcal{F}^c) \\
&\quad = \mathbb{P}_{s_0}(\mathcal{F}^c) \\
&\quad \ge 1 - \zeta\,.
\end{aligned}
$$

In words, with probability at least $1-\zeta$, $\hat\pi$ chooses 2ϵ-optimizing actions: The policy is **almost** 2ϵ**-optimizing**. While this is not as good as always choosing 2ϵ-optimizing actions, we expect that as $\zeta \to 0$ the difference in performance between $\hat\pi$ and a policy that always chooses 2ϵ-optimizing actions disappears because performance is expected to depend on action probabilities in a **continuous** fashion. The next lemma makes this precise:

Lemma (Policy error bound II): Let $\zeta \in [0,1]$, π be a memoryless policy that selects ϵ-optimizing actions with probability at least $1-\zeta$ in each state. Then,

$$
v^\pi \ge v^* - \frac{\epsilon + 2\zeta\|q^*\|_\infty}{1-\gamma}\mathbf{1}\,.
$$

Proof: By Part 1 of the previous lemma, it suffices to show that π is $\epsilon + 2\zeta\|q^*\|_\infty$-optimizing in every state. This follows from algebra and is left to the reader. ■

Error control

What remains is to show that with high probability, the error δ_H, defined in (2) is small. Intuitively, $\hat T \approx T$. To firm up this intuition, we may note that for any fixed $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ function over the state-action pairs such that $\|q\|_\infty \le \frac{1}{1-\gamma}$ and for any fixed $(s,a) \in \mathcal{S}\times\mathcal{A}$, by Eq. (1) and the choice of the sets $\mathcal{C}(s,a)$, with probability $1-\zeta$,

$$
\begin{aligned}
|\hat T q(s,a) - Tq(s,a)| &= \gamma \left| \frac{1}{m} \sum_{s' \in \mathcal{C}(s,a)} v(s') \;-\langle P_a(s), v\rangle \right| \le \gamma \|q\|_\infty \sqrt{\frac{\log\frac{2}{\zeta}}{2m}} \\
&\le \frac{\gamma}{1-\gamma}\sqrt{\frac{\log\frac{2}{\zeta}}{2m}} =: \Delta(\zeta, m),
\end{aligned} \tag{3}
$$

where, for brevity, we introduced $v = Mq$ in the above formula.

Union bounds

So we know that for any *fixed* state-action pair (s, a), outside of a low probability event, $(\hat{T}q)(s, a)$ is close to $(Tq)(s, a)$. But can we conclude from this that, outside of *some* low probability event, $(\hat{T}q)(s, a)$ is close to $(Tq)(s, a)$ *everywhere*?

To answer this question, it will be easier to turn it around and just try to come up with some event that, on the one hand, has low probability, while, in the other hand, outside of this event, $(\hat{T}q)(s, a)$ is close to $(Tq)(s, a)$ regardless of (s, a).

Denoting by $\mathcal{E}(s, a)$ the event when $(\hat{T}q)(s, a)$ is *not* close to $(Tq)(s, a)$, i.e.,

$$\mathcal{E}(s,a) = \left\{ |(\hat{T}q)(s,a) - (Tq)(s,a)| > \Delta(\zeta, m) \right\},$$

it is clear that if $\mathcal{E} = \cup_{(s,a)} \mathcal{E}(s, a)$ then outside of $\mathcal{E}$, none of $\mathcal{E}(s, a)$ holds and hence

$$\max_{(s,a)\in\mathcal{S}\times\mathcal{A}} |(\hat{T}q)(s,a) - (Tq)(s,a)| \le \Delta(\zeta, m)\,.$$

But how large can the probability of $\mathcal{E}$ be? For this, recall the following elementary result, which follows directly from the properties of measures:

Lemma (Union Bound): For any probability measure $\mathbb{P}$ and any countable sequence of events $A_1, A_2, \dots$ of the underlying measurable space,

$$\mathbb{P}\left(\cup_i A_i\right) \le \sum_i \mathbb{P}(A_i).$$

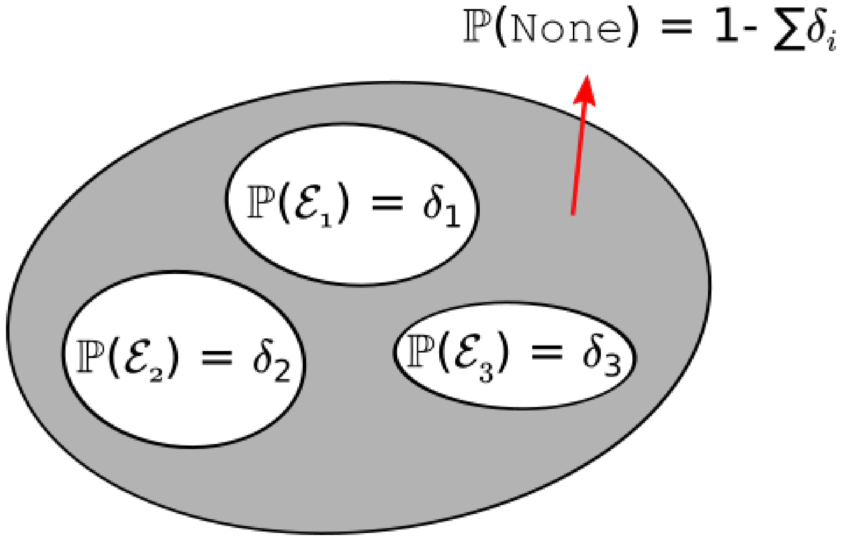
By this result, using that $\mathcal{S} \times \mathcal{A}$ is finite,

$$\mathbb{P}(\mathcal{E}) \le \sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} \mathbb{P}(\mathcal{E}(s,a)) \le \mathrm{SA}\zeta\,.$$

If we want this probability to be $0 \le \zeta' \le 1$, we can set $\zeta = \frac{\zeta'}{\mathrm{SA}}$ and conclude that with probability $1 - \zeta'$, for *any* state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$,

$$|(\hat{T}q)(s,a) - (Tq)(s,a)| \le \Delta\left(\frac{\zeta'}{\mathrm{SA}}, m\right) = \frac{\gamma}{1-\gamma}\sqrt{\frac{\log\frac{2\mathrm{SA}}{\zeta'}}{2m}}\,. \tag{4}$$

The following diagram summarizes the idea of union bounds:


$\mathbb{P}(\text{None}) = 1 - \sum \delta_i$
$\mathbb{P}(\mathcal{E}_1) = \delta_1$
$\mathbb{P}(\mathcal{E}_2) = \delta_2$
$\mathbb{P}(\mathcal{E}_3) = \delta_3$

To control the error of some bad event happening, we can break the the bad event into a number of elementary parts. By controlling the probability of each such part, we can control the probability of the bad event, or, alternatively, control the probability of the complementary "good" event. The worst case for controlling the probability of the bad event is if the elementary parts do not overlap, but the argument of course works even in this case.

Returning to our calculations, from the last formula we see that the errors grew a little compared to (3) , but the growth is modest: the errors scale with the logarithm of the number of state-action pairs. While this logarithmic error-growth is mild, it is unfortunate that the number of states appears here. To control the errors, by this formulae we would need to choose m to be proportional to the logarithm of the size of the state space, which is better than a linear dependence, but still. One must wonder whether this dependence is truly necessary? If it was, there would be a big gap between the complexity of planning in deterministic and stochastic MDPs. We should not give in for this just yet!

Avoiding dependence on state space cardinality

The key to avoiding the dependence on the cardinality of the state is to avoid taking union bounds over the whole state-action set. That this may be possible follows from that, thinking back to the recursive implementation of the planner, we can notice that the planner does not necessarily rely on *all* the sets $\mathcal{C}(s,a)$.

To get a handle on this, it will be useful to introduce a notion of a distance induced by the set $\mathcal{C}(s) := \cup_{a\in\mathcal{A}}\mathcal{C}(s,a)$ between the states. This distance between states s and s' (denoted by $\text{dist}(s,s')$) will be the smallest number of steps that we can take to get from s to s', if in each step we choose one "neighbouring" state to the last state, starting from state s. Formally, this is the length n of the shortest sequence $s_0, s_1, \dots, s_n$ such that $s_0 = s$, $s_n = s'$ and for each $i \in [n]$, $s_i \in \mathcal{C}(s_{i-1})$ (this is the distance between states in the directed graph over the states with edges induced by $\mathcal{C}$).

With this, for $h \ge 0$, define

$$\mathcal{S}_h = \{s \in \mathcal{S} | \text{ dist}(s_0, s) \le h\}$$

as the set of states accessible from s_0 by at most h steps. Note that this is a nested sequence of sets and $\mathcal{S}_0 = s_0$, $\mathcal{S}_1$ contains s_0 and its immediate "neighbors", etc.

We may now observe that in the calculation of $Q_H(s_0, \cdot)$ when function q is called with a certain value of $0 \le k \le H$, for the state that appears in the call we have

$$s \in \mathcal{S}_{H-k} \,.$$

This can be proved by induction on k, starting with $k = H$.

▶ Click here for the proof.

Taking into account that when q is called with $k = 0$, the sets $\mathcal{C}(s, a)$ are not used (line 2), we see that only states s from $\mathcal{S}_{H-1}$ are such that the calculation ever uses the set $\mathcal{C}(s, a)$. Since $|\mathcal{C}(s, a)| = m$,

$$\mathcal{S}_h \le 1 + (mA) + \dots + (mA)^h \le (mA)^{h+1}$$

and in particular, $\mathcal{S}_{H-1} \le (mA)^H$, which is independent of the size of the state space. Of course, all along, we knew this very well: This is why the total runtime is also independent of the size of the state space.

The plan is to take advantage of this to avoid a union bound over all possible state-action pairs. We start with a recursive expression for the errors.

Recall that $\delta_H = \|(\hat{T}^H \mathbf{0})(s_0, \cdot) - q^*(s_0, \cdot)\|_\infty$. By the triangle inequality,

$$\delta_H = \|(\hat{T}^H \mathbf{0})(s_0, \cdot) - q^*(s_0, \cdot)\|_\infty \le \|(\hat{T}\hat{T}^{H-1}\mathbf{0})(s_0, \cdot) - \hat{T}q^*(s_0, \cdot)\|_\infty + \|\hat{T}q^*(s_0, \cdot) - q^*(s_0, \cdot)\|_\infty \,.$$

Now, observing that

$$|\hat{T}q(s,a) - \hat{T}q^*(s,a)| \le \frac{\gamma}{m} \sum_{s' \in \mathcal{C}(s,a)} |Mq - v^*|(s') \le \gamma \max_{s' \in \mathcal{C}(s)} |Mq - v^*|(s') \,,$$

we see that

$$\delta_H \le \gamma \max_{s' \in \mathcal{C}(s_0), a \in \mathcal{A}} |(\hat{T}^{H-1}\mathbf{0})(s', a) - q^*(s', a)| + \|\hat{T}q^*(s_0, \cdot) - q^*(s_0, \cdot)\|_\infty \,.$$

In particular, defining

$$\delta_h = \underbrace{\max_{s' \in \mathcal{S}_{H-h}, a \in \mathcal{A}} |\hat{T}^h \mathbf{0}(s', a) - q^*(s', a)|}_{=:\|\hat{T}^h\mathbf{0} - q^*\|_{\mathcal{S}_{H-h}}} \,,$$

we see that

$$\delta_H \le \gamma \delta_{H-1} + \|\hat{T}q^* - q^*\|_{\mathcal{S}_0} \,,$$

where we use the notation $\|q\|_{\mathcal{U}} = \max_{s\in\mathcal{U},\max_{a\in\mathcal{A}}} |q(s,a)|$. More generally, we can prove by induction on $1 \le h \le H$ (starting with $h = H$) that

$$\delta_h \le \gamma\delta_{h-1} + \|\hat{T}q^* - q^*\|_{\mathcal{S}_{H-h}} \le \gamma\delta_{h-1} + \underbrace{\|\hat{T}q^* - q^*\|_{\mathcal{S}_{H-1}}}_{=:\varepsilon'/(1-\gamma)},$$

while

$$\delta_0 = \|q^*\|_{\mathcal{S}_H} \le \|q^*\|_\infty \le \frac{1}{1-\gamma},$$

where the last inequality uses that $r_a(s) \in [0,1]$, which we shall assume for simplicity. Unfolding this recursion for $(\delta_h)_h$, letting

we get

$$\delta_H \le \frac{\gamma^H + \varepsilon'(1+\gamma+\cdots+\gamma^{H-1})}{1-\gamma} \le \left(\gamma^H + \frac{\varepsilon'}{1-\gamma}\right)\frac{1}{1-\gamma}. \tag{5}$$

We see that the first term in the sum on the right-hand side (in the parenthesis) is controlled by H. It remains to show that ε' can also be controlled (by choosing m appropriately).

In fact, notice that $\varepsilon'/(1-\gamma)$ is the maximum-norm error with which $\hat{T}q^*$ approximates $q^* = Tq^*$, but only for states in $\mathcal{S}_{H-1}$ we need to control this error. By our earlier argument, this set has at most $(mA)^H$ states, hence, it is believable that this error can be controlled even when m is chosen independently of the number of states.

Controlling $\|\hat{T}q^* - q^*\|_{\mathcal{S}_{H-1}}$

Since $\mathcal{S}_{H-1}$ has only $(mA)^H$ states in it, one's first instinct is to take a union bound over the error events for the states in this set. The trouble is that the set $\mathcal{S}_{H-1}$ itself is random. As such, it is not clear, what the failure events should be? And how many failure events are we going to have? The size of this set is also random! Notice that if $(A_i)_{i\in[n]}$ are some events with $\mathbb{P}(A_i) \le \delta$ and $I_1,\dots,I_k \in [n]$ are random indices, it does not hold that $\mathbb{P}(\cup_{j=1}^k A_{I_j}) \le k\delta$: One cannot apply the union bound to randomly chosen events. In fact, in the worst case, $\mathbb{P}(\cup_{j=1}^k A_{I_j}) = n\delta$.

To exploit that $\mathcal{S}_{H-1}$ is a small set, we need to use one more time the structure. The reason that the randomness of $\mathcal{S}_{H-1}$ is not going to matter too much is because of the special way this set is constructed. First of all, clearly, $s_0 \in \mathcal{S}_{H-1}$ always and at this state the error $\|(\hat{T}q^*)(s_0,\cdot) - Tq^*(s_0,\cdot)\|_\infty$ is under control by Hoeffding's inequality. Next, we may consider the neighbors of s_0. If $S \in \mathcal{C}(s_0)$, either $S = s_0$, in which case we already know that the error at S is under control, or S is a "bona fide neighbor" and we can think of then generating the elements in $\mathcal{C}(S,a)$ just inside the call of q. Ultimately, the error at such a neighbor is under control because, by definition, all the sets $\mathcal{C}(s,a)$ (with (s,a) sweeping through all possible state-action pairs) are **independently chosen**.

This suggests that we should consider the chronological order in which in the recursive call of function q the states in $\mathcal{S}_{H-1}$ appear. Let this order be $S_1, S_2, \dots, S_n$, where $n = 1 + (mA) + \dots + (mA)^{H-1}$, $S_1 = s_0$, S_2 is the second state that q is called on (necessarily, $S_2 \in \mathcal{C}(s_0)$), S_3 is the third such state. Note that states may reappear in this sequence multiple times. Furthermore, by construction, $\mathcal{S}_{H-1} = \{S_1, \dots, S_n\}$. Also note that the length of this sequence is not random: This length is exactly the number of times q is called, which is clearly not random.

That $\|\hat{T}q^* - q^*\|_{\mathcal{S}_{H-1}} = \|\hat{T}q^* - Tq^*\|_{\mathcal{S}_{H-1}}$ is under control directly follows from the next lemma:

Lemma: Assume that the immediate rewards belong to the $[0, 1]$ interval. For any $0 \le \zeta \le 1$ with probability $1 - \mathrm{A}n\zeta$, for any $1 \le i \le n$,

$$\|\hat{T}q^*(S_i, \cdot) - q^*(S_i, \cdot)\|_\infty \le \Delta(\zeta, m)\,,$$

where Δ is given by (3).

Proof: Recall that $\mathcal{C}(s, a) = (S'_1(s, a), \dots, S'_m(s, a))$ where *(i)* the $(\mathcal{C}(s, a))_{(s,a)}$ are mutually independent and *(ii)* for any (s, a), $(S'_i(s, a))_i$ is an i.i.d. sequence with common distribution $P_a(s)$.

For $s \in \mathcal{S}$, $a \in \mathcal{A}$, $C \in \mathcal{S}^m$, let

$$g(s, a, C) = |\frac{\gamma}{m} \sum_{s' \in C} v^*(s') \; - \langle P_a(s), v^* \rangle|$$

(as earlier, $s' \in C$ means that s' is an element of the set composed of the elements in the sequence C). Recall that by the definition of $\hat{T}$ and the properties of q^*,

$$|\hat{T}q^*(s, a) - q^*(s, a)| = |\frac{\gamma}{m} \sum_{s' \in \mathcal{C}(s,a)} v^*(s') \; - \langle P_a(s), v^* \rangle| = g(s, a, \mathcal{C}(s, a))\,. \tag{6}$$

Fix $1 \le i \le n$. Let $\tau = \min\{1 \le j \le i \,:\, S_j = S_i\}$. That is, τ is the time when S_i first appears in the sequence $\{S_i\}_i$.

Fix $a \in \mathcal{A}$. We claim that given S_τ, $(S'_j(S_\tau, a))_{j=1}^m$ is i.i.d. with common distribution $P_a(S_\tau)$. That is, for any $s, s'_1, \dots, s'_m \in \mathcal{S}$,

$$\mathbb{P}(S'_1(S_\tau, a) = s'_1, \dots, S'_m(S_\tau, a) = s'_m \,|\, S_\tau = s) = \prod_{j=1}^m P(s, a, s'_j) \tag{7}$$

Note that given this, for any $\Delta \ge 0$, by (6),

$$\begin{aligned}
\mathbb{P}(|\hat{T}q^*(S_i,a)-q^*(S_i,a)|>\Delta) &= \mathbb{P}(g(S_i,a,\mathcal{C}(S_i,a))>\Delta) \\
&= \mathbb{P}(g(S_\tau,a,\mathcal{C}(S_\tau,a))>\Delta) \\
&= \sum_s \mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta, S_\tau = s) \\
&= \sum_s \sum_{1\le j\le i} \mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta, S_j = s, \tau = j) \\
&= \sum_s \sum_{1\le j\le i} \sum_{\substack{s_{1:j-1}\in\mathcal{S}^{j-1}:\\ s\notin s_{1:j-1}}} \mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta, S_j = s, S_{1:j-1}=s_{1:j-1}) \\
&= \sum_s \sum_{1\le j\le i} \sum_{\substack{s_{1:j-1}\in\mathcal{S}^{j-1}:\\ s\notin s_{1:j-1}}} \mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta, \phi_j(s,s_{1:j-1},\mathcal{C}(s_1),\dots,\mathcal{C}(s_{j-1}))=1)\,,
\end{aligned}$$

for some binary valued functions $\phi_1,\dots,\phi_i$ where for $1\le j\le i$, ϕ_j is defined so that

$$\phi_j(s,s_{1:j-1},\mathcal{C}(s_1),\dots,\mathcal{C}(s_{j-1}))=1$$

holds if and only if $S_j=s, S_{1:j-1}=s_{1:j-1}$ holds, where $s\in\mathcal{S}$ and $s_{1:j-1}\in\mathcal{S}^{j-1}$ are arbitrary so that $s\notin s_{1:j-1}$. That such functions exist follows because for any sequence $s_{1:j}$ to verify whether $S_{1:j}=s_{1:j}$ the knowledge of the sets $\mathcal{C}(s_1),\dots,\mathcal{C}(s_{j-1})$ suffices: The appropriate function should first check $S_1=s_1$, then move on to checking $S_2=s_2$ only if $S_1=s_1$ holds, etc.

Now, notice that by our assumptions, for $s\notin s_{1:j-1}$, $\mathcal{C}(s,a)$ and $\phi_j(s,s_{1:j-1},\mathcal{C}(s_1),\dots,\mathcal{C}(s_{j-1}))=1$ are independent of each other. Hence,

$$\begin{aligned}
&\mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta, \phi_j(s,s_{1:j-1},\mathcal{C}(s_1),\dots,\mathcal{C}(s_{j-1}))=1) \\
&\quad= \mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta)\cdot\mathbb{P}(\phi_j(s,s_{1:j-1},\mathcal{C}(s_1),\dots,\mathcal{C}(s_{j-1}))=1)\,.
\end{aligned}$$

Plugging this back into the previous displayed equation, "unrolling" the expansion done using the law of total probability, we find that

$$\mathbb{P}(|\hat{T}q^*(S_i,a)-q^*(S_i,a)|>\Delta) = \sum_s \mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta)\mathbb{P}(S_\tau=s)\,.$$

Now, choose $\Delta=\Delta(\zeta,m)$ from (3) so that, thanks to $|q^*|_\infty\le 1/(1-\gamma)$, for any fixed (s,a), $\mathbb{P}(g(s,a,\mathcal{C}(s,a))>\Delta(\zeta,m))\le\zeta$ Plugging this in into the previous display we get

$$\mathbb{P}(|\hat{T}q^*(S_i,a)-q^*(S_i,a)|>\Delta(\zeta,m)) \le \zeta\sum_s \mathbb{P}(S_\tau=s)=\zeta\,.$$

The claim the follows by a union bound over all actions and all $1\le i\le n$. ■

Final error bound

Putting everything together, we get that for any $0\le\zeta\le1$, the policy $\hat\pi$ induced by the planner is $\epsilon(m,H,\zeta)$-optimal with

$$\epsilon(m,H,\zeta) := \frac{2}{(1-\gamma)^2}\left[\gamma^H + \frac{1}{1-\gamma}\sqrt{\frac{\log\left(\frac{2n\mathrm{A}}{\zeta}\right)}{2m}} + \zeta\right].$$

Thus, to obtain a planner that induces a δ-optimal policy, we can set H, ζ and m so that each term above contributes at most $\delta/3$:

$$\begin{aligned} \frac{2\gamma^H}{1-\gamma} &\le (1-\gamma)\frac{\delta}{3}\,, \\ \zeta &\le (1-\gamma)^2\frac{\delta}{6} \qquad \text{and} \\ \frac{m}{\log\left(\frac{2n\mathrm{A}}{\zeta}\right)} &\ge \frac{18}{\delta^2(1-\gamma)^6}\,. \end{aligned}$$

For H we get that we can set $H = \lceil H_{\gamma,(1-\gamma)\delta/6}\rceil$. We can also set $\zeta = (1-\gamma)^2\delta/6$. To solve for the smallest m that satisfies the last inequality, recall that $n = (mA)^H$. To find the critical value of m note the following elementary result which we cite without a proof:

Proposition: Let $a > 0, b \in \mathbb{R}$. Let $t^* = \frac{2}{a}\left[\log\left(\frac{1}{a}\right) - b\right]$. Then, for any positive real t such that $t \ge t^*$,

$$at + b > \log(t)\,.$$

From this, defining

$$c_\delta = \frac{18}{\delta^2(1-\gamma)^6}$$

and

$$m^*(\delta,\mathrm{A}) = 2c_\delta\left[H\log(c_\delta H) + \log\left(\frac{12}{(1-\gamma)^2\delta}\right) + (H+1)\log(\mathrm{A})\right] \tag{8}$$

if $m \ge m^*$ then all the inequalities are satisfied. Putting things together, we thus get the following result:

Theorem: Assume that the immediate rewards belong to the $[0,1]$ interval. There is an online planner such that for any $\delta \ge 0$, in any discounted MDP with discount factor γ, the planner induces a δ-optimal policy and uses at most $O((m^*\mathrm{A})^H)$ elementary arithmetic and logic operations per its calls, where $m^*(\delta,\mathrm{A})$ is given by (8) and $H = \lceil H_{\gamma,(1-\gamma)\delta/3}\rceil$.

Overall, we see that the runtime did increase compared to the deterministic case (apart from logarithmic factors, in the above result $m = H^7/\delta^2$ whereas in the deterministic case $m = 1$!), but we managed to get a runtime that is independent of the cardinality of the state space. Again, what is troubling is the **exponential dependence** on the effective horizon, though as we have seen, in the worst-case, this is unavoidable. In the next lectures we will consider proving the planner with extra information so that this exponential dependence can be avoided.

Notes

Sparse lookahead trees

The idea of the algorithm that we analyzed comes from a paper by Kearns, Mansour and Ng from 2002. In their paper they consider the version of the algorithm which creates a fresh "new" random set $\mathcal{C}(s,a)$ in *every* recursive call. This makes it harder to see their algorithm as approximating the Bellman operator, but in effect, the two approaches are by and large the same. In fact, if we introduce H random operators, $\hat{T}_1, \dots, \hat{T}_H$ which are the same as $\hat{T}$ above but $\hat{T}_h$ has its own "private" sets $(\hat{C}_h(s,a))_{(s,a)}$, then their algorithm can be written as computing

$$A = \arg\max_a (\hat{T}_1 \dots \hat{T}_h \mathbf{0})(s_0, a)\,.$$

It is not hard to modify the analysis given here to accommodate this change. With this, one can also interpret the calculations done by the algorithm as backing up values in a "sparse lookahead tree" built recursively from s_0.

Much work has been devoted to improving these basic ideas and eventually these ideas led to various Monte-Carlo tree search algorithms, including yours truly's UCT. In general, these algorithms attempt to improve on the runtime by building the trees when they need to be built. As it turns out, a useful strategy here is to expand nodes which in a way hold the greatest promise to improve the value at the "root". This is known as the "optimisism in planning". Note that A* (and its MDP relative, AO*) are also based on optimism: A*'s admissible heuristic functions in our language correspond to functions that upper bound the optimal value. The definite source on MCTS theory as of today is Remi Munos's monograph.

Measure concentration

Hoeffding's inequality is a special case of what is known as measure concentration. This phrase refers to that the empirical measure induced by a sample is a good approximation to the whole measure. The simplest case is when one just compares the means of the measures (the empirical and the sample-generating one), giving rise to concentration inequalities around the mean. Hoeffding's inequality is an example. What we like about Hoeffding's inequality (besides that it is simple) is that the failure probability, δ (later ζ) appears inside a logarithm. That means, that the price of being more stringent is mild. When the exact dependence is of type that appears in Hoeffding's inequality (i.e., $\sqrt{\log(1/\delta)}$)), we say that the deviation of the subgaussian type because Gaussian random variables also satisfy an inequality like this. Concentration of measure and concentration inequalities are a central topic in probability theory, with separate books devoted to them. A few favourites are given at the end of this notes . For learning purposes, Pollard's mini-book is nice (but all these books have pros and cons), or Vershynin's book.

The comparison inequality

The comparison inequality between the logarithm and the linear function is given as Proposition 4 here. The proof is based on two observations: First, it is enough to consider the case when $b = 0$. Then, if $a \geq 1$, the result is trivial, while for $a < 1$, the guess is based on doubling the value where the growth rate of $t \mapsto at$ matches that of $t \mapsto \log(t)$.

A model-centered view and random operators

A key idea of this lecture is that $\hat{T}$ is a good (random) approximation to T, hence, it can be used in place of T. One can also tell this story by saying that the data underlying $\hat{T}$ gives a random approximation to the MDP; the transition probabilities of this random approximating MDP would be defined using

$$\hat{P}(s, a, s') = \frac{1}{m} \sum_{s'' \in C(s,a)} \mathbb{I}\{s'' = s'\}$$

It may seem quite miraculous that with only a few elements in $C(s, a)$ (i.e., small m) we get a good approximation to the next state distribution. But so is the magic of randomness! Using a random operator (or a sequence of them, if, as outlined above, one uses a fresh set of random next state every time an update is calculated) in a dynamic programming method has been coined *empirical dynamic programming* by Haskell et al..

A bigger point is that for a model to be a "good" approximation to the "true MDP", it suffices that the Bellman optimality operator that it induces is a "close" approximation to the Bellman optimality operator of the true MDP.

This in fact brings us to our next topic, which is what happens when the simulator is imperfect?

Imperfect simulation model?

We can rarely expect simulators to be perfect. Luckily, not all is lost in this case. As noted above, if the simulator induced an MDP whose Bellman optimality operator is in a way close to the Bellman optimality operator of the true MDP, we expect the outcome of planning to be still a good policy in the true MDP.

In fact, the above proof has already all the key elements in place to show this. In particular, it is not hard to show that if $\hat{T}$ is a γ max-norm contraction and $\hat{q}^*$ is its fixed point then

$$\|\hat{q}^* - q^*\|_\infty \leq \frac{\|\hat{T}q^* - Tq^*\|_\infty}{1-\gamma},$$

which, combined with the our first lemma of this lecture on the policy error bound gives that the policy that is greedy with respect to $\hat{q}^*$ is

$$\frac{2\|\hat{T}q^* - Tq^*\|_\infty}{(1-\gamma)^2}$$

optimal in the MDP underlying T. We will return to this in later lectures. In particular, in batch reinforcement learning, one of the basic methods is to learn a "model" of the environment and as such it is inevitable to study the error that results from modelling errors. See Lecture 17 and Lecture 18.

Monte-Carlo methods

We saw in homework 0 that randomization may help a little, and today we saw that it can help in a more significant way. A major lesson again is that representations do matter: If the MDP is not given with a "generative simulator", getting such a simulator may be really hard. This is good to remember when it comes to learning models:

One should insist on learning models that make the job of planners easier.

Generative models are one such case, provably, as we have seen in today's lecture put together with our previous lower bound that involved the number of states. Randomization, more generally, is a powerful tool in computing science, which brings us to a somewhat philosophical question: What is randomness? Does "true randomness" exist? Can we really build computers to harness this?

True randomness?

What is the meaning of "true" randomness? The margin is definitely not big enough to explain this. Hence, we just leave this there, hanging, for everyone to ponder about. But let's also note that this is a thoroughly studied question in theoretical computing science, with many beautiful results and even books. Arora and Barak's book on computational complexity (Chapters 7, 20 and 21) is a good start for exploring this.

