ECE 8803: Online Decision Making in Machine Learning **Fall 2021**

Lecture 22: November 10

Lecturer: Vidya Muthukumar

Thus far, we have examined *model-based* approaches to RL, which proceeded in two steps:

- Estimate the parameters of the MDP, and
- Run a dynamic programming algorithm on the estimated MDP.

Model-based approaches are among some of the conceptually simplest to understand (and in fact turn out to be more sample-efficient than the so-called "model-free" methods, which we will discuss today). However, they turn out to not be very scalable in practice: a central reason is the *storage* requirement of $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$, as we need to store all of the parameters of the MDP. An alternative is to try to directly estimate (or learn) the iterative procedure that is used in dynamic programming. This turns out to reduce the storage requirement to $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ (and also plays well with function approximation schemes that are dominant in RL practice). In today's lecture, we will get a birds' eye view of various types of iterative RL approaches and understand how/why they work.

22.1. Recap of notation and models

Our model for state evolution (dynamics) constitutes a *Markov Decision Process* with a finite number of states and actions. Concretely,

- $\mathcal{S}$ denotes the finite state space.
- $\mathcal{A}$ denotes the finite set of actions that can be taken at any state.
- $\boldsymbol{P}$ denotes the set of transition probabilities; in particular, $P(s'|s,a)$ denotes the probability that we transition from state s to state s' given that action a was played.
- $\boldsymbol{r}$ denotes the set of immediate rewards; in particular, $r(s,a)$ denotes the immediate reward that we would obtain if we played action a in state s.

To model delayed impacts of actions, we need to introduce a *horizon* for sequential decision-making. So far, we have seen

- The finite-horizon model, in which we begin at state s_1 at time step 1 and we sequentially make decisions until we end at time step H. We will not engage with this model in depth today, so we do not recap the details.

- The infinite-horizon discounted model, in which we begin at state s_0 at time step 0 and the horizon $H = \infty$, but rewards are *discounted* over time at a rate given by $\gamma \in (0, 1)$. Under this model, the value function of a policy $\boldsymbol{\pi} := (\pi_1, \pi_2, \dots,)$ is given by

$$V^{\boldsymbol{\pi}}(s_0) := \mathbb{E}_{\boldsymbol{\pi}, \boldsymbol{P}} \left[\sum_{h=0}^{\infty} \gamma^h \cdot r(s_h, \pi_h(s_h)) \right], \tag{22.1}$$

  and again, an optimal policy $\boldsymbol{\pi}^*$ is one that maximizes the value function $V^{\boldsymbol{\pi}}(s_1)$. The infinite-horizon nature of the discounted model leads to an elegant solution to optimal policy computation: there always exists a *stationary* policy $\boldsymbol{\pi}^*_{\mathsf{stat}} := (\pi^*, \pi^*, \dots,)$ that maximizes the value function $V^{\boldsymbol{\pi}}(s_0)$. As a result, it suffices to optimize over (and learn) stationary policies under the discounted model. Henceforth, while discussing the discounted model we will only consider stationary policies $\boldsymbol{\pi}_{\mathsf{stat}} := (\pi, \pi, \dots,)$, and we will denote them by π as shorthand. Finally, we saw that we can write the optimal stationary policy π^* succinctly as

$$\begin{aligned} \pi^*(s) &:= \max_{a \in \mathcal{A}} Q^*(s, a) \text{ where} \\ Q^*(s, a) &:= r(s, a) + \gamma \cdot \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[V^*(s') \right]. \end{aligned}$$

  Above, $Q^*(s, a)$ denotes an *action-optimal-value* function: what will happen if we take action a in the current step and the optimal policy π^* in all the future (discounted steps). This means that the optimal policy essentially maximizes this value-to-go, and it also means that we have the recursive property that $V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a)$. These elegant properties will be repeatedly used in today's lecture.

Our discussion today will focus on the discounted model for notational convenience (but there are equivalents of all of this for finite-horizon RL).

22.2. Computing the optimal policy in discounted MDPs

To jumpstart our discussion of "iterative RL", we begin by addressing a question that we have swept under the rug for the past few lectures. We have seen that there exists an optimal stationary policy $\boldsymbol{\pi}_{\mathsf{stat}} := (\pi, \pi, \dots,)$ for the discounted MDP problem. How do we compute this efficiently? We now present two iterative algorithms that efficiently compute the optimal stationary policy. (See the additional notes and references section for a third.) The first is called *value iteration*, which is effectively an adaptation of dynamic programming for the discounted MDP problem. The second is called *policy iteration*, which iteratively evaluates a policy and then improves it. Both of these can be adapted to the RL setting in natural and scalable ways.

