Parallel Curricula on RL and Control: Opening Lecture

David and Will

January 30, 2020

1 Logistics:

- Course websites: know where they are
  - http://mathsdl-spring20.willwhitney.com
- The course has one main assignment
  - Write a review paper in a group on an area you choose
  - Abstract (proposal) due in March
  - Paper due at the end of April
- Grading is mostly the paper, partly participation
- Explain idea of the parallel curriculum
  - This is the only time we'll be lecturing
  - Most of the parallel curriculum is focused on mandatory readings; in class we'll go further
- Section usually Fridays at 11, but no session tomorrow

2 The Problem: sequential decision making under uncertainty

decision making

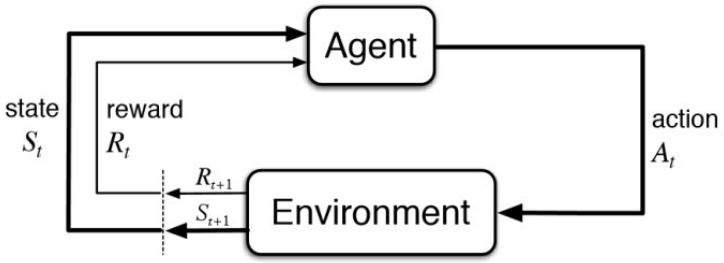
- Not just making predictions or estimates
- You only see the outcomes of the chosen actions
- Actions affect the world

sequential

- Actions are made in sequence so that past decisions affect future options and states of the world

under uncertainty

- The way that the world responds to actions is unknown and stochastic



3 Examples

Atari, manipulation (rubik's cube), medical treatments, portfolio optimization, gridworld (our example)

4 Formalizing the problem: the Markov decision process (MDP)

- states $s \in \mathcal{S}$
- actions $a \in \mathcal{A}$
- rewards $r \in \mathbb{R}$ (although often assumed to be positive and bounded)
- transition dynamics $s' \sim p(\cdot|s, a)$
- initial state distribution $s_0 \sim p_0(\cdot)$
- policy $a \sim \pi(\cdot|s)$

4.1 The episodic setting

Typically we divide time up into *episodes*. At the beginning of each episode, the environment is reset according to $s_0 \sim p_0(\cdot)$. The agent interacts with the environment for a fixed number of steps or until some condition is reached, e.g. the agent loses the game.

4.2 Objectives

Our goal is to maximize the *expected reward* obtained by our policy. However, there are a couple of variations in how we might treat time.

The **finite horizon** objective:

$$J^H(\pi) = \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0 \\ s_{t+1} \sim p(\cdot|s_t, a_t) \\ a_t \sim \pi(\cdot|s_t)}} \sum_{t=0}^{H} r(s_t, a_t) \tag{1}$$

where H is often the length of an episode.

The **discounted** objective, for $\gamma \in [0, 1)$:

$$J^\gamma(\pi) = \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0 \\ s_{t+1} \sim p(\cdot|s_t, a_t) \\ a_t \sim \pi(\cdot|s_t)}} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{2}$$

One or the other of these variants is required because the un-discounted infinite-time problem doesn't work out mathematically. (Almost every policy tends to get $\pm\infty$ reward, and infinitesimal changes in policy yield arbitrary changes in reward.)

Question: Does anything *not* fit into the framework?

4.3 The Markov property

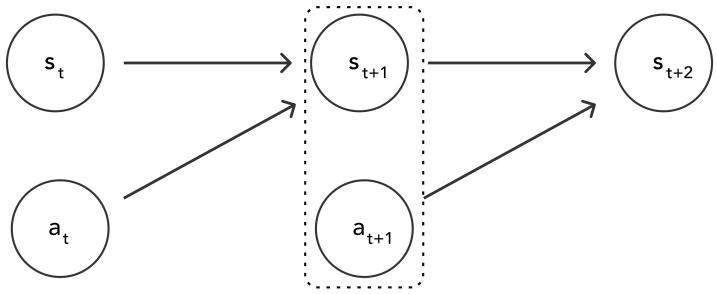
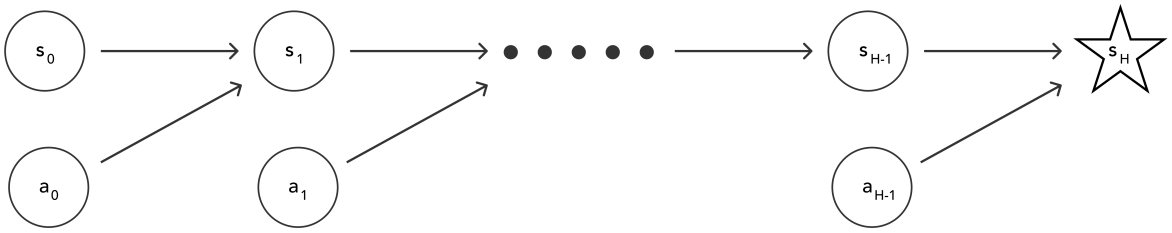