Can we recycle the sets $C(s,a)$ between the calls?

If simulation is expensive, it may be tempting to recycle the sets between calls of the planner. After all, even if we recycle these sets, $\hat{\pi}$ will have the property that it selects ϵ-optimizing actions with high probability at every state. However, this may not be a good idea. The reader is challenged to think about what can go wrong? The proof actually uses that the planner construct a new random operator $\hat{T}$ with every call. But where is this used?

The ubiquity of continuity arguments in the MDP literature

All the computations that we do with MDPs tend to be approximate. We evaluate policies approximately. We compute a Bellman back approximately. We have approximate models. We greedify approximately. If any of these operations could enlarge small errors, none of the approximate methods would work. The study of approximate computations (which is a necessity if one faces large MDPs) is a study of the sensitivity of the values of the resulting policies to the errors introduced in the computations. This, in numerical analysis, would be called error analysis. In other areas of mathematics, this is called sensitivity analysis. In fact, sensitivity analysis often involves computing derivatives to see how fast outputs change as the inputs change (which is that data that will be approximated). What should we be taking derivatives with respect to here? Well, it is always the data that is being changed. One can in fact use differentiation based sensitivity analysis everywhere. This has been tried a little in the "older" MDP literature and is also related to policy gradient theorems (that we will learn about laters). However, perhaps there are more nice things to be discovered about this approach.

From local to online access

The algorithm that is analyzed in this lecture requires local access simulators. This is better than requiring global access, but worse than requiring online access. It remains an open question of whether with online access, one can also get a similar result than shown in the present lecture and if not, whether the sample complexity of planning remains finite under this setting.

When the state space is small

For finite state-action MDPs where the rewards and transition probabilities are represented using tables, a previous lecture's main result established that an optimal policy of the MDP can be calculated by using at most $O(H\text{poly}(S, A))$ arithmetic and logic operations ($H = 1/(1-\gamma)$ here). In the current lecture we saw that even when S is unbounded, given a simulator with local access, $\tilde{O}((AH^7/\delta^2)^H)$ such elementary operations and calls to a simulator are sufficient. In a finite MDP, depending on the values of S, A and H, either policy iteration, or the online planner that builds the tree will be faster. But policy iteration (and value iteration) as described previously used a table representation. The question then arises of what is the sample complexity of planning with a simulator access to a finite MDP? If planning means outputting a policy, the complexity needs to scale with S. In the presence of global access simulators, a simple approach, is to sample an appropriate number of next states for each state-action pair to build an empirical (but "sparse") transition model and use this in connection with any MDP solver. We will see later in Lecture 18 that in this case $O(H^3SA/\delta^2)$ samples (or H^3/δ^2 samples per state-action pair) are sufficient to obtain a δ-optimal policy.

In the case of online planning with global access, the sample complexity cannot be worse, but it is unclear whether it can be improved. Similarly, it is unclear what the complexity is in the case of either local or online access.

References

- Kearns, M., Mansour, Y., & Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. Machine learning, 49(2), 193-208. [link]
- David Pollard (2015). A few good inequalities. Chapter 2 of a book under preparation with working title "MiniEmpirical". [link]
- Stephane Boucheron, Gabor Lugosi and Pascal Massart (2012). Concentration inequalities: A nonasymptotic theory of indepndence. Clarendon Press – Oxford. [link]
- Roman Vershynin (2018). High-Dimensional Probability: An Introduction with Applications in Data Science. [link]

- M. J. Wainwright (2019) High-dimensional statistics: A non-asymptotic viewpoint. Cambridge University Press.
- Lafferty J., Liu H., & Wasserman L. (2010). Concentration of Measure. [link]
- Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.
- William B. Haskell, Rahul Jain, and Dileep Kalathil. Empirical dynamic programming. Mathematics of Operations Research, 2016.
- Sanjeev Arora and Boaz Barak (2009). Computational Complexity: A Modern Approach. Cambridge University Press.
- Remi Munos (2014). From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. Foundations and Trends in Machine Learning: Vol. 7: No. 1, pp 1-129.

7. Function Approximation

PDF Version

Our lower bound for online planners show that there are no online planners that lead to good policies in all MDPs while satisfying the following three requirements

1. the planner induces policies that achieve some positive fraction of the optimal value in all MDPs;
2. the per-state runtime shows polynomial dependence on the planning horizon H and
3. it shows a polynomial dependence on the number of actions and
4. it shows no dependence on the number of states in the MDP.

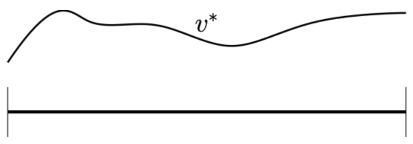
Thus, one is left with no choice than to give up on one of the requirements. Since efficiency is clearly nonnegotiable (otherwise the runner just would not be practical), the only requirement that can be replaced is the first one. In what follows we will look at ways of relaxing this requirement.

In all the relaxations we will look at, we will essentially restrict the set of MDPs that the planner is expected to work on. However, we will do this in such a way that no MDP will be ever ruled out. We achieve this by giving the planner some **extra hint** about the MDP and we demand good performance only when the hint is correct. Since the hint will take a **general form**, some hint is always correct for **any** MDP. Hence, no MDP is left behind and the planner can again demanded to be efficient and effective.

Hints on value functions

The hints that we start with will concern the value functions. In particular, they state that either the optimal value, or the value function of all policies are effectively compressible.

For motivation, consider the figure on the right. Imagine the state space is an interval of the real line and the optimal value function in an MDP looks like as shown on the figure: It is a nice, smooth function over the interval. As is well known, such relatively slowly changing functions can be



well approximated by using the linear combination of a few fixed basis functions, like an appropriate polynomial, or Fourier basis, or using splines. Then, one hopes that even though the state space is large or even infinite as in this example, there could perhaps be a method that calculates the few coefficients needed get a good approximation to v^* with a runtime that depends polynomially on the horizon, the number of actions and the number of coefficients that one needs to calculate. Given the knowledge of v^* and simulator access to the MDP, good actions can then be efficiently obtained by performing one-step lookahead computations.

Linear function approximation

If the basis functions mentioned are $\phi_1, \dots, \phi_d : \mathcal{S} \to \mathbb{R}$ then, formally, the hope is that with some coefficients $\theta = (\theta_1, \dots, \theta_d)^\top \in \mathbb{R}^d$, we will have

$$v^*(s) = \sum_{i=1}^{d} \theta_i \phi_i(s) \qquad \text{for all } s \in \mathcal{S}\,. \tag{1}$$

In the reinforcement learning literature, the vector $(\phi_1(s), \dots, \phi_d(s))^\top$ is called the feature vector assigned to state s. For a more compact notation we also use ϕ to be a map from $\mathcal{S}$ to $\mathbb{R}^d$ which assigns the feature vectors to the states:

$$\phi(s) = (\phi_1(s), \dots, \phi_d(s))^\top\,.$$

Conversely, given $\phi : \mathcal{S} \to \mathbb{R}^d$, its component are denoted using $\phi_1, \dots, \phi_d$. It will also be useful to introduce a matrix notation: Recall that the number of states is S and without loss of generality we may assume that $\mathcal{S} = [\mathrm{S}]$. Then, we can treat each of $\phi_1, \dots, \phi_d$ as S-dimensional vectors: The ith component of ϕ_j is $\phi_j(i)$. Then, we can stack $\phi_1, \dots, \phi_d$ next to each other to form a matrix:

$$\Phi = \begin{pmatrix} | & | & \dots & | \\ \phi_1 & \phi_2 & \dots & \phi_d \\ | & | & \dots & | \end{pmatrix} \in \mathrm{R}^{\mathrm{S} \times d}\,.$$

That is, Φ is a $\mathrm{S} \times d$ matrix. The set of real-valued functions over the state space that can be described with the linear combination of the basis functions is

$$\mathcal{F} = \{ f : \mathcal{S} \to \mathbb{R} \,:\, \exists \theta \in \mathbb{R}^d \text{ s.t. } f(s) = \langle \phi(s), \theta \rangle \}\,.$$

Identifying the space of real-valued functions with the vector space $\mathbb{R}^{\mathrm{S}}$ in the natural way, $\mathcal{F}$ is a d-dimensional subspace of $\mathbb{R}^{\mathrm{S}}$, which is the same as the "column space", or the span, or the range space of Φ:

$$\mathcal{F} = \{ \Phi\theta \,:\, \theta \in \mathbb{R}^d \} = \text{span}(\Phi)$$

If we need to indicate the dependence of $\mathcal{F}$ on the choice of features, we will write either $\mathcal{F}_\phi$ or $\mathcal{F}_\Phi$.

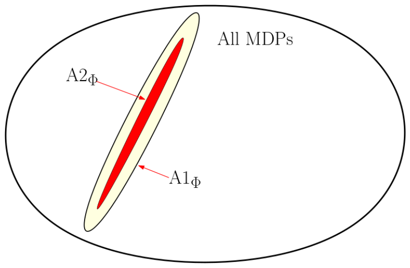
Now, we have three equivalent ways of specifying the "features", either by specifying the basis functions $\phi_1, \dots, \phi_d$, or the feature-map ϕ, or the feature matrix Φ, and we have a four equivalent way of specifying the functions that can be obtained via the linear combination of features.

Delivering the hint

Note that in the above problem description it is tacitly assumed that the feature-map, in some form or another, is available to the planner. In fact, the feature map can be made available in multiple ways. When we argue for lower bounds, especially for query complexity, we often assume that the whole feature-map is available for the algorithm. For upper bounds with online planning, the most natural assumption is that the planner gets from the simulator the feature vector of the states that it encounters. In particular, when it comes to online planning, the natural assumption is that the planner gets the feature vector of the initial state together with the state and with any subsequent calls to the simulator, the simulator returns the feature vector of the next states, together with the next states.

Typical hints

In what follows we will study planning under a number of different hints (or assumptions) that connect the MDP and a feature-map. The simplest of this just states that (1) holds:



Assumption A1 (v^*-realizibility): The MDP M and the featuremap ϕ are such that $v^* \in \mathcal{F}_\phi$

A second variation is when all value functions are realizable:

Assumption A2 (universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $v^\pi \in \mathcal{F}_\phi$.

Clearly, A2 implies A1, because by the fundamental theorem of MDPs, there exists a memoryless policy π such that $v^\pi = v^*$. The figure on the right illustrates the set of all finite MDPs with some state space and within those the set of those MDPs that satisfy A1 with a specific feature map ϕ (denoted by A1$_\phi$ on the figure), as well as those MDPs that satisfy A2 with the same feature map (denoted by A2$_\phi$). Both of these sets represent a very small fraction of all MDPs. However, of one changes the feature map, the union of all these sets clearly covers the set of all MDPs: The hint is general.

There are many variations of these assumptions. Often, we will find it useful to relax the assumption value functions are exactly realizable. Under the modified assumptions the value function does not need to lie in the span of the feature-map, but only in some vicinity of it. The natural error metric to be used is the maximum norm for reasons that will become clear later. To help with stating these assumptions in a compact form, introduce the notation

$$v \in_\varepsilon \mathcal{F}$$

to denote that

$$\inf_{f\in\mathcal{F}} \|f - v\|_\infty \le \epsilon\,.$$

That is, $v \in_\varepsilon \mathcal{F}$ means that the best approximator to v from $\mathcal{F}$ approximates it within a uniform error of ε.

Fixing $\varepsilon \ge 0$ and replacing $\in$ with $\in_\varepsilon$ in the above two assumptions gives the following:

Assumption A1$_\varepsilon$ (approximate v^* realizability): The MDP M and the featuremap ϕ are such that $v^* \in_\varepsilon \mathcal{F}_\phi$

Assumption A2$_\varepsilon$ (approximate universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $v^\pi \in_\varepsilon \mathcal{F}_\phi$.

Action-value hints

We obtain new variants if we consider feature-maps that map state-action pairs to vectors. Concretely, (by abusing notation) let $\phi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^d$. Then, the analog of A1 is as follows:

Assumption B1 (q^*-realizibility): The MDP M and the featuremap ϕ are such that $q^* \in \mathcal{F}_\phi$

Here, as expected, $\mathcal{F}_\phi$ is defined as the set of functions that lie in the span of the feature-map. The analog of A2 is as follows:

Assumption B2 (universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $q^\pi \in \mathcal{F}_\phi$.

We can also introduce positive approximation errors $\varepsilon > 0$, which lead to B1$_\varepsilon$ and B2$_\varepsilon$:

Assumption B1$_\varepsilon$ (approximate q^*-realizibility): The MDP M and the featuremap ϕ are such that $q^* \in_\varepsilon \mathcal{F}_\phi$

Assumption B2$_\varepsilon$ (approximate universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $q^\pi \in_\varepsilon \mathcal{F}_\phi$.

One may wonder why not choose one of these assumptions? When one assumption implies another, then clearly there is a preference to choose the weaker assumption. But often, there is going to be a price and sometimes the assumptions are just not comparable.

Notes

Origin

The idea of using value function approximation in planning dates back to at least the 1960s if not earlier. I include some intriguing early references at the end. That these ideas already appeared at the down of computing where computers hardly even existed is quite intriguing.

Infinite spaces

Function approximation is especially appealing when the state space, or the action space, or both are "continuous" (i.e., they are a subset of a Euclidean space). In this case, the compression is "infinite". Experimental evidence suggests that function approximation can work quite well in the context of MDP planning in a surprisingly large number of different scenarios. When the spaces are infinite, all the "math" will still go through, except that occasionally one has to be a bit more careful. For example, one cannot clearly say that Φ is a matrix, but Φ can clearly be defined as a linear operator mapping $\mathbb{R}^d$ to the vector space of all real-valued functions over the (say) state space (when the feature map is also over states).

Where do the features come from?

It will be instructive to start with a special case. **Low-rank MDPs** are those where the transition kernel factorizes: For any s, a, s' state-action-state triple,

$$P(s'|s,a) = \langle \phi(s,a), \nu(s') \rangle$$

for some $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and $\nu(s') \in \mathbb{R}^d$. If in addition to the above,

$$r(s,a) = \langle \phi(s,a), \nu' \rangle \tag{2}$$

also holds for some $\nu' \in \mathbb{R}^d$, it is not hard to see that any action-value function lies in the space of the features ϕ.

But what are the cases when the transition kernel factorizes? (If the transition kernel factorizes with some feature map ϕ_0, one can always arrange for (2) to hold by adding an extra dimension to the feature map, filled with the values of the rewards.) A simple case is when state-action pairs can be clustered into non-overlapping groups such that for any two pairs $(s_1, a_1), (s_2, a_2)$ that belong to the same group, the transitions are identical: $P(\cdot|s_1, a_1) = P(\cdot|s_2, a_2)$. Assuming d groups number from 1 to d, $\phi_i(s,a)$ can be chosen as the indicator that (s,a) belongs to the ith group ($i \in [d]$).

Another interesting case which leads to a factored transition kernel is when the state-space is $\mathbb{R}^p$ with some $p > 0$ and the dynamics takes the form

$$S_{t+1} = f(S_t, A_t) + \eta_{t+1}$$

with some function f, and $(\eta_t)_t$ is a sequence of independent random variables with common density g. Then, the transition kernel takes the form $P(ds'|s,a) = g(s' - f(s,a))ds'$. The important point here is that the noise introduced is homoscedastic (does not change with (s,a)). Take, for example, the case when $g(x) = \frac{1}{\sqrt{2\pi}}\exp(-x^2/2)$, i.e., $(\eta_t)_t$ are standard normal random variables. It is well known then that

$$g(x-y) = \langle u(x;\cdot,\cdot), u(y;\cdot,\cdot) \rangle \,,$$

where

$$u(x;\omega,b) = \sqrt{2}\cos(\omega^\top x + b)$$

and for $n, m : \mathcal{D} \to \mathbb{R}$, $\mathcal{D} := \mathbb{R}^p \times [0, 2\pi]$,

$$\langle n, m \rangle = \int_{\mathbb{R}^p} \frac{1}{2\pi} \int_0^{2\pi} n(\omega,b)m(\omega,b)\ db \prod_{i=1}^p g(\omega_i)\, d\omega \,.$$

From this, we get

$$P(ds'|s,a) = g(s' - f(s,a))ds' = \langle u(s';\cdot,\cdot), u(f(s,a);\cdot,\cdot) \rangle ds' \,.$$

It follows that if we define $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{\mathcal{D}}$ via

$$(\phi(s,a))(\omega,b) = u(f(s,a);\omega,b)$$

then

$$P(ds'|s,a) = \langle \phi(s,a), u(s';\cdot,\cdot) \rangle ds' \,,$$

which is the same as above, except here ϕ is infinite dimensional. In a way, what happens here is that the noise introduces smoothness of the value functions. Smoothness of value functions can arise in some other ways. In the related topic of numerical computation of solutions of partial differential equations, Galerkin's method also starts from assuming that the solution lies in the span of some features. In the relavant literature, various methods have been proposed to find appropriate features (or, basis functions, as they are called there). The book of Quarteroni et. al. gives several methods for automating the construction of these basis functions, and they also make a connection to optimal control.

Nonlinear value function approximation

The most successful use of the idea of compressing value functions uses **neural networks**. Readers are most likely are already familiar with the ideas underlying neural networks. The hope here is that whatever we find in the case of linear function approximation will have implications in how to use nonlinear function approximation in MDP planning. In a way, the very first question is whether one can **decouple** the design of the planning algorithm from what function approximation technique it is used with. We will study this question by asking for planners that work with any feature map. If we find that we can identify planners that are performant no matter the feature map, the decoupling is successful and we can hope that the ideas will generalize to nonlinear function approximation. However, if we find that successful planners need to use intricate properties of the feature maps, then this is must be taken as a warning that complications may arise when the results are generalized to nonlinear function approximation. In any case, it appears to be a prudent strategy to first investigate the simpler, more straightforward linear case, before considering the nonlinear case.

Computation with advice/Non-uniform Computation

Computation with advice is a general approach in computer science where a problem of computing a map is changed to computing a map which has an additional input, the advice. Clearly, the approach taken here can be seen as a special case of computation with advice. There is also the closely related notion of non-uniform computation studied in computability/complexity theory. In non-uniform computation, the Turing machine, in addition to its input, also receives some "advice" string.

References

The classical reference is a paper of Bellman et al. from 1963, where they proposed to use linear function approximation in a specific context for approximating the optimal value functions (Bellman et al. 1963). Other early papers are by Daniel (1976) and Schweitzer and Seidmann (1985). In the latter paper, the authors generalized the earlier constructions of Bellman and others and, with modern terminology, they introduced fitted value iteration, fitted policy iteration and approximate linear programming as possible approaches.

The observation that homoscedastic noise makes it so that the transition kernel factorizes is due to Ren et al. (2022). The book of Quarteroni et al. (2016) describes various methods for automating the construction of basis functions for the solution of parametric family of partial differential equations.

- Richard Bellman, Robert Kalaba and Bella Kotkin. 1963. Polynomial Approximation–A New Computational Technique in Dynamic Programming: Allocation Processes. Mathematics of Computation, 17 (82): 155-161
- Daniel, James W. 1976. “Splines and Efficiency in Dynamic Programming.” Journal of Mathematical Analysis and Applications 54 (2): 402–7.
- Schweitzer, Paul J., and Abraham Seidmann. 1985. “Generalized Polynomial Approximations in Markovian Decision Processes.” Journal of Mathematical Analysis and Applications 110 (2): 568–82.
- Brattka, Vasco, and Arno Pauly. 2010. Computation with Advice. arXiv [cs.LO].
- Quarteroni, Alfio, Andrea Manzoni, and Federico Negri. “Reduced Basis Methods for Partial Differential Equations”. Springer International Publishing. 2016.
- Ren, T., T. Zhang, C. Szepesvári, and B. Dai. 2022. “A Free Lunch from the Noise: Provable and Practical Exploration for Representation Learning.” UAI. abstract

8. Approximate Policy Iteration

PDF Version

Note: On March 13, 2021, these notes were updated as follows:

1. Tighter bounds are derived; the old analysis was based on bounding ; the new analysis directly bounds , which leads to a better dependence on the approximation error;
2. Unbiased return estimates are introduced that use rollouts of random length.

One simple idea to use function approximation in MDP planning is to take a planning method that uses internal value functions and add a constraint that restrict the value functions to have a compressed representation.

As usual, two questions arise:

- Does this lead to an **efficient** planner? That is, can the computation be carried out in time polynomial in the relevant quantities, but not the size of the state space? In the case of linear functions the question is whether we can calculate the coefficients efficiently.
- Does this lead to an **effective** planner? In particular, how good a policy can we arrive at with a limited compute effort?

In this lecture, as a start into exploring the use of value function approximation in planning, we look at modifying policy iteration in the above described way. The resulting algorithm belongs to the family of **approximate policy iteration** algorithms, which consists of all algorithms derived from policy iteration by adding approximation to it.

We will work with linear function approximation. In particular, we will assume that the planner is given as a hint a feature-map . In this setting, since policy iteration hinges upon evaluating the policies obtained, the hint given to the planner is considered to be “good” if the (action-)value functions of **all** policies are well-represented with the features.

This means, that we will work under assumption B2 from the previous lecture, which we copy here for convenience. In what follows we fix .

Assumption B2 (approximate universal value function realizibility) The MDP and the featuremap are such that for any memoryless policy of the MDP, .

Recall that here the notation means that can be approximated up to a uniform error of using linear combinations of the basis functions underlying the feature-map :

For any policy ,

One may question whether it is reasonable to expect that the value functions of all policies can be compressed. We will come back to this question later.

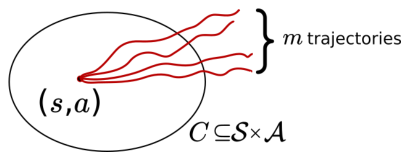
Approximate Policy Evaluation: Done Well

Recall that in phase of policy iteration, given a policy , the next policy is obtained as the policy that is greedy with respect to . If we found some coefficients such that

then when it comes to “using” policy , we could just use when an action is needed at state . Note that this action can be obtained at the cost of elementary operations, a small overhead compared to a table lookup (with idealized access times).

Hence, the main question is how to obtain this parameter in an efficient manner. To be more precise, here we want to control the uniform error committed in approximating .

To simplify the notation, let . A simple idea is **rolling out** with the policy from a fixed set to “approximately” measure the value of at the pairs in . For concreteness, let . Rolling out with policy this pair means using the simulator to simulate what would happen if we used policy for a number of consecutive time steps when the initial state is , the first



action , but for subsequent time steps the actions are chosen using policy for whatever states are encountered. If the simulation goes on for steps, this way we get trajectories starting in . For let the trajectory obtained be . Thus,

,

where , , and for , , and . The figure on the right illustrates these trajectories.

Given these trajectories, the empirical mean of the discounted sum of rewards along these trajectories is used for approximating :

—

Under the usual condition that the rewards are in the interval, the expected value of is in the vicinity of the and by averaging a large number of independent trajectories, we also achieve that the empirical means are tightly concentrated around their mean.

Using a randomization device, it is possible to remove the error ("bias") introduced by truncating the trajectories at a fixed time. For this, just let be independent **geometrically distributed** random variables with parameter , which are also independently chosen from the trajectories. By definition is the number of - parameter Bernoulli trials needed to get one success. With the help of these variables, define now by

—

Note that in the expression of the discount factor is eliminated. To calculate one can just perform a rollout with policy as before, just in each time step , after obtaining , draw a Bernoulli variable with parameter to decide whether the rollout should continue.

To see why the above definition works, fix and note that by definition, for , and thus . Therefore, All in all, this means, that we expect that if we solve for the **least-squares problem**

we expect to be a good approximation to . Or at least, we can expect this hold at the points of , where we are taking our measurements. The question is what happens **outside of** : That is, what guarantees can we get for **extrapolating** to points of . The first thing to observe that unless we are choosing carefully, there is no guarantee about the extrapolation error will be kept under control. In fact, if the choice of is so unfortunate that all the feature vectors for points in are **identical**, the least-squares problem will have many solutions.

Our next lemma gives an explicit error bound on the extrapolation error. For the coming results we slightly generalize least-squares by introducing a weighting of the various errors in . For this, let be a weighting function assigning a positive weight to the various error terms and let

be the minimizer of the resulting weighted squared-loss. A simple calculation gives that provided the (weighted) **moment matrix**

is nonsingular, the solution to the above weighted least-squares problem is unique and is equal to

From this expression we see that there is no loss of generality in assuming that the weights in the weighting function sum to one: . We will denote this by writing (here, refers to the fact that we can see as an element of a simplex). To state the lemma recall the notation that for a positive definite, matrix and vector ,

Lemma (extrapolation error control in least-squares): Fix any , , and such that the moment matrix is nonsingular. Define

Then, for any we have

Before the proof note that what his lemma tells us is that as long as we guarantee that the moment matrix is full rank, the extrapolation errors relative to predicting with some can be controlled by controlling

1. the value of ; and
2. the maximum deviation of the targets used in the weighted least-squares problem and the predictions with .

Proof: First, we relate to :

Then for a fixed ,

To get a sense of how to control the sum notice that if in the last sum was somehow replaced by , using the definition of could greatly simplify the last expression. To get here, one may further notice that having the term in absolute value squared would help. Now, to get the squares, recall Jensen's inequality, which states that for any convex function and probability distribution , . Of course, this also works when is a finitely supported, which is the case here. Thus, applying Jensen's inequality with , we thus get

Plugging this back into the previous inequality gives the desired result.

It remains to be seen of whether can be kept under control. This is the subject of a classic result of Kiefer and Wolfowitz:

Theorem (Kiefer-Wolfowitz): Let be finite. Let be such that the underlying feature matrix is rank . There exists a set and a distribution over this set, i.e. , such that

1. ;
2. —;

3 In the previous line, the inequality is achieved with equality and the value of is best possible under all possible choices of and .

We will not give a proof of the theorem, but we give references at the end where the reader can look up the proof. When is not full rank (i.e., is not rank), one may reduce the dimensionality (and the cardinality of reduces accordingly). The problem of choosing and such that is minimized is called the -optimal design problem in statistics. This is a specific instance of optimal experimental design.

Combining the Kiefer-Wolfowitz theorem with the previous lemma shows that least-squares amplifies the "measurement errors" by at most a factor of :

Corollary (extrapolation error control in least-squares via optimal design): Fix any full rank. Then, there exists a set with at most elements and a weighting function such that for any and any ,

where is given by

Importantly, note that **and are chosen independently of and** , that is, they are independent of the target. This suggests that in approximate policy evaluation, one should choose as in the Kiefer-Wolfowitz theorem and use the weighted moment matrix. This leads to

where is defined by Eq. and is defined by Eq. . We call this procedure **least-square policy evaluation based on rollouts from -optimal design points**, or LSPE- , for short. Note that we stick to the truncated rollouts, because this allows a simpler probabilistic analysis. That this properly controls the extrapolation error is as attested by the next result:

Lemma (LSPE- extrapolation error control): Fix any full-rank feature-map and take the set and the weighting function as in the Kiefer-Wolfowitz theorem. Fix an arbitrary policy and let and such that and assume that immediate rewards belong to the interval . Let be as in Eq. . Then, for any , with probability ,

Notice that that from the Kiefer-Wolfowitz theorem, and therefore nothing in the above expression depends on the size of the state space. Now, say we want to make the above error bound at most with some value of . From the above we see that it suffices to choose and so that

This, together with gives

Proof: In a nutshell, we use the previous corollary, together with Hoeffding's inequality and using that , which follows since the rewards are bounded in .

▶ Click here for the full proof.

In summary, what we have shown so far is that if the features can approximate well the action-value function of a policy, then there is a simple procedure (Monte-Carlo rollouts and least-squares estimation based on an optimal experimental design) to produce an reliable estimate of the action-value function of the policy. The question remains whether if we use these estimates in policy iteration, the whole procedure will still give good policies after a sufficiently large number of iterations.

Progress Lemma with Approximation Errors

Here we give a refinement of the geometric progress lemma of policy iteration that allows for "approximate" policy improvement steps. This previous lemma stated that the value function of the improved policy is at least as large as the Bellman operator applied to the value function of the policy to be improved. Our new lemma is as follows:

Lemma (Geometric progress lemma with approximate policy improvement): Consider a memoryless policy and its corresponding value function . Let be any policy and define via

Then,

Proof: First note that for the optimal policy , . We have

Using the value difference identity and that , we calculate

where the inequality follows because , the sum of positive linear operators, is a positive linear operator itself and hence is also monotone. Plugging the inequality obtained into gives

Taking the maximum norm of both sides and using the triangle inequality and that gives the desired result.

Approximate Policy Iteration

Notice that the progress lemma makes no assumptions about the origin of the errors. This motivates considering a generic version of **approximate policy iteration** where for in the th update set, the new policy is approximately greedy with respect to in that sense that

The progress lemma implies that the resulting sequence of policies will have value functions that converge to a neighborhood of where the size of the neighborhood is governed by the magnitude of the error terms .

Theorem (Approximate Policy Iteration): Let , be such that holds for all . Then, for any ,

Proof: Left as an exercise.

Consider now a version of approximate policy iteration where the sequence of policies is defined as follows:

That is, for each , is greedy with respect to .

Corollary (Approximate Policy Iteration with Approximate Action-value Functions): The sequence defined in is such that

Proof: To simplify the notation consider policies and functions over the state-action space such that and . We have where we used that is linear, monotone, and that is monotone, and both are nonexpansions in the maximum norm.

Hence, if is defined by then and the result follows from the previous theorem.

Global planning with least-squares policy iteration

Putting things together gives the following planning method:

```
1 Given the feature map , find and as in the Kiefer-Wolfowitz theorem
2 Let
3 For do
4     Roll out with policy for steps to get the targets where
      and
5     Solve the weighted least-squares problem given by Eq. to get .
6 Return
```

We call this method least-squares policy iteration (LSPI) for obvious reasons. Note that this is a **global planning method**: The method makes no use of an input state and the parameter vector returned can be used to get the policy (as in the method above).

