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

# Lecture 23: November 15

*Lecturer: Vidya Muthukumar*



Last lecture, we gained a birds' eye view of *value-based* methods to a) efficiently compute the optimal policy an an infinite-horizon discounted MDP with full knowledge of the MDP via *value-iteration*, and b) efficiently learn an (approximately) optimal policy from samples via *Q-learning*. Value-based methods take an indirect route to computing or learning an approximately optimal policy by first estimating the optimal value function, and computing the "optimal" policy under that estimate.

One may ask whether there exist more direct methods to compute or learn the optimal policy by directly optimizing over policy-space. In today's lecture, we will take a close look at these more direct methods, which are also very popular in RL practice. We will also glimpse some interesting connections between these methods and online learning algorithms, which you studied in the first month of this course.

## 23.1. 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 $s$.

For notational convenience, in today's lecture we will continue to engage with 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

$$V^{\boldsymbol{\pi}}(s_0) := \mathbb{E}_{\boldsymbol{\pi},\boldsymbol{P}}\left[\sum_{h=0}^{\infty} \gamma^h \cdot r(s_h, \pi_h(s_h))\right], \tag{23.1}$$

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 $\pi$ as shorthand. Finally, we saw that we can write the optimal stationary policy $\pi^*$ 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 $\pi^*$ 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 $V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$.

For the purposes of this lecture, it is also convenient to define *policy-specific* Q-functions: for any stationary policy $\pi$, we can define its corresponding Q-function as

$$Q^{\pi}(s,a) := r(s,a) + \gamma \cdot \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[ V^{\pi}(s') \right]. \tag{23.2}$$

You can verify for yourself from the definitions above that $Q^{\pi^*}(s,a) = Q^*(s,a)$, and the two are often used synonymously. Moreover, the discounted model gives us another recursive property here: for any stationary policy $\pi$, we have

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot \mathbb{E}_{s' \sim P(\cdot|s,\pi(s))} \left[ V^{\pi}(s') \right]$$

and so we get

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s)). \tag{23.3}$$

This elegant property will be used in today's lecture.

## 23.2. Policy-based methods for computation in discounted MDPs

We specialize our discussion in this lecture to the discounted MDP setting, and re-examine the question of algorithm design for computing the optimal policy. Last lecture, we introduced and studied the *value iteration* algorithm, which was a DP-inspired approach that iterated on an estimate of an optimal value and then used that estimate to compute an optimal policy. We saw that, through a nice principle of *contractivity* of the optimal value estimate over time, that value iteration eventually converges to the optimal value, and therefore the optimal policy.

Today, we will see an alternative approach to the computation problem that directly iterates over (and turns out to provably improve) the policy estimate. This approach is commonly called *policy iteration*.

### 23.2.1 The policy iteration algorithm

The policy iteration algorithm is described in Algorithm 1. Notice that it directly iterates over *policies*, not *values*. This iteration procedure occurs in two steps:

**Algorithm 1** The policy iteration algorithm.

**Initialize** stationary policy $\pi_0$.
**while** $t \geq 0$ and $\pi_t \neq \pi_{t-1}$ **do**
  **Evaluate** value of current policy $\boldsymbol{V}^{\pi_t}$
  **Improve** policy estimate:
  **for** all states $s \in \mathcal{S}$ **do**
    $\pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} [Q^{\pi_t}(s, a)]$
  **end for**
**end while**

- *Evaluate* the value of the currently estimated policy $\pi_t$, given by $V^{\pi_t}$.
- *Improve* the policy from $\pi_t \to \pi_{t+1}$ by maximizing the Q-function for the current policy, i.e. $Q^{\pi_t}(s, a)$.

Our hope is to show that $\pi_t \to \pi^*$ eventually. The second step of the iteration procedure appears, in a way, quite heuristic. However, it turns out that we can always show a strict *policy improvement* at every round for which policy iteration is active. This is formally shown below.