Figure 1: $s_{t+2} \perp\!\!\!\perp s_t, a_t, s_{t-1}, a_{t-1}, \ldots \mid s_{t+1}$

The environment is memoryless, and the optimal policy is a function of the current state only.

5 Why is this hard?

5.1 Credit assignment



You took many actions to get to s_H. Which ones were important?

5.2 Exploration: finding good trajectories

- there are exponentially many action sequences $a_1, \ldots, a_{H-1}$, can't try them all
- taking random actions often doesn't get you far

5.3 Exploration vs exploitation

- In some settings you care about total *lifetime* rewards
  - medicine
  - stock trading
  - . . . really any time you're learning with live fire
- Have to balance getting a reward now with learning something about the environment (and maybe finding a big reward)

From here we're going to move on to methods, so it's important that all of this makes sense. **What questions do you have?**

(wait for questions)

Now we're going to switch to talking about ways to solve the RL problem. Confusingly, people sometimes call both the *problem* and the *methods* "RL" ("we can use RL to solve X").

6 High level solution method 1: policy search

Remember: our goal is to find a stochastic policy π that maximizes our rewards. Policy search tries to directly find a good policy.

The main idea of policy search:

- define a *parametric* policy class, $\pi_\theta(a \mid s)$
- find the setting of θ that maximizes reward:

$$\theta^* = \arg\max_\theta J^{(H \text{ or } \gamma)}(\pi_\theta) \tag{3}$$

Different algorithms either choose the policy class or the search method differently.

6.1 Ways to represent policies

OK, so we want to choose a class of functions mapping from states to distributions over actions. **What are some options?**

(wait for audience suggestions)

6.1.1 Tabular policies

The simplest option is a tabular policy. We create a parameter $\theta_{sa} \geq 0$ for each (s, a) pair, then our policy is

$$\pi_\theta(a \mid s) = \frac{\theta_{sa}}{\sum_{a' \in \mathcal{A}} \theta_{sa'}} \tag{4}$$

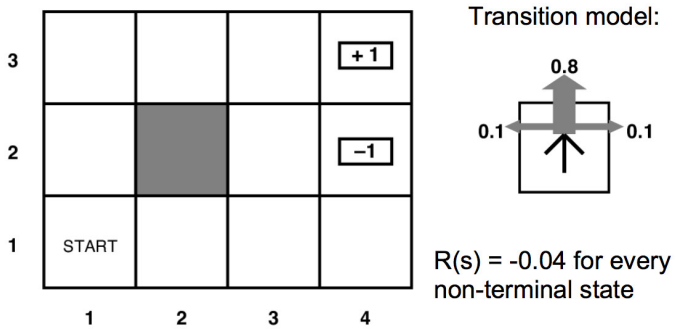


Figure 2: From https://courses.grainger.illinois.edu/cs440/fa2018/lectures/lect29.html

For something like this gridworld, that works great! However for more complex environments we have a problem: we have $|\mathcal{S}| \times |\mathcal{A}|$ parameters. Right now we have $N = \{\text{number of squares}\}$ states. But if the goal was in a random place each episode, suddenly we have N^2 states. Now imagine we want to play Pac-Man, and we have ghosts that move around and every square can have a pip in it or not. Scaling with $|\mathcal{S}|$ is *very* bad.

6.1.2 Function approximators

For larger spaces we use function approximators. We might have a linear function, like $\pi_\theta(a \mid s) \propto \theta^T s$. Or we could use a neural network.