Theorem (LSPI performance): Fix an arbitrary full rank feature-map and let . Assume that B2 holds. Then, for any , with probability at least , the policy which is greedy with respect to is -suboptimal with

In particular, for any , choosing so that policy is -optimal with

while the total computation cost is .

Thus, with a polynomial cost, LSPI with the specific configuration at the cost of polynomial computation cost, but importantly, with a cost that is independent of the size of the state space, can result in a good policy as long as , the worst-case error of approximating action-value functions of policies using the features provided, is sufficiently small.

Proof: Note that B2 and that is full rank implies that for any memoryless policy there exists a parameter vector such that (cf. Part 2 of Question 3 of Assignment 2). Hence, we can use the "LSPE extrapolation error bound" (cf.). By this result, a union bound and of course by B2 , we get that for any , with probability at least , for any ,

where we also used that . Call the quantity on the right-hand side in the above inequality .

Take the event when the above inequalities hold and for now assume this event holds. By the previous theorem, is -optimal with

To obtain the second part of the result, we split into two equal parts: is set to force the iteration error to be at most , while and are chosen to force the policy evaluation error to be at most . Here, to choose and , is again split into two equal parts. The details of this calculation are left to the reader.

Notes

Approximate Dynamic Programming (ADP)

Value iteration and policy iteration are specific instances of dynamic programming methods. In general, dynamic programming refers to methods that use value functions to calculate good policies. In **approximate dynamic programming** the methods are modified by introducing "errors" when calculating the values. The idea is that the origin of the errors does not matter (e.g., whether they come due to imperfect function approximation, linear, or nonlinear, or due to the sampling): The analysis is done in a general form. While here we met approximate policy iteration, one can also use the same ideas as shown here to study an approximate version of value iteration. A homework in problem set 2 asks you to study this method, which is usualy called **approximate value iteration**. In an earlier homework you were asked to study how linear programming can also be used to compute optimal value functions. Adding approximations we then get **approximate linear programming**.

What function approximation technique to use?

We note in passing that fans of neural networks should like that the general, ADP-style results, like the theorem in the middle of this lecture, can be also applied to the case when neural networks are used as the function approximation technique. However, one main lesson of the lecture is that to control extrapolation errors, one should be quite careful in how the training data is chosen. For linear prediction and least-squares fitting, optimal design gives a complete answer, but the analog questions are completely open in the case of nonlinear function approximation, such as neural networks. There is also a sizable literature that connects nonparametric techniques (an analysis friendly relative of neural networks) to ADP methods.

Concentrability coefficients and all that jazz

The idea of introducing approximate calculations has been introduced at the same time people got interested in Markov Decision Processes in the 1960s. Hence, the literature is quite enormous. However, the approach taken here which asks for error bounds where the algorithmic (not approximation-) error is uniformly controlled regardless of the MDP is quite recent and where the term that involves the approximation error is also uniformly bounded (for a fixed dimension and discount factor).

Earlier literature often presented bounds where the magnification factor of the approximation and the algorithmic error involved terms which depended on the MDP. Often these came in the form of "concentrability coefficients" (and yours truly was quite busy with working on these results a while ago). The main conclusion of this earlier analysis is that more stochasticity in the transitions means less control, less concentrability, which is advantageous for the ADP algorithms. While this makes sense and this indicates that these earlier results are complementary to the results presented here, the issue is that these results are quite pessimistic for example when the MDP is deterministic (as in this case the concentrability coefficients can be as large as the size of the state space).

While here we emphasized the importance of using a good design to control the extrapolation errors, in these earlier results, no optimal design was used. The upshot is that this saves the effort of coming up with a good design, but the obvious downside is that the extrapolation error may become uncontrolled. In the batch setting (which we will come back to later), of course, there is no way to control the sample collection, and this is in fact the setting where this earlier analysis was done.

The strength of hints

A critical assumption in the analysis of API was that the approximation error is controlled uniformly for all policies. This feels limiting. Yet, there are some interesting sufficient conditions when this assumption is clearly satisfied. In general, these require that the transition dynamics and the reward are both "compressible". For example, if the MDP is such that , the immediate reward as a function of the state-action pairs satisfies and the transition matrix, satisfies with some matrix , then for any policy policy , has a range which is a subset of . Since is the fixed-point of , i.e., , it follows that is also necessarily in the range space of . As such, and . MDPs that satisfy the above two constraints are called **linear in** (or sometimes, just "linear MDPs"). Exact linearity can be relaxed: If and , then for any policy , with —— . Nevertheless, later we will investigate whether this assumption can be relaxed.

The tightness of the bounds

It is not known whether the bound presented in the final result is tight. In fact, the dependence of on the is almost certainly not tight; in similar scenarios it has been shown in the past that replacing Hoeffding's inequality with Bernstein's inequality allows the reduction of this factor. It is more interesting whether the amplification factor of the approximation error, , is best possible. In the next lecture we will show that the approximation error amplification factor cannot be removed while keeping the runtime under control. In a later lecture, we will show that the dependence on cannot be improved either – at least for this algorithm. However, we will see that if the main concern is the amplification of the approximation error, while keeping the runtime polynomial (perhaps with a higher order though) then under B2 better algorithms exist.

The cost of optimal experimental design

The careful reader would not miss that to run the proposed method one needs to find the set and the weighting function . The first observation here is that it is not crucial to find the best possible pair. The Kiefer-Wolfowitz theorem showed that with this best possible choice, . However, if one finds a pair such that , the price of this is that wherever appears in the final performance bound, a submultiplicative factor of will also need to be introduced. This should be acceptable. In relation to this note that by relaxing this optimality requirement, the cardinality of can be reduced. For example, by introducing the factor of as suggested above allows one to reduce the cardinlity to ; which may actually be a good tradeoff as this can save much on the runtime.

However, the question still remains of who computes these (approximately) optimal designs and at what cost. While this calculation only needs to be done once and is independent of the MDP (just depends on the feature map), the value of these methods remains unclear because of this compute cost. General methods to compute approximately optimal designs needed here are known, but their runtime for our case will be proportional to the number of state-action pairs. In the very rare cases when simulating transitions is very costly but the number of state-action pairs is not too high, this may be a viable option. However, these cases are rare. For special choices of the feature-map, optimal designs may be known. However, this reduces the general applicability of the method presented here. Thus, a major question is whether the optimal experimental design can be avoided. What is known is that for linear prediction with least-squares, clearly, they cannot be avoided. One suspects that this is true more generally.

Can optimal designs be avoided while keeping the results essentially unchanged? Of particular interest would be if the feature-map would also be only "locally explored" as the planner interacts with the simulator. Altogether, one suspects that two factors contributed here for the appearance of optimal experimental design: One factor is that the planner is global: It comes up with a parameter vector that leads to a policy that can be used regardless of the state. The other (perhaps) factor is that the approach was based on simple "patching up" a dynamic programming algorithm with a function approximator. While this is a common approach, controlling the extrapolation errors in this approach is critical and is likely only possible with something like an optimal experimental design. As we shall see soon, there are indeed approaches that avoid the optimal experimental design step and which are based on online planning and they also deviate from the ADP approach.

Policy evaluation alternatives

The policy evaluation method presented here feels unsophisticated. It uses simple Monte-Carlo rollouts, with truncation, averaging and least-squares regression. The reinforcement learning literature offers many alternatives, such as the "temporal difference" learning type methods that are based on solving the fixed point equation . One can indeed try to use this equation to avoid the crude Monte-Carlo approach presented here, in the hope of reducing the variance (which is currently rather crudely upper bounded using the term in the Hoeffding bound). Rewriting the fixed point as , and then plugging in , we see that the trouble is that to control the extrapolation errors, the optimal design must likely depend on the policy to be evaluated (because of the appearance of).

Alternative error control: Bellman residuals

Let and be so that

Here, is called the "Bellman residual" of . The policy evaluation alternatives above aim at controlling these residuals. The reader is invited to derive the analogue of the "approximate policy iteration" error bound in for this scenario.

The role of in the Kiefer-Wolfowitz result

One may wonder about how critical is the presence of in the results presented. For this, we can say that it is not critical. Unweighted least-squares does not perform much worse.

Least-squares error bound

The error bound presented for least-squares does not use the full power of randomness. When part of the errors with are random, some helpful averaging effects can appear, which we ignored for now, but which could be used in a more refined analysis.

Optimal experimental design – a field on its own

Optimal exoerimental design is a subfield of statistics. The design considered here is just one possibility. In fact, this design which is called G-optimal design (G stands, uninspiringly, for the word "general"). The Kiefer-Wolfowitz theorem actually also states that this is equivalent to the D-optimal designs.

Lack of convergence

The results presented show convergence to a ball around the optimal target. Some people think this is a major concern. While having a convergent method may look more appealing, as long as one controls the size of the ball, I will not be too concerned.

Approximate value iteration (AVI)

Similarly to what is done here, one can introduce an approximate version of value-iteration. This is the subject of Question 3 of homework 2. While the conditions are different, the qualitative behavior of AVI is similar to that of approximate policy iteration.

In particular, as for approximate policy iteration, there are two steps to this proof: One is to show that the residuals can be controlled and the second is that if they are controlled then the policy that is greedy with respect to (say) is -optimal with controlled by . For this second part, we have the following bound:

where . The procedure that uses least-squares fitting to get the iterates is known under various names, such as **least-squares value iteration** (LSVI), **fitted Q-iteration** (FQI), **least-squares Q iteration** (LSQI). This proliferation of abbreviations and names is unfortunate, but there is not much that can be done at this stage. To add insult to injury, when neural networks are used to represent the iterates and an incremental stochastic gradient descent algorithm is used for "fitting" the weights of these networks by resampling old data from a "replay buffer", the resulting procedure is coined "Deep Q-Networks" (training), or DQN for short.

Bounds on the parameter vector

The Kiefer-Wolfowitz theorem implies the following:

Proposition: Let and be such that and . Then, there exist a matrix such that for

there exists such that the following hold:

1. , ;
2. ;
3. $-$.

Proof: Let be the -optimal design whose existence is guaranteed by the Kiefer-Wolfowitz theorem. Let be the underlying moment matrix. Then, by the definition of , .

Define and . The first property is clearly satisfied. As to the second property,

Finally, for the third property,

finishing the proof.

Thus, if one has access to the full feature-map then knowing that a function realized is bounded, one may as well assume that the feature map is bounded and the parameter vector is bounded just by $\overline{}$.

Regularized least-squares

The linear least-squares predictor given by a feature-map and data

predicts a response at via where

with

Here, by abusing notation for the sake of minimizing clutter, we use ,

. The problem is that may not be invertible (i.e., may not be defined as written above). "By continuity", it is nearly equally problematic when is ill-conditioned (i.e., its minimum eigenvalue is "much smaller" than its maximum eigenvalue). In fact, this leads to poor "generalization". One remedy, often used, is to modify by shifting it with a small constant multiple of the identity matrix:

Here, is a tuning parameter, whose value is often chosen based on cross-validation or with a similar process. The modification guarantees that is invertible and it overall improves the quality of predictions, especially when is tuned base on data.

Above, the choice of the identity matrix, while is common in the literature, is completely arbitrary. In particular, invertibility will be guaranteed if is replaced with any other positive definite matrix . In fact, the matrix one should use here should be one that makes small (while, say, keeping the minimum eigenvalue of at constant). That this is the choice that makes sense can be argued for by noting that with

the vector defined in is the minimizer of

and thus, the extra penalty has the least impact for the choice of that makes the norm of the smallest. If we only know that , by our previous note, a good choice is , where where is a -optimal design. Indeed, with this choice, . Note also that if we apply the feature-standardization transformation of the previous note, we have

showing that the choice of using the identity matrix is justified when the features are standardized as in the proposition of the previous note.

References

We will only scratch the surface now; expect more references to be added later.

The bulk of this lecture is based on

- Tor Lattimore, Csaba Szepesvári, and Gellért Weisz. 2020. "Learning with Good Feature Representations in Bandits and in RL with a Generative Model." ICML and

arXiv:1911.07676,

who introduced the idea of using -optimal designs for controlling the extrapolation errors. A very early reference on error bounds in "approximate dynamic programming" is the following:

- Whitt, Ward. 1979. "Approximations of Dynamic Programs, II." Mathematics of Operations Research 4 (2): 179–85.

The analysis of the generic form of approximate policy iteration is a refinement of Proposition 6.2 from the book of Bertsekas and Tsitsiklis:

- Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-Dynamic Programming. Athena Scientific, Belmont, Massachusetts, 1996.

However, there are some differences between the "API" theorem presented here and Proposition 6.2. In particular, the theorem presented here appears to capture all sources of errors in a general way, while Proposition 6.2 is concerned with value function approximation errors and errors introduced in the "greedification step". The form adopted here appears, for example, in Theorem 1 of a technical report of Scherrer, who also gives earlier references:

- Scherrer, Bruno. 2013. "On the Performance Bounds of Some Policy Search Dynamic Programming Algorithms." arxiv.

The earliest of these references is perhaps

- Munos, R. 2003. "Error Bounds for Approximate Policy Iteration." ICML.

Least-squares policy iteration appears in

- Lagoudakis, M. G. and Parr, R. Least-squares policy iteration. The Journal of Machine Learning Re-search, 4:1107–1149, 2003.

The particular form presented in this work though uses value function approximation based on minimizing the Bellman residuals (using the so-called LSTD method).

Two books that advocate the ADP approach:

- Powell, Warren B. 2011. Approximate Dynamic Programming. Solving the Curses of Dimensionality. Hoboken, NJ, USA: John Wiley & Sons, Inc.
- Lewis, Frank L., and Derong Liu. 2013. Reinforcement Learning and Approximate Dynamic Programming for Feedback Control. Hoboken, NJ, USA: John Wiley & Sons, Inc.

And a chapter:

- Bertsekas, Dimitri P. 2009. “Chapter 6: Approximate Dynamic Programming,” January, 1–118.

A paper that is concerned with API and least-squares methods, but uses concentrability is:

Antos, Andras, Csaba Szepesvári, and Rémi Munos. 2007. “Learning near-Optimal Policies with Bellman-Residual Minimization Based Fitted Policy Iteration and a Single Sample Path.” Machine Learning 71 (1): 89–129.

Optimal experimental design has a large literature. A nice book concerned with computation is this:

- M. J. Todd. Minimum-volume ellipsoids: Theory and algorithms. SIAM, 2016.

The Kiefer-Wolfowitz theorem is from:

- J. Kiefer and J. Wolfowitz. The equivalence of two extremum problems. Canadian Journal of Mathematics, 12(5):363–365, 1960.

More on computation here:

- E. Hazan, Z. Karnin, and R. Meka. Volumetric spanners: an efficient exploration basis for learning. Journal of Machine Learning Research, 17(119):1–34, 2016
- M. Grötschel, L. Lovász, and A. Schrijver. Geometric algorithms and combinatorial optimization, volume 2. Springer Science & Business Media, 2012.

The latter book is a very good general starting point for convex optimization.

That the features are standardized as shown in the notes is assumed (and discussed), e.g., in

- Wang, Ruosong, Dean P. Foster, and Sham M. Kakade. 2020. “What Are the Statistical Limits of Offline RL with Linear Function Approximation?” arXiv [cs.LG]. arXiv

which we will meet later.

9. Limits of query-efficient planning

PDF Version

In the last lecture we have seen that given a discounted MDP $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, a feature-map $\varphi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and a precomputed, suitably small core set, for any $\varepsilon' > 0$ target and any confidence parameter $0 \le \zeta \le 1$, interacting with a simulator of M, with at most $\text{poly}(\frac{1}{1-\gamma}, d, \mathrm{A}, \frac{1}{(\varepsilon')^2}, \log(1/\zeta))$, compute time, LSPI returns some weight vector $\theta \in \mathbb{R}^d$ such that with probability $1 - \zeta$, the policy that is greedy with respect to $q = \Phi\theta$ is δ-suboptimal with

$$\delta \le \frac{2(1+\sqrt{d})}{(1-\gamma)^2}\,\varepsilon + \varepsilon'\,, \qquad (1)$$

where ε is the error with which the features can approximate the action-value functions of the policies of the MDP:

$$\varepsilon = \varepsilon^*(M, \Phi) := \sup_{\pi \text{ memoryless}} \inf_{\theta \in \mathbb{R}^d} \|\Phi\theta - q^\pi\|_\infty\,. \qquad (2)$$

Here, following our earlier convention, Φ refers to the $|\mathcal{S} \times \mathcal{A}| \times d$ matrix that is obtained by stacking the feature vectors $\varphi^\top(s, a)$ of all possible state-action pairs on the top of each other in some fixed order. Setting ε' to match the first term in Eq. (1), we can keep the effort polynomial in the relevant quantities (including $1/\varepsilon$), but even in the limit of infinite computation, the best bound we can obtain is

$$\delta \le \frac{2(1+\sqrt{d})}{(1-\gamma)^2}\,\varepsilon\,. \qquad (3)$$

While it makes sense that with a reasonable compute effort δ cannot be better than ε or a constant multiple of ε, it is unclear whether the extra $\sqrt{d}/(1-\gamma)^2$ factor is an artifact of the proof. We may suspect that some power of $1/(1-\gamma)$ may be necessary, because even if we knew the parameter vector that gives the best approximation to q^*, the error incurred by acting greedily with respect to q^* could be as large as

$$\frac{\varepsilon}{1-\gamma}\,.$$

However, at this point, it is completely unclear whether the extra $\sqrt{d}$ factor is necessary. The main question asked in this lecture: Are the “extra” factors truly necessary in the above bound? Or are there some other polynomial runtime algorithms that are able to produce policies with smaller suboptimality?

In this lecture we will give a partial answer to this question: We will justify the presence of $\sqrt{d}$. We start with a lower bound that shows that when there is no limit on the number of actions, efficient algorithms are limited to $\delta = \Omega(\varepsilon\sqrt{d})$.

Query lower bound for MDPs with large action sets

For the statement of our results, the following definitions will be useful:

Definition (soundness): An online planner is (δ, ε)-sound if for any finite discounted MDP $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$ and feature-map $\varphi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ such that $\varepsilon^*(M, \Phi) \le \varepsilon$, when interacting with (M, φ), the planner induces a δ-suboptimal policy of M.

Definition (memoryless planner): Call a planner **memoryless** if it does not retain any information between its calls.

The announced result is as follows:

Theorem (Query lower bound: large action sets): For any $\varepsilon > 0$, $0 < \delta \le 1/2$, positive integer d and for any (δ, ε)-sound online planner $\mathcal{P}$ there exists a “featurized-MDP” (M, φ) with rewards in $[0, 1]$ with $\varepsilon^*(M, \Phi) \le \varepsilon$ such that when interacting with a simulator of (M, φ), the expected number of queries used by $\mathcal{P}$ is at least

$$\Omega\left(\exp\left(\frac{1}{32}\left(\frac{\sqrt{d}\varepsilon}{\delta}\right)^2\right)\right).$$

Note that if $\delta = \varepsilon$ or smaller, the number of queries is exponential in d. For the proof we need a result that shows that one can pack the d-dimensional unit sphere with exponential in d many vectors that are nearly orthogonal. The precise result, which is stated without proof, is as follows:

Lemma (Johnson-Lindenstrauss (JL) Lemma) For every $\tau > 0$ and integers d, k such that

$$\left\lceil \frac{8 \ln k}{\tau^2} \right\rceil \leq d \leq k$$

then there exists $v_1, \dots, v_k$ vectors of the d-dimensional unit sphere such that for all $1 \leq i < j \leq k$,

$$|\langle v_i, v_j \rangle| \leq \tau \,.$$

Note that for a fixed dimension d, the valid range for k is

$$d \leq k \leq \exp\left(\frac{d\tau^2}{8}\right) . \tag{4}$$

In particular, k can be "exponentially large" in d when τ is a constant. We can directly relate this lemma to our feature matrices. In particular, the lemma is equivalent to the following result:

Proposition (JL feature matrix): For any τ, d, k as in the JL lemma there exists a matrix $\Phi \in \mathbb{R}^{k \times d}$ such that for any $i \in [k]$,

$$\max_{i \in [k]} \inf_{\theta \in \mathbb{R}^d} \|\Phi\theta - e_i\|_\infty \leq \tau \,, \tag{5}$$

where e_i is the ith basis vector of standard Euclidean basis of $\mathbb{R}^k$, and in particular if $\varphi_i^\top$ is the ith row of Φ, $\|\Phi\varphi_i - e_i\|_\infty \leq \tau$ holds.

Proof: Choose $v_1, \dots, v_k$ from the JL lemma as the rows of Φ. Fix $i \in [k]$. Then, $\Phi v_i - e_i = (v_1^\top v_i, \dots, v_i^\top v_i, \dots, v_k^\top v_i)^\top - e_i = (v_1^\top v_i, \dots, 0, \dots, v_k^\top v_i)^\top$. Since by construction $|v_j^\top v_i| \le \tau$ for $j \ne i$, the statement follows. ■

Finally, we need a variation of the result of Question 6 of Assignment 0. This question asked for proving that any algorithm that identifies the single nonzero entry in a binary array of length k requires to look at at least $(k+1)/2 - 1/k$ entries of the array on expectation. A similar lower bound applies if we require the algorithm to be correct with, say, probability $1/2$:

Lemma (High-probability needle lemma): Let $p > 0$. Any algorithm that correctly identifies the single nonzero entry in any binary array of length k with probability at least p has the property that the expected number of queries that the algorithm uses is at least $\Omega(pk)$.

In fact, if q_k is the worst-case expected number of queries used by an algorithm that is correct with probability p then one can show that for $k \ge 2$, $q_k \ge p(\frac{k+1}{2} - \frac{1}{k})$.

Proof: Left as an exercise. ■

With this we are ready to give the proof of the theorem:

Proof (of the theorem): We only give a sketch.

Fix the planner $\mathcal{P}$ with the said properties. Let k be a positive integer to be chosen later. We construct a feature map $\varphi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and k MDPs $M_1, \dots, M_k$ that share $\mathcal{S} = \{s, s_{\text{end}}\}$ and $\mathcal{A} = [k]$ as state and action-spaces, respectively. Here s will be chosen as the initial state where the planners will be tested from and s_{end} will be an absorbing state with zero reward. The MDPs share the same deterministic transition dynamics: All actions in s end up in s_{end} with probability one and all actions taken in s_{end} end up in s_{end} with probability one. The rewards for actions taken in s_{end} are all zero. Finally, we choose the reward of MDP M_i in state s to be

$$r_a^{(i)}(s) = \mathbb{I}(a = i) r^* \,,$$

where the value of $r^* \in (0, 1]$ is left to be chosen later.

Then, denoting by A the action returned by the planner when called with state s, one can see that the value of the policy induced at s in MDP M_i is $r^* \mathbb{P}_i(A = i)$, where $\mathbb{P}_i$ is the distribution induced by the interconnection of the planner and MDP M_i. Thus, for $r^* = 2\delta$, the planner needs to return A so that $\mathbb{P}_i(A = i) \ge 1/2$. Hence, it needs at least $\Omega(k)$ calls by the high-probability needle lemma.

Finally, the JL feature matrix construction allows us to construct a feature-map for this MDP as the action-value functions take the form $q^\pi(s,a) = \mathbb{I}(a = i) r^*$, $q^\pi(s_{\text{end}}, a) = 0$ in this MDP. ■

A lower bound when the number of actions is constant

The previous result leaves open whether query-efficient planners exist with a fixed number of actions. Our next result shows that the problem does not get much easier in this setting either.

The result is stated for **fixed-horizon MDPs**. Given an MDP $M = (\mathcal{S}, \mathcal{A}, P, r)$, a policy π, a positive integer $h > 0$ and state $s \in \mathcal{S}$ of the MDP, let

$$v_h^\pi(s) = \mathbb{E}_s^\pi[\sum_{t=0}^{h-1} r_{A_t}(S_t)]$$

be the total reward collected by π when it is used for h steps. The action-value functions $q_h^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are defined similarly. The optimal h-step value function is

$$v_h^*(s) = \sup_\pi v_h^\pi(s)\,, \qquad s \in \mathcal{S}\,.$$

The Bellman optimality operator $T : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ is defined via

$$Tv(s) = \max_{a \in \mathcal{A}} r_a(s) + \langle P_a(s), v \rangle\,.$$

The policy evaluation operator $T_\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ of a memoryless policy π is

$$T_\pi v(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\left(r_a(s) + \langle P_a(s), v \rangle\right).$$

A policy π is h-step optimal if $v_h^\pi = v_h^*$. Also, π is greedy with respect to $v : \mathcal{S} \to \mathbb{R}$ if $T_\pi v = Tv$. The analogue of the fundamental theorem looks as follows:

Theorem (fixed-horizon fundamental theorem): We have $v_0^* \equiv \boldsymbol{0}$ and for any $h \ge 0$, $v_{h+1}^* = T v_h^*$. Furthermore, for any $\pi_0^*, \dots, \pi_h^*, \dots$ such that for $i \ge 0$, π_i^* is greedy with respect to v_i^*, for any $h > 0$ it holds that $\pi = (\pi_{h-1}^*, \dots, \pi_0^*, \dots)$ (i.e., the policy which in step 1 uses π_{h-1}^*, in step 2 uses π_{h-2}^*, …, in step h uses π_0^*, after which it continues arbitrarily) is h-step optimal:

$$v_h^\pi = v_h^* \,.$$

Proof: Left as an exercise. Hint: Use induction. ■

In the theorem our earlier notion of policies is slightly abused: π is only specified for h steps. In any case, according to this result for a fixed horizon $H > 0$, the natural analogue for memoryless policies are these H-step nonstationary memoryless policies. Let us denote the set of these by Π_H.

In the next result, we will only care about optimality with respect to a fixed initial state $s_0 \in \mathcal{S}$. Then, without loss of generality, we also assume that the set of states $\mathcal{S}_h$ reachable from s_0 in $h \ge 0$ steps are disjoint: $\mathcal{S}_h \cap \mathcal{S}_{h'} = \emptyset$ for $h \ne h'$ (why?). It follows that we can also find a **memoryless policy** π that is optimal **at** s_0: $v_H^\pi(s_0) = v_H^*(s_0)$. In fact, one can even find a memoryless policy that also satisfies

$$v_{H-i}^\pi(s) = v_{H-i}^*(s), \qquad s \in \mathcal{S}_i \tag{6}$$

simultaneously for all $0 \le i \le H-1$. Furthermore, the same holds for the action-value functions:

$$q_{H-i}^\pi(s,a) = q_{H-i}^*(s,a), \qquad s \in \mathcal{S}_i, a \in \mathcal{A}, 0 \le i \le H-1 \,. \tag{7}$$

Thus, the natural analogue that all action-value functions are well-approximated with some feature-map is that there are feature-maps $(\varphi_h)_{0 \le h \le H-1}$ such that for $0 \le h \le H-1$, $\varphi_h : \mathcal{S}_h \times \mathcal{A} \to \mathbb{R}^d$ and for any memoryless policy π, the $H-h$-step action value function of π, when restricted to $\mathcal{S}_h$, is well-approximated by the linear combination of the basis functions induced by φ_h. Since we will not need q_{H-h}^π outside of $\mathcal{S}_h$, in what follows, **we assume that these are restricted to** $\mathcal{S}_h$. Writing Φ_h for the feature matrix induced by φ_h (the rows of Φ_h are the feature vectors under φ_h for some ordering of the state-action pairs from $\mathcal{S}_h \times \mathcal{A}$), we redefine $\varepsilon^*(M,\Phi)$ as follows:

$$\varepsilon^*(M,\Phi) := \sup_{\pi \text{ memoryless}} \max_{0 \le h \le H-1} \inf_{\theta \in \mathbb{R}^d} \|\Phi_h \theta - q_{H-h}^\pi\|_\infty \,. \tag{8}$$

Since we changed the objective, we also need to change the definition of (δ, ε)-sound online planners: These planners now need to induce policies that are δ-suboptimal or better when evaluated with the H-horizon undiscounted total reward criterion from the designated start-state s_0 provided that the MDP satisfies $\varepsilon^*(M, \Phi) \le \varepsilon$. In what follows, we call these planners (δ, ε)**-sound for the H-step criterion**.

With this, we are ready to state the main result of this section:

Theorem (Query lower bound: small action sets, fixed-horizon objective): For $\varepsilon > 0$, $0 < \delta \le 1/2$ and positive integer d, let

$$u(d, \varepsilon, \delta) = \left\lfloor \exp\left(\frac{d(\frac{\varepsilon}{2\delta})^2}{8}\right) \right\rfloor .$$

Then, for any $\varepsilon > 0, 0 < \delta \le 1/2$, positive integers A, H, d such that $d \le \mathrm{A}^H$ and for any online planner $\mathcal{P}$ that is (δ, ε)-sound for MDPs with at most A actions and the H-step criterion, there exists a "featurized-MDP" (M, φ) with A actions and rewards in $[0, 1]$ such that when interacting with a simulator of (M, φ), the expected number of queries used by $\mathcal{P}$ is at least

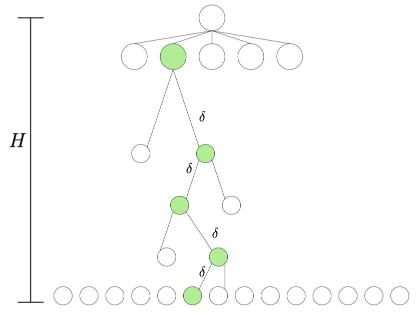
$$\tilde{\Omega}\left(\frac{u(d, \varepsilon, \delta)}{d(\varepsilon/\delta)^2}\right)$$

provided that $\mathrm{A}^H > u(d, \varepsilon, \delta)$ ("large horizons"), while it is

$$\tilde{\Omega}\left(\frac{\mathrm{A}^H}{H}\right)$$

otherwise ("small horizon").

