Reinforcement Learning

# Lecture 3

*Lecturer: Haim Permuter* *Scribe: Bashar Huleihel*

In the course of this lecture, we will first introduce the Bellman optimality operator, $T$, and then, in continuation of the previous lecture, we will talk about value iteration in finite MDPs for which the environment is fully observed and known to the agent. We will also discuss the case where the environment is unknown to the agent and talk about Monte Carlo methods.

In the last lecture, we proved some properties on the operator $T_\pi$. We will start this lecture by showing the same properties on the Bellman optimality operator, $T$.
First let us define the state-value vector by

$$\mathbf{v} = \begin{bmatrix} v(s_1) \\ v(s_2) \\ \vdots \\ v(s_{|\mathcal{S}|}) \end{bmatrix},$$

and the operator $T$ by

$$\begin{aligned} T(\mathbf{v})(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_a \left[ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right], \quad \forall s \in \mathcal{S}. \end{aligned} \tag{1}$$

**Theorem 1 (Properties of $T$)** The operator $T$ satisfies the following properties: $\forall \mathbf{v}, \mathbf{v}' \in \mathbb{R}^{|\mathcal{S}|}, \quad s \in \mathcal{S},$

1. *monotonicity* $\quad v(s) \leq v'(s) \Rightarrow T(v)(s) \leq T(v')(s)$
2. *additivity* $\quad \forall d \in \mathbb{R}: \quad \tilde{v}(s) = v(s) + d \Rightarrow T(\tilde{v})(s) = T_\pi(v)(s) + \gamma d$
3. *$\gamma$-contraction* $\quad \forall \mathbf{v}, \mathbf{v}' \in \mathbb{R}^{|\mathcal{S}|} \quad \|T(\mathbf{v}) - T(\mathbf{v}')\|_\infty \leq \gamma \|\mathbf{v} - \mathbf{v}'\|_\infty$

**Proof** Let us prove the properties:

*monotonicity*: By using directly the definition of operator $T$,

$$\begin{aligned} T(v)(s) &= \max_a \left[ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right] \\ &\overset{(a)}{\leq} \max_a \left[ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v'(s') \right] \\ &= T(v')(s), \end{aligned}$$

where $(a)$ follows by the assumption that $v(s) \leq v'(s)$. Hence, $T(v')(s) \geq T(v)(s)$.

*additivity*: Let us compute $T(\tilde{v})(s)$ directly.

$$\begin{aligned} T(\tilde{v})(s) &= \max_a \left[ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)(v(s') + d) \right] \\ &= \max_a \left[ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)(v(s')) \right] + \max_a \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) d \\ &\overset{(a)}{=} T(v)(s) + \gamma d, \end{aligned}$$

where (a) follows because $\max_a \sum_{s' \in \mathcal{S}} p(s'|s,a) = 1$, $\forall a \in \mathcal{A}$.

$\gamma$*-contraction*: Let,

$$\begin{aligned} a_v^* &= \operatorname*{argmax}_a r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s'), \\ a_{v'}^* &= \operatorname*{argmax}_a r(s,a) + \gamma \sum_{s'} p(s'|s,a) v'(s'). \end{aligned}$$

then,

$$\begin{aligned} |T(v)(s) - T(v')(s)| &= \left| \max_a \left[ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right] - \max_a \left[ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v'(s') \right] \right| \\ &= \left[ r(s,a_v^*) + \gamma \sum_{s'} p(s'|s,a_v^*) v(s') \right] - \left[ r(s,a_{v'}^*) + \gamma \sum_{s'} p(s'|s,a_{v'}^*) v'(s') \right] \end{aligned} \tag{2}$$

$$\begin{aligned} &\overset{(a)}{\leq} \gamma \sum_{s'} p(s'|s,a_v^*)(v(s') - v'(s')) \\ &\leq \gamma \sum_{s'} p(s'|s,a_v^*) \|v - v'\|_\infty \end{aligned}$$

$$= \gamma \|v - v'\|_\infty,$$

where $(a)$ follows because $a_v^*$ is not optimal for the right term in 2. Note that as this is true for all $s \in \mathcal{S}$, it is therefore also true for $\max_{s \in \mathcal{S}} |T(v)(s) - T(v)(s)|$, which proves the property.

## I. VALUE ITERATION

In the last lecture, we discussed on policy iteration, which one of his drawbacks is that each of its iterations involves policy evaluation that may entail a protracted, iterative computation. If policy evaluation is done iteratively, then convergence exactly to $v_\pi$ occurs only in the limit. The policy evaluation step can be truncated in several ways without losing the convergence guarantees of policy iteration. One special case of truncation is when policy evaluation is stopped after just one backup of each state. This algorithm is called *value iteration*, and it can be written as a backup operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right], \end{aligned} \tag{3}$$

for all $s \in \mathcal{S}$. For arbitrary $v_0$, the sequence $v_k$ can be shown to converge to $v_*$ under the same conditions that guarantee the existence of $v_*$.