**Theorem 1 (Policy improvement of policy iteration)** *For policy iteration (initialized anywhere), we either have $\pi_{t+1} = \pi_t$ (in which case policy iteration terminates with the optimal policy), or $V^{\pi_{t+1}}(s) > V^{\pi_t}(s)$ for all $s \in \mathcal{S}$.*

Theorem 1, in essence, shows that policy iteration terminates in at most $|\mathcal{A}|^{|\mathcal{S}|}$ steps (note that the total number of policies is in fact equal to $|\mathcal{A}|^{|\mathcal{S}|}$). This is because the policy improves at each step, and as long as $\pi_t \neq \pi_{t-1}$, we will find a new policy on every iteration. Thus, in the worst case we explore all the suboptimal policies and eventually terminate at the optimal policy $\pi^*$ (for which no improvement is possible). This proof is a bit more involved than the proofs for value iteration and uses a number of central ideas: it is detailed below.

**Proof** We need to show that for every $s \in \mathcal{S}$, we have $V^{\pi_{t+1}}(s) > V^{\pi_t}(s)$ when $\pi_{t+1} \neq \pi_t$. We only consider the case where $\pi_{t+1}(s) \neq \pi_t(s)$ (since $\pi_{t+1}(s) \neq \pi_t(s)$, at least one such state must exist). We start with two simple observations:

- Since $\pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} [Q^{\pi_t}(s, a)]$, we must have $Q^{\pi_t}(s, \pi_{t+1}(s)) > Q^{\pi_t}(s, \pi_t(s))$.
- By the aforementioned recursive property, we have $Q^{\pi_t}(s, \pi_t(s)) = V^{\pi_t}(s)$.

Thus, in sum, we have

$$r(s, \pi_{t+1}(s)) + \gamma \cdot \mathbb{E}_{s' \sim P(\cdot|s, \pi_{t+1}(s))} \left[ V^{\pi_t}(s') \right] > V^{\pi_t}(s). \tag{23.4}$$

Now, we consider the *on-policy* Bellman operator, given by

$$\mathcal{T}^{\pi_{t+1}} V(s) := r(s, \pi_{t+1}(s)) + \gamma \cdot \mathbb{E}_{s' \sim P(\cdot|s, \pi_{t+1}(s))} \left[ V(s') \right].$$

Notice, for example, that we can write $Q^{\pi_t}(s, \pi_{t+1}(s)) := \mathcal{T}^{\pi_{t+1}} V^{\pi_t}(s)$. Now, we can more compactly write Equation (23.4) as

$$\mathcal{T}^{\pi_{t+1}} V^{\pi_t}(s) > V^{\pi_t}(s) \text{ for all } s \in \mathcal{S}. \tag{23.5}$$

Now, the on-policy Bellman operator $\mathcal{T}^{\pi_{t+1}} \boldsymbol{V}$ is essentially linear in $\boldsymbol{V}$. Therefore, it can be shown to *preserve element-wise monotonicity* in the following sense: if we have $V(s) > V'(s)$ for all $s \in \mathcal{S}$, then we also have $\mathcal{T}^{\pi_{t+1}} V(s) > \mathcal{T}^{\pi_{t+1}} V'(s)$ for all $s \in \mathcal{S}$. Applying this to Equation (23.5) then gives us

$$(\mathcal{T}^{\pi_{t+1}})^2 V^{\pi_t}(s) > \mathcal{T}^{\pi_{t+1}} V^{\pi_t}(s) > V^{\pi_t}(s),$$

and continuing this line of argument would give us

$$(\mathcal{T}^{\pi_{t+1}})^\infty V^{\pi_t}(s) > \ldots > (\mathcal{T}^{\pi_{t+1}})^2 V^{\pi_t}(s) > \mathcal{T}^{\pi_{t+1}} V^{\pi_t}(s) > V^{\pi_t}(s).$$

To get our final step, we use one more important property of the on-policy Bellman operator: much like the *optimal* Bellman operator that we examined last lecture, it also contracts! In other words, for any two initializations of value functions $\boldsymbol{V}, \boldsymbol{V'}$ we have

$$|\mathcal{T}^{\pi_{t+1}} V(s) - \mathcal{T}^{\pi_{t+1}} V'(s)| \leq \gamma |V(s) - V'(s)| \text{ for all } s \in \mathcal{S}. \tag{23.6}$$