In words, if the horizon is large enough, the previous exponential-in-d lower bound continues to hold, while for horizons that are smaller, a lower bound that is exponential in the horizon holds. Note that above $\tilde{\Omega}(\cdot)$ hides logarithmic terms. Note that the condition $d \le \mathrm{A}^H$ is reasonable: We do not expect the feature-space dimension to be comparable to A^H.



Proof: Fix a planner $\mathcal{P}$ with the required properties. We consider $k = \mathrm{A}^H$ MDPs $M_1, \dots, M_k$ that share the state space $\mathcal{S} = \cup_{0 \le h \le H} \mathcal{A}^h$ and action space $\mathcal{A}$. Here, by convention, $\mathcal{A}^0$ is a singleton with the single element $\perp$, which will play the role of the start state s_0. The transition dynamics are also shared by these MDPs: When in state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ is taken, the next state is $s' = (a)$ when $s = \perp$, while if $s = (a_1, \dots, a_h)$ with some $1 \le h \le H-1$ then $s' = (a_1, \dots, a_h, a)$ and when $h = H$ then the next state is s (ever state in $\mathcal{A}^H$ is absorbing). The MDPs differ in their reward functions. To describe the rewards let f be a bijection from $[k]$ to $\mathcal{A}^H$.

Now, fix $1 \le i \le k$ and define $(a_0^*, \dots, a_{H-1}^*)$ by $f(i) = (a_0^*, \dots, a_{H-1}^*)$. Let $s_0^* = s_0$, $s_1^* = (a_0^*)$, $s_2^* = (a_0^*, a_1^*)$, …, $s_H^* = (a_0^*, \dots, a_{H-1}^*)$. Then, in MDP M_i, $r_{a_{H-1}^*}(s_{H-1}^*) = 2\delta$ while $r_a(s) = 0$ for any other state-action pair.

Note that the optimal reward in H steps from $\perp$ is 2δ and the only policy that achieves this reward is the one that goes through the states in $s_0^*, s_1^*, \dots, s_{H-1}^*$. We can visualize MDP M_i as a tree, as seen on the figure on the right. The green nodes on the figure correspond to the states $s_0^*, s_1^*, \dots, s_{H-1}^*, s_H^*$. Note also that $\mathcal{S}_h = \mathcal{A}^h$ for $0 \le h \le H$.

We will now describe the action-value functions of the memoryless policies in M_i as this will be useful later. Fix $0 \le h \le H-1$. Then, q_{H-h}^π, by our convention, is defined over $\mathcal{S}_h$. Then, for any $s \in \mathcal{S}_h (= \mathcal{A}^h)$ and $a \in \mathcal{A}$,

$$q_{H-h}^\pi(s,a) = \begin{cases} 2\delta\,, & \text{if } h = H-1, s = s_{H-1}^*, a = a_{H-1}^*\,; \\ v_{H-h-1}^\pi(s_{h+1}^*)\,, & \text{if } h < H-1, s = s_h^*, a = a_h^*\,; \\ 0\,, & \text{otherwise}\,. \end{cases} \tag{9}$$

Note that here $0 \le v_{H-h-1}^\pi(g(s,a)) \le 2\delta$. We see that for each stage $0 \le h \le H-1$, there is only one state-action pair such that the value of q_{H-h}^π is nonzero, and in this case the value is in the $[0, 2\delta]$ interval.

Now, since the planner induces a policy with suboptimality δ, for the action A it returns it holds that $\mathbb{P}(A \ne a_0^*) \le 1/2$ (any other action than a_0^* incurs zero total expected reward in our construction). Then with $b \ge \log(2H)/\log(2) = \log_2(2H)$ fresh calls, by taking the action A_0 that is returned most often in these calls, we get $\mathbb{P}(A_0 \ne a_0^*) \le 1/(2H)$. Repeating this process in state $S_1 = g(s_0, A_0)$ we get action A_1 so that

$$\begin{aligned}\mathbb{P}(A_0 \ne a_0^* \text{ or } A_1 \ne a_1^*) &= \mathbb{P}(A_0 \ne a_0^*) + \mathbb{P}(A_0 = a_0^*, A_1 \ne a_1^*) \\ &\le \mathbb{P}(A_0 \ne a_0^*) + \mathbb{P}(A_1 \ne a_1^* | A_0 = a_0^*) \le \frac{1}{2H} + \frac{1}{2H}\,.\end{aligned}$$

Now, repeating again the process in state $S_2 = g(S_1, A_1)$ gives A_2, etc. Eventually, we get a sequence of actions $A_0, \dots, A_{H-1}$ such that $\mathbb{P}(A_0 \ne a_0^* \text{ or } \dots \text{ or } A_{H-1} \ne a_{H-1}^*) \le 1/2$.

By our previous argument (reduction to the "needle" problem), this whole process needs $\Omega(k)$ queries. If the expected number of queries issued by $\mathcal{P}$ is q, the expected number of queries issued here is $H\log_2(2H)q$. Hence,

$$q = \Omega\left(\frac{k}{\log_2(2H)H}\right).$$

Let us now consider a choice for $\Phi = (\Phi_h)_{0\le h\le H-1}$ such that $\varepsilon^*(M,\Phi) \le \varepsilon$. For Φ_h choose first a "JL feature matrix" $\tilde\Phi_h \in \mathbb{R}^{|\mathcal{S}_h|\times d}$ such that Eq. (5) holds. Then let $\Phi_h = \sqrt{2\delta}\tilde\Phi_h$. Choose $\theta_h = v^\pi_{H-h-1}(s^*_{h+1})\varphi_h(s_h^*, a_h^*)/(2\delta)$ if $h < H-1$ and choose $\theta_h = \varphi_h(s_h^*, a_h^*)$, otherwise. Then, by Eq. (9), for $(s,a) \ne (s_h^*, a_h^*)$, $|\varphi_h(s,a)^\top\theta_h - q_{H-h}(s,a)| \le |v^\pi(s^*_{h+1})|\,|\tilde\varphi_h(s,a)^\top\tilde\varphi_h(s_h^*,a_h^*)| \le 2\delta\tau$ and for $(s,a) = (s_h^*, a_h^*)$, $\varphi_h(s,a)^\top\theta_h = q_{H-h}(s,a)$. Hence, $\varepsilon^*(M,\Phi) \le \varepsilon$ holds if we set $\tau = \varepsilon/(2\delta)$.

From Eq. (4), $\tilde\Phi_h$ exists if $d \le k$ and

$$k \le u := \left\lfloor \exp\left(\frac{d(\frac{\varepsilon}{2\delta})^2}{8}\right)\right\rfloor.$$

Recall that $k = \mathrm{A}^H$. Thus, the required claim holds for the case when $\mathrm{A}^H \le u$ ("small horizon case"). In the opposite case ("large horizon"), let $\tilde H$ be the largest positive number such that $\mathrm{A}^{\tilde H} \le u$ holds. Repeating the above argument with horizon $\tilde H$ gives the lower bound $q = \Omega\left(\frac{\mathrm{A}^{\tilde H}}{\log_2(2\tilde H)\tilde H}\right) = \Omega\left(\frac{u}{\log_2(2\tilde H)\tilde H}\right)$, which finishes the proof. ■

Proof of the JL lemma

For completeness, we include a proof of the JL lemma. The proof uses the so-called probabilistic method The idea of this is that sometimes it is easier to establish the existence of some "good configuration" (like the nearly orthogonal vectors on the unit sphere in the JL lemma) by establishing that such a configuration has positive probability under some probability distribution over possible configurations.

In our case, this works as follows: Let $V_1, \dots, V_k$ be random vectors, each uniformly distributed on the d-dimensional unit sphere and so that the distinct vectors in this sequence are pairwise independent of each other. Take $i \neq j$. If we show that $|\langle V_i, V_j \rangle| \le \tau$ holds with probability at least $1 - 1/k^2$, by a union bound over the $k(k-1)/2$ pairs $1 \le i < j \le k$, it follows that $\max_{i \neq j} |\langle V_i, V_j| \le \tau$ holds with probability at least $1/2$, from which, the lemma follows.

Thus, it remains to show that the angle between the random vectors V_i and V_j is "small" with the claimed probability. Since the uniform distribution is rotation invariant and V_i and V_j are independent of each other, $\langle V_i, V_j \rangle$ has the same distribution as $\langle e_1, V_1 \rangle = V_{11} \in [-1, 1]$. To see this take a rotation R that rotates V_i to e_1; then $\langle V_i, V_j \rangle = \langle RV_i, R^{-1}V_j \rangle = \langle e_1, R^{-1}V_j \rangle$. Now, since R and V_j are independent of each other, $R^{-1}V_j$ is still uniformly distributed on the sphere, hence, $\langle e_1, R^{-1}V_j \rangle$ and $\langle e_1, V_1 \rangle$ share the same distribution.

A tedious calculation shows that for any $x \ge 6$,

$$\mathbb{P}(V_{11}^2 > x/d) \le \exp(-x/4)\,. \tag{10}$$

(The idea of proving this is to notice that if X is d-dimensional standard normal variable then $V = X/\|X\|_2$ is uniformly distributed on the sphere. Then, one proceeds using Chernoff's method.) The result now follows from (10) by choosing x so that $\tau^2 = x/d$ holds. ■

Notes

- The lower bound for the discounted case is missing the planning horizon. In the fixed-horizon setting, the lower bound is again missing the horizon when the horizon is large. It remains to be seen whether the extra "horizon terms" in Eq. (3) are necessary.
- In any case, the main conclusion is that even when we require "strong features", high-accuracy planning is intractable.
- The reader familiar with the TCS literature may recognize a close resemblance to questions studied there which are concerned with the existence of "fully polynomial time approximation schemes" (FPTAS).

- There are many open questions. For one, is there a counterpart of the second theorem for the discounted setting?

Bibliographical notes

The idea of using the Johnson-Lindenstrauss lemma in this context is due to Du, Kakade, Wang and Yang (DKWY, for short). The first theorem is a variant of a result from this paper. The second theorem is a variation of Theorem 4.1 from the paper of Du et al. mentioned above who prove the analoge result for global planners. The proof of the lemma also follows the proof given in this paper. The proof of inequality (10) is given in a paper of Dasgupta and Gupta, which also gives the "full version" of the Johnson-Lindenstrauss lemma which states that logarithmically many dimensions are sufficient to keep pairwise distances between a finite set of points.

- Dasgupta, Sanjoy; Gupta, Anupam (2003), "An elementary proof of a theorem of Johnson and Lindenstrauss" link, Random Structures & Algorithms, 22 (1): 60–65

The presentation of the first result which is for "bandits" (fixed horizon problems with $H = 1$) follows closely that of a paper by Lattimore, Weisz and yours truly. This, and a paper by van Roy and Dong were both prompted by the DKWY paper, whose initial version focused on the case when $\delta \ll \sqrt{d}\varepsilon$, which made the outlook for designing robust RL methods quite bleak. While it is true that in this high-precision regime nothing much can be done (unless further restricting the features), both papers emphasized that the hardness result disappears when the algorithm can deliver δ optimal policies with $\delta \gtrsim \sqrt{d}\varepsilon$.

0 Comments

1 Login ▾

G

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

• Share

Best Newest Oldest

Be the first to comment.

Subscribe Privacy Do Not Sell My Data

DISQUS

10. Planning under realizability

PDF Version

The lesson from the last lecture is that efficient planners are limited to induce policies whose suboptimaly gap is polynomially larger than the misspecification error of the feature-map supplied to the planner. We have also seen) that if we accept this polynomial in the feature-space-dimension error amplification, a relatively straightforward adaptation of policy iteration gives rise to a computationally efficient (global) planner – at least, when the planner is furbished with the solution to an underlying optimal experimental design problem. In any case, the planner is query efficient.

All this was shown in the context when the misspecification error is relative to the set of action value functions underlying all possible policies. In this lecture we look into whether this error metric could be changed so that the misspecification error is measured by how well the optimal action-value function, , is approximated by the features, while still retaining the positive result. As the negative result already implies that there are no efficient planners unless the suboptimality gap of the induced policy is polynomially larger than the approximation error, we look into the case when **the optimal action-value function is perfectly representable** with the features supplied to the planner. This assumption is also known as " -realizability", or, " linear realizability", if we want to be more specific about the nature of the function approximation technique used.

Planning under realizability

We consider fixed horizon online planning in large finite MDPs . As usual, the horizon is denoted by and we consider planning with a fixed initial state , as in the previous lecture. Let us denote by the states that are reachable from in steps. As before, we assume that when . Recall that in this case the action-value functions depend on the number of steps left, of the current stage. For a fixed , let be the optimal action-value function with stages in the process, stages left. Since we do not need the values of outside of , we abuse notation by redefining it restricted to this set.

Important note: The indexing of used here is not consistent with the indexing used in the previous lecture, where it was more convenient to index value functions based on the number of stages left.

The planner will be given a feature map for every stage such that . The realizability assumption means that

Note that we demand that **the same parameter vector is shared between all stages**. As it turns out, this makes our result stronger. Regardless, at the price of increasing the dimension from to , one can always assume that the parameter vector is shared. Since we will give a negative result concerning the query-efficiency of planners, we allow the planners access to the full feature-map: The negative result still applies even if the planner is allowed to perform any sort of computation with the feature-map during or before the planning process.

For , we call an online planner **-sound for the -step criterion** if for any MDP and feature map pair such that the optimal action-value function of is realizable with the features in the sense that holds, the planner induces a policy that is -suboptimal or better when evaluated with the -horizon undiscounted total reward criterion from the designated start-state in MDP . Note that this is very much the same as the previous soundness criterion, except that the definition of the approximation error is relaxed, while we demand .

The result below uses MDPs where the immediate reward (obtained from the simulator) can be random. The random reward is used to make the job of the planners harder and it allows us to consider MDPs with deterministic dynamics. (The result could also be proven for MDPs with deterministic rewards and random transitions.)

The usual **definition of MDPs with random transitions and rewards** is in a way even simpler: Such a (finite) MDP is given by the tuple where is a collection of distributions over state-reward pairs. In particular, for all state-action pairs , . Letting (i.e., is drawn from at random), we can recover as the distribution of and as the expected value of . That the reward can be random forces a change to the notion of the canonical probability spaces, since histories now also show include rewards, incurred in each time step . With appropriate modifications, we can nevertheless still introduce and the corresponding expectation operator, , as well. The natural definition of the value of a policy at state , say, in the discounted setting is then . However, it is easy to see that for any , , and, as such, nothing changes in the theoretical results derived so far.

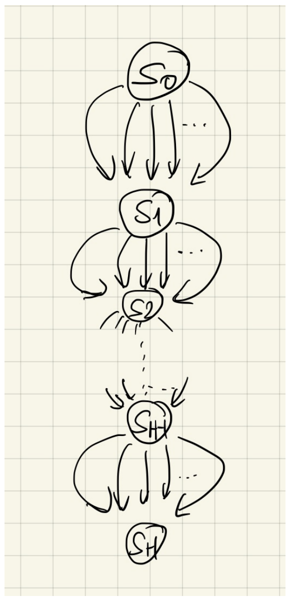
For reals, let . The main result of this lecture is as follows:

Theorem (worst-case query-cost is exponential under -realizability): For any large enough and any online planner that is -sound for the -horizon planning problem, there exists a triplet where is a finite MDP with random rewards taking values in and deterministic transitions, is a state of this MDP and is a -dimensional feature-map such that holds for the optimal action-value function and the expected number of queries that uses when interconnected with satisfies

---

Note that with random rewards with no control on their tail behavior (e.g., unbounded variance) it would not be hard to make the job of any planner arbitrarily hard. As such, it is quite important that the MDPs that are constructed for the result, the rewards, while random, lie in a fixed interval. Note that the specific choice of this interval does not matter: If there is a hard example with some interval, that example can be translated into another by shifting and scaling, and at the price of introducing an extra dimension in the feature map to account for the shifts. A similar comment applies to (which, nevertheless, needs to be scaled to the range of the rewards).

The main ideas of the proof



Rather than giving the full proof, we will just explain the main ideas behind it. At a high-level, the proof merges the ideas behind the lower bound for the small action-set case and the lower bound of the large action-set case. That is, we will consider an action set that is exponentially large in . In particular, we will consider action sets that have elements.

Note that because realizability holds, having a large action set but with a trivial dynamics (as in the lower bound in the last lecture) does not lead to the lower bound of the desired form. In particular, if the dynamics are trivial (i.e., , see the figure on the right) then the optimal action to be taken at does not depend on what actions are taken at later stages and can be efficiently found by just maximizing for the reward received in that stage, which can be done efficiently due to our realizability assumption, even in the presence of random rewards. Whether an example exists with only a few actions but with a more complicated dynamics remains open. With the construction provided here (which is based on tree dynamics and zero intermediate reward in the tree), this clearly fails, as we will make it clear below.

In any case, since the "chain dynamics" does not work, the next simplest approach is to have a tree, but with exponentially many actions in every node. Since this creates many many states (states at stage) the next question then is **how to ensure realizability**. There are two issues: We need to be able to keep the dimension fixed at at every stage and somehow we will need to have a way of controlling which action should be optimal at each state at each stage. Indeed, realizability means that we need to ensure that for all and

,

Here, stands for the state that is reached by taking action in state (in the tree, every node, or state is uniquely indexed by the action sequence that reaches it). Now, in the definition of , for all , we also have , which calls for the need to know the identity of the maximizing action. What is more, since the solution to the Bellman optimality equations is unique, if we guarantee that holds at all state-action pairs for with some features and parameter vectors, it also follows that for all , that is, is realizable with the features.

A simple approach to resolve all of these issues is to **let a fixed action be the optimal action at all the states**, together with using the JL features from the previous lecture (the identity of this action is of course hidden from the planner). In particular, the **JL feature-matrix lemma** from the previous lecture furnishes us with -dimensional unit vectors such that for ,

—

Fix these vectors. That should be optimal at all states is equivalent to that

In our earlier proof we used and . Will this still work? Unfortunately, it does not. The first observation is that from this it follows that for any , , ,

As such, for almost all the actions , we expect to be close to . Now, under this choice we also have that for all states and all stages . This creates essentially the same problem as what we saw above with the trivial chain dynamics. In particular, from we get that . As such, we expect to be close to either or (since is close to). Putting aside the issue that we wanted the immediate reward be in , we see that if the reward noise is not large, and thus the identity of can be obtained with just a few queries: The signal to noise ratio is just too good!

This problem replicates itself at the very last stage: Here, for any state , hence

for any pair. Unless we choose to be small, say, , a planner will succeed with fewer queries than in our desired bound.

This motivates us to introduce a **scaling of the features** (recall that the parameter vector is shared between the stages) with some scaling factors. For maximum generality, we allow for the scaling factor of the feature vector of to depend on itself (since states between stages are not shared, scaling can depend on the stage with this choice). Let be the scaling factor we intend to use with where we intend to **keep in a constant range** (so the scaling with the stage index works as intended) while we aim to use .

Now, we can explain the **need for many actions**. By the Bellman optimality equation we have that for any suboptimal action, ,

where uses that . From this we see that close to the initial state the reward gaps are of constant order. In particular, **if there were only a few actions per state**, a planner could identify the optimal action by finding the action whose reward is significantly larger than that of the others. By choosing to have many actions, the planner faces a "needle-in-a-haystack" situation, which makes their job hopeless even with perfect signal (no noise).

The next idea is to **force "clever" planners to only experiment with actions in the last stage**. Since here, the signal-to-noise ratio will be very poor, if we manage to achieve this, even clever planners will need to use a large number of queries. A simple way of forcing this is to **choose all the rewards while transitioning in the tree and taking suboptimal actions to be identically zero** except for stage , where, in accordance to our earlier plan, the rewards are chosen at random to ensure consistency but the signal to noise ratio will be poor.

Since the dynamics in the tree is known, and it is known that all rewards are zero with the possible exception of when using the optimal action (one of exponentially many actions and is thus hard to find), planners are either left with either solving the needle in a haystack problem of identifying the optimal action by randomly stumbling upon it, or they need to experiment with actions in the last stage. That the rewards are chosen to be identically zero is not critical: From the point of view of this argument, what is critical is that they are all the same.

It remains to be seen that consistency can be achieved and also that the optimal action at has a large value compared to the values of suboptimal actions at the same state. Here, we still face some challenges with consistency. Since we want the immediate rewards to belong to the interval, all the action values have to be nonnegative. As such, it will be easier if we introduce an additional bias component in the feature vectors, which we allow to scale with the stage.

To summarize, we let

while we propose to use

—

It remains to show that and can be satisfied with , while also keeping the suboptimal gap of at large, and while the last stage rewards ( ) are in and are of size as planned.

Assume for a moment that is optimal in all states, i.e., that holds. Then, is also optimal in state , hence, under , for any is equivalent to

where we also used that by assumption because . Plugging in the definitions,

— —

Define so that

— —

with — — (i.e., is a decreasing geometric sequence) This has two implications: simplifies to

— —

and also for the last stage rewards, from we get

— — — —

Clearly, if , since for , , while also .

With this, to satisfy , on the one hand we choose to define with the following "downward recursion" in the tree: For any in the tree and actions ,

— —

Note that this is consistent with . The next challenge is to show that stays within a constant range. In fact, with the above definition, this will not hold. In particular, when , the right-hand side can be as large as , which means that the scaling coefficients will exponentially increase with a base of . Note, however, that if , then provided that (which can be ensured at the root by choosing for all actions),

— — — — — — —

and thus will also hold.

Hence, we modify the construction so that **the definition is never needed for** . This is achieved by changing the dynamics: We introduce a special set of states, , the **exit lane**. Once, the process gets into this lane, there is now return and in fact all the remaining rewards up the end are zero. Specifically, all the actions in lead to state and we set the feature vector of all states in the exit-lane zero:

This way, regardless the choice of the parameter vector, we ensure that the Bellman optimality equations hold at these state and the optimal values are correctly set to zero.

The exit lane is introduced to remove the need to use with repeat actions. In particular, for any with some , say, (i.e., is obtained by following these actions) then if for , the next state is . Since the optimal value of is zero and we don't intend to introduce an immediate reward, we set

making the value of repeat actions zero. The next complication is that this ruins our plan to keep optimal at all states: Indeed, could be applied multiply times in a path from to a leaf of the tree, and by the second application, the new rule forces the value of to be zero. Hence, we need to modify this rule when the action is .

Clearly, whether a suboptimal action, or is repeated is problematic for the recursive definition of . Hence, it is better if is also forced to use the exit lane. Thus, if is used in with , the next state is . However, we do not zero out , but keep the recursive definition and we rather introduce an immediate reward to match . It is not hard to check that this reward is also in the range. Note that here if then by definition . This completes the description of the structure of the MDPs.

That the action gap at is large follows from the choice of the JL feature vectors.

It remains to be seen that is indeed the optimal action at any state. This boils down to checking that for , . When is a repeat action, this is trivial. When is not a repeat action, we have

where we used that and and thus — by the choice of and since .

Let denote the MDP constructed this way when the optimal action is (the feature maps, of course, are common between these MDPs). For a formal proof, one also needs to argue that planners that do not use many queries cannot distinguish between these MDPs. Intuitively, this is because such planners will receive, with high probability, identical observations under different MDPs in this class. As such, these planners can at best randomly choose an action ("needle in a haystack") and since in MDP only action incurs high values, they cannot induce a policy with a near-optimal value.

Computation with many actions

In the construction given the number of actions was allowed to scale exponentially with the dimension. The above proof would show a separation between the query and computation complexity of planning, if one could demonstrate that there is a choice of the JL feature vectors when the optimization problems

admits a computationally efficient solver regardless of the choice of and (for simplicity, we suppress dependence on). Whether such a solver exist will depend on the choice of the feature-map and this is a fascinating question on its own. One approach to arrive at such a solver is to rewrite this problem as the problem of finding

where is the convex hull of the feature vectors . Provided that this problem admits an efficient solution and given any extreme point of , we can efficiently recover an action such that (this amounts to "inverting" the feature map), the first problem can also be solved efficiently.

Note that is a linear optimization problem over a convex set and the question whether this problem admits an efficient solver lies at the heart of computer science. The general lesson is that the answer can be expected to be yes when has some "convenient" description other than the one that is used to define it. The second problem of inverting the feature map is known as the "decomposition problem" and the same conclusions hold for this problem.

Notes

- It is possible to modify the construction to make it work in the discounted setting. The paper cited below shows how.
- Back to the finite horizon setting, for an upper bound, one can employ the **least-squares value iteration algorithm with -optimal** design (LSVI-G), which we have met in Homework 2. What results is that to get a -sound (global) planner with this approach,

  queries are sufficient (and the compute cost is also of similar order). We see that as far as the exponents in the lower and upper bounds are concerned, in the upper bound the exponent is while in the lower bound it is . Thus, there remains a logarithmic gap between them when , while the **gap is unbounded** when , i.e., for **long horizon problems**. In particular, in the constant dimension and long-horizon featurized planning problem, the LSVI-G algorithm seems to be suboptimal because it calculates the optimal action-value function stage-wise. One conjectures that the upper bound for LSVI-G is tight, while the lower bound in this lecture is also essentially correct. This would means that there is an alternate algorithm that could perform much better than LSVI-G in large-horizon planning with constant feature-dimension. Clearly, for the specific construction used in this lecture, a planner that tries all actions, say at , will find the optimal action and the **cost of this planner is independent of the horizon**. Hence, at least in this case, the lower bound can be matched with an alternate algorithm. One may think that this problem is purely of theoretical interest. To counter this note that long-horizon planning is a really important practical question: Many applications require thousands of steps, if not millions, while perhaps the feature space dimension does not need to be very large. Whether there exist an algorithm that works better than LSVI-G thus remains to be a fascinating open problem with good potential for having a real impact on applications.
- For infinite horizon undiscounted problems and realizability, there is a simple example that shows that with actions and -dimensional features, any query efficient planner that guarantees a constant suboptimality gap needs queries per state. This is based on a shortest path problem on a regular grid. Here, the obstruction is simply algebraic: There is no noise in either the transitions or the rewards.

Bibliographical notes

This lecture is entirely based on the paper

- Weisz, Gellert, Philip Amortila, and Csaba Szepesvári. 2020. "Exponential Lower Bounds for Planning in MDPs With Linearly-Realizable Optimal Action-Value Functions.",

which is available on arXiv and which will also soon appear at ALT.

The second lower for the undiscounted setting mentioned in the notes is from

- Weisz, Gellert, Philip Amortila, Barnabás Janzer, Yasin Abbasi-Yadkori, Nan Jiang, and Csaba Szepesvári. 2021. "On Query-Efficient Planning in MDPs under Linear Realizability of the Optimal State-Value Function."

available on arXiv.

A beautiful book that is a very good source on reading about the linear optimization problem mentioned above is

- Grotschel, Martin, László Lovász, and Alexander Schrijver. 1993. Geometric Algorithms and Combinatorial Optimization. Vol. 2. Algorithms and Combinatorics. Berlin, Heidelberg: Springer Berlin Heidelberg.

11. Planning under v^* realizability (TensorPlan I.)

PDF Version

In the last lecture we saw that under q^* linear realizability, query-efficient fixed-horizon online planning with a constant suboptimality gap is intractable provided that there is no limit on the number of actions. In particular, the MDPs that were used to show intractability use $e^{\Theta(d)}$ actions, where d is the dimension of the feature-map that realizes the optimal action-value function. At the end of the lecture, we also noted that intractabality also holds for undiscounted infinite horizon problems under v^* linear realizability in the regime when the number of actions scales linearly with d. In this lecture we further dissect v^* realizability, but return to the fixed horizon setting and we will consider the case when the number of actions is fixed. As it turns out, in this case, query-efficient online planning is possible.

Before giving the details of this result, we need to firm up some and refine other definitions. First, v^* **realizability** under a feature map $\phi = (\phi_h)_{0 \le h \le H-1}$ in the H-horizon setting means that

$$\inf_{\theta \in \mathbb{R}^d} \max_{0 \le h \le H-1} \|\Phi_h \theta - v_h^*\|_\infty = 0\,, \qquad (1)$$

where v_h^* is the optimal-value function when $H-h$ steps are left (in particular, $v_H^* = \mathbf{0}$). Again, this uses the indexing introduced in the previous lecture. In what follows, without the loss of generality we assume that the feature map is such that all the feature-vectors lie within the a (2-norm) ball of radius one. When realizability holds with a parameter vector bounded in 2-norm by B, we say that v^* is B-realizable under the feature map ϕ.

We also slightly modify the interaction protocol between the planner and the simulator, as shown on the figure below. The main new features are introducing stages, and restricting the planners to access states and features only through local calls to the simulator.

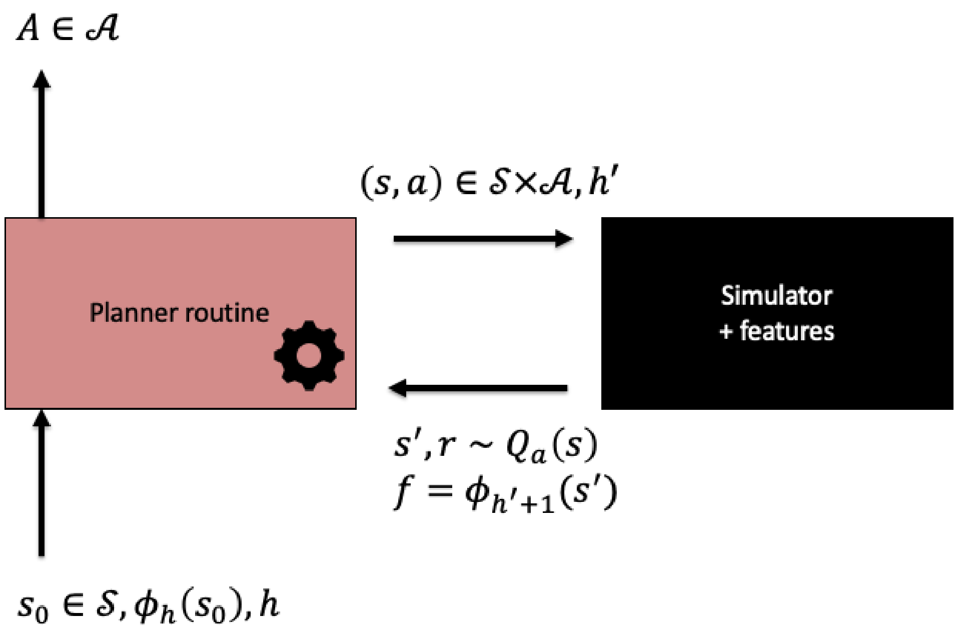


Illustration of the interaction protocol between the planner and the simulator.