22.2.1 From finite-horizon DP to value iteration

Before we introduce the value iteration algorithm, it is useful to recall the structure of the dynamic programming algorithm that we introduced last Monday for a finite-horizon MDP.

- $h = H$ (base case): for every $s \in \mathcal{S}$, define $V^*_h(s) = \max_{a \in \mathcal{A}} r(s, a)$ and $\pi^*_h(s) := \arg\max_{a \in \mathcal{A}} r(s, a)$.

- For $h = H-1, \ldots, 1$ (induction step): for every $s \in \mathcal{S}$, define the expected values-to-go

$$V_h^*(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \mathbb{E}_{P(\cdot|s,a)} \left[V_{h+1}^*(s') \right] \right],$$

  and optimal-policy-to-go

$$\pi_h^*(s) = \arg\max_{a \in \mathcal{A}} \left[r(s,a) + \mathbb{E}_{P(\cdot|s,a)} \left[V_{h+1}^*(s') \right] \right].$$

- Optimal policy is given by $(\pi_1^*, \ldots, \pi_H^*)$, and optimal value starting from state s_1 is given by $V_1^*(s_1)$.

Recall that for every step h, we also defined $Q_h^*(s,a) := r(s,a) + \mathbb{E}_{P(\cdot|s,a)}[V_{h+1}^*(s')]$ as the *action-optimal value function*: what is the reward gained from step h onwards if we play action a in step h, and optimally thereafter? With this notation, we note that the updates at step h of DP look like

$$V_h^*(s) = \max_{a \in \mathcal{A}} Q_h^*(s,a) \tag{22.2a}$$

$$\pi_h^*(s) = \arg\max_{a \in \mathcal{A}} Q_h^*(s,a). \tag{22.2b}$$

It is conceptually unclear how to generalize the DP algorithm to an infinite-horizon setting: its underpinnings rely on knowing the end of the horizon H, after all. However, we can take inspiration from the structure in Equation (22.2) to write down an iterative procedure to solve the discounted MDP, which is called *value iteration*.

Definition 1 (Value iteration algorithm) *The value iteration (written in "synchronous"*[1] *form) aims to compute the optimal value and policy $(\boldsymbol{V}^*, \pi^*)$. It begins with an (arbitrarily chosen) set of value functions $\boldsymbol{V}_0$ and performs the following update at step t:*

$$V_{t+1}(s) = \max_{a \in \mathcal{A}} Q_t(s,a) \tag{22.3a}$$

$$\pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} Q_t(s,a). \tag{22.3b}$$

for every value of $s \in \mathcal{S}$. Above, $Q_t(\cdot,\cdot)$ denotes our current estimate of the action-optimal-value (optimal Q) function, given by

$$Q_t(s,a) := r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V_t(s')]. \tag{22.4}$$

*In sum, the update $\boldsymbol{V}_{t+1}$ is itself a function of $\boldsymbol{V}_t$ and the MDP parameters $r(s,a), P(s'|s,a)$. As shorthand, the value iteration update is often written as $\boldsymbol{V}_{t+1} = \mathcal{T}^* \boldsymbol{V}_t$ where $\mathcal{T}^*$ is called the **optimal Bellman operator**. In this notation, we have*

$$V_{t+1}(s) = \mathcal{T}^* V_t(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V_t(s')] \right] \text{ for every } s \in \mathcal{S}.$$

*It is worth verifying for yourself that $\boldsymbol{V}^* = \mathcal{T}^* \boldsymbol{V}^*$; in other words, the optimal value function is, itself, a **fixed point** of the optimal Bellman operator.*

1. There are also "asynchronous" versions of value iteration where at every step t the update is only made for one of the states. This distinction does not matter much for practical purposes, but will be useful to specify various forms of Q-learning in this lecture.

