Lecture 11: Actor Critic Methods

1 Prologue

238,900 miles from Earth.

- 00:08 [Mission Control] Altitude 5200 feet.
- 00:10 [Armstrong] Manual Attitude Control is good. *That meant all the calculations for how much fuel would be used on a computer-controlled descent were now moot.*
- 00:18 [Mission Control] Houston. You are go for landing. Over
- 00:28 [Armstrong] Roger that. Go for landing, we have an alarm.
- 00:30 [Armstrong] 1201
- 00:32 [Mission Control] 1201, Roger 1201 alarm.
- 00:36 [Mission Control] We're go. Hang tight. We're go. *The 1201 alarm meant the main navigational computer for landing (the Primary Guidance, Navigation and Control System (PGNCS)was not working because it ran out of memory. That meant they were not able to calculate the difference in altitude measured by their radar and what was being calculated by Mission Control. The computers were designed to reboot when something like that happened. It did. During that time, Armstrong was flying blind. The computer came back on line and this time they reported a new error.*
- 00 [Armstrong] It's a 1202 ... What is that? Give us a reading on the 1202 Program Alarm ...
- 01:07 [Mission Control] Roger, 1202. We copy it. *This is not a better error. That means the computer is trying to do too many things at once and is postponing some of them.*
- 03:16 [Mission Control] Sixty seconds.
- 03:46 [Mission Control] Thirty seconds. *Seconds until fuel runs one.*
- 03:52 [Aldrin?] Contact light engaged.
- 03:55 [Aldrin?] Ok. Engine Stopped.
- 04:10 [Mission Control] We copy you down Eagle.
- 04:12 [Armstrong] Houston, uh [long pause] Tranquility Base here, the Eagle has landed.
- 04:18 [Mission Control] Roger Tranquility. We copy you on the ground. You got a lot of guys about to turn blue here. We're breathing again.

2 Logistics

- Midterm on Thur. 1 page (2 sided) cheat sheet. You'll scan your midterm afterwards, so bring a phone/tablet that you can use for scanning. Proposed midterm questions on the shared sheet are excellent. Many are very very similar to questions on the quiz.
- We're having an additional office hours today 2pm - 4pm in CS 301. Precepts for Thur/ Fri will be canceled.
- Brief break in the middle of today's class
- Check out the polls on Ed about (1) due date for future HWs and (2) final project preferences.

3 Review

- RL objective: $\mathbb{E}_\pi \left[\sum_t \gamma^t r(s_t, a_t)\right]$. Bandits are a special case where $\gamma = 0$. Recall that the value function and Q-function are very similar; they estimate the expected discounted reward conditioned on an initial state and action.
- The goal in RL is to produce the policy, $\pi(a \mid s)$. At the end of the day, it's the policy that we're going to return.
- Link: Notice that these two things are different. There's the objective, which we aim to estimate and optimize. And there's the policy, which is the thing that we're actually optimizing. We've had separate lectures focusing on these two separate components.
  - Policy: imitation learning, policy gradient
  - value functions, value iteration

  Today's lecture will show that these two things are two sides of the same coin.

4 Where are we going?

After talking about value functions, we've focused on two ideas: policies and value functions. This lecture will bring together these two threads. It neatly ties together much of the material that we've seen in the first half of this course. The new material in today's lecture won't be covered on the exam, but much of it will rehash things that we've covered in previous lectures, material that will be covered on Thursday's midterm.

At this point, you've seen most of the building blocks for RL, the key ideas. The second half of the course will talk about how you can ensemble these pieces to build practical and powerful RL algorithms. Considerations will include discussions of the bias/variance tradeoff, exploration, and how to use models. The second half of the course will also ease off on the homeworks and introduce the final project. The final project will be fairly open ended, will let you work with classmates to implement practical RL algorithms.

5 (Generalized) Policy Iteration

Two sides of the same coin:

- policy evaluation: given π, find Q^π
- policy improvement: given Q^π, find π' s.t. $Q^{\pi'} > Q^\pi$ for all states and actions.