6.2 Ways to search for policies

The dumbest option first: BogoSearch (à la BogoSort):

1. randomly pick a set of parameters,
2. evaluate it,
3. see if it's better than your best so far.

Clearly this is a bad algorithm. However it's useful to realize that you can search for good parameters however you like, and that every policy search method is black box; we only have access to samples of the objective.

A smarter option is to use gradients and ascend $\nabla_\theta J$.

6.2.1 Score function (REINFORCE) gradients

REINFORCE is one of the fundamental methods of RL, and is also known in statistics as the *score function* gradient. The goal is to approximate $\nabla_\theta J$; however, as I mentioned above, we only have access to samples of J, not analytic gradients. The idea is to use algebra and calculus to write down $\nabla_\theta J$ as an expectation which we can approximate with samples.

$$J^H(\pi_\theta) = \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0 \\ s_{t+1} \sim p(\cdot|s_t, a_t) \\ a_t \sim \pi_\theta(\cdot|s_t)}} \sum_{t=0}^{H} r(s_t, a_t) \tag{5}$$

$$= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [r(\tau)] \qquad \text{overloading } r \tag{6}$$

$$= \int_\tau p_\theta(\tau) r(\tau) d\tau \tag{7}$$

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \int_\tau p_\theta(\tau) r(\tau) d\tau \tag{8}$$

$$= \int_\tau \nabla_\theta p_\theta(\tau) r(\tau) d\tau \qquad \text{under technical conditions} \tag{9}$$

$$= \int_\tau \frac{p_\theta(\tau)}{p_\theta(\tau)} \nabla_\theta p_\theta(\tau) r(\tau) d\tau \tag{10}$$

$$= \int_\tau p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} r(\tau) d\tau \tag{11}$$

$$= \int_\tau p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \qquad \text{due to trick below} \tag{12}$$

$$= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \nabla_\theta \log p_\theta(\tau) r(\tau) \tag{13}$$

Here we're able to write Eq. 12 due to the *likelihood ratio trick*, which is just the derivative of log plus the chain rule, used backwards:

$$\frac{d}{dx} \log x = \frac{1}{x} \tag{14}$$

$$\frac{d}{dx} \log f(x) = \frac{1}{f(x)} f'(x) \tag{15}$$

In deriving REINFORCE, we've simply used this rule backwards. See Sergey's slides for more detail of this derivation.

In order to reduce the variance of these gradients we can introduce a baseline:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \nabla_\theta \log p_\theta(\tau) (r(\tau) - b) \tag{16}$$

where b doesn't depend on the parameters. This works because the expectation of the score function is zero. Lots more to say on this, but we'll skip it for now.

What questions do you have?

take a break

7 High level solution method 2: value estimation

We will use discounted setting throughout this section.

For now we will assume finite, tabular representations. We will show how to move these ideas to parametric function approximators in a few weeks.

7.1 Value estimation

It is useful to estimate the expected value of a given policy.

7.1.1 The value function

The value function for a policy π is the expected future value from following π starting at state s:

$$V^\pi(s) := \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s\right] \tag{17}$$

7.1.2 The Bellman equation

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi|s}[r(s,a)] + \gamma \mathop{\mathbb{E}}_{\substack{a\sim\pi|s \\ s'|s,a}}[V^\pi(s')] \tag{18}$$

7.1.3 Dynamic programming for value estimation

Assume that we have access to the transition dynamics and reward function, but we want to find the value function.

The idea is to find the value function that satisfies the Bellman equation. You could do this just by solving the system of $|\mathcal{S}|$ linear equations.