Notice that value iteration makes updates in the style of DP, but in kind of "forward" sense rather than backward: it begins with some arbitrary guess of the optimal values (given by $\boldsymbol{V}_0$), and iteratively adjusts it by computing an optimal value-to-go with the current guess of what the optimal value is. The ultimate hope is that $\boldsymbol{V}_t \to \boldsymbol{V}^*$, and so $\pi_t \to \pi^*$. The following result shows that this is, indeed, the case for any strictly discounted MDP (i.e. $\gamma < 1$).

Theorem 2 *Assume a discount factor $\gamma < 1$ and immediate rewards bounded between $[0, 1]$. Then, value iteration yields $\|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty \leq \frac{\gamma^t}{1-\gamma}$, that is, $|V_t(s) - V^*(s)| \leq \frac{\gamma^t}{1-\gamma}$ for all $s \in \mathcal{S}$. Since $|\mathcal{S}||\mathcal{A}|$ computations are done at every step, the total runtime of value-iteration to yield an ϵ-close approximation of the optimal value (from which we can approximate the optimal policy) can be shown to be given by $\mathcal{O}\left(\frac{|\mathcal{S}||\mathcal{A}|}{1-\gamma}\log\left(\frac{1}{(1-\gamma)\epsilon}\right)\right)$.*

Proof The proof of convergence of value iteration uses a really foundational idea of the optimal Bellman operator, which is the property of γ-*contractivity*. Intuitively, this says that the distance between two different initializations of the value function will decrease after one application of the Bellman operator (and, therefore, one iteration of value iteration!). Formally, this says that for *any* two different value functions $\boldsymbol{V}, \boldsymbol{V}'$, we have

$$\|\mathcal{T}^*\boldsymbol{V}' - \mathcal{T}^*\boldsymbol{V}\|_\infty \leq \gamma\|\boldsymbol{V}' - \boldsymbol{V}\|_\infty. \tag{22.5}$$

We will prove Equation (22.5) at the end of this proof. Equation (22.5) tells us that the distance between two different initializations of value iteration would decrease exponentially over time, and eventually they would converge to the same value function (which turns out to be the optimal value function). In fact, if one of the initializations was already at the optimum (i.e. $\boldsymbol{V}_0 = \boldsymbol{V}^*$), this equation can be used to show that value iteration initialized *anywhere* will yield values that get closer and closer to $\boldsymbol{V}^*$. Formally, we apply Equation (22.5) to $\boldsymbol{V}' := \boldsymbol{V}_t$ and $\boldsymbol{V} := \boldsymbol{V}^*$ to get

$$\begin{aligned}\|\boldsymbol{V}_{t+1} - \boldsymbol{V}^*\|_\infty &= \|\mathcal{T}^*\boldsymbol{V}_t - \mathcal{T}^*\boldsymbol{V}^*\|_\infty \\ &\leq \gamma\|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty.\end{aligned}$$

Above, the equality used the definition of the value iteration update *and* the fact that $\boldsymbol{V}^*$ is a fixed point of the Bellman operator. The inequality uses Equation (22.5). Applying this recursively, we get

$$\|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty \leq \gamma\|\boldsymbol{V}_{t-1} - \boldsymbol{V}^*\|_\infty \leq \gamma^2\|\boldsymbol{V}_{t-2} - \boldsymbol{V}^*\|_\infty \ldots \leq \gamma^t\|\boldsymbol{V}_0 - \boldsymbol{V}^*\|_\infty.$$

Therefore, we have

$$\|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty \leq \gamma^t\|\boldsymbol{V}_0 - \boldsymbol{V}^*\|_\infty \leq \frac{\gamma^t}{1-\gamma}.$$

Thus, assuming Equation (22.5), we have completed our proof. It remains to prove Equation (22.5). We prove the statement for any $s \in \mathcal{S}$. We will show that

$$\mathcal{T}^*V'(s) - \mathcal{T}^*V(s) \leq \gamma(V'(s) - V(s))$$

and a similar reverse inequality will hold. Let $a^* \in \mathcal{A}$ be the action such that $r(s,a^*) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a^*)}[V'(s')] = \mathcal{T}^* V'(s)$. Then, we have $\mathcal{T}^* V(s) \geq r(s,a^*) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a^*)}[V(s')]$ (as while a^* may coincide with the optimal Bellman operator for value function $\boldsymbol{V}'$, it does not for $\boldsymbol{V}$!). All of this gives us

$$\begin{aligned}\mathcal{T}^* V'(s) - \mathcal{T}^* V(s) &\leq r(s,a^*) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a^*)}[V'(s')] - (r(s,a^*) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a^*)}[V(s')]) \\ &= \gamma \mathbb{E}_{s' \sim P(\cdot|s,a^*)} \left[V'(s') - V(s')\right] \leq \gamma \|\boldsymbol{V}' - \boldsymbol{V}\|_\infty.\end{aligned}$$