Combined, these steps give us a way of getting better and better policies. We can simply alternate between evaluating the policy and estimating the policy.[1]

5.1 How to Evaluate a Policy?

Given a policy $\pi(a \mid s)$, how do we get $Q^\pi(s, a)$? We assume that you have a dataset of $\{(s, a, r, s')\}$. We can use the Bellman *expectation* equation to estimate Q^π:

$$Q(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{\pi(a'|s')}[Q(s', a')]. \tag{1}$$

Note that the expectation is just taken w.r.t. the policy, not the dynamics, so it's feasible to do this in MDPs where the dynamics are unknown. This same update can also be written as a loss function:

$$Q_{\text{new}} \leftarrow \arg\min_Q \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}, a'\sim\pi(a'|s')} \left[(Q(s, a) - (r + \gamma Q_{\text{old}}(s', a')))^2\right]. \tag{2}$$

[1] Convergence of policy iteration: https://rltheory.github.io/lecture-notes/planning-in-mdps/lec4/

It is tempting to use the *observed* next action a' instead of sampling it anew from the policy you want to evaluate. This would require data (s, a, r, s, a), and is hence known as SARSA:

$$Q(s,a) \leftarrow r(s,a) + \gamma Q(s',a'). \tag{3}$$

What does SARSA end up converging to? When will SARSA perform policy evaluation?

5.2 How to Improve a Policy?

If you have Q^π, how do you get a policy that's better than π? Simply by acting greedily w.r.t. Q^π!

$$\pi'(a \mid s) = \mathbb{1}(a = \arg\max_a Q^\pi(s,a)). \tag{4}$$

Claim: If π is not already optimal, the π' is better than π. More precisely, we have $Q^{\pi'}(s,a) \geq Q^\pi(s,a)$ for all states and actions. If and only if $\pi = \pi^*$ will this hold with equality at all states and actions. Said in other words, if you can improve the Q value at any state (i.e., the current policy doesn't select the argmax), then you get a new policy with a higher expected return.

5.3 Actor Critic Methods – What's in the name?

Methods that implement generalized policy iteration are often called *actor critic* methods. *Actor* is another name for the policy – it's the thing that chooses the *act*ions. *Critic* is another name for the Q-function, because it's the thing that critiques the actions, telling you which are better than the others.

In practice, we'll alternative between updating the policy and updating the value function. This is a general recipe, and there are lots of examples of specific algorithms that do this:

- Initialize Q, π
- Policy evaluation (update Q)
- Policy improvement (update π)
- (Optional) Collect data.
- Go to step 2.

An off-policy algorithm Recall that we can use off-policy methods to do policy evaluation. Thus, thus, this actor-critic algorithm can be implemented as an off-policy algorithm. Note that, in contrast, REINFORCE is an on-policy algorithm because the Q function is estimated in an on-policy manner.

6 Connections with methods we've seen before

One of the beautiful things about this result is that there are connections with many of the things we've already covered in this course.

- Q: How does this relate to Q-learning?
- Q: How does this relate to the policy gradient?

Q-learning. This is just policy iteration where the improvement step is done analytically:

$$\pi(a \mid s) = \arg\max_a Q(s,a). \tag{5}$$

What happens when we evaluate this policy?

$$\mathbb{E}_{\pi(a|s)}[Q(s,a)] = \max_a Q(s,a). \tag{6}$$

Note that this looks just like Q-learning! For example, consider what happens if we do policy *evaluation* using the updated policy:

$$Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{\pi(a|s)}[Q(s',a')] = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)}[\max_{a'} Q(s',a')]. \tag{7}$$

Policy gradient. Why do we need policy iteration? It helps for settings where we have a large number (or infinite number) of actions, so we can't compute the maximizing action exactly. How do we update the policy in this case?

One way to think about this is through the lens of *amortized optimization*. We'd like the train the policy so that it outputs Q-maximizing actions. The objective for doing this could look something like this:

$$\mathbb{E}_{\pi(a|s)}[Q(s,a)]. \tag{8}$$

Let's consider taking the gradient of this w.r.t. π. Again, recall the log derivative trick: $\nabla_\theta \log \pi_\theta(a \mid s) = \frac{1}{\pi_\theta(a|s)} \nabla_\theta \pi_\theta(a \mid s)$

$$\nabla_\theta = \int \nabla_\theta \pi(a \mid s) Q(s,a) da \tag{9}$$

$$= \int \pi(a \mid s) \nabla_\theta \log \pi(a \mid s) Q(s,a) da \tag{10}$$

$$= \mathbb{E}_{\pi(a|s)} \left[Q(s,a) \nabla_\theta \log \pi(a \mid s) \right]. \tag{11}$$

Q: Where have you seen this before? This looks like the policy gradient!

Differences from REINFORCE. There are two differences with the policy gradient that we've seen before:

1. When deriving the policy gradient, we had to apply the product rule, giving us an expression like $\nabla_\pi Q^\pi + \pi \nabla_\pi Q^\pi$. But here, we see that we can ignore that second term.
2. In the policy gradient, we had to take an expectation over the states and actions visited by π. But here, we see that we only need the distribution over actions to be correct. This means that we can use previously-collected states to perform the actor update. This means taht we can share data across actor updates, improving data efficiency.