The proof of this contractivity property turns out to be quite similar to the proof of contractivity that we provided for the *optimal* Bellman operator last lecture; we do not provide it here. The contractivity property implies that applying the on-policy Bellman operator repeatedly gives us $(\mathcal{T}^{\pi_{t+1}})^\infty V(s) = V^{\pi_{t+1}}(s)$ for every $s \in \mathcal{S}$, i.e. it eventually gives us the *policy evaluation* for the policy $\pi_{t+1}$. Applying this for the special case $V(s) := V^{\pi_t}(s)$ then gives us

$$V^{\pi_{t+1}}(s) := (\mathcal{T}^{\pi_{t+1}})^\infty V^{\pi_t}(s) > V^{\pi_t}(s)$$

and so we have shown our strict policy improvement. ■

Theorem 1 demonstrate a remarkable property of *strict policy improvement*, that uses the special structure in the policy iteration update and two special properties of the on-policy Bellman operator: monotonicity and contractivity. Despite this elegance, Theorem 1 by itself does not guarantee a fast convergence rate for policy iteration, as the total number of policies is equal to $|\mathcal{A}|^{|\mathcal{S}|}$ and it appears that in the worst case we would cycle through all of them. It turns out that policy iteration can be shown to converge in a much faster number of steps (that is polynomial in $|\mathcal{S}|, |\mathcal{A}|, \frac{1}{1-\gamma}$) via deep connections to linear programming theory: see the additional notes and references for more details. Moreover, policy iteration is often observed to converge faster than value iteration in practice, making it an extremely attractive practical algorithm.

## 23.3. From discrete to continous optimization

The policy iteration algorithm is effectively a *discrete policy optimization* algorithm, as it aims to efficiently optimize over the set of stationary policies for the discounted MDP. As

with all of the other algorithms we have seen for optimal policy computation, it assumes exact knowledge of the MDP. To make it more amenable to RL settings, as well as to function approximation of policy space in very large MDPs (which we will investigate in some more depth next lecture), we would like to see whether we can effectively write the problem of policy optimization in a more *continuous* form. In the finite-state-finite-action setting that we are considering, it turns out to be convenient to make the policy space continuous through a *soft-max* policy representation, as detailed below.

**Definition 2** *For parameters $\boldsymbol{\theta} := \{\theta_{s,a}\}_{(s,a)\in\mathcal{S}\times\mathcal{A}}$, the soft-max policy representation is written as*

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp(\theta_{s,a})}{\sum_{a'\in\mathcal{A}}\exp(\theta_{s,a})}. \tag{23.7}$$

*Note that this policy is, in general stochastic, but includes the set of all deterministic policies (which can be obtained by setting the corresponding $\theta_{s,\pi(s)} \to \infty$ and $\theta_{s,a} < \infty$ for all $a \neq \pi^*(s)$). For any $\boldsymbol{\theta}$, we denote the value of the corresponding stochastic policy by $V^{\boldsymbol{\theta}}(s) := V^{\pi_{\boldsymbol{\theta}}}(s)$.*

Definition 2 is convenient because we can now think of the problem of computing an optimal policy as a continuous and *unconstrained* optimization problem (as we do not need to constrain the $\theta_{s,a}$ values at all). In other words, we want to solve

$$\max_{\boldsymbol{\theta}\in\mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}} V^{\boldsymbol{\theta}}(s_0) \tag{23.8}$$

for starting state $s_0$.

### 23.3.1 The policy gradient algorithm

The policy gradient algorithm is perhaps the first approach one would think of trying to solve Equation (23.8): it simply runs gradient ascent on the objective with respect to $\boldsymbol{\theta}$. In other words, the update is given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta\nabla V^{\boldsymbol{\theta}_t}(s_0). \tag{23.9}$$

Note that the plus sign is because we are doing maximization; thus a gradient *ascent* algorithm. The policy gradient update is conceptually simple to explain and is also flexible in implementation. While today, we are focusing on the parameter $\boldsymbol{\theta}$ representing a softmax policy, richer parameterizations are possible that incorporate powerful function approximators of the policy class, including linear models and neural networks. We will see this next lecture. In the meantime, a natural question that arises is whether the update can be computed or approximated efficiently; in other words, is the gradient efficiently computable? It turns out that this is the case, in a certain sense, for the soft-max parameterization that we have chosen. We state this result without the proof, which turns out to be an elegant algebraic manipulation.