A similar inequality holds for the reverse direction ($\mathcal{T}^* V(s) - \mathcal{T}^* V'(s)$), and for all states $s \in \mathcal{S}$; therefore, the proof of γ-contractivity is complete. ■

This proof can also be used to show that $\boldsymbol{V}^{\pi_t}$ is close in value to $\boldsymbol{V}^*$; we do not provide the details here.

22.3. From value iteration to Q-learning

Like the DP algorithm for finite-horizon MDPs, it is clear that value iteration requires exact knowledge of the MDP $\boldsymbol{P}, \boldsymbol{r}$. Our central goal in this lecture is to see how we can adapt the value iteration procedure to the RL setting in which we only have *samples* of the MDP. This will lead to the celebrated Q-learning algorithm. To see how value iteration leads to Q-learning, we first rewrite the update entirely in terms of Q-functions.

Definition 3 *From Definition 1, we see that value iteration maintains Q-function estimates given by $Q_t(s,a)$ for all state-action pairs at the t-th iteration. We can rewrite Equation* (22.3) *entirely in terms of Q-functions as:*

$$Q_{t+1}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a \in \mathcal{A}} Q_t(s,a)\right] \tag{22.6}$$

This gives us a single update by which we can specify value iteration, and is often called ***Q-value iteration****. Note that the specification of the optimal policy remains the same, i.e. $\pi_t = \arg\max_{a \in \mathcal{A}} Q_t(s,a)$ at iteration t.*

The Q-value iteration form as written in Equation (22.6) will be convenient for us to define Q-learning. Note that the update in Equation (22.6) requires exact knowledge of the MDP in two ways: it uses the knowledge of the immediate rewards $r(s,a)$ and the transition $P(\cdot|s,a)$. Suppose that we (somehow) had access to the exact value of $Q_t(s,a)$ that we would have obtained via perfect value iteration. Then, suppose that we are at a particular state s and we sample action a, subsequently observing reward $\widehat{r}(s,a)$ and transiting to state s'. Then, we can *estimate* the Q-function update going from step t to $t+1$ as

$$\widehat{Q}_{t+1}(s,a) = \widehat{r}(s,a) + \gamma \max_{a \in \mathcal{A}} Q_t(s',a).$$

As $s' \sim P(\cdot|s,a)$ and $\widehat{r}(s,a)$ is in expectation equal to $r(s,a)$, this would be an *unbiased* update of Equation (22.6). However, because we are only taking one sample of the reward and transition, it would be of very high variance. The simplest and most intuitive way to reduce the variance of the update would be to repeatedly restart the process at (s,a) and take repeated samples. However, Q-learning does something much simpler (and, it turns

out, much more sample efficient overall!): it partially *bootstraps* the information present in the previous iteration and combines it with the high-variance estimate taken from a new sample, given above. As we will see, this property makes Q-learning flexibly implementable in a large number of simulation environments. Formally, the Q-learning algorithm is defined below.

Algorithm 1 The Q-learning algorithm (in synchronous form)

Input: Step-size schedule $\{\alpha_t\}_{t\geq 0}$ (typically decreasing)
Initialize $Q_0(s,a)$ arbitrarily for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.
while $t \geq 0$ **do**
  for all state-action pairs $(s,a) \times \mathcal{S} \times \mathcal{A}$ **do**
    $Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t \widehat{Q}_{t+1}(s,a)$
  end for
end while

Algorithm 1 shows that the Q-learning update can be written in just one line! Moreover, it only requires storage space $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ (one value corresponding to each state-action pair). For this simplicity and storage efficiency in part, Q-learning is popular in practice.

For convenience, the Q-learning update is often written as

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha_t(\widehat{Q}_{t+1}(s,a) - Q_t(s,a)). \tag{22.7}$$

The weights $\{\alpha_t\}_{t\geq 0}$ with which we combine these vary as more iterations are run in a manner that is interpretable and that we will explain shortly.