Broadly, this policy iteration view allows us to lift some of the assumptions of the policy gradient derivation that we saw before.

6.1 Another perspective.

Seen in another way, again thinking about amortized optimization.

Let's start by reviewing the policy improvement step, this time considering a deterministic policy: $\pi : s \mapsto a$. The loss is then

$$\max_\theta \mathbb{E}_s[Q(s, a = \pi_\theta(s))]. \tag{12}$$

The gradient w.r.t. the policy parameters θ are

$$\nabla_\theta = \mathbb{E}_s \left[\nabla_a Q(s,a) \nabla_\theta \pi(s) \right]. \tag{13}$$

This is closely related to the *deterministic policy gradient*, but we'll ignore that for now. Note that we can't use the regular $\log \pi(a \mid s)$ trick for deterministic policies as the probability is either 0 or infinity for deterministic policies in continuous spaces.

Now, I want to consider an alternative approach to policy optimization, what you might do if you hadn't taken this course. In regular supervised learning, we have (x, y) pairs and we learn to map one to the other. For the purpose of policy improvement, we ideally would have examples $(s, a^* = \arg\max_a Q(s,a))$.

However, actually computing these optimal actions a^* can be quite expensive. One way to do this would be to run gradient descent on the Q-function. But, this is expensive; it requires many steps to converge. So, we could consider a heuristic – what if we just take one gradient update? That is, we pretend that $a^* = \pi(s) + \nabla_a Q(s,a)$. Then, let's do supervised learning on these data, using a MSE:

$$\min_\theta \tfrac{1}{2} \mathbb{E}_{(s,a^*)} \left[(\pi_\theta(s) - a^*)^2 \right]. \tag{14}$$

Let's take the gradient of this, substituting our approximation for a^*:

$$\nabla_\theta \tfrac{1}{2}\mathbb{E}_{(s,a^*)}\left[(\pi(s)-a^*)^2\right] = \mathbb{E}_{(s,a^*)}\left[(\pi(s)-a^*)\nabla_\theta\pi(s)\right] \tag{15}$$

$$= \mathbb{E}_s\left[(\cancel{\pi(s)} - (\cancel{\pi(s)} + \nabla_a Q(s,a)))\nabla_\theta\pi(s)\right] \tag{16}$$

$$= -\mathbb{E}_s\left[\nabla_a Q(s,a)\nabla_\theta\pi(s)\right] \tag{17}$$

Note that this is exactly the same as the deterministic policy gradient! The sign is different just because the objective in Eq. 12 is a maximization problem while Eq. 14 is a minimization problem.

6.2 Thinking in terms of code.

Neural networks:

- Actor (policy)
- Critic (Q-function)

Modern deep learning libraries are magical, making it very easy to compute all sorts of policy improvement losses.

Policy gradient. Let's start with the regular policy gradient. We'd like to compute the gradient $\mathbb{E}_\pi[Q(s,a)\nabla_\theta\pi(a \mid s)]$. But, it's often easier to think in terms of losses than in terms of gradients; doing so let's us feed the loss function to some optimizer (e.g., Adam, SGD, Adagrad) and let it do the optimization from there. So, how do we construct a loss function whose gradient is the policy gradient?

To do this, we can simply sample states and actions from the policy π, and then do Q-weighted behavioral cloning

```
s = env.reset()
loss = 0
while not done:
  a = policy(a).sample().detach()
  loss -= Q(s, a) * pi(s).log_prob(a)
  s, r, done, _ = env.step(a)
policy_gradient = torch.grad(loss, policy.parameters)
```

We have previously talked about the connections between behavioral cloning and the policy gradient. In particular, we can think about the policy gradient as doing behavioral cloning *on the policy itself*, except that each sample is weighted by the Q-function. Said in other words, the policy gradient is equivalent to imitating optimized data. We now see that these are not only useful mental models for understanding the mechanics of the the policy gradient, but they also directly correspond to how you'd implement these methods!

Policy improvement. In turns out we can go a step further, and write down this loss in an even simpler form by exploiting the fact that modern deep learning libraries do automatic differentiation.

```
def actor_loss(s):
  a = policy(s).sample()
  return -Q(s, a)
```

Mathematically, this loss is the same as the one above. Under the hood, the deep learning library will compute the correct gradients. However, this loss can actually work better in practice because of the reparametrization trick. Let's assume that $\pi(a \mid s)$ is a Gaussian. Sampling from that Gaussian is typically implemented as

```
def gaussian_sample(loc, scale):
  std = standard_normal.sample()
  return loc + std * scale
```

Note that this function is differentiable w.r.t. the location and scale. This is what's happening under the hood in many deep learning libraries.

