

# 1 Administrivia

- Final project presentations: for grad students, presentations will be May 1 (next Thurs!) and May 6. There will be 2 pres on May 1, and 3 on May 6. Draw lots for the lucky people to go on May 1.
- If you go on May 1, I don't expect you to be able to present fully complete work, but I do expect you to be able to show that you're making good progress/are on target to complete.
- Presentations will be 20 min long. You can use whatever materials you want. I've reserved a projector, but you'll have to provide your own laptop.
- Goal of your presentation: tell us what the problem you worked on was, why it was interesting/hard, a little of what other people did and how you extended it, and what your results were. Tell us whether your results were as good as what you hoped for or not. What shortcomings did your work have? How would you extend it in the future?
- You will be graded on both **content** and **presentation style**. Practice, practice, practice!
- Undergrads: final playoffs on May 8. You must be prepared to explain what you did in your program!
- Does anyone know whether we can get wireless access in this room? Can someone volunteer to bring a laptop?

# 2 Reinforcement Learning

- We've now seen a bit about how to represent a decision problem as a Markov Decision Process (MDP).
- We showed how we can represent uncertainty about *outcome* using an MDP and describe how an agent can act *optimally* with respect to a fixed world (policy  $\pi$ ).
- Using algorithms like policy iteration, we know how to do *planning* – how to select optimal actions in a *known* environment. I.e., you know what all of the components of  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$  are, but you don't know what  $\pi$  you like.
- Today, we'll talk about the harder problem of **reinforcement learning** – how to *learn* how to act when you *don't* know all the components that describe your MDP.
- Usually, we assume that you know what states are in the world, and what actions are available to you ( $\mathcal{S}, \mathcal{A}$ ), but you don't know the dynamics of those actions ( $T$ ) or the rewards associated with them ( $R$ ).
- Goal is to start with only  $\mathcal{S}$  and  $\mathcal{A}$ , no knowledge of  $T$  and  $R$ , and to learn to act optimally.
- Have to interact with world, try different things, to get it all over time.
- May make mistakes!

- Note that this only makes sense when you intend to operate multiple times in the world. If you're in it for one shot, then you're kinda hosed (if you have no prior knowledge).
- Fundamental problem: how do you know which actions of yours contributed to superior (or poor) performance? Called the **credit assignment** problem.
- Example: playing checkers
- Example: robomouse
- Two potential things you could do:

**Model-based learning** Find explicit model of  $\mathcal{M}$  (i.e., find all parameters) and then use planning algorithm (e.g., policy iteration) to find  $\pi^*$ . Note that the only parts that are at all difficult to find are  $T$  and  $R$

**Model-free learning** Figure out how to act well directly, without ever finding  $T$  and  $R$ . I.e., find an estimate of  $\pi^*$ ,  $\tilde{\pi}^*$ , directly.

## 2.1 Model based approach

- Simplest thing you can do:
- Note that each state/action pair is just a multinomial distribution: you just need to estimate the params by which it transitions to every other state.
- E.g., need to estimate  $\Pr[s_{t+1} = q_i | s_t = q_j, a_k]$  for every  $i, j$ , and  $k$ .
- So, for example, if you could start at  $q_{37}$  a bunch of times, you could take action  $a_3$  a bunch of times and see where you end up. From this, estimate  $T(q_{37}, a_3, q_j)$ . (Q: what number do you fill in here?)
- Thus, if you can start at every state an infinite number of times, you can just take every action there enough to estimate  $T$  directly.
- You get  $R$  just by observing each state (if deterministic  $R$ ).
- Problem 1: this may sting a lot. You may spend a bunch of experience learning about things you don't care about (e.g., low reward states).
- Problem 2: You don't always have the opportunity to restart from an arbitrary state whenever you want to.
- Tradeoff: **exploration vs. exploitation**
- Example: getting lunch on Central Ave.
- Need some way to trade off between explore and exploit
- One way is: initialize all states to  $R_{\max}$  and all transitions to uniform.

- Now find  $\pi^*$  for this (highly optimistic) model. (Policy iteration)
- Act according to  $\pi^*$  until you accumulate enough experience to change some value ( $R(s)$  or  $T(s, a)$  for any  $s, a$ ).
- Update model, solve for *new*  $\pi^*$ , and continue.
- This is the  $E^3$  algorithm – efficient explore and exploit
- Uses principle of **optimism under uncertainty**
- Can show that this is, in some sense, optimal. You're always either optimally exploring ('cause you want to go get reward from places you haven't been yet) or exploiting ('cause you can take advantage of things that you're close to).

## 2.2 Model Free Learning

- What if you want to skip that whole tedious model building phase?
- Let's go back to what good action really is, in terms of value:

$$V^*(s) = R(s) + \gamma \sum_{s'} T(s, \pi^*(s), s') V^*(s')$$

- Last time, we used the  $Q$  function to find  $\pi^*$ :

$$Q^\pi(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^\pi(s, a')$$

- Problem is that we don't know the distribution  $T$ : all we have is one experience at a time.
- Start by initializing  $Q(s, a)$  (almost) arbitrarily.
- If you have a current estimate of  $Q(s, a)$ , and you're at  $s$  and take action  $a$ , then you'll observe that you end up somewhere else,  $s'$ .
- But you have some estimate of  $V(s') = \max_{a'} \{Q(s', a')\}$  already.
- So you can treat  $R(s) + \gamma V(s')$  as a *sample* of the true  $R(s) + \gamma \sum_{s'} T(s, a, s') V(s')$ .
- This is just a sample, and it's based on a mythical  $Q(s', a')$ , but it is closer to the true value than you were, b/c it incorporates the real  $R(s)$  and a sample of the actual  $T$ .
- Still, it would be overshooting to completely replace your current  $Q(s, a)$  with this new estimate (your current  $Q$  may reflect real, valuable experience). You basically want to average in your new value into your old one.
- This is like our idea for updating weights on eval function in the game playing context from a couple of lectures back.

- This leads us to the **Q-learning** algorithm, by Watkins:

1. Initialize  $Q(s, a)$  (nearly) arbitrarily
2. Pick initial state  $s$
3. While (!converged)
  - (a) Get reward  $R(s)$
  - (b) Choose  $a = \arg \max_{a'} Q(s, a')$
  - (c) Take action  $a$  and move to  $s'$
  - (d) Update  $Q(s, a)$ :

$$Q(s, a) := Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

- (e) New state is  $s'$
  - (f) Decay  $\alpha$
- Can show that this converges to true  $Q^*$  (and, therefore, true  $\pi^*$ ) eventually. (May take a long time...) Also, have to guarantee that you try *every*  $s, a$  pair infinitely often.
  - (May need  $\epsilon$  exploration to do that...)
  - Note that this only updates  $Q(s, a)$  for one state at a time, but that state is “in the future” for some other state (possibly every other state)
  - May also want to update other states based on this experience. But want to update states that are “close” to  $s$  more than ones that are “far” from it.
  - Define an “eligibility” for each state,  $e(s)$  for every  $s$ . Multiply  $e(s)$  into the update above
  - Think of radioactive breadcrumbs....
  - The breadcrumb decay factor is  $\lambda$  and this is a “TD( $\lambda$ )” method