Another way of understanding value iteration is by citing the Bellman optimality equation,

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]. \end{aligned} \tag{4}$$

Value iteration is obtained by turning this equation into an update rule. Also, note that the value iteration backup is identical to the policy evaluation backup, except that the former requires that the maximum be taken over all actions.

Further note that value iteration requires an infinite number of iterations to converge exactly to $v_*$. In practice, in each of its sweeps, value iteration combines one sweep of policy evaluation and one sweep of policy improvement, and we stop once the value

function changes by only a small amount in a sweep.

An algorithm for value iteration with this kind of termination condition is depicted in Algorithm 1.

**Algorithm 1** Value iteration

**input:** Initialize arbitrarily an array $V$

**output:** deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')].$$

---

**repeat**

$\Delta = 0$

**For each** $s \in \mathcal{S}$**:**

$v = V(s)$

$V(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta = \max(\Delta, |v - V(s)|)$

**until** $\Delta > \epsilon$ (where $\epsilon$ is a small positive number)

---

### Asynchronous Dynamic Programming

A major drawback of the DP methods that we have discussed so far is that they require sweeps over the entire state set of the MDP. For very large state sets, therefore, even a single sweep may take long time. Indeed, even if we could perform the value iteration backup on a million states per second, it would take over a thousand years to complete a single sweep. *Asynchronous* DP algorithms are iterative DP algorithms that are not organized in terms of systematic sweeps of the state set. These algorithms back up the values of states in some order, using whatever values of other states happen to be available. The values of some states may be backed up several times before the values of others are backed up once. To converge, the algorithm must continue to back up the values of all the states. That is, it cannot ignore any state after some point in the computation.

### Generalized Policy Iteration

The term *generalized policy iteration* (GPI) refers to the general idea of letting policy evaluation and policy improvement processes interact. Almost all reinforcement learning methods are well described as GPI, since all have policies and value functions such that the policy is always being improved with respect to the value function and the value function is always being driven toward the value function for the policy. If both the evaluation process and the improvement process no longer generate changes, then the Bellman optimality equation is satisfied, and therefore, the value function and policy must be optimal. In practice, for the long run, the evaluation and improvement processes in GPI interact to find the optimal value function and an optimal policy.

## II. Monte Carlo

Here we consider our first learning methods for estimating value functions and discovering optimal policies. Unlike above, we do not assume complete knowledge of the environment. That is, these methods require only sample sequences of states, actions, and rewards from actual or simulated interaction with an environment. Monte Carlo methods are used to solve the reinforcement learning problem based on averaging sample returns. We define them only for episodic tasks, that is, we assume experience is divided into episodes and that episodes will eventually terminate (no matter what actions are selected). Only on the completion of an episode are value estimates and policies changed. Monte Carlo methods sample and average *returns* for each state - action pair and learn value functions from sample returns with the MDP.

### Monte Carlo Prediction

We begin by considering Monte Carlo methods for learning the state-value function for a given policy. Recall that the value of a state is the expected return from that state, and one way to estimate it from experience is simply to average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value. In particular, suppose we wish to estimate $v_\pi(s)$, the value of a state $s$ under policy $\pi$, given a set of episodes obtained by following $\pi$. Each occurrence of

state $s$ in an episode is called a *visit* to $s$. Let us call the first time $s$ is visited in an episode the *first visit* to $s$. The *first visit* MC method estimates $v_\pi(s)$ as the average of the returns following first visits to $s$, whereas the *every-visit* MC method averages the returns following all visits to $s$. The algorithm for the First-visit MC is depicted in Algorithm 2.

**Algorithm 2** First-visit MC prediction, for estimating $V \approx v_\pi$

**Initialize:**

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

*Returns(s)*$\leftarrow$ an empty list, for all $s \in \mathcal{S}$

**repeat**

Generate an episode using $\pi$

For each state $s \in \mathcal{S}$ appearing in the episode:

G$\leftarrow$ return following the first occurrence of $s$

*Returns(s)*$\leftarrow$ *Returns(s)* $+$ G

$V(s) \leftarrow$ average(*Returns(s)*)

**until** forever

Both first-visit MC and every-visit MC converge to $v_\pi(s)$ as the number of visits to $s$ goes to infinity. Unlike the case in DP, in Monte Carlo methods the estimate for one state does not build upon the estimate of any other state.

**Monte Carlo Estimation of Action Values**

When the model is not available, then it is particularly useful to estimate the state-action pair values rather than *state* values. With a model, state values alone are sufficient to determine a policy. However, without a model, state values alone are not sufficient, and one must explicitly estimate the value of each action for the values to be useful in suggesting a policy. Thus, one of our primary goals for Monte Carlo methods is to

estimate $q_*$. We want to estimate $q_\pi(s, a)$, the expected return when starting in state $s$, taking action $a$, and thereafter, following policy $\pi$. The first-visit MC method averages the returns following the first time in each episode that the state was visited and the action was selected. These methods converge quadratically, as before, to the true expected values as the number of visits to each stateaction pair goes to infinity. The only drawback is that many stateaction pairs may never be visited. Therefore, we need to estimate the values of all the actions from each state, not just the state we currently favor. This is the general problem of *maintaining exploration*. One way to do so is by specifying that the episodes start in a stateaction pair, and that every pair has a nonzero probability of being selected as the start. This guarantees that all stateaction pairs will be visited in the limit of an infinite number of episodes. The most common alternative approach is to consider only policies that are stochastic with a nonzero probability of selecting all actions in each state.