7 Practical Actor-Critic Algorithms

There are many practical and widely-used algorithms that are instantiations of this general recipe:

- A2C
- A3C
- DDPG
- SVG(0)
- SAC
- NAF
- D4PG

These are good papers to refer to to learn more.

In the latter half of the course, we'll see that these algorithms differ in some design decisions that end up being important:

- Replay buffers – keeping around old and new data, doing both value and policy udpates on both. Note that this is easier than doing importance weighting.
- Min Q trick – avoid OOD actions

8 Short Break

9 Midterm Review

These are the most checked questions from the "what are you confused about" sheet last week.

There are so many variables. What do we have and what are we trying to find? Here are the two main problems that we're interested in:

1. Off-policy: The inputs are a dataset of trajectories $\tau = (s, a, r, s, a, r, \cdots)$ from a potentially suboptimal policy, and the output is a return maximizing policy $\pi^*(a \mid s)$. We have seen one algorithm for off-policy learning: Q-learning. You can think about value iteration as a "mathematicians" version of Q-learning where the dynamics are known; this isn't very practical, but it's useful for starting to wrap your head around what Q-learning does.
2. On-policy: The input is a simulator (i.e., something that supports the Gym API) and the output is a return maximizing policy $\pi^*(a \mid s)$. The two algorithms we've seen here are REINFORCE and actor critic methods; note that actor critic methods won't be covered on the midterm.

Relationship between policy gradient, Q-learning, TD learning, and value iteration? One main axis of variation is whether you learn a policy, a value function, or both. Policy gradient methods just learn a policy, whereas Q-learning and value iteration just learn a value function.

TD learning refers to a large group of methods (including Q-learning and value iteration) that have some term that looks like $Q' - (r + gammaQ)$. TD learning is part of an even larger class of methods known as approximate dynamic programming.

Value iteration refers to the tabular algorithm with known dynamics:

$$V(s) \leftarrow \max_a r(s,a) + \mathbb{E}_{p(s'|s,a)}[V(s')] \tag{18}$$

Q-learning is very very similar, but doesn't assume that the dynamics are known. As such, the expectation is replaced with a sampled next state. To make this work, you need to do the iteration on Q-values instead of values:

$$Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)}[\max_{a'} Q(s',a')]. \tag{19}$$

Because Q-learning doesn't assume that the dynamics are known, it is applicable to higher dimensional tasks. In these settings, we assume that we have a bunch of data $\{(s,a,r,s')\}$ and apply a loss that looks like the following:

$$\min_Q \mathbb{E}_{s,a,r,s'}\left[(Q(s,a) - y(r,s')^2\right] \quad \text{where } y(r,s') = r + \gamma \max_{a'} Q(s',a'). \tag{20}$$

What is the difference between value iteration, Monte Carlo, TD, and Q-learning? As noted above, TD is a broader term for methods that resemble dynamic programming. For the other methods, there are a few factors of variation:

- Are the dynamics known? Value iteration requires this
- Do you want to estimate Q^* or Q^π? Value iteration and Q-learning estimate Q^*, whereas SARSA and Monte Carlo methods estimate Q^π.

When to use policy iteration vs value iteration? Hopefully today's discussion helped clarify this. Namely, that value iteration can be seen as doing policy iteration, where the improvement step is done "implicitly" via the "max" introduced in the Bellman equation. If actions are discrete and if you want to estimate Q^* and if you know the model, do value iteration. If the actions are discrete and you want to estimate Q^* but don't know the model, do Q-learning. If actions are continuous, do generalized policy iteration (today's lecture).

What is the Bellman optimality equation? The underlying intuition is that the expected return starting at one state can be decomposed into two terms: the reward plus the expected return starting at the next state. There are a few variations of this identity, corresponding to whether you're talking about a particular policy $\pi(a \mid s)$ or the optimal policy $\pi^*(a \mid s)$:

$$Q(s,a) = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)}[\max_{a'} Q(s',a')] \qquad \text{(Bellman optimality equation)}$$

$$Q(s,a) = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)\pi(a'|s')}[Q(s',a')]. \qquad \text{(Bellman expectation equation)}$$

Derivation of the policy gradient? Check out this Ed post: `https://edstem.org/us/courses/54890/discussion/4499279?answer=10393995`

References