22.3.1 Convergence under the parallel sampling model

The convergence properties of the Q-learning algorithm depend intricately on the choice of step-size $\{\alpha_t\}_{t\geq 0}$. By definition, we will always have $\alpha_t \in [0,1]$. On one hand, a higher value of α_t leads to more weight being put on the noisy estimate of the Q-function, and so will lead to decreased bias but increased variance. On the other hand, a lower value of α_t leads to more weight being put on the previous estimate of the Q-function, and so will lead to decreased variance but increased bias. The right trade off between these factors turns out to be achieved by a *time-varying* step size, where $\alpha_t \to 0$ as $t \to \infty$. The intuition is roughly two-fold:

- At the initial stages of Q-learning, even the true Q-function from the idealized process of Q-value iteration (that has access to the MDP) would anyway be quite far away from the optimal Q-function. Therefore, the variance from the noise in estimating it makes little diference, and we might as well set our step size high in the hope of making progress one way or another.
- At the later stages of Q-learning, *if* learning has been successful we expect to be quite close to the optimal-Q-function already. Therefore, it is better to "bootstrap" these more reliable samples from previous iterations, rather than overweight the new sample which will have a lot of noise in it. We can do this by setting a smaller value of α_t as t increases, thus downweighting future samples as time goes on.

Essentially, although there is noise in each evaluation, this turns out to get "averaged out" in a nice way across iterations under these conditions. Incidentally, similar intuition (although we did not discuss it in this class) is also used to explain the convergence behavior of SGD! Although SGD and Q-learning apply to very different settings in practice, their convergence behavior can be explained through a common framework called *stochastic approximation* that was conceived in the 1950's. We provide the substance of the main result for Q-learning below.

Theorem 4 (Eventual convergence of Q-learning) *Suppose that we set the step sizes* $\{\alpha_t\}_{t\geq 0}$ *in a way such that*

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and} \tag{22.8a}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{22.8b}$$

Then, we have that $Q_t(s,a) \to Q^*(s,a)$ *as* $t \to \infty$ *for all* $(s,a) \in \mathcal{S} \times \mathcal{A}$.

The details behind Theorem 4 are out of scope for this lecture, but essentially demonstrate a delicate balance in the choice of step size decay: it can neither be too fast nor too slow. For example, Equation (22.8) is satisfied by $\alpha_t = \frac{1}{t}$ but not $\frac{1}{t^2}$ or $\sqrt{\frac{1}{t}}$. Finite-time convergence results for this setting also have been proved recently, using a sophisticated combination of the convergence of value iteration (proof of Theorem 2) and a careful control of the bias-variance tradeoff as intuitively discussed above. For example, Even-Dar et al. (2003); Wainwright (2019) show that Q-learning (in this parallel-sampling model) with a certain cleverly chosen step size needs $T := \mathcal{O}\left(\frac{1}{(1-\gamma)^5} \cdot \frac{1}{\epsilon^2}\right)$ iterations (for each state-action-pair) to converge. A careful comparison with the model-based results that we saw last Wednesday would reveal that the dependence of this result on the number of states and actions (noting that the total number of iterations is equal to $|\mathcal{S}||\mathcal{A}|T$) is the same, but the dependence on the effective horizon is worse for Q-learning than for the model-based method. This is one way to interpret the oft-made claim that "model-based methods are more sample-efficient in RL".

22.3.2 Convergence with a pre-run policy

Q-learning as presented in Algorithm 1 assumes perfect access to a "parallel-sampling" simulator that is able to query every state-action pair at every iteration. However, as we discussed in the previous lecture, we may not always have this flexibility. For example, we may have access to a trajectory $(s_1, a_1, s_2, a_2, \ldots, s_T, a_T, \ldots)$ from a specific policy π_b that was run, commonly called a *behavioral policy*, and the corresponding immediate reward values. As long as this policy has covered the MDP (i.e. visited all state-action pairs sufficiently often), we would still hope to be able to run the Q-learning algorithm in an iterative fashion and eventually see enough samples of all state-action pairs to converge.