**Theorem 3** *Let $\tau := (s_0, a_0, s_1, a_1, \dots, )$ denote a trajectory that could be induced by the stochastic policy $\pi_{\boldsymbol{\theta}}$. Corresponding to this trajectory, we define a total reward $G(\tau) :=$*

$\sum_{h=0}^{\infty} \gamma^h r(s_h, a_h)$ *and a probability distribution* $\mathbb{P}_{\boldsymbol{\theta}}[\tau] := \pi_{\boldsymbol{\theta}}(a_0|s_0) \cdot P(s_1|s_0, a_0) \cdot \pi_{\boldsymbol{\theta}}(a_1|s_1) \cdot P(s_2|s_1, a_1) \ldots$. *Then, we can write the value of the policy* $\pi_{\boldsymbol{\theta}}$ *as* $V^{\boldsymbol{\theta}}(s_0) = \sum_\tau \mathbb{P}_{\boldsymbol{\theta}}[\tau] \cdot G(\tau)$, *and we can write the policy gradient as*

$$\nabla V^{\boldsymbol{\theta}}(s_0) = \mathbb{E}_{\tau \sim \mathbb{P}_{\boldsymbol{\theta}}[\cdot]} \left[ G(\tau) \cdot \sum_{h=0}^{\infty} \nabla \log \pi_{\boldsymbol{\theta}}(a_h|s_h) \right], \tag{23.10}$$

*or*

$$\nabla V^{\boldsymbol{\theta}}(s_0) = \mathbb{E}\left[ Q^{\pi_{\boldsymbol{\theta}}}(s, a) \cdot \nabla \log \pi_{\boldsymbol{\theta}}(a|s) \right], \tag{23.11}$$

*where in the second case the expectation is taken over a notion of "stationary" distribution over all state-action pairs induced by the policy* $\pi_{\boldsymbol{\theta}}$.

The first expression above (Equation (23.10)) is commonly known as the REINFORCE algorithm. Note that these still do not imply an efficient *exact* computation of the gradients, but these turn out to be easily approximatable by sampling trajectories. We will return to this point at the end of the lecture.

Interestingly, the objective $V^{\boldsymbol{\theta}}(s_0)$ is quite non-concave in $\boldsymbol{\theta}$, and so it is not at all obvious that policy gradient will converge; nevertheless, recent theoretical results have shown that policy gradient can converge under suitable initialization properties (see the additional notes). These proofs are quite technically involved and we do not discuss them in detail here. Moreover, there is a more "natural" type of gradient-style algorithm that turns out to do much better, and also enjoys interesting connections to the discrete policy optimization algorithms that we have discussed.

### 23.3.2 A "natural" policy gradient algorithm

A well-known issue with policy gradient methods is that the gradients (at iteration $t$) turn out to scale *roughly directly proportionally* to the parameter vector $\boldsymbol{\theta}_t$; therefore, there is an issue that can arise where we move faster and faster towards a possible deterministic suboptimal policy (corresponding to $\boldsymbol{\theta}_t$ with one very large coordinate for each state) and it becomes difficult to escape this. A natural fix to this would be to add a prefactor to the gradient step that cancels out this undesirable scaling effect. This leads to the *natural policy gradient* method, defined below.

**Definition 4 (Natural policy gradient (NPG))** *Define the Fisher information matrix* $\boldsymbol{F}_t := \mathbb{E}\left[\nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla \log \pi_{\boldsymbol{\theta}}(a|s)^\top\right]$, *where the expectation as before is over the stationary distribution over state-action pairs. (Note that, because* $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, *the matrix* $\boldsymbol{F}_t \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|}$.*) Then, the NPG update takes the form*

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \boldsymbol{F}_t^\dagger \nabla V^{\boldsymbol{\theta}_t}(s_0),$$

*where for any matrix* $\boldsymbol{M}$, *its pseudoinverse is denoted by* $\boldsymbol{M}^\dagger$.