Because in fixed-horizon problems the stage index influences what actions should be taken, the planner is called with an initial state s_0 and a stage index h. For defining the policy induced the planner, it is assumed that the planner is first called with $h = 0$ at some state, then it is called with $h = 1$ with a state obtained following a transition by taking the action returned by the planner, etc. While interacting with the simulator, the planner is restricted to use only states that it has encountered before. Also, the planner can feed a stage index to the simulator, to get the features of the next state corresponding to the incremented input stage index. There is no other access to the features. Note also that just like in the previous lecture, we allow the MDPs to generate random rewards.

In this setting a δ-sound planner is one which, under the above protocol, induces a policy of the MDP whose simulator it interacts with which is at most δ-suboptimal.

Theorem (query-efficient planning under v^*-realizability): For any integers $A, H > 0$ and reals $B, \delta > 0$, there exists an online planner $\mathcal{P}$ with the following properties:

1 The planner $\mathcal{P}$ is δ-sound for the H-horizon planning problem and the class of MDP-feature-map pairs (M, ϕ) such that v^* is B-realizable under ϕ and M has at most A actions and its rewards are bounded in $[0, 1]$;
2 The number of queries used by the planner in each of its call is at most

$$\mathrm{poly}\left(\left(\frac{dH}{\delta}\right)^A, B\right)$$

Note that for $A > 0$ fixed the query-cost is polynomial in d, H, $1/\delta$ and B. It remains to be seen whether this bound can be improved. However, this is somewhat of a theoretical question as under v^*-realizability, even if the coefficients $\theta \in \mathbb{R}^d$ that realize v^* are known, in the lack of extra information, one needs to perform $\Theta(A)$ simulation calls to be able to get good approximations to the action-value function q^*, which seems necessary for inducing a good policy. Hence, the query cost must scale at least linearly with A, hence, no algorithm is expected to be even query-efficient when the number of actions is large.

TensorPlan: An optimistic planner

The planner that is referred to in the previous theorem is called **TensorPlan**. The reason for this name will become clear after we describe the algorithm.

TensorPlan belongs to the class of optimistic algorithms. Since knowing θ^*, the parameter vector that realizes v^*, would be sufficient for acting near-optimally, the algorithm aims to find a good approximation to this vector.

A suitable estimate is constructed in a two-step process:

1. The algorithm maintains a non-empty "hypothesis" set $\Theta \subset \mathbb{R}^d$, which contains those parameter vectors that are **consistent** with the data that the algorithm has seen. The details of the construction of this set are at the heart of the algorithm and will come soon.
2. Given Θ, an estimate θ^+ is produced by solving a maximization problem:

$$\theta^+ = \arg\max_{\theta\in\Theta} \phi_0(s_0)^\top \theta\,. \qquad (2)$$

Here, s_0 is the initial state of the episode, i.e., this is the state the planner is called when $h = 0$. Recalling that $\phi_0(s_0)^\top \theta^* = v_0^*(s_0)$, we see that provided that $\theta^* \in \Theta$,

$$v_0(s_0;\theta^+) \ge v_0^*(s_0)\,,$$

where, for convenience, we introduce $v_h(s;\theta) = \phi_h(s)^\top\theta$. When θ^+ is close enough to θ^*, one hopes that the policy induced by θ^+ will be near-optimal. Hence, the approach is to

“roll out” with the induced policy (using the simulator) and verify whether during the rollout the data received is consistent with the Bellman equation, and as a result of this, also whether the episode return observed is close to $v_0(s_0;\theta^+)$. When a contradiction to any of these is detected, the data can be used to shrink the set Θ of consistent parameter vectors.

The approach described leaves open the question of what we mean by a policy “induced” by θ^+. The naive approach is to base this on the Bellman optimality equation, which states that

$$v_h^*(s) = \max_a r_a(s) + \langle P_a(s), v_{h+1}^* \rangle \tag{3}$$

holds for $h = 0, 1, \dots, H-1$ with $v_H^* = \mathbf{0}$. If $\theta^+ = \theta^*$, $v_h(\cdot;\theta^+)$ will also satisfy this equation and thus one might define the policy induced by θ^+ that achieves the maximum above when v_{h+1}^* is replaced by $v_{h+1}(\cdot;\theta^+)$. Consistency of θ^+ would also mean checking whether (3) holds (approximately) when $v_\cdot^*(\cdot)$ is replaced in this equation by $v_\cdot(\cdot;\theta^+)$, which, one may imagine can be checked by generating data from the simulator.

While this may approach work, it is not easy to see whether it does. (It is open problem whether this works!) TensorPlan defines induced policies and consistency slightly differently. The changed definition allows not only for proving that TensorPlan is query-efficient, but it even makes the guarantees for TensorPlan stronger than what was announced above in the theorem.

What makes the analysis of the algorithm that is based on the Bellmean optimality equation difficult is the presence of the maximum in this equation. Hence, TensorPlan removes this maximum. Accordingly, the policy induced by θ^+ is defined as any policy π_{θ^+} which in state s and stage h chooses **any action** $a \in \mathcal{A}$ which ensures that

$$v_h(s;\theta^+) = r_a(s) + \langle P_a(s), v_{h+1}(\cdot;\theta^+) \rangle \,. \tag{4}$$

If there is no such action, π_θ is free to choose any action. We say that **local consistency** holds at (s, h, θ^+) when there exists an action $a \in \mathcal{A}$ such that (4) holds.

If there are multiple actions that satisfy (4), any of them will do: Choosing the maximizing action is not enforced. However, when v^* is realizable and $\theta^+ = \theta^*$, any action that satisfies (4) will be a maximizing action and the policy induced will be optimal.

The advantage of the relaxed notion of induced policy is that with this choice, TensorPlan **will also be able to compete with any deterministic policy whose value-function is**

realizable. This expands the scope of TensorPlan: Perhaps the optimal value function is not realizable with the features handed to TensorPlan, but if there is any deterministic policy whose value-function is realizable with them, then TensorPlan will be guaranteed to produce almost as much as reward as that policy. In fact, it will produce nearly as much reward as the policy that achieves the best value.

TensorPlan

To summarize, after generating a hypothesis θ^+, TensorPlan will run a number of rollouts using the simulator so that for each state s encountered TensorPlan first finds an action a satisfying (4). If this succeeds, the rollout continues by TensorPlan getting a next state from the simulator at (s, a, h) and h is incremented. This continues up to $h = H$, which ends a rollout. TensorPlan will run m rollouts of this type and if all of them succeeds, TensorPlan stops and will use the parameter vector θ^+ in the rest of the episode and the same policy π_{θ^+} as used during the rollouts. If during the rollouts an inconsistency is detected, TensorPlan will decrease the hypothesis set Θ and continue with a next experiment.

It remains to be seen why TensorPlan (1) stops with a bounded number of queries and (2) why it is sound.

Boundedness

We start with boundedness. This is where the change of how policies are induced by parameters is used in a critical manner. Introduce the discriminants:

$$\Delta(s, a, h, \theta) = r_a(s) = \langle P_a(s)\phi_{h+1}, \theta\rangle - \phi_h(s)^\top \theta \,.$$

Note that $\Delta(s, a, h, \theta)$ is just the difference between the right-hand and the left-hand side of (4), where we plugged in the definition v_h and v_{h+1} and we define

$$P_a(s)\phi_{h+1} = \sum_{s' \in \mathcal{S}} P_a(s, s')\phi_{h+1}(s')\,;$$

thus $P_a(s)\phi_{h+1}$ is the "expected next feature vector" given (s, a). Then, by definition, local consistency holds for (s, h, θ) if and only if there exists some action $a \in \mathcal{A}$ such that $\Delta(s, a, h, \theta) = 0$. Exploiting that the product of numbers is zero if and only if some of them is zero, we see that local consistency is equivalent to

$$\prod_{a \in \mathcal{A}} \Delta(s, a, h, \theta) = 0\,. \qquad (5)$$

The reason this purely algebraic reformulation of local consistency is helpful is because the product of the discriminants can be see as a **linear function of the A-fold tensor product** of $(1, \theta^\top)^\top$.

To see why this holds, it will be useful to introduce some extra notation: For a real r and a finite-dimensional vector u, we will denote by $\overline{ru}$ the vector $(r, u^\top)^\top$ (i.e., adding r to the first position and shifting down all other entries in u). With this notation, we can write the discriminants as an inner product:

$$\Delta(s,a,h,\theta) = \langle \overline{r_a(s)\,(P_a(s)\phi_{h+1} - \phi_h(s))}, \overline{1\,\theta} \rangle$$

Now, recall that the tensor product $\otimes$ of vectors satisfies the following property:

$$\prod_a \langle x_a, y_a \rangle = \langle \otimes_a x_a, \otimes_a y_a \rangle\,,$$

where the inner product between two tensors is defined in the usual way, by overlaying them and then taking the sum of the products of the entries that are on the top of each other.

Based on this identity, we see that (5), and thus local consistency, is equivalent to

$$\langle \underbrace{\otimes_a \overline{r_a(s)\,(P_a(s)\phi_{h+1} - \phi_h(s))}}_{D(s,h)}, \underbrace{\otimes_a \overline{1\,\theta}}_{F(\theta)} \rangle = 0\,.$$

Note that while $F(\theta) \in \mathbb{R}^{(d+1)^A}$ is a nonlinear function of θ, the above equation is **linear in $F(\theta)$**.

Imagine for a moment that the data $D(s,h)$ above can be obtained with no errors and assume that $v^*$ is realizable. Let $k = (d+1)^A$. We can think of both $D(s,h)$ and $F(\theta)$ taking values in $\mathbb{R}^k$ (this corresponds to "flattening" these tensors).

TensorPlan can be seen as an algorithm that generates a sequence $(\theta_1, x_1), (\theta_2, x_2), \dots$ such that $\theta_i \in \mathbb{R}^d$ is the ith hypothesis that TensorPlan chooses, $x_i \in \mathbb{R}^k$ is the ith data of the form $D(s,h)$ with some (s,h) where TensorPlan detects an inconsistency. When inconsistency is detected, the hypothesis set is shrunk:

$$\Theta_{i+1} = \Theta_i \cap \{\theta \,:\, F(\theta)^\top x_i = 0\},$$

and θ_{i+1} is chosen in Θ_{i+1} by (2). Together with $\Theta_1 = B_2^d(B)$ (the ℓ^2 ball of radius B in $\mathbb{R}^d$), we have that for $i > 1$,

$$\Theta_i = \{\theta \in B_2^d(B) \,:\, F(\theta)^\top x_1 = 0, \dots, F(\theta)^\top x_{i-1} = 0\}.$$

Let $f_i = F(\theta_i)$. By its construction, for any $i \ge 1$, $\theta_i \in \Theta_i$ and hence f_i is orthogonal to $x_1, \dots, x_{i-1}$. Also by its construction, x_i is **not** orthogonal to f_i. Because of this, x_i cannot lie in the span of $x_1, \dots, x_{i-1}$ (if it did, it would be orthogonal to f_i). Hence, the vectors $x_1, x_2, \dots$ are linearly independent. As there are at most k linearly independent vectors in $\mathbb{R}^k$, Tensorplan will generate at most k of these data vectors (in fact, for TensorPlan, this is $k-1$, can you explain why?). This means that after at most k "contradictions" to local consistency, TensorPlan will cease to detect more inconsistencies and thus it will stop.

Soundness

It remains to be seen that TensorPlan is sound. Let θ^+ be the parameter vector that TensorPlan generated when it stops. This means that during the m rollouts, TensorPlan did not detect any inconsistencies.

Take a trajectory $S_0^{(i)}, A_0^{(i)}, \dots, S_{H-1}^{(i)}, A_{H-1}^{(i)}, S_H^{(i)}$ generated during the ith rollout of m rollouts. Since there is no inconsistency along it, for any $0 \le t \le H-1$ we have

$$r_{A_t^{(i)}}(S_t^{(i)}) = v_t(S_t^{(i)}; \theta^+) - \langle P_{A_t^{(i)}}(S_t^{(i)}), v_{t+1}(\cdot; \theta^+)\rangle \,. \tag{6}$$

Hence, with probability $1-\zeta$,

$$\begin{aligned}
v_0^{\pi_{\theta^+}}(s_0) &\ge \frac{1}{m}\sum_{i=1}^m \sum_{t=0}^{t-1} r_{A_t^{(i)}}(S_t^{(i)}) - H\sqrt{\frac{\log(1/\zeta)}{2m}} \\
&= \frac{1}{m}\sum_{i=1}^m \sum_{t=0}^{t-1} v_t(S_t^{(i)}; \theta^+) - \langle P_{A_t^{(i)}}(S_t^{(i)}), v_{t+1}(\cdot; \theta^+)\rangle - H\sqrt{\frac{\log(2/\zeta)}{2m}} \\
&\ge \frac{1}{m}\sum_{i=1}^m \sum_{t=0}^{t-1} v_t(S_t^{(i)}; \theta^+) - v_{t+1}(S_{t+1}^{(i)}; \theta^+) - (H+2B)\sqrt{\frac{\log(2/\zeta)}{2m}} \\
&= v_0(s_0; \theta^+) - (H+2B)\sqrt{\frac{\log(2/\zeta)}{2m}}\,,
\end{aligned}$$

where the first inequality is by Hoeffding's inequality and uses that rewards are bounded in $[0,1]$, the equality after it uses (6), the second inequality is again by Hoeffding's inequality and uses that

$$\langle P_{A_t^{(i)}}(S_t^{(i)}), v_{t+1}(\cdot; \theta^+)\rangle = \mathbb{E}[v_{t+1}(S_{t+1}^{(i)}; \theta^+) | S_t^{(i)}, A_t^{(i)}]$$

and that v_t is bounded between $[-B, B]$ (note that we could truncate v_t to $[0, H]$ to replace $H + 2B$ above by $2H$), while the last equality uses that $v_H(\cdot; \theta^+) = \mathbf{0}$ by definition and that $S_0^{(i)} = s_0$ by definition. Setting m high enough ($m = \tilde{O}((H + B)^2/\delta^2)$) we can guarantee

$$v_0^{\pi_{\theta^+}}(s_0) \geq v_0(s_0; \theta^+) - \delta.$$

We now argue that this implies soundness.

Letting $\Theta^\circ \subset B_2^d(B)$ be the set of B-bounded parameter vectors θ such that for some deterministic policy π, $v^\pi = \Phi\theta$. By the definition of D and F, for any $i \geq 1$, $\Theta^\circ \subset \Theta_i$ (no correct hypothesis is ever eliminated). It also follows that at any stage of the process,

$$v_0(s_0; \theta^+) \geq \max_{\theta \in \Theta^\circ} v_0^{\pi_\theta}(s_0).$$

Hence, when TensorPlan stops with parameter θ^+, with high probability,

$$v_0^{\pi_{\theta^+}}(s_0) \geq v_0(s_0; \theta^+) - \delta \geq \max_{\theta \in \Theta^\circ} v_0^{\pi_\theta}(s_0) - \delta\,.$$

In particular, if v^* is B-realizable, $v_0^{\pi_{\theta^+}}(s_0) \geq v_0^*(s_0) - \delta$. Thus, after stopping, for the rest of the episode, TensorPlan can safely use the policy induced by θ^+.

Summary

So far we have seen that if somehow TensorPlan would be able to get $\Delta(s, a, h, \theta)$ with no errors, (1) it would stop after refining its hypothesis set at most k times and (2) when it stops, with high probability it would return with a parameter vector that induces a policy with high value. Regarding the number of queries used, if obtaining $\Delta(s, a, h, \theta)$ is counted as a single query, TensorPlan would need at most $maHk = maH(d + 1)^A$ queries (m rollouts, for each of the H states in the rollout, A queries are needed).

It remains to be seen how to adjust this argument to the case when $\Delta(s, a, h, \theta)$ need to be estimated based on interactions with a stochastic simulator.

Notes

- It is not known whether TensorPlan can be computationally efficiently implemented. I suspect it cannot. This is because Θ_i is specified with a number of highly nonlinear constraints (in the parameter vector).

- The essence of the construction here is lifting the problem into a higher-dimensional linear space. This is a standard technique in machine learning but in a very different context when data is mapped to a higher dimensional space to strengthen the power of linear predictors. The once popular RKHS methods take this to the extreme. Note that here, in contrast to this classic lifting procedure, the parameter vector is mapped through a nonlinear function to a higher dimensional space and the purpose is to simply have a clear grasp on why learning stops.
- We call Δ here the discriminant function because what is important about it is that it discriminates between "good" and "bad" cases and it does it by using the special value of zero. Readers familiar with the RL literature will note, however, that Δ is nothing but, what is known as the "temporal difference error" (under some fixed action).
- It is curious that the algorithm builds up a data-bank of critical data that it uses to restrain the set of parameter vectors and that it is quite selective in adding new data here. That is, TensorPlan may generate a lot more data then goes on the list $x_1, x_2, \dots$. If we wanted to be philosophical and would not mind antropomorphising algorithms, we could say that TensorPlan remembers what it is "surprised by". This is very much unlike other algorithms, like LSVI-G, which may generate a lot of redundant data. The other difference is that TensorPlan uses the data to generate a hypothesis set. The choice of the parameter vector from this set is dictated by the optimization (reward maximization) problem solved by TensorPlan.
- There are quite a few examples of optimistic algorithms in planning; there is a considerable literature of using optimisim in tree search. However, classics, such as the A^* algorithm can also be seen as an optimistic algorithm (at least when used with an "admissible heuristic", which is just a way of saying that A^* uses an optimistic estimate of the values). The LAO^* algorithm is another example. However, the real "homeland" of optimistic algorithms in online learning, a topic that will be covered later in the course.

Bibliographical notes

This lecture is entirely based on the paper

- Weisz, Gellert, Philip Amortila, Barnabás Janzer, Yasin Abbasi-Yadkori, Nan Jiang, and Csaba Szepesvári. 2021. "On Query-Efficient Planning in MDPs under Linear Realizability of the Optimal State-Value Function."

available on arXiv.

12. TensorPlan and eluder sequences

Under construction.

13. From API to Politex

PDF Version

In the lecture on approximate policy iteration, we proved that for any MDP feature-map pair (M, ϕ) and any $\varepsilon' > 0$ excess suboptimality target, with a total runtime of

$$\text{poly}\left(d, \frac{1}{1-\gamma}, A, \frac{1}{\varepsilon'}\right),$$

least-squares policy iteration with G-optimal design (LSPI-G) can produce a policy π such that the suboptimality gap δ of π satisfies

$$\delta \le \frac{2(1+\sqrt{d})}{(1-\gamma)^{2}}\varepsilon + \varepsilon', \qquad (1)$$

where ε is the worst-case error with which the d-dimensional features can approximate the action-value functions of memoryless policies of the MDP M. In fact, the result continues to hold if we restrict the memoryless policies to those that are **ϕ-measurable** in the sense that the probability assigned by such a policy to taking some action a in some state s depends only on $\phi(s, \cdot)$. Denote the set of such policies by Π_ϕ. Then, for an MDP M and associated feature-map ϕ, let

$$\tilde{\varepsilon}(M, \phi) = \sup_{\pi \in \Pi_\phi} \inf_\theta \|\Phi\theta - q^\pi\|_\infty .$$

Checking the proof, noticing that LSPI produces ϕ-measurable policies only, it follows that provided the first policy it uses is also ϕ-measurable, ε in (1) can be replaced by $\tilde{\varepsilon}(M, \phi)$.

Earlier, we also proved that the amplification of ε by the $\sqrt{d}$-factor is unavoidable by **any** efficient planner. However, this leaves open the question of whether the amplification by a polynomial power of $1/(1-\gamma)$ is necessary, and whether in particular, the quadratic dependence is necessary? Our first result, which is given without proof, shows that in the case of LSPI this amplification is real and the quadratic dependence cannot be improved.

Theorem (LSPI error amplification lower bound): The quadratic dependence in (1) is tight: There exists a constant $c > 0$ such that for every $0 \le \gamma < 1$ and every $\varepsilon > 0$ there exists a featurized MDP (M, ϕ), a policy π of the MDP, a distribution μ over the states such that LSPI when it is allowed infinitely many rollouts of infinite length produces a sequence of policies $\pi_0 = \pi, \pi_1, \dots$ such that

$$\inf_{k \geq 1} \mu(v^* - v^{\pi_k}) \geq \frac{c\tilde{\varepsilon}(M, \phi)}{(1-\gamma)^2} \,.$$

The result of the theorem holds even when LSPI is used with **state-aggregation**. Intuitively, state-aggregation means that states are groups into a number of groups and states belonging to the same group are treated identically when it comes to representing value functions. This, value-functions based on state-aggregation are constant over any group. When we are concerned with state-value functions, aggregating the states based on a partitioning of the states $\mathcal{S}$ into the groups $\{\mathcal{S}_i\}_{1 \leq i \leq d}$ (i.e., $\mathcal{S}_i \subset \mathcal{S}$ and all the subsets are disjoint from each other), a feature-map that allows to represent these piecewise constant functions is

$$\phi_i(s) = \mathbb{I}(s \in \mathcal{S}_i) \,, \qquad i \in [d] \,,$$

where $\mathbb{I}$ is the indicator function that takes the value of one when its argument (a logical expression) is true, and is zero otherwise. In other words, $\phi : \mathcal{S} \to \{e_1, \dots, e_d\}$. Any feature map of this form defines a partitioning of the state-space and thus corresponds to the state-aggregation. Note that the piecewise constant functions can also be represented if we rotate all the features by the same rotation. The only important aspect here is that the features of different states are either identical, or orthogonal to each other, making the rows of the feature matrix an **orthonormal** system.

For approximating action-value functions, state-aggregation uses the same partitioning of states regardless of the identity of the actions: In effect, for each action, one uses the feature map from above, but with a private parameter vector. This effectively amounts to stacking $\phi(s)$ A-times, to get one copy of it for each action $a \in \mathcal{A}$. Note that for state-aggregation, there is no $\sqrt{d}$ amplification of the approximation errors: State-aggregation is extrapolation friendly, as will be explained at the end of the lecture.

Returning to the result, an inspection of the actual proof reveals that in this case LSPI leads to a sequence of policies that alternate between the initial policy and π_1. "Convergence" is fast, yet, the guarantee is far from satisfactory. In particular, in the same example, an alternate algorithm, which we will cover next can **reduce the quadratic dependence on the horizon to a linear dependence**.

Politex

Politex comes from **Po**licy **It**eration with **Ex**pert Advice. Assume that one is given a featurized MDP (M, ϕ) with state-action feature-map ϕ and access to a simulator, and a G-optimal design $\mathcal{C} \subset \mathcal{S} \times \mathcal{A}$ for ϕ.

Politex generates a sequence of policies $\pi_0, \pi_1, \dots$ such that for $k \ge 1$,

$$\pi_k(a|s) \propto \exp\left(\eta \bar{q}_{k-1}(s,a)\right),$$

where

$$\bar{q}_k = \hat{q}_0 + \dots + \hat{q}_j,$$

with

$$\hat{q}_j = \Pi \Phi \hat{\theta}_j,$$

where for $j \ge 0$, $\hat{\theta}_j$ is the parameter vector obtained by running the least-squares policy evaluation algorithm based on G-optimal design (LSPE-G) to evaluate policy π_j (see this lecture). In particular, recall that this algorithm rolls out policy π_j from the points of a G-optimal design to produce m independent trajectories of length H each, calculates the average return for each of these design points and then solves the (weighted) least-squares regression problem where the features are used to regress on the obtained values.

Above, $\Pi : \mathbb{R}^{\mathcal{S}\times\mathcal{S}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$ truncates its argument to the $[0, 1/(1-\gamma)]$ interval:

$$(\Pi q)(s,a) = \max(\min(q(s,a), 1/(1-\gamma)), 0), \qquad (s,a) \in \mathcal{S} \times \mathcal{A}\,.$$

Note that to calculate $\pi_k(a|s)$, one does need to calculate $E_k(s,a) = \exp\left(\eta \Pi[\phi(s,a)^\top \bar{\theta}_{k-1}]\right)$ and then compute $\pi_k(a|s) = E_k(s,a)/\sum_{a'} E_k(s,a')$.

Unlike in policy iteration, the policy returned by Politex after k iterations is either the "mixture policy"

$$\bar{\pi}_k = \frac{1}{k}(\pi_0 + \dots + \pi_{k-1})\,,$$

or the policy which gives the best value with respect to the start state, or start distribution. For simplicity, let us just consider the case when $\bar{\pi}_k$ is used as the output. The meaning of a mixture policy is simply that one of the k policies is selected uniformly at random and then the selected policy is followed for the rest of time. Homework 3 gives precise definitions and asks you to prove that the value function of $\bar{\pi}_k$ is just the mean of the value functions of the constituent policies:

$$v^{\bar{\pi}_k} = \frac{1}{n}(v^{\pi_0} + \dots + v^{\pi_{k-1}})\,. \tag{2}$$

We now argue that the dependence on the approximation error of the suboptimality gap of $\bar{\pi}_k$ only scales with $1/(1-\gamma)$, unlike the case of approximate policy iteration.

For this, recall that by the value difference identity

$$v^{\pi^*} - v^{\pi_j} = (I - \gamma P_{\pi^*})^{-1} \left[T_{\pi^*} v^{\pi_j} - v^{\pi_j}\right].$$

Summing up, dividing by k, and using (2) gives

$$v^{\pi^*} - v^{\bar{\pi}_k} = \frac{1}{k}(I - \gamma P_{\pi^*})^{-1} \sum_{j=0}^{k-1} T_{\pi^*} v^{\pi_j} - v^{\pi_j}.$$

Now, $T_{\pi^*} v^{\pi_j} = M_{\pi^*}(r + \gamma P v^{\pi_j}) = M_{\pi^*} q^{\pi_j}$. Also, $v^{\pi_j} = M_{\pi_j} q^{\pi_j}$. Let $\hat{q}_j = \Pi \Phi \hat{\theta}_j$. Elementary algebra then gives

$$\begin{aligned} v^{\pi^*} - v^{\bar{\pi}_k} &= \frac{1}{k}(I - \gamma P_{\pi^*})^{-1} \sum_{j=0}^{k-1} M_{\pi^*} q^{\pi_j} - M_{\pi_j} q^{\pi_j} \\ &= \underbrace{\frac{1}{k}(I - \gamma P_{\pi^*})^{-1} \sum_{j=0}^{k-1} M_{\pi^*} \hat{q}_j - M_{\pi_j} \hat{q}_j}_{T_1} + \underbrace{\frac{1}{k}(I - \gamma P_{\pi^*})^{-1} \sum_{j=0}^{k-1} (M_{\pi^*} - M_{\pi_j})(q^{\pi_j} - \hat{q}_j)}_{T_2}. \end{aligned}$$

We see that the approximation errors $\varepsilon_j = q^{\pi_j} - \hat{q}_j$ appear only in term T_2. In particular, taking pointwise absolute values, using the triangle inequality, we get that

$$\|T_2\|_\infty \le \frac{2}{1-\gamma} \max_{0 \le j \le k-1} \|\varepsilon_j\|_\infty,$$

which shows the promised dependence. It remains to show that $\|T_1\|_\infty$ above is also under control. However, this is left to the next lecture.

Notes

State aggregation and extrapolation friendliness

The $\sqrt{d}$ in our results comes from controlling the extrapolation errors of linear prediction. In the case of state-aggregretion, however, this extra $\sqrt{d}$ error amplification is completely avoided: Clearly, if we measure a function with a precision ε and there is at least one measurement per part, then by using the value measured at each part (at an arbitrary state there) over the whole part, the worst-case error is bounded by ε. Weighted least-squares in this context just takes the weighted average of the responses over each part and uses this as the prediction, so it also avoids amplifying approximation errors.

In this case, our analysis of extrapolation errors is clearly conservative. The extrapolation error was controlled in two steps: In our first lemma, for ρ weighted least-squares we reduced this problem to that of controlling $g(\rho) = \max_{z \in \mathcal{Z}} \|\phi(z)\|_{G_\rho^{-1}}$ where G_ρ is the moment matrix for ρ. In fact, the proof of this lemma is the culprit: By carefully inspecting the proof, we can see that the application of Jensen's inequality introduces an unnecessary term: For the case of state aggregation (orthonormed feature matrix),

$$\sum_{z' \in C} \varrho(z') |\phi(z')^\top G_\varrho^{-1} \phi(z')| = 1$$

as long as the design ρ is such that it chooses any group exactly once. Thus, the case of state-aggregation shows that some feature-maps are more **extrapolation friendly** than others. Also, note that the Kiefer-Wolfowitz theorem, of course, still gives that $\sqrt{d}$ is the smallest value that we can get for g when optimizing for ρ.

It is a fascinating question of how extrapolation errors behave for various feature-maps.

Least-squares value iteration (LSVI)

In homework 2, Question 3 was concerned with least-squares value iteration. The algorithm concerned (call it LSVI-G) uses a random approximation of the Bellman operator, based on a G-optimal design (and action-value functions). The problem was to show a result similar to what holds for LSPI-G holds for LSVI-G, as well. That is, for any MDP feature-map pair (M, ϕ) and any $\varepsilon' > 0$ excess suboptimality target, with a total runtime of

$$\text{poly}\left(d, \frac{1}{1-\gamma}, A, \frac{1}{\varepsilon'}\right),$$

least-squares policy iteration with G-optimal design (LSPI-G) can produce a policy π such that the suboptimality gap δ of π satisfies

$$\delta \le \frac{4(1+\sqrt{d})}{(1-\gamma)^{2}} \varepsilon_{\text{BOO}} + \varepsilon' \,. \tag{3}$$

Thus, the dependence on the horizon of the approximation error is similar to the one that was obtained for LSPI. Note that the definition of ε_{BOO} is different from what we have used in analyzing LSPI:

$$\varepsilon_{\text{BOO}} := \sup_\theta \inf_{\theta'} \|\Phi\theta' - T\Pi\Phi\theta\|_\infty \,.$$

Above, T is the Bellman optimality oerator for action-value functions and Π is defined so that for $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, Πf is also a $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ function which is obtained from f by truncating for each input (s, a) the value $f(s, a)$ to $[0, 1/(1-\gamma)]$:

$(\Pi(f))(s,a) = \max(\min(f(s,a), 1/(1-\gamma)), 0)$. In ε_{BOO}, "BOO" stands for "Bellman-optimality operator" in reference to the appearance of T in the definition.