This avatar of Q-learning, executed with samples from a *pre-run policy* (this is often called batch reinforcement learning, or *asynchronous* Q-learning), is described in Algorithm 2. It

Algorithm 2 The Q-learning algorithm (in asynchronous form)

Input: Step-size schedule $\{\alpha_t\}_{t\geq 0}$ (typically decreasing)
Initialize $Q_0(s,a)$ arbitrarily for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.
while $t \geq 0$ **do**
  $Q_{t+1}(s_t,a_t) = (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t \widehat{Q}_{t+1}(s_t,a_t)$
end while

can be shown (again in Even-Dar et al. (2003)) that this version of Q-learning also converges, but with a larger number of iterations that is effective multiplied by the expected visit time of all states (formally, called the *covering time*) of the behavioral policy π_b. The reason for this blow-up is that we need the expected covering time number of iterations to be able to see all state-action pairs at least once (which we could obtain in one iteration if we had access to the parallel-sampling simulator).

22.3.3 Online Q-learning: Exploration-vs-exploitation

We conclude this lecture by briefly describing yet another variant of Q-learning that is best suited for *online RL*, or the setting of RL that necessitates exploration. This was the setting that we discussed last lecture: we have control over the actions $\{a_t\}_{t\geq 1}$, but not the states (which will evolve as a function of the actions that we take and the underlying MDP). We have seen thus far that we would like our Q-learning algorithm to be able to visit all state-action pairs after a sufficient number of iterations (with high probability): the online RL setting allows us to adaptively pick our actions $\{a_t\}_{t\geq 1}$ to make this happen. Algorithm 3 depicts Q-learning run with an ϵ-greedy style exploration schedule: it balances

Algorithm 3 Online Q-learning with ϵ-greedy

Input: Step-size schedule $\{\alpha_t\}_{t\geq 0}$ (typically decreasing)
Initialize $Q_0(s,a)$ arbitrarily for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.
while $t \geq 0$ **do**
  $a_t = \arg\max_{a\in\mathcal{A}} Q_{t-1}(s_t,a)$ with probability $1-\epsilon_t$, $\text{Uniform}(\mathcal{A})$ with probability ϵ_t.
  $Q_{t+1}(s_t,a_t) = (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t \widehat{Q}_{t+1}(s_t,a_t)$.
end while

exploitation (picking the estimated optimal action at that round) and exploration (picking actions uniformly at random). Of course, there are many better ways of conducting this exploration. One way is to use UCB instead of ϵ-greedy (which has been done in recent work). Another way is to try and choose actions to facilitate the long-term exploration benefits of visiting infrequently seen states in future rounds; we saw these ideas last lecture in model-based RL. It is a currently exciting research direction to apply these ideas to the model-free Q-learning algorithm.

22.4. Additional references and notes

- Value iteration was conceived by Richard Bellman in 1957, but Q-learning was conceived more recently by Watkins and Dayan. The latter showed a possibility result

of the following form: *provided* that every state-action pair is visited sufficiently often, the Q-learning algorithm will eventually converge. Several such "asymptotic" results were then shown for all three variants of Q-learning described here, via the stochastic approximation framework. Stochastic approximation itself goes back to deeply mathematical ideas by mathematicians Robbins and Munro, and is widely used in optimization, numerical analysis and reinforcement learning.

- The "model-based" v.s. "model-free" debate is unresolved. While the discussion here (for finite-state-finite-action RL) implies that model-based methods require fewer overall samples to learn a good policy, model-free methods are more storage efficient (as they only need to store a Q-function, not an entire MDP). Moreover, as we will see next lecture, model-free methods are currently more amenable to function approximation when the state and action space of the MDP is extremely large (in which case we may not wish to run any of the algorithms that we have described thus far out of the box: even linear time in $|\mathcal{S}|, |\mathcal{A}|$ may be undesirable!).

- Note that value iteration does not give an approach to *exactly* solve for the optimal value and policy, unlike dynamic programming in the finite-horizon MDP — it only gives us approximations to these. Two alternative approaches that would solve the optimal value and policy *exactly* are *policy iteration* and *linear-programming based*. We will discuss policy iteration and the ensuing "policy-based" methods for RL next week. Unfortunately, we do not have time to cover LP-based methods for solving MDPs in this class (but they are a fascinating topic).

References

Eyal Even-Dar, Yishay Mansour, and Peter Bartlett. Learning rates for q-learning. *Journal of machine learning Research*, 5(1), 2003.

Martin J Wainwright. Stochastic approximation with cone-contractive operators: Sharp infty-bounds for q-learning. *arXiv preprint arXiv:1905.06265*, 2019.