**Monte Carlo Control**

Here we will consider how Monte Carlo estimation can be used to approximate optimal policies. First let us consider a Monte Carlo version of classical policy iteration which is depicted by the following diagram:

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*}$$

We start with an arbitrary policy $\pi_0$, use policy evaluation ($\xrightarrow{E}$) to evaluate $v_{\pi_0}$ and then use policy improvement ($\xrightarrow{I}$) to find $\pi_1$. This process continues repeatedly until $v(\cdot)$ stops improving, i.e., $v(\cdot)$ the Bellman optimality condition is satisfied. In this section, we will assume that the episodes are generated with exploring starts, and we observe an infinite number of episodes. Under these assumptions, the Monte Carlo methods will compute each $q_{\pi_k}$ exactly, for arbitrary $\pi_k$, and the policy improvement theorem assures us that each $\pi_{k+1}$ is uniformly better than $\pi_k$, or just as good as $\pi_k$. This in turn assures us that the overall process converges to the optimal policy and optimal value function. Here, after each episode, the observed returns are used for policy evaluation, and then the policy is improved at all the states visited in the episode. A complete algorithm called Monte Carlo ES, for Monte Carlo with Exploring Starts, is depicted in Algorithm 3.

**Algorithm 3** Monte Carlo Exploring Starts, for estimating $\pi \approx \pi_*$

**Initialize:**

Arbitrary $Q(s, a)$

$\pi_s \leftarrow$ arbitrary policy

*Returns(s)*$\leftarrow$ an empty list

---

**repeat**

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S0)$ so that all pairs have probability $> 0$

Generate an episode starting from $S_0$, $A_0$ following $\pi$

For each state $s, a$ appearing in the episode:

G$\leftarrow$ the return that follows the first occurrence of $s, a$

*Returns(s,a)*$\leftarrow$ *Returns(s,a)* + G

$Q(s, a) \leftarrow$ average(*Returns(s,a)*)

For each $s$ in the episode:

$\pi(s) \leftarrow \text{argmax}_a\, Q(s, a)$

**until** forever

---

In this algorithm, all the returns for each stateaction pair are accumulated and averaged. One can see that Monte Carlo ES cannot converge to any suboptimal policy, as the value function would eventually converge to the value function for that policy, and that, in turn, would cause the policy to change.

**Monte Carlo Control without Exploring Starts**

Here we want to avoid the unlikely assumption of exploring starts. To do so, we will introduce two approaches which ensure that the agent select all the actions. The first approach is called *on-policy* methods, by which we attempt to evaluate or improve the policy that is used to make decisions. The second approach called *off-policy* methods, we use to evaluate or improve a policy different from that used to generate the data. In this section, we will show how an on-policy Monte Carlo control method can be

designed. We assume that the policy is generally *soft*, meaning that $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$(s). This method uses $\epsilon$-greedy policies, meaning that most of the time they choose an action that has maximal estimated action value, but with probability $\epsilon$, they select an action at random instead. That is, all nongreedy actions are given the minimal probability of selection, $\frac{\epsilon}{|\mathcal{A}(s)|}$, and the remaining probability, $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$, is given to the greedy action. For any $\epsilon$-soft policy, $\pi$, any $\epsilon$-greedy policy with respect to $q_\pi$ is guaranteed to be better than or equal to $\pi$. The complete algorithm is depicted below.

**Algorithm 4** On-policy first-visit MC control (for $\epsilon$-soft policies) estimates $\pi \approx \pi_*$

**Initialize (for all** $s \in \mathcal{S}, a \in \mathcal{A}$**):**

Arbitrary $Q(s, a)$

$\pi_{a|s} \leftarrow$ an arbitrary $\epsilon$-soft policy

*Returns(s,a)*$\leftarrow$ empty list

---

**repeat**

Generate an episode using policy $\pi$

For each state $s, a$ appearing in the episode:

G$\leftarrow$ the return that follows the first occurrence of $s, a$

*Returns(s,a)*$\leftarrow$ *Returns(s,a)* + G

$Q(s, a) \leftarrow$ average(*Returns(s,a)*)

For each $s$ in the episode:

$A^* \leftarrow \text{argmax}_a Q(s, a)$

For all $a \in \mathcal{A}$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{if } a = A^*, \\ \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{if } a \neq A^*. \end{cases}$$

**until** forever

# REFERENCES

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 2017.