In general, the error measures ε used in LSPI and ε_{BOO} are incomparable. The latter quantity measures a "one-step error", while ε is concerned with approximating functions defined over an infinite-horizon.

Linear MDPs

Call an **MDP linear** if both the reward function and the next state distributions for each state lie in the span of the features: $r = \Phi\theta_r$ with some $\theta_r \in \mathbb{R}^d$ and P, as an $\mathrm{SA} \times \mathrm{S}$ matrix takes the form $P = \Phi W$ with some $W \in \mathbb{R}^{d\times \mathrm{S}}$. Clearly, this is a notion that captures how well the "dynamics" (including the reward) of the MDP can be "compressed".

When an MDP is linear, $\varepsilon_{\text{BOO}} = 0$. We also have in this case that $\varepsilon = 0$. More generally, defining $\zeta_r = \inf_\theta \|\Phi\theta_r - r\|_\infty$ and $\zeta_P = \inf_W \|\Phi W - P\|_\infty$, it is not hard to see that $\varepsilon_{\text{BOO}} \le \zeta_r + \gamma\zeta_P/(1-\gamma)$ and $\varepsilon \le \zeta_r + \gamma\zeta_P/(1-\gamma)$, which shows that both policy iteration (and its soft versions) and value iteration are "valid" approaches, though, by ignoring the fact that we are comparing upper bounds, this also shows that value iteration may have an edge over policy iteration when the MDP itself is compressible. This should not be too surprising given that value-iteration is "more direct" in aiming to calculate q^*. Yet, they may exist cases when the action-value functions are compressible, while the dynamics is not.

Stationary points of a policy search objective

Let $J(\pi) = \mu v^\pi$. A stationary point of J with respect to some set of memoryless policies Π is any $\pi \in \Pi$ such that

$$\langle \nabla J(\pi), \pi' - \pi \rangle \le 0\,.$$

It is known that if ϕ are state-aggregation features then any stationary point π of J satisfies

$$\mu v^\pi \ge \mu v^* - \frac{4\varepsilon_{\text{apx}}}{1-\gamma}\,,$$

where ε_{apx} is defines as the worst-case error of approximation action-value functions of ϕ-measurable policies with the features (the same constant as used in the analysis of approximate policy iteration).

Soft-policy iteration with Averaging

Politex can be seen as a "soft" version of policy iteration with averaging. The softness is controlled by η: When $\eta \to \infty$, Politex uses a greedy policy w.r.t. to an average of all previous Q-functions. Notice that in this case if Politex were to use a greedy policy w.r.t. the last Q-function, then it would reduce exactly to LSPI-G. As we have seen, in LSPI-G the approximation error can get quadratically amplified with the horizon $1/(1-\gamma)$. Thus, one way to avoid this quadratic amplification is to stay soft with averaging. As we shall see in the next lecture, the price of this is a relatively slower convergence to a target suboptimality excess value. Nevertheless, the promise is that the algorithm will still stay polynomial in all the relevant quantities.

References

Politex was introduced in the paper

- POLITEX: Regret Bounds for Policy Iteration using Expert Prediction. Abbasi-Yadkori, Y.; Bartlett, P.; Bhatia, K.; Lazic, N.; Szepesvári, C.; and Weisz, G. In ICML, pages 3692–3702, May 2019. pdf

However, as this paper also notes, the basic idea goes back to the MDP-E algorithm by Even-Dar et al:

- Even-Dar, E., Kakade, S. M., and Mansour, Y. Online Markov decision processes. Mathematics of Operations Research, 34(3):726–736, 2009.

This algorithm considered a tabular MDP with nonstationary rewards – a completely different setting. Nevertheless, this paper introduces the basic argument presented above. The Politex paper notices that the argument can be extended to the case of function approximation. In particular, it also notes the nature of the function approximator is irrelevant as long as the approximation and estimation errors can be tightly controlled.

The Politex paper presented an analysis for online RL and average reward MDPs. Both add significant complications. The argument shown here is therefore a simpler version. Connecting Politex to LSPE-G in the discounted setting is trivial, but has not been presented before in the literature.

The first paper to use the error decomposition shown here together with function approximation is

- Abbasi-Yadkori, Y., Lazic, N., and Szepesvári, C. Modelfree linear quadratic control via reduction to expert prediction. In AISTATS, 2019.

14. Politex

PDF Version

The following lemma can be extracted from the calculations found at the end of the last lecture:

Lemma (Mixture policy suboptimality): Fix an MDP M. For any sequence $\pi_0,\dots,\pi_{k-1}$ of policies, any sequence $\hat q_0,\dots,\hat q_{k-1}: \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ of functions, and any policy π^*, the mixture policy $\bar\pi_k = 1/k(\pi_0+\dots+\pi_{k-1})$ satisfies

$$v^{\pi^*} - v^{\bar\pi_k} \le \frac{1}{k}(I-\gamma P_{\pi^*})^{-1}\underbrace{\sum_{j=0}^{k-1} M_{\pi^*}\hat q_j - M_{\pi_j}\hat q_j}_{T_1} + \frac{2\max_{0\le j\le k-1}\|q^{\pi_j}-\hat q_j\|_\infty}{1-\gamma}\,. \qquad (1)$$

In particular, the only restriction is on policy π^* so far and that is that it has to be a memoryless policy. To control the suboptimality of the mixture policy, one just needs to control the action-value approximation errors $\|q^{\pi_j}-\hat q_j\|_\infty$ and the term T_1 and for this we are free to choose the policies $\pi_0,\dots,\pi_{k-1}$ in any way we want them to be chosen. To help with this choice, let us now inspect $T_1(s)$ for a fixed state s:

$$T_1(s) = \sum_{j=0}^{k-1}\langle \pi^*(s,\cdot),\hat q_j(s,\cdot)\rangle - \langle \pi_j(s,\cdot),\hat q_j(s,\cdot)\rangle\,, \qquad (2)$$

where, abusing notation, we use $\pi(s,a)$ for $\pi(a|s)$. Now, recall that $\hat q_j$ will be computed based on π_j while π^* is unknown. One must thus wonder whether it is possible to control this term?

Online linear optimization

As it happens, the problem of controlling terms of this type is the central problem studied in a subfield of learning theory, **online learning**. In particular, in **online linear optimization**, the following problem is studied:

An adversary and a learner are playing a zero-sum minimax game in k discrete rounds, taking actions in an alternating manner. In round j ($0 \le j \le k-1$), first, the learner needs to choose a vector $x_j \in \mathcal{X} \subset \mathbb{R}^d$. Then, the adversary chooses a vector, $y_j \in \mathcal{Y} \subset \mathbb{R}^d$. Before its choice, the adversary learns about all previous choices of the learner, and the learner also learns about all previous choices of the adversary. They also remember their own choices. For simplicity, let us constraint the adversary and the learner to be deterministic. The payoff to the adversary at the end of the k rounds is

$$R_k = \max_{x \in \mathcal{X}} \sum_{j=0}^{k-1} \langle x, y_j \rangle - \langle x_j, y_j \rangle \,. \tag{3}$$

In particular, the adversary's goal is maximize this, while the learner's goal is to minimize this (the game is zero-sum). Both the adversary and the learner are given k and the sets $\mathcal{X}, \mathcal{Y}$. Letting L to denote the learner's strategy (a sequence of maps of histories to $\mathcal{X}$) and A to denote the adversary's strategy (a sequence of maps of histories to $\mathcal{Y}$), the above quantity depends on L and A: $R_k = R_K(A, L)$.

Taking the perspective of the learner, the quantity defined in (3) is called the learner's **regret**. Denote the minimax value of the game by R_k^*: $R_k^* = \inf_L \sup_A R_k(A, L)$.

Thus, this only depends on k, $\mathcal{X}$ and $\mathcal{Y}$. The dependence is suppressed when it is clear from the context. The central question then is how R_k^* depends on k and also on $\mathcal{X}$ and $\mathcal{Y}$. In online linear optimization both sets $\mathcal{X}$ and $\mathcal{Y}$ are convex.

Connecting these games to our problem, we can see that $T_1(s)$ in (2) matches the regret definition in (3) if we let $d = \mathrm{A}$, $\mathcal{X} = \mathcal{M}_1(\mathrm{A}) = \{p \in [0,1]^{\mathrm{A}} \,:\, \sum_a p_a = 1\}$ be the $\mathrm{A} - 1$ simplex of $\mathbb{R}^{\mathrm{A}}$ and $\mathcal{Y} = [0, 1/(1-\gamma)]^{\mathrm{A}}$. Furthermore, $\pi_j(s, \cdot)$ needs to be chosen first, which is followed by the choice of $\hat{q}_j(s, \cdot)$. While $\hat{q}_j(s, \cdot)$ will not be chosen in an adversarial fashion, a bound B on the regret against arbitrary choices will also serve as a bound for the specific choice we will need to make for $\hat{q}_j(s, \cdot)$.

Mirror descent

Mirror descent (MD) is an algorithm that originates in optimization theory. In the context of online linear optimization, MD is a strategy for the learner which is known to guarantee near minimax regret for the learner under a wide range of circumstances.

To align with the large body of literature on online linear optimization, it will be beneficial to switch signs. Thus, in what follows we assume that the learner will aim at minimizing $\langle x, y \rangle$ by its choice $x \in \mathcal{X}$ and the adversary will aim at maximizing the same expression over its choice $y \in \mathcal{Y}$. This means that we also redefine the regret to

$$R_k = \max_{x \in \mathcal{X}} \sum_{j=0}^{k-1} \langle x_j, y_j \rangle - \langle x, y_j \rangle$$
$$= \sum_{j=0}^{k-1} \langle x_j, y_j \rangle - \min_{x \in \mathcal{X}} \sum_{j=0}^{k-1} \langle x, y_j \rangle \,. \qquad (4)$$

Everything else remains the same: The game is zero-sum, minimax, the regret is the payoff for the adversary and the negative regret is the payoff of the learner. This version is called a loss-game. The reason to prefer the loss game is because most of optimization theory is written for minimizing convex functions rather than for maximizing concave functions. However, clearly, this is an arbitrary choice. The second form of the regret shows that the player's goal is to compete with the best single decision from $\mathcal{X}$ but chosen given the hindsight of knowing all the choices of the adversary. That is, the learner's goal is to keep its cumulative loss $\sum_{j=0}^{k-1} \langle x_j, y_j \rangle$ close to, or even below the best cumulative loss in hindsight, $\min_{x \in \mathcal{X}} \sum_{j=0}^{k-1} \langle x, y_j \rangle$. (With this, $T_1(s)$ matches R_k when we change $\mathcal{Y} = [-1/(1-\gamma), 0]^{\mathrm{A}}$.)

MD is recursively defined and in its simplest form it has two design parameters. The first is an extended real-valued convex function $F : \mathbb{R}^d \to \bar{\mathbb{R}}$, called the "regularizer", while the second is a stepsize, or learning rate parameter $\eta > 0$. (The extended reals is just $\mathbb{R}$ together with $+\infty$, $-\infty$ and an appropriate extension of basic arithmetic. By allowing convex functions to take the value $+\infty$ allows to merge "constraints" with objectives in a seamless fashion. The value $-\infty$ is added because sometimes we have to work with negated extended real-valued convex functions.)

The specification of MD is as follows: In round 0, $x_0 \in \mathcal{X}$ is picked to minimize F:

$$x_0 = \arg\min_{x \in \mathcal{X}} F(x) \,.$$

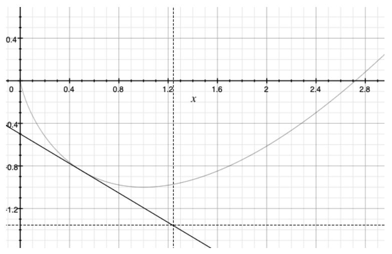
In what follows, we assume that all the minimizers that we need in the definition of MD do exist. In the specific case that we need, $\mathcal{X}$ is the $d-1$ simplex, which is a closed convex set, and since convex functions are also continuous, the minimizers that we will need are guaranteed to exist.

Then, in round $j > 0$, MD chooses x_j as follows:

$$x_j = \arg\min_{x \in \mathcal{X}} \ \eta \langle x, y_{j-1} \rangle + D_F(x, x_{j-1}) \qquad (5)$$

Here,

$$D_F(x, x') = F(x) - (F(x') + \langle \nabla F(x'), x - x' \rangle)$$



is the remainder term in the first-order Taylor-series expansion of the value of F at x when the expansion is carried out at x' and, for simplicity, we assume that F is differentiable on the interior of its domain $\text{dom}(F) = \{x \in \mathbb{R} \,:\, F(x) < +\infty\}$. Since for any convex function and any linear approximation of it stays below the graph of the convex function, we immediately get that D_F is nonnegative valued. For an illustration see the figure on the right, which shows a convex function, the first-order Taylor approximation of the function at some point.

One should think of F is a "nonlinear distance inducing function"; above $D_F(x, x')$ can be thought of penalty imposed on deviating from x'. However, D_F is more often than not is not a distance, i.e., often it is not even symmetric. Because of this, we can't really call D_F a distance. Hence, it is called a divergence. In particular, $D_F(x, x')$ is called the **Bregman divergence** of x from x'.

In the definition of the MD update rule, we tacitly assumed that $D_F(x, x_{j-1})$ is well-defined. This requires that F should be differentiable at x_{j-1}, which one needs to check when applying MD. In our specific case, this will hold, again.

The idea of the MD update rule is to (1) allow the learner to react to the last loss y_{j-1} vector chosen by the adversary, while also (2) limiting how much x_j can depart from x_{j-1}, thus, effectively stabilizing the algorithm, the tradeoff governed by the choice of $\eta > 0$. (Separating η from F only makes sense because there are some standard choices for F, but η is really just a scale parameter for F). In particular, the larger the value of η is, the less "data-sensitive" MD will be (here, $y_0, \dots, y_{k-1}$ constitute the data), and vice versa, the smaller η is, the more data-sensitive MD will be.

Where is the mirror?

Under some technical conditions on F, the update rule (5) has a two step-implementation:

$$\tilde{x}_j = (\nabla F)^{-1}(\nabla F(x_{j-1}) - \eta y_{j-1})\,, \tag{6}$$
$$x_j = \arg\min_{x \in \mathcal{X}} D_F(x, \tilde{x}_j)\,. \tag{7}$$

The first equation above explains the name: To obtain $\tilde{x}_j$, one first transforms x_{j-1} using $\nabla F : \text{dom}(\nabla F) \to \mathbb{R}^d$ to the "mirror" (dual) space where "gradients"/"slopes live", where one then adds to the result $-\eta y_{j-1}$, which can be seen as a "gradient step" (interpreting y_{j-1} as the gradient of some loss). Finally, the result is then mapped back to the original (primal) space using the inverse of ∇F.

The second step of the update takes the resulting point $\tilde{x}_j$ and "projects" it to $\mathcal{X}$ in a way that respects the "geometry induced by F" on the space $\mathbb{R}^d$.

The use of complex terminology, like "primal" and "dual" spaces, which happen to be the same old Euclidean space, $\mathbb{R}^d$, probably sounds like an overkill. Indeed, in the simple case we consider when these spaces are identical it is. The distinction would become important when working with infinite dimensional spaces, which we leave to others for now.

Besides helping with understanding the terminology, the two-step update shown can also be useful for computation. In fact, this will be the case in the special case that we need.

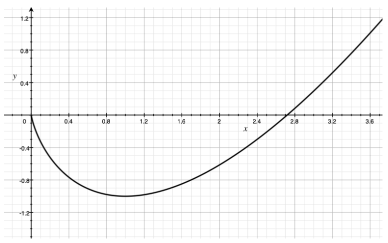
Mirror descent on the simplex

We have seen that in the special case we need,

$$\begin{aligned} \mathcal{X} &= \mathcal{P}_{d-1} := \{p \in [0,1]^d \,:\, \sum_a p_a = 1\}\,, \\ \mathcal{Y} &= [-1/(1-\gamma), 0]^d \,, \text{and} \\ d &= \mathrm{A}\,. \end{aligned}$$

To use MD we need to specify the regularizer F and the learning rate. For the former, we choose

$$F(x) = \sum_i x_i \log(x_i) - x_i\,,$$



which is known as the **unnormalized negentropy** function. Note that F takes on finite values when $x \in [0,\infty]^d$ (since $\lim_{x\to 0+} x\log(x) = 0$, we set $x_i \log(x_i) = 0$ whenever $x_i = 0$). Outside of this quadrant, we define the value of F to be $+\infty$. The plot of $x\log(x) - x$ for $x \ge 0$ is shown on the right.

It is not hard to verify that F is convex: First, $\mathrm{dom}(F) = [0,\infty]^d$ is convex. Taking the first derivative, we find that for any $x \in (0,\infty)^d$,

$$\nabla F(x) = \log(x)\,,$$

where log is applied componentwise. Taking the derivative again, we find that for $x \in (0,\infty)^d$,

$$\nabla^2 F(x) = \mathrm{diag}(1/x)\,,$$

i.e., the matrix whose (i,i)th diagonal entry is $1/x_i$. Clearly, this is a positive definite matrix, which suffices to verify that F is a convex function.

The Bregman divergence induced by F is

$$\begin{aligned} D_F(x,x') &= \langle \mathbf{1}, x\log(x) - x - x'\log(x') + x' \rangle - \langle \log(x'), x - x' \rangle \\ &= \langle \mathbf{1}, x\log(x/x') - x + x' \rangle\,, \end{aligned}$$

where again we use an "intuitive" notation when operations are first applied componentwise (i.e., $x\log(x)$ denotes a vector whose ith component is $x_i\log(x_i)$). Note that the domain of D_F is $[0,\infty)^d \times (0,\infty)^d$. If both x and x' lie in the $d-1$-simplex, D_F becomes the well-known **relative entropy**, or **Kullback-Leibler (KL) divergence**.

It is not hard to verify that x_j can be obtained as shown in (6)-(7) and in particular this two-step update takes the form

$$\tilde{x}_{j,i} = x_{j-1,i}\exp(-\eta y_{j-1,i})\,, \qquad x_{j,i} = \frac{\tilde{x}_{j,i}}{\sum_{i'}\tilde{x}_{j,i'}}\,, \quad i \in [d]\,.$$

Unrolling the recursion, we can also that this is the same as

$$\tilde{x}_{j,i} = \exp(-\eta(y_{0,i}+\dots+y_{j-1,i}))\,, \qquad x_{j,i} = \frac{\tilde{x}_{j,i}}{\sum_{i'}\tilde{x}_{j,i'}}\,, \quad i \in [d]\,. \tag{8}$$

Based on this, it is obvious that MD can be efficiently implemented with this choice of F. As far as the regret is concerned, the following theorem holds:

Theorem (MD with negentropy on the simplex): Let $\mathcal{X} = \mathcal{P}_{d-1}$ amd $\mathcal{Y} = [0,1]^d$. Then, no matter the adversary, a learner using MD with

$$\eta = \sqrt{\frac{2\log(d)}{k}}$$

is guaranteed that its regret R_k in k rounds is at most

$$R_k \le \sqrt{2k\log(d)}\,.$$

When the adversary plays in $\mathcal{Y} = [a,b]^d$ with $a < b$, we can use MD on the transformed sequence $\tilde{y}_j = (y_j - b\mathbf{1})/(b-a) \in [0,1]^d$. Then, for any $x \in \mathcal{X}$,

$$\begin{aligned} R_k(x) &:= \sum_{j=0}^{k-1} \langle x_j - x, y_j \rangle \\ &= \sum_{j=0}^{k-1} \langle x_j - x, (b-a)\tilde{y}_j + b\mathbf{1} \rangle \\ &= (b-a) \sum_{j=0}^{k-1} \langle x_j - x, \tilde{y}_j \rangle \\ &\le (b-a)\sqrt{2k\log(d)}\,, \end{aligned}$$

where the third equality used that $\langle x_j, \mathbf{1} \rangle = \langle x, \mathbf{1} \rangle = 1$. Taking the maximum over $x \in \mathcal{X}$ gives that

$$R_k \le (b-a)\sqrt{2k\log(d)}\,. \tag{9}$$

By the update rule in (8),

$$\tilde{x}_{j,i} = \exp(-\eta(\tilde{y}_{0,i} + \cdots + \tilde{y}_{j-1,i})) = \exp(-\eta/(b-a)(y_{0,i} + \cdots + y_{j-1,i} - jb))\,, \qquad i \in [d]\,.$$

Note that the "shift" by $-jb$ cancels out in the normalization step. Hence, MD in this case takes the form

$$\tilde{x}_{j,i} = \exp(-\eta/(b-a)(y_{0,i} + \cdots + y_{j-1,i}))\,, \qquad x_{j,i} = \frac{\tilde{x}_{j,i}}{\sum_{i'} \tilde{x}_{j,i'}}\,, \quad i \in [d]\,, \tag{10}$$

which is the same as before, except that the learning rate is scaled by $1/(b-a)$. In particular, in this case one can set

$$\eta = \frac{1}{b-a}\sqrt{\frac{2\log(d)}{k}}\,. \tag{11}$$

and use update rule (8).

MD applied to MDP planning

As agreed, $T_1(s)$ from (2) takes the form of a k-round regret against $\pi^*(s,\cdot)$ in online linear optimization on the simplex with losses in $[-1/(1-\gamma), 0]^{\mathrm{A}}$. This suggest to use MD in a state-by-state manner to control $T_1(s)$. Using (8) and (11) gives

$$E_j(s,a) = \exp(\eta(\hat{q}_0(s,a) + \cdots + \hat{q}_{j-1}(s,a)))\,, \qquad \pi_j(a|s) = \frac{E_j(s,a)}{\sum_{a'} E_j(s,a')}\,, \quad a \in \mathcal{A}$$

to be used with

$$\eta = (1-\gamma)\sqrt{\frac{2\log(\mathrm{A})}{k}}\,.$$

Note that this is the update used by Politex. Then, (9) gives that simultaneously for all $s \in \mathcal{S}$,

$$|T_1(s)| \le \frac{1}{1-\gamma}\sqrt{2k\log(\mathrm{A})}\,. \tag{12}$$

Putting things together, we get the following result:

Theorem (Politex suboptimality gap bound): Pick a featurized MDP (M,ϕ) with a full rank feature-map $\varphi : \mathcal{S}\times\mathcal{A}\to\mathbb{R}^d$ and let $K, m, H \ge 1$. Assume that B2$_\varepsilon$ holds for (M,ϕ) and the rewards in M are in the $[0,1]$ interval. For $0 \le \zeta < 1$, define

$$\kappa(\zeta) = \varepsilon(1+\sqrt{d}) + \sqrt{d}\left(\frac{\gamma^H}{1-\gamma} + \frac{1}{1-\gamma}\sqrt{\frac{\log(d(d+1)K/\zeta)}{2m}}\right),$$

Then, in K iterations, Politex produces a mixed policy $\bar\pi_K$ such that with probability $1-\zeta$, the suboptimality gap δ of $\bar\pi_K$ satisfies

$$\delta \le \frac{1}{(1-\gamma)^2}\sqrt{\frac{2\log(\mathrm{A})}{K}} + \frac{2\kappa(\zeta)}{1-\gamma}\,.$$

In particular, for any $\varepsilon' > 0$, choosing K, H, m so that

$$\begin{aligned} K &\ge \frac{32\log(A)}{(1-\gamma)^4(\varepsilon')^2}\,, \\ H &\ge H_{\gamma,(1-\gamma)\varepsilon'/(8\sqrt{d})} \qquad \text{and} \\ m &\ge \frac{32d}{(1-\gamma)^4(\varepsilon')^2}\log((d+1)^2K/\zeta)\,, \end{aligned}$$

policy π_K is δ-optimal with

$$\delta \le \frac{2(1+\sqrt{d})}{1-\gamma}\,\varepsilon + \varepsilon'\,,$$

while the total computation cost is $\mathrm{poly}(\frac{1}{1-\gamma}, d, \mathrm{A}, \frac{1}{(\varepsilon')^2}, \log(1/\zeta))$.

Note that as compared to the result of LSPI with G-optimal design, the amplification of the approximation error ε is reduced by a factor of $1/(1-\gamma)$, as it was promised. The price is that now the number of iterations K, is a polynomial of $\frac{1}{(1-\gamma)\varepsilon'}$, whereas before it was logarithmic. This suggest that perhaps a higher learning rate can help initially to speed up convergence to get the best of both words.

Proof: As in the proof of the suboptimality gap for LSPI, we get that for any $0 \le \zeta \le 1$, with probability at least $1-\zeta$, for any $0 \le k \le K-1$,

$$\|q^{\pi_k} - \hat{q}_k\|_\infty = \|q^{\pi_k} - \Pi\Phi\hat{\theta}_k\|_\infty \le \|q^{\pi_k} - \Phi\hat{\theta}_k\|_\infty \le \kappa(\zeta)\,,$$

where the first inequality uses that q_{π_k} takes values in $[0,1]$. On the event when the above inequalities hold, by (1) and (12),

$$\delta \le \frac{1}{(1-\gamma)^2}\sqrt{\frac{2\log(\mathrm{A})}{K}} + \frac{2\kappa(\zeta)}{1-\gamma}\,.$$

The details of this calculation are left to the reader. ■

Notes

Online convex optimization, online learning

Online linear optimization is a special case of **online convex/concave optimization**, where the learner chooses elements of some nonempty convex set $\mathcal{X} \subset \mathbb{R}^d$ and the adversary needs to choose an element of a nonempty set $\mathcal{Y}$ of concave functions over $\mathcal{X}$: $\mathcal{Y} \subset \{f : \mathcal{X} \to \mathbb{R} \,:\, f \text{ is concave}\}$. Then, the definition of regret is changed to

$$R_k = \max_{x\in\mathcal{X}} \sum_{j=0}^{k-1} y_j(x) - y_j(x_j)\,, \tag{13}$$

where as before $x_j \in \mathcal{X}$ is the choice of the learner for round j and $y_j \in \mathcal{Y}$ is the choice of the adversary for the same round. Identifying any vector u of $\mathbb{R}^d$ with the linear map $x \mapsto \langle x, u\rangle$, we see that online linear optimization is a special case of this problem.

Of course, by negating all functions in $\mathcal{Y}$ (i.e., letting $\tilde{\mathcal{Y}} = \{-y \,:\, y \in \mathcal{Y}\}$) and redefining the regret to

$$R_k = \max_{x\in\mathcal{X}} \sum_{j=0}^{k-1} \tilde{y}_j(x_j) - \tilde{y}_j(x) \tag{14}$$

we get a definition that is used in the literature, which prefers the convex case to the concave. Here, the interpretation is that $\tilde{y}_j \in \tilde{\mathcal{Y}}$ is a "loss function" chosen by the adversary in round j.

The standard function notation (y_j is applied to x) injects unwarranted asymmetry in the notation. After all, from the perspective of the learner, they need to choose a value in $\mathcal{X}$ that works for the various functions in $\mathcal{Y}$. Thus, we can consider any element of $\mathcal{X}$ as a function that maps elements of $\mathcal{Y}$ to reals through $y \mapsto y(x)$. Whether $\mathcal{Y}$ has functions in them or $\mathcal{X}$ has functions in them does not matter that much; it is the interconnection between $\mathcal{X}$ and $\mathcal{Y}$ that matters more. For this reason, one can study online learning when $y(x)$ above is replaced by $b(x, y)$, where $b : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a specific map that assigns payoffs to every pair of points in $\mathcal{X}$ and $\mathcal{Y}$. When the map is fixed, one can spare an extra symbol by just using $[x, y]$ in place of $b(x, y)$, which makes things almost a full circle given that we started with the linear case when $[x, y] = \langle x, y \rangle$.

Truncation or no truncation?

We introduced truncation to simplify the analysis. The proof can be made to go through even without it, with a mild increase of the suboptimality gap (or runtime). The advantage of removing the projection is that without projection, $\hat{q}_0 + \dots + \hat{q}_{j-1} = \Phi(\hat{\theta}_0 + \dots + \hat{\theta}_{j-1})$, which leads to a practically significant reduction of the runtime.

15. From policy search to policy gradients

PDF Version

In the previous lectures we attempted to reduce the complexity of planning by assuming that value functions over the large state-action spaces can be compactly represented with a few parameters. While value-functions are an indispensable component of poly-time MDP planners (see Lectures 3 and 4), it is far from clear whether they should also be given priority when working with larger MDPs.

Indeed, perhaps it is more natural to consider sets of policies with a compact description. Formally, in this problem setting the planner will be given a black-box simulation access to a (say, γ-discounted) MDP $M = (\mathcal{S}, \mathcal{A}, P, r)$ as before, but the interface also provides access to a parameterized family of policies over $(\mathcal{S}, \mathcal{A})$, $\pi = (\pi_\theta)_{\theta \in \mathbb{R}^d}$, where for any fixed parameter $\theta \in \mathbb{R}^d$, π_θ is a memoryless stochastic policy: $\pi_\theta : \mathcal{S} \to \mathcal{M}_1(\mathcal{A})$.