We could also define a Bellman operator:

$$\mathcal{T}^\pi V(s) := \mathop{\mathbb{E}}_{a\sim\pi|s}[r(s,a)] + \gamma \mathop{\mathbb{E}}_{\substack{a\sim\pi|s \\ s'|s,a}}[V(s')] \tag{19}$$

Then we can define an algorithm by initializing $V_0(s) = 0$ for all s and then setting

$$V_{t+1} = \mathcal{T}^\pi V_t \tag{20}$$

which can be implemented by looping over all s and setting

$$V_{t+1}(s) = \sum_a \pi(a|s)\left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_t(s')\right) \tag{21}$$

Convergence: We can prove convergence from the Banach fixed point theorem by showing that $\mathcal{T}^\pi$ is a contraction in ∞ norm.

$$|\mathcal{T}^\pi V(s) - \mathcal{T}^\pi V'(s)| = \left| \mathop{\mathbb{E}}_{a\sim\pi|s}[r(s,a)] + \gamma \mathop{\mathbb{E}}_{\substack{a\sim\pi|s \\ s'|s,a}}[V(s')] - \mathop{\mathbb{E}}_{a\sim\pi|s}[r(s,a)] - \gamma \mathop{\mathbb{E}}_{\substack{a\sim\pi|s \\ s'|s,a}}[V'(s')] \right| \tag{22}$$

$$= \gamma \left| \mathop{\mathbb{E}}_{\substack{a\sim\pi|s \\ s'|s,a}}[V(s') - V'(s')] \right| \tag{23}$$

$$\leq \gamma \max_{s'} |V(s') - V'(s')| = \gamma \|V - V'\|_\infty \tag{24}$$

7.1.4 Making it stochastic: TD learning

We can also just follow the policy repeatedly and make updates on any observed $s, a, r(s,a), s'$ transition by:

$$V_{t+1}(s) = (1-\alpha_t)V_t(s) + \alpha_t(r(s,a) + \gamma V_t(s')) \tag{25}$$

$$= V_t(s) + \alpha_t(r(s,a) + \gamma V_t(s') - V_t(s)) \tag{26}$$

This is a stochastic approximation to the full dynamic programming algorithm. So under proper conditions on the learning rates and the way we sample transitions from the MDP, we are guaranteed to converge to the correct value function.

7.1.5 Policy iteration

Once we have the value function for a given policy we may want to improve the policy. An easy way to do this is to just select a policy to maximize the expected value at the next state:

$$\pi'(a|s) = \arg\max_a \mathbb{E}[r(s,a)] + \gamma \mathop{\mathbb{E}}_{s'|s,a}[V^\pi(s')] \tag{27}$$

This guarantees to improve the policy, so that π' is better than π, since it can be thought of as changing the first action at s and then following π and having higher expected value.

Since there are a finite number of policies, this monotone algorithm is guaranteed to find the optimal one.

Stop here for visualization

7.2 Learning a policy with value functions (control)

Instead of learning a policy by iterating value estimation and policy improvement, we can actually directly iterate on policies implicitly defined by value functions. It is easier to see if we define the action-value or Q function.

7.2.1 The Q function

$$Q^{\pi}(s,a) := \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\middle| s_0 = s, a_0 = a\right] \tag{28}$$

We will denote the optimal policy π^* and the optimal value and Q functions V^*, Q^*.

7.2.2 The Bellman optimality equation

The key insight for this algorithm is to replace the Bellman equation by the Bellman optimality equation.

$$Q^*(s,a) = \mathbb{E}[r(s,a)] + \gamma \mathop{\mathbb{E}}_{s'|s,a}[\max_{a'} Q^*(s',a')] \tag{29}$$

7.2.3 Q value iteration

We can also define a Bellman optimality operator:

$$\mathcal{T}^*Q(s,a) := \mathbb{E}[r(s,a)] + \gamma \mathop{\mathbb{E}}_{s'|s,a}[\max_{a'} Q(s',a')] \tag{30}$$

Then we can define an algorithm by initializing $Q_0(s,a) = 0$ for all s, a and then setting

$$Q_{t+1} = \mathcal{T}^{\pi}Q_t \tag{31}$$

This converges by a similar fixed point argument (left as an exercise)

7.2.4 Making it stochastic: Q-learning

We can also make a stochastic approximation of this algorithm as before:

$$Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t(r(s,a) + \gamma \max_{a'} Q_t(s',a')) \tag{32}$$

$$= Q_t(s,a) + \alpha_t(r(s,a) + \gamma \max_{a'} Q_t(s',a') - Q_t(s,a)) \tag{33}$$

and again convergence follows under appropriate conditions on learning rates and sampling procedure.

Stop here for visualization

8 Next steps:

- We will post links to RL background material on the website
- Reading assignment for next week will also be on the website. COME PREPARED! We will not be lecturing again.