It turns out that the NPG update can equivalently written as a direct update on the stochastic policies as follows. We denote $\pi_t := \pi_{\boldsymbol{\theta}_t}$ as shorthand (we write $\pi_t(s, a)$ as the

probability that the policy prescribes policy $a$ in state $s$). Then, the NPG update turns out to be equivalent to the following policy update:

$$\pi_{t+1}(s,a) = \pi_t(s,a) \cdot \frac{\exp\left(\frac{\eta Q^{\pi_t}(s,a)}{(1-\gamma)}\right)}{\sum_{a' \neq a} \exp\left(\frac{\eta Q^{\pi_t}(s,a')}{1-\gamma}\right)}. \tag{23.12}$$

Notice that this is basically like the multiplicative weights update, except with a constant value of $\eta$! The $Q$-functions for the current policy constitute the current reward functions, and the update to $\pi_{t+1}(s,a)$ is a type of "soft" policy iteration (which is more soft, the smaller $\eta$ is).

It turns out that one could plug in regret bounds from the multiplicative weights algorithm (MWA) to prove convergence rates of NPG as a result of this connection. This would require the step size $\eta$ to decay with $t$ at the rate $1/\sqrt{t}$. However, the effective "rewards" given to the MWA, i.e. $Q^{\pi_t}(s,a)$, are relatively benign: they are a function of a smoothly varying policy, and in later rounds this policy may not vary much at all! It turns out to suffice to use a constant learning rate, which in turn turns out to lead to a much faster convergence rate of $\mathcal{O}\left(\frac{1}{T}\right)$ rate[1] to the optimal policy (as opposed to the slower $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$ rates that we saw for stochastic optimization long back). This proof also uses ideas from MWA together with the special structure in the reward sequence, and essentially could be thought of as a "constant-regret"guarantee.

## 23.4. From optimization to learning

The optimization theory that we have presented above is fascinating in its own right. However, to make it work with RL we need to make it amenable to samples of the MDP. Our sampling model for direct policy-based methods for RL is a little different from the ones that we have seen thus far, as it explicitly evaluates policies at every step. In particular, for policy $\pi_t$ what we will observe is typically a *trajectory* $\tau_t := (s_0, a_0, s_1, a_1, \ldots,)$ and corresponding reward sequence $\{r(s_0,a_0), r(s_1,a_1), \ldots$. As we saw in Section 23.3.1, the trajectory will be generated according to the probability model $\pi_t(a_0|s_0) \cdot P(s_1|s_0,a_0) \cdot \pi_t(a_1|s_1) \cdot P(s_2|s_1,a_1) \ldots$. It turns out that we can use this information to compute unbiased estimates of the policy gradient, as well as the Fisher information matrix that is used in the natural policy gradient update. All of these are roughly equivalent procedures, and can be analyzed through a stochastic optimization framework (recall the tools that we developed in Lecture 9!).

### 23.4.1 The best of both worlds: Actor-critic algorithms

We conclude this lecture by describing an interesting algorithm that has flavors of value-based and policy-based methods in it. While imbuing policy optimization methods like NPG with a stochastic optimization framework yields powerful direct-policy methods for RL,

1. You may wonder where the dependences on $|\mathcal{S}|, |\mathcal{A}|$ show up here: these arise because the NPG update needs to be made for every state $s \in \mathcal{S}$, and the complexity of implementing the soft policy iteration update for any state $s$ is also $\mathcal{O}(|\mathcal{A}|)$. This shows you that the computational complexity of optimization methods needs to be measured carefully, both in terms of number of iterations *and* complexity per iteration!