For example, π_θ could be such that for some feature-map $\varphi : \mathcal{S} \times \mathcal{A} \to \mathcal{R}^d$,

$$\pi_\theta(a|s) = \frac{\exp(\theta^\top \varphi(s,a))}{\sum_{a'} \exp(\theta^\top \varphi(s,a'))}\,, \qquad (s,a) \in \mathcal{S} \times \mathcal{A}\,. \tag{1}$$

In this case "access" to π_θ means access to φ, which can be either global (i.e., the planner is given the "whole" of φ and can run any preprocessing on it), or local (i.e., $\varphi(s',a)$ is returned by the simulator for the "next states" $s' \in \mathcal{S}$ and for all actions a). Of course, the exponential function can be replaced with other functions, or, one can just use a neural network to output "scores", which are turned into probabilities in some way. Dispensing with stochastic policies, a narrower class is the class of policies that are greedy with respect to action-value functions that belong to some parametric class.

One special case that is worthy of attention due to its simplicity is the case when $\mathcal{S}$ is partitioned into m (disjoint) subsets $\mathcal{S}_1, \dots, \mathcal{S}_m$ and for $i \in [m]$, we have A basis functions defined as follows:

$$\varphi_{i,a'}(s,a) = \mathbb{I}(s \in \mathcal{S}_i, a = a')\,, \qquad s \in \mathcal{S}, a, a' \in \mathcal{A}, i \in [m]\,. \tag{2}$$

Here, to minimize clutter, we allow the basis functions to be indexed by pairs and identified $\mathcal{A}$ with $1, \dots, \mathrm{A}$, as usual. Then, the policies are given by $\theta = (\theta_1, \dots, \theta_m)$, the collection of m probability vectors $\theta_1, \dots, \theta_m \in \mathcal{M}_1(\mathcal{A})$:

$$\pi_\theta(a|s) = \sum_{i=1}^m \sum_{a'} \varphi_{i,a'} \theta_{i,a'} \,. \tag{3}$$

Note that because of the special choice of φ, $\pi_\theta(a|s) = \theta_{i,a}$ for the unique index $i \in [m]$ such that $s \in \mathcal{S}_i$. This is known as state-aggregretion: States belonging to the same group give rise to the same probability distribution over the actions. We say that the featuremap $\varphi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ is of the **state-aggregation type** if it takes the form (2) with an appropriate reindexing of the basis functions.

Fix now a state-aggregation type featuremap. We can consider both the **direct parameterization** of policies given in (3), or the "Boltzmann" parameterization given in (1). As it is easy to see the set of possible policies that can be expressed with the two parameterizations are nearly identical. Letting Π_{direct} be the set of policies that can be expressed using φ and the direct parameterization and letting $\Pi_{\text{Boltzmann}}$ be the set of policies that can be expressed using φ but with the Boltzmann parameterization, first note that $\Pi_{\text{direct}}, \Pi_{\text{Boltzmann}} \subset \mathcal{M}_1(\mathcal{A})^{\mathcal{S}} \subset ([0,1]^{\mathrm{A}})^{\mathrm{S}}$, and if we take the closure, $\text{clo}(\Pi_{\text{Boltzmann}})$ of $\Pi_{\text{Boltzmann}}$ then we can notice that

$$\text{clo}(\Pi_{\text{Boltzmann}}) = \Pi_{\text{direct}} \,.$$

In particular, the Boltzmann policies cannot express point-mass distributions with finite parameters, but letting the parameter vectors grow without bound, any policy that can be expressed with the direct parameterization can also be expressed by the Boltzmann parameterization. There are many other possible parameterizations, as also mentioned earlier. The important point to notice is that while the parameterization is necessary so that the algorithms can work with a compressed representation, different representations may describe an identical set of policies.

Policy search

A reasonable goal then is to ask for a planner that competes with the best policy within the parameterized family, or the ε-best policy for some positive ε. Since there may not be a parameter θ such that $v^{\pi_\theta} \ge v^{\pi_{\theta'}} - \varepsilon \mathbf{1}$ for any $\theta' \in \mathbb{R}^d$, we simplify the problem by requiring that the policy computed is nearly best when started from some initial distribution $\mu \in \mathcal{M}_1(\mathcal{S})$.

Defining $J : \mathrm{ML} \to \mathbb{R}$ as

$$J(\pi) = \mu v^\pi (= \sum_{s\in\mathcal{S}} \mu(s) v^\pi(s)),$$

the **policy search problem** is to find a parameter $\theta \in \mathbb{R}^d$ such that

$$J(\pi_\theta) = \max_{\theta'} J(\pi_{\theta'})\,.$$

The approximation version of the problem asks for finding $\theta' \in \mathbb{R}^d$ such that

$$J(\pi_\theta) \ge \max_{\theta'} J(\pi_{\theta'}) - \varepsilon\,.$$

The formal problem definition then is as follows: a planning algorithm is given the MDP M and a policy parameterization $(\pi_\theta)_\theta$ and we are asking for an algorithm that returns the solution to the policy search problem in time polynomial in the number of actions A and the number of parameters d that describes the policy. An even simpler problem is when the MDP has finitely many states, and the algorithm needs to run in polynomial time in S, A and d. In this case, it is clearly advantageous for the algorithm if it is given the exact description of the MDP (as described in Lecture 3) Sadly, even this mild version of policy search is intractable.

Theorem (Policy search hardness): Unless $\mathrm{P} = \mathrm{NP}$, there is no polynomial time algorithm for the finite policy search problem even when the policy space is restricted to the constant policies and the MDPs are restricted to be deterministic with binary rewards.

The constant policies are those that assign the same probability distribution to each state. This is a special case of state aggregation when all the states are aggregated into a single class. As the policy does not depend on the state, the problem is also known as the **blind policy search problem**. Note that the result holds regardless of the representation used to express the set of constant policies.

Proof: Let $\mathcal{S} = \mathcal{A} = [n]$. The dynamics is deterministic: The next state is a if action $a \in \mathcal{A}$ is taken regardless of the state. A policy is simply a probability distribution $\pi \in \mathcal{M}_1([n])$ over the action space, which we shall view as a column vector taking values in $[0,1]^n$. The transition matrix of π is $P_\pi(s, s') = \pi(s')$, or, in matrix form, $P_\pi = \mathbf{1}\pi^\top$.

Clearly, $P_\pi^2 = \mathbf{1}\pi^\top \mathbf{1}\pi^\top = P_\pi$ (i.e., P_π is idempotent). Thus, $P_\pi^t = \mathbf{1}\pi^\top$ for any $t > 0$ and hence

$$J(\pi) = \mu(r_\pi + \sum_{t \ge 1} \gamma^t P_\pi^t r_\pi) = \mu \left(I + \frac{\gamma}{1-\gamma} \mathbf{1}\pi^\top \right) r_\pi \,.$$

Defining $R_{s,a} = r_a(s)$ so that $R \in [0,1]^{n \times n}$, we have $r_\pi = R\pi$. Plugging this in into the previous displayed equation and using that $\mu \mathbf{1} = 1$, we get

$$J(\pi) = \mu R \pi + \frac{\gamma}{1-\gamma} \pi^\top R \pi \,.$$

Thus we see that the policy search problem is equivalent to maximizing the quadratic expression in the previous display over the probability simplex. Since there is no restriction on R, one may at this point conjecture that this will be hard to do. That this is indeed the case can be shown by a reduction to the **maximum independent set problem**, which asks for checking whether the independence number of a graph is above a threshold and which is known to be NP-hard even for 3-regular graphs (i.e., graphs where every vertex has exactly three neighbours).

Here, the independence number of a graph is defined as follows: We are given a simple graph $G = (V, E)$ (i.e., there are no self-loops, no double edges, and the graph is undirected). An independent set in G is a neighbour-free subset of vertices. The independence number of G is defined as

$$\alpha(G) = \max\{|V'| \,:\, V' \subset \text{ independent in } G\} \,.$$

Quadratic optimization has close ties to the maximum independent set problem:

Lemma (Motzkin-Strauss '65): Let $G \in \{0,1\}^n$ be the vertex-vertex adjacency matrix of simple graph (i.e., $G_{ij} = 1$ if and only if (i,j) is an edge of the graph). Then, for $I \in \{0,1\}^{n \times n}$ the $n \times n$ identity matrix,

$$\frac{1}{\alpha(G)} = \min_{y \in \mathcal{M}_1([n])} y^\top (G + I) y \,.$$

We now show that if there is an algorithm that solves policy search in polynomial time then it can also be used to solve the maximum independent set problem for simple, 3-regular graphs. For this pick a 3-regular graph G with n vertices. Define the MDP as above with n states and actions and the rewards chosen so that $R = E - (I + G)$ where G is the vertex-vertex adjacency matrix of the graph and E is the all-ones matrix: $E = \mathbf{1}\mathbf{1}^\top$. We add E so that the rewards are in the $[0, 1]$ interval and in fact are binary as required. Choose μ as the uniform distribution over the states. Note that $\mathbf{1}^\top (I + G) = 4\mathbf{1}^\top$ because the graph is 3-regular. Then, for $\pi \in \mathcal{M}_1(\mathcal{A})$,

$$\begin{aligned} J(\pi) &= \frac{1}{1-\gamma} - \mu(E + I + G)\pi - \frac{\gamma}{1-\gamma}\pi^\top (E + I + G)\pi \\ &= \frac{1}{1-\gamma} - \frac{1}{n}\mathbf{1}^\top (I + G)\pi - \frac{\gamma}{1-\gamma}\pi^\top (I + G)\pi \\ &= \frac{1}{1-\gamma} - \frac{4}{n} - \frac{\gamma}{1-\gamma}\pi^\top (I + G)\pi \,. \end{aligned}$$

Hence, $\max_{\pi \in \mathcal{M}_1([n])} J(\pi) = \frac{1}{1-\gamma} - \frac{4}{n} - \frac{\gamma}{1-\gamma}\frac{1}{\alpha(G)} \ge \frac{1}{1-\gamma} - \frac{4}{n} - \frac{\gamma}{1-\gamma}\frac{1}{m}$ holds if and only if $\alpha(G) \ge m$. Thus, the decision problem of deciding that $J(\pi) \ge a$ is at least as hard as the maximum independent set problem. As noted, this is an NP-hard problem, hence the result follows. $\blacksquare$

Potential remedy: Local search

Based on the theorem just proved it is not very likely that we can find computationally efficient planners to compete with the best policy in a restricted policy class, even if the class looks quite benign. This motivates aiming at some more modest goal, one possibility of which is to compute local maxima of the map $J : \pi \mapsto \mu v^\pi$. Let $\Pi = \{\pi_\theta \,:\, \theta \in \mathbb{R}^d\} \subset [0, 1]^{\mathcal{S}\times\mathcal{A}}$ be the set of policies that can represented; we view these now as “large vectors”. Then, in this approach we aim to identify $\pi^* \in \Pi$ (and its parameters) so that for any $\pi' \in \Pi$ and small enough $\delta > 0$ so that $\pi^* + \delta(\pi' - \pi^*) \in \Pi$, $J(\pi^* + \delta(\pi' - \pi^*)) \le J(\pi^*)$. For δ small, $J(\pi^* + \delta(\pi' - \pi^*)) \approx J(\pi^*) + \delta\langle J'(\pi^*), \pi' - \pi^*\rangle$. Plugging this in into the previous inequality, reordering and dividing by $\delta > 0$ gives

$$\langle J'(\pi^*), \pi' - \pi^*\rangle \le 0 \,, \qquad \pi' \in \Pi \,. \tag{4}$$

Here, $J'(\pi)$ denotes the derivative of J. What remains to be seen is whether (1) relaxing the goal to computing π^* helps with the computation (and when) and (2) whether we can get some guarantees for how well π^* satisfying (4) will do compared to $J^* = \max_{\pi\in\Pi} J(\pi)$, that is obtaining some **approximation guarantees**. For the latter we seek for some function ε of the MDP M and Π (or φ, when Π is based on some featuremap) so that

$$J(\pi^*) \ge J^* - \varepsilon(M,\Pi)$$

As to the computational approaches, we will consider a simple approach based on (approximately) following the gradient of $\theta \mapsto J(\pi_\theta)$.

Notes

Access models

The reader may be wondering about what is the appropriate "access model" when π_θ is not restricted to the form given in (1). There are many possibilities. One is to develop planners for specific parametric forms. A more general approach is to let the planner access $\pi_\theta(\cdot|s)$ and $\frac{\partial}{\partial\theta}\pi_\theta(\cdot|s)$ for any s it has encountered and any value of $\theta \in \mathbb{R}^d$ it chooses. This is akin to the **first-order black-box oracle** model familiar from optimization theory.

From function approximation to POMDPs

The hardness result for policy search is taken from a paper of Vlassis, Littman and Barber, who actually were interested in the computational complexity of planning in partially observable Markov Decision Problems (POMDPs). It is in fact an important observation that with function approximation, planning in MDPs becomes a special case of planning in POMDPs: In particular, if policies are restricted to depend on the states through a feature-map $\varphi : \mathcal{S} \to \mathbb{R}^d$ (any two states with identical features will get the same action distribution assigned to them), then planning to achieve high reward with this restricted class is almost the same as planning to achieve high reward in a partially observable MDP where the observation function is φ. Planners for the former problem could still have some advantage though if they can also access the states: In particular, an online planner which is given a feature-map to help its search but is also given access to the states is in fact not restricted to return actions whose distribution follows a policy from the feature-restricted class of policies. In machine learning, in the analogue problem of competing with a best predictor within a class but using predictors that do not respect the restrictions put on the competitors are called **improper** and it is known that improper learning is often more powerful than proper learning. However, when it comes to learning online or in a batch fashion then feature-restricted learning and learning in POMDPs become exact analogs. Finally, we note in passing that Vlassis et al. (2012) also add an argument that shows that it is not likely that policy search is in NP.

Open problem: Hardness of approximate policy search

Provided that from an approximate solution to the Motzkin-Straus problem one can efficiently extract an approximate solution to the maximum independent set problem, it follows that the approximate version of policy search is also NP-hard. In particular, it is not hard to see with the same construction that if one has an efficient method to find a policy with $J(\pi) \ge \max_\pi J_\pi - \varepsilon$ then this gives an efficient method to find an independent set of size $c\alpha(G)$ for the said 3-regular graphs where

$$c = \frac{1}{1 + \frac{1-\gamma}{\gamma}\varepsilon\alpha(G)} \ge \frac{1}{1 + \frac{1-\gamma}{\gamma}\varepsilon n} \ge 94/95\,,$$

where the last inequality follows if $\varepsilon \le 0.5$, $\gamma \ge 0.5$ and $H := \frac{1}{1-\gamma} \ge \frac{n}{95/94-1} = 94n$ holds. Now, it is known that, unless P=NP, there is no polynomial time approximation algorithm for the maximal independent set problem with approximation factor $c = 94/95$ or better. Hence, we get that, unless P=NP, there is no polynomial time approximation algorithm for the policy search problem for any fixed $0 \le \epsilon \le 0.5$ provided the planning horizon is scaled with n so that $H = \text{const} n$. (This is somewhat unsatisfactory given that the range of the optimal values is $1/(1-\gamma)$: It would be more natural to scale ϵ with $1/(1-\gamma)$, i.e., consider relative errors as in complexity theory.) Also, it remains an open problem to get a hardness result for a "constant" γ (independent of n).

The above is still dependent on whether an approximate solution to the maximum independent set problem can be extracted from an approximate solution to the Motzkin-Straus optimization problem.

Dealing with large action spaces

A common reason to consider policy search is because working with a restricted parametric family of policies holds the promise of decoupling the computational cost of learning and planning from the cardinality of the action-space. Indeed, with action-value functions, one usually needs an efficient way of computing greedy actions (with respect to some fixed action-value function). Computing $\arg\max_{a\in\mathcal{A}} q(s,a)$ in the lack of extra structure of the action-space and the function $q(s,\cdot)$ takes linear time in the size of $\mathcal{A}$, which is highly problematic unless $\mathcal{A}$ has a small cardinality. In many applications of practical interest this is not the case: The action space can be "combinatorially sized", or even a subset of some (potentially multidimensional) continuous space.

If **sampling from $\pi_\theta(\cdot|s)$ can be done efficiently**, one may then potentially avoid the above expensive calculation. Thus, policy search is often proposed as a remedy to extend algorithms to work with large action spaces. Of course, this only applies if the sampling problem can indeed be efficiently implemented, which adds an extra restriction on the policy representation. Nevertheless, there are a number of options to achieve this: One can use for example an implicit representation (perhaps in conjunction with a direct one that uses probabilities/densities) for the policy.

For example, the policy may be "represented" as a map $f_\theta : \mathcal{S} \times \mathcal{R} \to \mathcal{A}$ so that sampling from $\pi_\theta(\cdot|s)$ is accomplished by drawing a sample $R \sim P$ from a fixed distribution over the set $\mathcal{R}$ and then returning $f(s, R) \in \mathcal{A}$. Clearly, this is efficient as long as f_θ can be efficiently evaluated at any of its inputs and the random value R can be efficiently produced. If f_θ is sufficiently flexible, one can in fact choose a very simple distribution for P, such as the standard normal distribution, or the uniform distribution.

Note that when $\mathcal{A}$ is continuous and the policies are deterministic is a special case: The key is still to be able to efficiently produce a sample from $\pi_\theta(\cdot|s)$, just in this case this means a deterministic computation.

The catch is that one may also still need the derivatives of $\pi_\theta(\cdot|s)$ with respect to the parameter θ and with an implicit representation as described above, it is unclear whether these derivatives can be efficiently obtained. As it turns out, this can be arranged if $f_\theta(\cdot|s)$ is made of composition of elementary (invertible, differentiable) transformations with this property (by the chain rule). This observation is the basis of various approaches to "neural" density estimation (e.g., Tabak and Vanden-Eijnden, 2010, Rezende, Mohamed, 2015, or Jaini et al. 2019).

References

- Vlassis, Nikos, Michael L. Littman, and David Barber. 2012. "On the Computational Complexity of Stochastic Controller Optimization in POMDPs." ACM Trans. Comput. Theory, 12, 4 (4): 1–8.
- Esteban G. Tabak. Eric Vanden-Eijnden. "Density estimation by dual ascent of the log-likelihood." Commun. Math. Sci. 8 (1) 217 - 233, March 2010.

- Rezende, Danilo Jimenez, and Shakir Mohamed. 2015. “Variational Inference with Normalizing Flows” link.
- Rezende, D. J., and S. Mohamed. 2014. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models.” ICML. link.
- Jaini, Priyank, Kira A. Selby, and Yaoliang Yu. 2019. “Sum-of-Squares Polynomial Flow.” In Proceedings of the 36th International Conference on Machine Learning, edited by Kamalika Chaudhuri and Ruslan Salakhutdinov, 97:3009–18. Proceedings of Machine Learning Research. PMLR.
- Arora, Sanjeev, and Boaz Barak. 2009. Computational Complexity. A Modern Approach. Cambridge: Cambridge University Press.

The hardness of the maximum independent set problem is a classic result; see, e.g., Theorem 2.15 in the book of Arora and Barak (2009) above, though this proof does not show that the hardness also applies to the case of 3-regular graphs. Below is the paper that shows that approximating the maximum independent set size within a factor of $94/95 = 0.9894\dots$ is NP-hard even for 3-regular graphs. The precise statement is in the main theorem statement on page 29 (this is the first, unnumbered and unnamed theorem on pdf page 3). In particular, the 2nd bullet point has this bound, specifically the hardness kicks in for approximation factors at least as large as $94/95$. I am very grateful for Zachary Friggstad who pointed me to this paper.

- Miroslav Chlebík, Janka Chlebíková: Inapproximability Results for Bounded Variants of Optimization Problems. FCT 2003: 27-38 DBLP page

16. Policy gradients

PDF Version

In this last lecture on planning, we look at policy search through the lens of applying gradient ascent. We start by proving the so-called policy gradient theorem which is then shown to give rise to an efficient way of constructing noisy, but unbiased gradient estimates in the presence of a simulator. We discuss at a high level the ideas underlying gradient ascent and stochastic gradient ascent methods (as opposed to more common case in machine learning where the goal is to minimize a loss, or objective function, we are maximizing rewards, hence ascending on the objective rather than descending). We then find out about the limitations of policy gradient even in the presence of "perfect representation" (unrestricted policy classes, tabular case) and perfect gradient information, which motivates the introduction of a variant known as "natural policy gradients" (NPG). We then uncover a close relationship between this method and Politex. The lecture concludes with comparing results for NPG and Politex.

The policy gradient theorem

Fix an MDP $M = (\mathcal{S}, \mathcal{A}, P, r)$ and a discount factor $0 \le \gamma < 1$. Continuing from the last lecture for $\theta \in \mathbb{R}^d$ let π_θ be a stochastic policy: $\pi_\theta : \mathcal{S} \to \mathcal{M}_1(\mathcal{A})$. Further, fix a distribution $\mu \in \mathcal{M}_1(\mathcal{S})$ over the states and for a policy $\pi : \mathcal{S} \to \mathcal{M}_1(\mathcal{A})$ let

$$J(\pi) = \mu v^\pi$$

denote the expected value of using policy π in M from an initial state randomly chosen from μ. The policy gradient theorem gives sufficient conditions under which the map $\theta \mapsto J(\pi_\theta)$ is differentiable at some parameter $\theta = \theta_0$ and gives a "simple" expression for the gradient as a function of $\frac{d}{dx} M_{\pi_x} q^{\pi_{\theta_0}}$. Just to demistify this, for finite (or discrete) action spaces, for a memoryless policy π and function $q : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}$, $M_\pi q$ is a function mapping states to reals defined via

$$(M_\pi q)(s) = \sum_a \pi(a|s) q(s,a)\,.$$

Hence, the derivative, $\frac{d}{dx}M_{\pi_x}q$ is actually quite simple. It is a function mapping states to d dimensional vectors which satisfies

$$\frac{d}{dx}(M_{\pi_x}q)(s) = \sum_a \frac{d}{dx}\pi_x(a|s)q(s,a)\,.$$

The theorem we give though is not limited to this case and also applies to when the action space is infinite and even when the policy is deterministic. For the theorem statement, recall that for a policy π we used $\tilde{\nu}_\mu^\pi$ to denote its discounted state occupancy measure. Also, for a function f, we use f' to denote its derivative.

Theorem (Policy Gradient Theorem): Fix an MDP $(\mathcal{S}, \mathcal{A}, P, r)$. For $x \in \mathbb{R}^d$, define the maps $f_\pi : x \mapsto \tilde{\nu}_\mu^\pi M_{\pi_x} q^\pi$ and $g_\pi : x \mapsto \tilde{\nu}_\mu^{\pi_x} v^\pi$. Fix $\theta_0 \in \mathbb{R}^d$. Assume that at least **one** of the following two conditions is met:

1. $\theta \mapsto f'_{\pi_\theta}(\theta_0)$ exists and is continuous in a neighborhood of θ_0 and $g'_{\pi_{\theta_0}}(\theta_0)$ exists;
2. $\theta \mapsto g'_{\pi_\theta}(\theta_0)$ exists and is continuous in a neighborhood of θ_0 and $f'_{\pi_{\theta_0}}(\theta_0)$ exists;

Then, $x \mapsto J(\pi_x)$ is differentiable at $x = \theta_0$ and

$$\frac{d}{dx}J(\pi_x)|_{x=\theta_0} = \frac{d}{dx}\tilde{\nu}_\mu^{\pi_{\theta_0}} M_{\pi_x} q^{\pi_{\theta_0}}|_{x=\theta_0} = \tilde{\nu}_\mu^{\pi_{\theta_0}} \frac{d}{dx} M_{\pi_x} q^{\pi_{\theta_0}}|_{x=\theta_0}\,, \tag{1}$$

where the last equality holds if $\mathcal{S}$ is finite.

For the second expression, we treat $\frac{d}{dx}M_{\pi_x}q^{\pi_{\theta_0}}$ as an $\mathrm{S} \times d$ matrix. Note that this fits well with our convention of treating functions as "column vectors" (hence $M_{\pi_x})q^{\pi_{\theta_0}}$ is a vector of dimension S) and with the standard convention that a "vector derivative" creates "row vectors".

Above, the second expression where we moved the derivative with respect to the parameter inside the expression will only be valid in infinite state spaces when some additional regularity assumption is met. One such assumption is that $s \mapsto \|\frac{d}{dx}(M_{\pi_x}q^{\pi_{\theta_0}})(s)|_{x=\theta_0}\|$ is $\tilde{\nu}_\mu^{\pi_{\theta_0}}$-integrable.

In words, the theorem shows that the derivative of the performance of a policy can be obtained by integrating a simple derivative that involves the action-value function of the policy.

Of the two conditions of the theorem, the first condition is the one that is generally easier to verify. In particular, the condition on the continuous differentiability of $x \mapsto f_{\pi_x}$ at $x = \theta_0$ is usually easy to verify. To show the differentiability of $x \mapsto g_{\pi_x}$ at $x = \theta_0$ just recall that if the partial derivatives of a function exist and are continuous the function is differentiable. Then recall that $\tilde{\nu}_\mu^{\pi_x} v = \sum_{t=0}^\infty \gamma^t \nu P_{\pi_x}^t v$ and hence its differentiability with respect to (say) x_1 follows if $x \mapsto \nu M_{\pi_x} P v$ is continuously differentiable at $x = \theta_0$. In effect, for finite state-action spaces, differentiability at θ_0 follows (and the conditions of the theorem are satisfied) as long as for any (s, a) state-action pair, the maps $x \mapsto \pi_x(a|s)$ have continuous partial derivatives at $x = \theta_0$.

Proof: The proof is based on a short calculation that starts with writing the value difference identity for $v^{\pi_x} - v^{\pi_{\theta_0}}$, multiplied from the right by μ, taking derivatives and then letting $x = \theta_0$.

The details are as follows: Recall from Calculus 101 the following result: Assume that $f = f(u, v)$ satisfies at least one of the two conditions:

1. $z \mapsto \frac{\partial}{\partial v} f(z, x)$ exists and is continuous in a neighborhood of $z = x$ and $\frac{\partial}{\partial u} f(u, x)|_{u=x}$ exists;
2. $z \mapsto \frac{\partial}{\partial u} f(x, z)$ exists and is continuous in a neighborhood of $z = x$ and $\frac{\partial}{\partial v} f(x, v)|_{v=x}$ exists.

Then $z \mapsto f(z, z)$ is differentiable at $z = x$ and

$$\frac{d}{dx} f(x, x) = \frac{\partial}{\partial u} f(x, x) + \frac{\partial}{\partial v} f(x, x)\,. \tag{2}$$

Let π', π be two memoryless policies. By the value difference identity,

$$\begin{aligned} v^{\pi'} - v^\pi &= (I - \gamma P_{\pi'})^{-1} [T_{\pi'} v^\pi - v^\pi] \\ &= (I - \gamma P_{\pi'})^{-1} [M_{\pi'} q^\pi - v^\pi]\,, \end{aligned}$$

where the last equality just used that that $T_{\pi'} v^\pi = M_{\pi'}(r + \gamma P v^\pi) = M_{\pi'} q^\pi$. Now let $\pi' = \pi_x$ and $\pi = \pi_{\theta_0}$ and multiply the value difference identity from the left by μ to get

$$\mu(v^{\pi_x} - v^{\pi_{\theta_0}}) = \tilde{\nu}_\mu^{\pi_x} [M_{\pi_x} q^{\pi_{\theta_0}} - v^{\pi_{\theta_0}}]\,. \tag{3}$$

Now, focusing on the first term on the right-hand-side, let

$$f(u, v) = \tilde{\nu}_\mu^{\pi_u} M_{\pi_v} q^{\pi_{\theta_0}}\,. \tag{4}$$

Provided that f is sufficiently regular in a neighborhood of (x, x) (to be discussed later), (2) gives that

$$\frac{d}{dx} f(x,x) = \frac{d}{du} \tilde{\nu}_\mu^{\pi_u} M_{\pi_x} q^{\pi_{\theta_0}}|_{u=x} + \frac{d}{dv} \tilde{\nu}_\mu^{\pi_x} M_{\pi_v} q^{\pi_{\theta_0}}|_{v=x}$$

Taking the derivative of both sides of (3) with respect to x and using the above display we get

$$\frac{d}{dx} J(x) = \frac{d}{dx} \mu(v^{\pi_x} - v^{\pi_{\theta_0}}) = \frac{d}{du} \tilde{\nu}_\mu^{\pi_u} M_{\pi_x} q^{\pi_{\theta_0}}|_{u=x} + \frac{d}{dv} \tilde{\nu}_\mu^{\pi_x} M_{\pi_v} q^{\pi_{\theta_0}}|_{v=x} + \frac{d}{dx} \tilde{\nu}_\mu^{\pi_x} v^{\pi_{\theta_0}} \,.$$

Now let $x = \theta_0$. Then, $M_{\pi_x} q^{\pi_{\theta_0}} = M_{\pi_{\theta_0}} q^{\pi_{\theta_0}} = v^{\pi_{\theta_0}}$. Hence, the first and the third term of the above display cancel each other and we get

$$\frac{d}{dx} J(\pi_x)|_{x=\theta_0} = \frac{d}{dv} \tilde{\nu}_\mu^{\pi_{\theta_0}} M_{\pi_v} q^{\pi_{\theta_0}}|_{v=\theta_0} \,.$$

Finally, the conditions to apply (2) to our f in (4) are met by our assumption on f_π and g_π, finishing the proof. ∎

When the action-space is discrete and π_θ are stochastic policies, we can further manipulate the expression we obtained. In particular, in this case

$$(M_{\pi_x} q^{\pi_{\theta_0}})(s) = \sum_a \pi_x(a|s) q^{\pi_{\theta_0}}(s,a)$$

and thus, for finite $\mathcal{A}$,

$$\frac{d}{dx} (M_{\pi_x} q^{\pi_{\theta_0}})(s) = \sum_a \frac{d}{dx} \pi_x(a|s) q^{\pi_{\theta_0}}(s,a) \,. \qquad (5)$$

While this can be used as the basis of evaluating (or approximating) gradient, it may be worthwhile to point out an alternate form which is available when $\pi_x(a|s) > 0$. In this case, using the chain rule we get

$$\frac{d}{dx} \log \pi_x(a|s) = \frac{\frac{d}{dx} \pi_x(a|s)}{\pi_x(a|s)} \,.$$

Using this in (5) we get

$$\frac{d}{dx} (M_{\pi_x} q^{\pi_{\theta_0}})(s) = \sum_a \pi_x(a|s) \left(\frac{d}{dx} \log \pi_x(a|s) \right) q^{\pi_{\theta_0}}(s,a) \,, \qquad (6)$$

which has the pleasant property that it takes the form of an expected value over the actions of the **score function** of the policy map correlated with the action-value function.