it does suffer from some drawbacks as well. For example, the stochastic NPG algorithm (used in RL) can be equivalently viewed as follows: we use the trajectory information $\tau_t$ (corresponding to stochastic policy $\pi_t$) to directly estimate the on-policy Q-function as $\widehat{Q}^{\pi_t}(s,a)$. Then, we plug that estimate into Equation (23.12). Note that (just like the estimate of the optimal Q-function $\widehat{Q}^*(s,a)$ that we introduced last lecture), this will be an unbiased but high-variance estimator, as we are only taking one sample. Therefore, just like we did in Q-learning, we may hope to reduce the variance of the estimate by *bootstrapping* it with the previous estimate. This would be a particularly good idea if $\pi_t \approx \pi_{t-1}$, i.e. in later stages of the process. This leads to an alternative estimator of the Q-function, that is given by

$$\widetilde{Q}^{\pi_t}(s,a) = (1-\alpha_t)\widetilde{Q}^{\pi_{t-1}}(s,a) + \alpha_t \widehat{Q}^{\pi_t}(s,a). \tag{23.13}$$

for some step-size $\alpha_t \in (0,1)$. Like in Q-learning, this estimate of the on-policy Q-function bootstraps the previous sample, and only partially uses the new unbiased-but-high-variance information. This new estimate leads to what is commonly called an *actor-critic* algorithm[2], described below.

**Algorithm 2** The actor-critic algorithm (in sychronous form)

**Input**: Step-size schedule $\{\alpha_t\}_{t\geq 0}$ (typically decreasing)
**Initialize** $\pi_0(s,a)$ arbitrarily for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.
**while** $t \geq 0$ **do**
  **Policy evaluation (critic):** $\widetilde{Q}^{\pi_t}(s,a) = (1-\alpha_t)\widetilde{Q}^{\pi_{t-1}}(s,a) + \alpha_t \widehat{Q}^{\pi_t}(s,a)$
  **Policy optimization (actor):**
  **for** all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$ **do**
    $\pi_{t+1}(s,a) = \pi_t(s,a) \cdot \dfrac{\exp\left(\frac{\eta \widetilde{Q}^{\pi_t}(s,a)}{(1-\gamma)}\right)}{\sum_{a' \neq a} \exp\left(\frac{\eta \widetilde{Q}^{\pi_t}(s,a')}{1-\gamma}\right)}.$
  **end for**
**end while**

In essence, the actor-critic method employs a type of variance reduction on value estimation in conjunction with policy optimization. Like Q-learning, it can also be employed with a significantly more flexible simulator model, e.g. in an online fashion, or only updating some states per iteration. Although the precise benefits and shortcomings of actor-critic algorithms are still being investigated, this "best-of-both-worlds" property makes it very popular in practice. Asymptotic guarantees for actor-critic methods do exist, and they have their roots in ordinary differential equations theory. Recent research efforts have been devoted to make these guarantees more finite-sample in nature.

## 23.5. Additional references and notes

- The proof that we provided for strict policy improvement does not imply a fast convergence rate for policy iteration. However, Puterman and Shin (1978) showed that policy iteration converges *at least* as fast as value iteration, implying that the fast

2. This is one such variant: there are several in the literature.

convergence guarantees for value iteration directly carry over. Moreover, Ye (2011) showed that policy iteration can be written as a "faster" version of an update of the simplex algorithm on the well-known linear programming formulation of the MDP (unfortunately, we did not have time to discuss this LP formulation in detail in the class!). This equivalence and connection to linear programming implies improved computation times *that do not depend* on the number of bits in the MDP; this is called a notion of "strongly polynomial time" that is popular in the algorithms literature. In practice, policy iteration is observed to work extremely well and converge fast, and is a very popular method for solving large-scale discounted MDPs (much like the simplex method remains the gold standard in practice for solving LPs, despite there being algorithms with improved worst-case run times).

- See Chapters 11 and 12 of the RL theory book draft: `https://rltheorybook.github.io/rltheorybook_AJKS.pdf` for further details on PG and NPG. While the ideas in PG (e.g. REINFORCE) go back to the 1990's, PG is barely used in practice today owing to the aforementioned scaling issues. On the other hand, NPG arises in some form or another in several popular optimization heuristics used in RL. One popular example is trust region policy optimization (TRPO) Schulman et al. (2015), which turns out to be equivalent to NPG in finite-state-finite-action RL.

- A big driver for the popularity of policy-based methods is their flexibility in incorporating function approximation (but the debate between value and policy-based methods remains wide open even in practice). We will touch upon this next lecture.

## References

Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

Yinyu Ye. 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, 2011.