Before moving on it is worth pointing out that an equivalent expression is obtained if $q^{\pi_{\theta_0}}(s,a)$ above is shifted by an arbitrary constant which may depend on θ_0 or s but not a. Indeed, since $\sum_a \pi_x(a|s) b(s,\theta_0) = b(s,\theta_0)$, differentiating both sides with respect to x gives $\sum_a \frac{d}{dx}\pi_x(a|a) b(s,\theta_0) = 0$. Hence, we also have

$$\frac{d}{dx}(M_{\pi_x} q^{\pi_{\theta_0}})(s) = \sum_a \pi_x(a|s)\left(\frac{d}{dx}\log \pi_x(a|s)\right)(q^{\pi_{\theta_0}}(s,a) - b(s,\theta_0))\,. \qquad (7)$$

This may have significance when using simulation to evaluate derivatives: One may attempt to use an appropriate "bias" term to reduce the variance of the estimate of the gradient. Before discussing simulation any further, it may be also worthwhile to discuss what happens when the action-space is infinite.

For countable infinite action spaces, the only difference is that (5) may not always hold. An easy sufficient condition for this to hold is that $\sum_a \|\frac{d}{dx}\pi_x(a|s)\| \, |q^{\pi_{\theta_0}}(s,a)|$ is summable, or equivalently, $\|\frac{d}{dx}\log \pi_x(a|s)\| \, |q^{\pi_{\theta_0}}(s,a)|$ is $\pi_x(\cdot|s)$-summable/integrable.

For uncountably infinite action spaces, this argument works with the minimal necessary changes. In the most general case, $\pi_\theta(\cdot|s)$ is a probability measure over $\mathcal{A}$ and its derivative is a vector-valued measure. The formulae derived above (e.g., (7)) remain valid if we replace the sum with an integral when $\pi_\theta(\cdot|s)$ is given in the form of a density with respect to some fixed measure λ over $\mathcal{A}$:

$$\frac{d}{dx}(M_{\pi_x} q^{\pi_{\theta_0}})(s) = \int_{\mathcal{A}} \pi_x(a|s)\left(\frac{d}{dx}\log \pi_x(a|s)\right)(q^{\pi_{\theta_0}}(s,a) - b(s,\theta_0))\lambda(da)\,. \qquad (8)$$

In fact, this is a strictly more general form: (7) is a special case of (8) when λ is set to the counting measure over $\mathcal{A}$.

In the special case when $\pi_\theta(\cdot|s) = \delta_{f_\theta(s)}(\cdot)$ (a Dirac at $f_\theta(s)$), in words, when we have a deterministic policy map and f is differentiable with respect to θ, it is better to start from the formula given in the theorem.

Indeed, in this case,

$$(M_{\pi_x} q^{\pi_{\theta_0}})(s) = q^{\pi_{\theta_0}}(s, f_\theta(s))$$

and hence

$$\frac{d}{dx}(M_{\pi_x} q^{\pi_{\theta_0}})(s) = \frac{d}{dx} q^{\pi_{\theta_0}}(s, f_x(s))$$

and thus, if either $\mathcal{S}$ is finite or an appropriate regularity condition holds,

$$\frac{d}{dx}J(\pi_x)|_{x=\theta_0} = \tilde{\nu}_\mu^{\pi_{\theta_0}} \frac{d}{dx} q^{\pi_{\theta_0}}(\cdot, f_x(\cdot))|_{x=\theta_0} \,.$$

If $a \mapsto q^{\pi_{\theta_0}}(s,a)$ is differentiable and $x \mapsto f_x(s)$ is also differentiable at $x=\theta_0$ for every s then

$$\frac{d}{dx}J(\pi_x)|_{x=\theta_0} = \tilde{\nu}_\mu^{\pi_{\theta_0}} \frac{\partial}{\partial a} q^{\pi_{\theta_0}}(\cdot, f_{\theta_0}(\cdot)) \frac{d}{dx} f_x(\cdot)|_{x=\theta_0} \,,$$

which is known as the "deterministic policy gradient formula".

Gradient methods

The idea of gradient methods is to make small steps in the parameter space in the direction of the gradient of an objective function that is to be maximized. In the context of policy search, this works as follows: If $x_i \in \mathbb{R}^d$ denotes the parameter vector in round i,

$$x_{i+1} = x_i + \alpha_i \nabla_x J(\pi_x)|_{x=x_i} \,,$$

where for f differentiable, $\nabla_x f = (\frac{d}{dx} f)^\top$ is the "gradient" (transpose of derivative). Above, α_i is a positive tuning parameter, called the "stepsize" of the update. The idea is that the "gradient" points in the direction where the function is expected to grow. Indeed, since by definition,

$$f(x') = f(x) + f'(x)(x'-x) + o(\|x'-x\|)$$

if $x' = x + \delta(f'(x))^\top$,

$$f(x+\delta(f'(x))^\top) = f(x) + \delta \|f'(x)\|_2^2 + o(|\delta|) \,,$$

or

$$\frac{f(x+\delta(f'(x))^\top) - f(x)}{\delta} = \|f'(x)\|_2^2 + o(1) \,,$$

For **any** δ sufficiently small so that the $o(1)$ term (in absolute value) is below $\|f'(x)\|_2^2$, we see that the right-hand side is positive, hence so is the left-hand side, as claimed. This simple observation is the basis of a huge number of algorithmic variants. In the lack of extra structure the best we can hope from a gradient method is that it will end up in the vicinity of a stationary point. In the presence of extra structure (.e.g, concave function to be maximized), convergence to a global maximum can be guaranteed.

In all cases the key to the success of gradient methods is the appropriate choice of the stepsizes; these choices are based on a refinement of the above simple argument that shows that moving towards the direction of the gradient helps. There are also ways of "speeding up" convergence; these "acceleration methods" use a refined iteration (two iterates updated simultaneously) and can greatly speed up convergence. As there are many excellent texts that describe various aspects of gradient methods which cover these ideas, we will not delve into them any further, but I will rather give some pointers to this literature in the endnotes.

The elephant in the room here is that the gradient of J is not readily available. The next best thing then is to attempt to build an estimate G of $\nabla_x J(\pi_x)$. In the planning setting, the question is whether one can get reasonable estimates of this gradient using a simulator.

Gradient estimation

Generally speaking there are two types of errors when construction an estimate of the gradient: The one that is purely random, and the one that is not. Defining $g(x) = \mathbb{E}[G]$, $b(x) = \nabla_x J(\pi_x) - g(x)$ measures the "bias" of the gradient estimate, while $G - g(x)$ is the noise. Gradient methods with decreasing (or small) stepsizes naturally "average out" the noise. The version of gradient methods that are able to do this are called **stochastic gradient** methods. Naturally, these methods are slower when the noise is larger and in general cannot converge faster than how fast the noise averages out. In particular, in persistent noise (i.e., noise with nonvanishing variance), the best rate available for stochastic gradient methods is $O(1/\sqrt{t})$. While this can be slower than what can be achieved without noise, if the iteration cost is polynomial in the relevant quantities, the total cost of achieving an $\varepsilon > 0$ stationary point can be bounded by a polynomial in these quantities and $1/\varepsilon^2$.

When the gradient estimates are biased, the bias will in general put a limit on how close a gradient method can get to a stationary point. While generally a zero bias is preferred to a nonzero bias, a nonzero bias which is positively aligned with the gradient ($\langle b(x), \nabla_x J(\pi_x)\rangle \ge 0$) does not hurt (again, for small stepsizes). When there is no way to guarantee that the bias is positively aligned with the gradient, one may get back into control by making sure that the magnitude of the bias is small relative to the magnitude of the gradient.

The next question is of course, how to estimate the gradient. For this many approaches have been proposed in the literature. When a simulator is available, as in our case, a straightforward approach is to start from the policy gradient theorem. Indeed, under mild regularity conditions (e.g., if there are finitely many states) (1) together with (8) gives

$$\frac{d}{dx} J(\pi_x) = \int_{\mathcal{S}} \tilde{\nu}_\mu^{\pi_x}(ds) \int_{\mathcal{A}} \pi_x(a|s) \left(\frac{d}{dx} \log \pi_x(a|s)\right) (q^{\pi_x}(s,a) - b(s,x))\lambda(da)\,. \quad (9)$$

Now note that $(1-\gamma)\tilde{\nu}_\mu^{\pi_x}$ is a probability measure over $\mathcal{S}$. Let $S_0, A_0, S_1, A_1, \dots$ be an infinite sequence of state-action pairs obtained by simulating policy π_x starting from $S_0 \sim \mu$. In particular, $A_t \sim \pi_x(\cdot|S_t)$ and $S_{t+1} \sim P_{A_t}(S_t)$ for any $t \ge 0$. In addition, define T_1, T_2 to be independent of each other and from the trajectory $S_0, A_0, S_1, A_1, \dots$ and have a geometric distribution with parameter $1-\gamma$. Then,

$$G = \frac{1}{1-\gamma} \frac{d}{dx} \log \pi_x(A_{T_1}|S_{T_1}) \left(\sum_{t=0}^{T_2-1} r_{A_{T_1+t}}(S_{T_1+t}) - b(S_{T_1}, x) \right)$$

is an unbiased estimate of $\frac{d}{dx} J(\pi_x)$:

$$\mathbb{E}[G] = \frac{d}{dx} J(\pi_x) \,.$$

The argument to show this has partially be given earlier in Lecture 8. One can also show that G has a finite covariance matrix, as well as that the expected effort to obtain G is $O(\frac{1}{1-\gamma})$.

Vanilla policy gradients (PG) with some special policy classes

Given the hardness result presented in the previous lecture, there is no hope that gradient methods or any other method will find the global optima of the objective function in policy search in a policy-class agnostic manner. To guarantee computational efficiency, one then

1. either needs to give up on convergence to a global optima, or
2. give up on generality, i.e., give up on that the method should work for any policy class and/or policy parameterization.

Gradient ascent to find a good policy ("vanilla policy gradients") is one possible approach to take even if it faces these restrictions. In fact, gradient ascent in some cases will find a globally optimal policy.

In particular, it has been long known that with small enough stepsizes gradient ascent converges at a reasonable speed to a global optimum provided that two conditions hold:

1. The objective function f is smooth (its derivative is Lipschitz continuous);
2. The objective function is gradient dominated, i.e., with some constants $c > 0, p \ge 1$, f satisfies $\sup_x f(x) - f(x') \le c\|f'(x')\|_2^p$ for any $x' \in \mathbb{R}^d$.

An example when both of these conditions are met is the **direct policy parameterization**, which does not allow any compression and is thus not helpful per se, but can serve as a test-case to see how far policy gradient (PG) methods can be pushed.

In this case, the parameter vector θ is SA dimensional. By allowing "two-dimensional index", $\pi_\theta(a|s) = \theta_{s,a}$, that is, the parameters encode the action selection probabilities in a direct manner. In this case, since the components of θ represent probabilities, they need to be nonnegative and the appropriate components needs to sum to one. Hence, $\theta \in \Theta$ for an appropriate set $\Theta \subset [0,1]^{\mathrm{SA}}$. Accordingly, one needs to change gradient ascent.

This is done as follows: When a proposed update moves the parameter vector outside of Θ, the proposed updated parameter vector is "back-projected" to Θ. For the projection there are a number of reasonable options, such as choosing the point within Θ which is closest to the proposed point in the standard Euclidean distance. With this modification, gradient ascent can be shown to converge at a reasonable speed in this case. This parallels the methods that were developed for the tabular case (policy iteration, value iteration). In fact, the algorithm can be seen as a "smoother", incremental version of policy iteration, which gradually adjusts the probabilities assigned to the individual actions. Using π_i to denote the i th policy, from the policy gradient theorem one gets

$$\tilde{\pi}_{i+1}(a|s) = \pi_i(a|s) + \alpha_i \tilde{\nu}_\mu^{\pi_i}(s) q^{\pi_i}(s,a)\,,$$

and

$$\pi_{i+1}(\cdot|s) = \arg\min_{p\in\mathcal{M}_1(\mathcal{A})} \|p - \pi_{i+1}(\cdot|s)\|_2, \qquad s \in \mathcal{S}\,.$$

Thus, the probability of an action in a state is increased in proportion to the value of that state.

That the action-value of action a at state s is multiplied with the discounted occupancy at s induced by using policy π_i started from μ is a bit of a surprise. In particular, if a state is inaccessible under policy π_i, the corresponding probabilities will not be updated. In fact, because this, the above iteration may get stuck at a suboptimal policy. The reader is invited to construct an example when this happens. To prevent this, it turns out to be sufficient if there is a constant $C > 0$ such that it holds that

$$\tilde{\nu}_\mu^{\pi^*}(s) \ge C\mu(s)\,, \qquad \text{for all } s \in \mathcal{S}\,, \tag{10}$$

where π^* is an optimal policy. Since μ appears on both sides and π^* is unknown, this condition does not look to helpful. However, if one chooses μ to be positive everywhere, the condition is clearly met. In any case, when (10) holds, gradient dominance and smoothness can be both verified, which in turn implies that the above update will converge at a geometric speed, the geometric speed involves an instance dependent constant which has no polynomial bound in terms of $H_\gamma = 1/(1-\gamma)$ and the size of the state-action space. Needless to say this is quite unattractive.

Policy gradient methods can be sensitive to how policies are parameterized. For illustration, consider still the "tabular case", just now change the way the memoryless policies are represented. One possibility is to use the Boltzmann, also known as the **softmax** representation. In this case $\theta \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ and

$$\pi_\theta(a|s) = \frac{\exp(\theta_{s,a})}{\sum_{a'} \exp(\theta_{s,a'})}\,, \qquad (s,a) \in \mathcal{S}\times\mathcal{A}\,.$$

A straightforward calculation gives

$$\frac{\partial}{\partial \theta_{s,a}} \log \pi_\theta(a'|s') = \mathbb{I}(s=s', a=a') - \pi_\theta(a|s)\mathbb{I}(s=s')$$

and hence

$$\begin{aligned}\frac{\partial}{\partial \theta_{(s,a)}} J(\pi_\theta) &= \sum_{s'} \tilde{\nu}_\mu^{\pi_\theta}(s') \sum_{a'} \pi_\theta(a'|s) \frac{\partial}{\partial \theta_{s,a}} \log \pi_\theta(a'|s') q^{\pi_\theta}(s',a') \\ &= \nu_\mu^{\pi_\theta}(s,a)\left(q^{\pi_\theta}(s,a) - v^{\pi_\theta}(s)\right),\end{aligned}$$

where recall that ν_μ^π is the discounted state-occupancy measure over the state-action pairs of policy π when the initial state distribution is μ. The difference in the bracket on the right-hand side is known as the **advantage** of action a and, accordingly, the function

$$\mathfrak{a}^\pi = q^\pi - v^\pi\,,$$

which is a function mapping state-action pairs to reals, is called the **advantage function** underlying policy π. To justify the terminology, note that policy iteration can be seen as choosing in each state the action that maximizes the "advantage". Thus, we expect that we get a better policy if the "probability mass" in the action distribution is shifted towards actions with a larger advantage. Note though that advantages (as defined above) can also be negative and in fact if π is optimal, all actions have nonnegative advantages only.

The gradient ascent rule prescribes that

$$\theta_{i+1} = \theta_i + \alpha_i \nu_\mu^{\pi_{\theta_i}} \circ \mathfrak{a}^{\pi_{\theta_i}}\,,$$

where $\circ$ denotes componentwise product. While this is similar to the previous update, now the meaning of parameters is quite different. In fact, just because a parameter is increased does not necessarily mean that the probability of the corresponding action is increased: This will only happen if the increase of this parameter exceeds that of the other parameters "at the same state". By slightly abusing notation with defining $\pi_i = \pi_{\theta_i}$, we have

$$\pi_{i+1}(a|s) \propto \pi_i(a|s) \exp(\alpha_i \nu_\mu^{\pi_i}(s,a) \mathfrak{a}^{\pi_i}(s,a)) \,. \tag{11}$$

Just like in the previous update rule, we also see the occupancy measure “weighting” the update. This is again not necessarily helpful and if anything, again, speaks to the arbitrariness of gradient methods. And while this does not entirely stop policy gradient to find an optimal policy, and again, one can even show that the speed is geometric, though, as before, the algorithm altogether fails to run in polynomial time in the relevant quantities. For this theorem which we give without proof recall that $H_\gamma = 1/(1-\gamma)$.

Theorem (PG is slow with Boltzmann policies): There exists universal constants $\gamma_0, c, C > 0$ such that for any $\gamma_0 < \gamma < 1$, if $\mathrm{S} > CH_\gamma^6$ then one can find a discounted MDP with S states and 3 actions, setting μ to be the uniform distribution and initializing the parameters so that π_0 is the uniform random policy, softmax PG with a constant stepsize of $\alpha > 0$ takes at least

$$\frac{c}{\alpha} \mathrm{S}^{2^{\Omega(H_\gamma)}}$$

iterations.

As one expects that without any compression, the chosen planner should behave reasonably, this rules out the “vanilla” version of policy gradient.

Natural policy gradient (NPG) methods

In fact, a quite unsatisfactory property of gradient ascent that the speed at which it converges can greatly depend on the parameterization used. Thus, for the same policy class, there are many possible “gradient directions”, depending on the parameterization chosen. What is a gradient direction for one parameterization is not necessarily a gradient direction for another one. But what is common about these directions that an infinitesimal step along them is guaranteed increase the objective. One can in fact take a direction obtained with a parameterization and look at what direction it gives with another parameterizations. To get some order, consider transforming all these directions into the space that corresponds to the direct parameterization. It is not hard to see that all possible directions that are within 90 degrees of the gradient direction with this parameterization can be obtained by considering an appropriate parameterization.

More generally, regardless of parameterization, all directions within 90 degrees of the gradient direction are ascent directions. This motivates changing the stepsize α_i from a scalar to a matrix A_i. Clearly, to keep the angle between the original gradient direction g and the transformed direction $A_i g$ below 90 degrees, $g^\top A_i g \ge 0$ has to hold. For A_i symmetric, this restricts the set of matrix "stepsizes" to the set of positive definite matrices (still, a large set).

There are many ways to choose a matrix stepsize. Newton's method is to choose it so that the direction is the "best" if the function is replaced by its local quadratic approximation. This provably helps to reduce the number of iterations when the objective function is "ill-conditioned", though all matrix stepsize methods incur additional cost per each iteration, which will often offset the gains.

Another idea, which comes from statistical problems where one often works with distributions is to find the direction of update which coincides with the direction one would obtain if one used the steepest descent direction directly in the space of distributions where distances are measured with respect to relative entropy. In some cases, this approach, which was coined the "natural gradient" approach, has been shown to give better results, though the evidence is purely empirical.

As it turns out, the matrix stepsize to be used with this approach is the (pseudo)inverse of the so-called Fisher information matrix. In our context, for every state, we have distributions over the actions. Fixing a state s, the Fisher information matrix becomes

$$F_x(s) = \frac{d}{dx}\log \pi_x(\cdot|s)\, \frac{d}{dx}\log \pi_x(\cdot|s)^\top \,.$$

To get the "information rate" over the states, one can sum these matrices up, weighted by the discounted state occupancy measure underlying μ and π_x to get

$$F(x) := \nu_\mu^{\pi_x} F_x \,.$$

The update rule then takes the form

$$x_{i+1} = x_i + \alpha_i F(x_i)^\dagger \nabla_x J(\pi_x)\,,$$

where for a square matrix A, $A^\dagger$ denotes the pseudoinverse of A. Interestingly, the update direction can be obtained without calculating F and inverting it:

Proposition: We have

$$(1-\gamma)F(x)^\dagger \nabla_x J(\pi_x) = \arg\min_{w\in\mathbb{R}^d} \nu_\mu^{\pi_x}\left(w^\top \nabla_x \log \pi_x(\cdot|\cdot) - \mathfrak{a}^{\pi_x}\right)^2 ,$$

where $\mathfrak{a}^{\pi_x} = q^{\pi_x} - v^{\pi_x}$ and arg min chooses the minimum $\|\cdot\|_2$-norm solution if multiple minimizers exist.

Proof: Just recall the formula that gives the solution to a least-squares problem. The details are left to the reader. ■

As an example of how things look like consider the case when π_x takes the form of a Boltzmann policy:

$$\pi_x(a|s) \propto \exp(x^\top \phi(s,a)) ,$$

where $\phi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^d$ is a feature-map. Then, assuming that there are finitely many actions,

$$\nabla_x \log \pi_x(a|s) = \underbrace{\phi(s,a) - \sum_{a'} \pi_x(a'|s)\phi(s,a')}_{\psi_x(s,a)} .$$

Then, the natural policy gradient update takes the form

$$x_{i+1} = x_i + \alpha_i w_i ,$$

where

$$w_i = \arg\min_{w\in\mathbb{R}^d} \nu_\mu^{\pi_x}\left(w^\top \psi_x - \mathfrak{a}^{\pi_{x_i}}\right)^2$$

In the tabular case ($d = \mathrm{SA}$, no compression),

$$w_i(s,a) = \mathfrak{a}^{\pi_{x_i}}(s,a)$$

and thus

$$\pi_{i+1}(a|s) \propto \pi_i(a|s)\exp(\alpha_i \mathfrak{a}^{\pi_i}(s,a)) = \pi_i(a|s)\exp(\alpha_i q^{\pi_i}(s,a)) .$$

Note that this update rule eliminates the term $\nu_\mu^{\pi_i}(s,a)$ term that we have previously seen (cf. (11)).

NPG is known to enjoy a reasonable speed of convergence, which gives altogether polynomial planning time. This is promising. No similar results are available for the nontabular case.

Note that if we (arbitrarily) change the definition of w_i by replacing ψ_x above with ϕ and a^{π_x} with q^{π_x}, we get what has been called in the literature Q-NPG:

$$w_i = \arg\min_{w \in \mathbb{R}^d} \nu_\mu^{\pi_x} \left(w^\top \phi - q^{\pi_x}\right)^2 .$$

Note that the only difference between Q-NPG and Politex is that in Politex one uses

$$w_i = \arg\min_{w \in \mathbb{R}^d} \hat{\nu}\left(w^\top \phi - q^{\pi_x}\right)^2 ,$$

where $\hat{\nu}$ is the measure obtained from solving the G-optimal design problem.

The price of not using $\hat{\nu}$ but using $\nu_\mu^{\pi_x}$ in Q-NPG is that the approximation error in Q-NPG becomes

$$\frac{C\varepsilon}{(1-\gamma)^{1.5}}$$

where

$$C = \left\| \frac{d\tilde{\nu}_\mu^{\pi^*}}{d\mu} \right\|_\infty$$

gives a bound on how much the distribution μ differs from that of obtained when the optimal policy π^* is followed from μ. As was argued before, it is necessary that C is finite for policy gradient methods not to "get stuck" at local optima. However, C can be arbitrarily large even for finite state-action MDPs; an in fact it is the presence of C that makes the policy gradient with the direct parameterization a slow algorithm.

In contrast, the same quantity in Politex is

$$\frac{\sqrt{d}\varepsilon}{1-\gamma} .$$

Not only the uncontrolled constant C is removed, but the dependence on the planning horizon is also improved. Other than these differences, the results available for Q-NPG are similar to that of Politex and in fact the proof technique to obtain the results is also the same.

The proof of the Calculus 101 result

For completeness, here is the proof of (2). For the proof recall that for a function $g : \mathbb{R}^d \to \mathbb{R}$, $\frac{d}{dx} g(x_0)$ is the unique linear operator (row vector, in the Euclidean case) that satisfies

$$g(x)=g(x_0)+\frac{d}{dx}g(x_0)(x-x_0)+o(\|x-x_0\|) \text{ as } x\to x_0\,.$$

Hence, it suffices to show that

$$f(x',x')=f(x,x)+\left(\frac{\partial}{\partial u}f(u,x)|_{u=x}+\frac{\partial}{\partial v}f(x,v)|_{v=x}\right)(x'-x)+o(\|x'-x\|)\,.$$

To minimize clutter we will write $\frac{\partial}{\partial u}f(x',x)$ for $\frac{\partial}{\partial u}f(u,x)|_{u=x'}$ (and similarly we write $\frac{\partial}{\partial v}f(x,x')$ for $\frac{\partial}{\partial v}f(x,v)|_{v=x'}$).

By definition we have

$$f(x',x')=f(x',x)+\frac{\partial}{\partial v}f(x',x)(x'-x)+o(\|x'-x\|)$$

and

$$f(x',x)=f(x,x)+\frac{\partial}{\partial u}f(x,x)(x'-x)+o(\|x'-x\|)\,.$$

Putting these together we get

$$\begin{aligned}f(x',x')&=f(x,x)+\left(\frac{\partial}{\partial v}f(x',x)+\frac{\partial}{\partial u}f(x,x)\right)(x'-x)+o(\|x'-x\|)\\&=f(x,x)+\left(\frac{\partial}{\partial v}f(x,x)+\frac{\partial}{\partial u}f(x,x)\right)(x'-x)\\&\qquad+\left(\frac{\partial}{\partial v}f(x',x)-\frac{\partial}{\partial v}f(x,x)\right)(x'-x)+o(\|x'-x\|)\\&=f(x,x)+\left(\frac{\partial}{\partial v}f(x,x)+\frac{\partial}{\partial u}f(x,x)\right)(x'-x)+o(\|x'-x\|)\,.\end{aligned}$$

where the last equality follows if $\frac{\partial}{\partial v}f(x',x)-\frac{\partial}{\partial v}f(x,x)=o(1)$ as $x'\to x$, i.e., if $x'\mapsto\frac{\partial}{\partial v}f(x',x)$ is continuous at $x'=x$.

That the result also holds under the assumption that $x'\mapsto\frac{\partial}{\partial u}f(x,x')$ is continuous at $x'=x$ follows from a symmetric argument. $\blacksquare$

Summary

While policy gradient methods remain extremely popular and the idea of directly searching in the set of policies is attractive, at the moment it appears that they not only lack theoretical support, but the theoretical results suggest that it is hard to find any setting where policy gradient methods would be provably competitive with alternatives. At minimum, they need careful choices of policy parameterizations and even in that case the update rule may need to be changed to guarantee efficiency and effectiveness, as we have seen above. As an approach to algorithm design their main advantage is their generality and a strong support through various software libraries. Compared to vanilla "dynamic programming" methods they make generally smaller, more incremental changes to the policies, which seems useful. However, this is also achieved by methods like Politex, which is derived using a "bound minimization" approach. While this may seem more ad hoc than following gradients, in fact, one may argue that following gradients is more ad hoc as it fails to guarantee good performance. However, perhaps the most important point here is that one should not care too much about how a method is derived, or what "interpretation" it may have (is Politex a gradient algorithm? does this matter?). What matters is the outcome: In this case how the methods perform. It is thus wise to learn about all possible ways of designing algorithms, especially since there is much room for improving the performance of current algorithms.

Notes

Philip Thomas (2014, see citation below) takes a careful look at the claims surrounding natural gradient descent. One claim that is often heard is that natural gradient descent will speed up convergence. This is usually back up by giving a demonstration (e.g., Kakade, 2002, or Amari, 1998). However, it is far from clear whether this speedup will necessarily happen. As it turns out, this is far from being true. In fact, natural policy gradient can cause divergence even where following the normal gradient is guaranteed to converge to a global optimum. An example of this is given in Section 6.5 of the paper of Thomas (2014).

References

- Amari, S. Natural gradient works efficiently in learning.Neural Computation, 10:251–276, 1998.
- Kakade, S. A natural policy gradient. In Advances in Neural Information Processing Systems, volume 14, pp.1531–1538, 2002.
- Bagnell, J. A. and Schneider, J. Covariant policy search. In Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1019–1024, 2003.
- Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforce-ment learning with function approximation. In Neural Information Processing Systems 12, pages 1057–1063.
- Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. "Deterministic Policy Gradient Algorithms." In ICML. http://hal.inria.fr/hal-00938992/.

- Bhandari, Jalaj, and Daniel Russo. 2019. "Global Optimality Guarantees For Policy Gradient Methods," June. https://arxiv.org/abs/1906.01786v1.
- Agarwal, Alekh, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. 2019. "On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/1908.00261.
- Mei, Jincheng, Chenjun Xiao, Csaba Szepesvari, and Dale Schuurmans. 2020. "On the Global Convergence Rates of Softmax Policy Gradient Methods." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/2005.06392.
- Zhang, Junyu, Alec Koppel, Amrit Singh Bedi, Csaba Szepesvari, and Mengdi Wang. 2020. "Variational Policy Gradient Method for Reinforcement Learning with General Utilities." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/2007.02151.
- Bhandari, Jalaj, and Daniel Russo. 2020. "A Note on the Linear Convergence of Policy Gradient Methods." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/2007.11120.
- Chung, Wesley, Valentin Thomas, Marlos C. Machado, and Nicolas Le Roux. 2020. "Beyond Variance Reduction: Understanding the True Impact of Baselines on Policy Optimization." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/2008.13773.
- Li, Gen, Yuting Wei, Yuejie Chi, Yuantao Gu, and Yuxin Chen. 2021. "Softmax Policy Gradient Methods Can Take Exponential Time to Converge." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/2102.11270.
- Thomas, Philip S. "GeNGA: A Generalization of Natural Gradient Ascent with Positive and Negative Convergence Results." ICML 2014. http://proceedings.mlr.press/v32/thomasb14.pdf.

The paper to read about natural gradient methods:

- Martens, James. 2014. "New Insights and Perspectives on the Natural Gradient Method," December. https://arxiv.org/abs/1412.1193v9. Last update: September, 2020.