Lecture 18: Exploration

1 Introduction

Admin:

- Final projects – keep pushing ahead. Note that homeworks are lighter now so you can allocate time for the projects. Since some experiments require running experiments, starting earlier will give you more time to get the experiments done. Next deadline is to submit one slide of results (Apr 24).
- HW7 – Last homework of the semester, due on Friday. Involves implementing SAC (covered last week).

Learning objectives for today At the end of this lecture, you will be able to

1. Explain important desiderata for exploration algorithms.
2. Explain how key exploration methods work, and compare their strengths and weaknesses.
3. Be able to implement key exploration methods.

Plan for today

1. Exploration methods seen so far
2. Parameter space noise
3. Bootstrap DQN

1.1 Exploration

Exploration is one of the key capabilities of RL is its ability to uncover solutions/strategies that are better than those done by humans. We've seen a couple ways this can happen:

- reweighting. This is what REINFORCE does. It looks at all the trajectories, weights by their return, and then mimics. This means that higher-return strategies are "reinforced."
- stitching. This is what SARSA and Q-learning do. Then can piece together bits of previously-seen trajectories to create new solutions. This enables these algorithms to identify a solution that has never been demonstrated before. This is why we call these methods off-policy methods.

Both these forms of learning relate to the *data*: searching within the data or recombining it to find better solutions. But there's another avenue for finding new solutions: exploration. Exploration is the problem of thinking about what data to collect. This is as key capability of RL methods – the fact that they can alter their own data distribution.

Should efficient exploration be possible?

- Combination lock – how long do you expect it to take to solve a lock with 3 numbers? Can you do better than this? Does testing one code tell you anything about the likelihood of other codes?
- Chain environment – how long would an efficient method take to find a reward hidden at one state? How long would a random policy take? A random policy diffuses at $\mathcal{O}(\sqrt{T})$, so exploring S states would take S^2 time steps.

- Wordle, mastermind – how long do you expect it to take to find the correct code? In mastermind, we have 6 colors and 4 pegs, so the number of possible codes is $6^4 = 1,296$. Yet good players can always solve the game in 5 moves or fewer [3]. What makes this different from the combination lock? The fact that making one guess tells you something about other guesses. So, when making guesses, you want to make guesses that tell you things about lots of other states.
- Navigation – let's say that you want to find you keys in your dorm. Let's say your dorm room is a 10m x 5m x 2m cube, and that the keys are occupying a 1 cubic cm area. Then the number of positions for the keys is 1e3 x 5e2 x 2e2 = 1e8 (100 million). Yet you can find your keys much much faster because (1) you have priors on where you keys are, and (2) you can use observations from one state to rule out other states (e.g., you can very quickly rule out the vast majority of positions where the keys would be floating in mid-air).

Today's class will look at a couple different strategies for doing exploration. We're going to primarily focus on exploration methods for *deep* RL (i.e., with function approximation), though some of the techniques will also apply to the tabular setting.

2 Review: Exploration methods we've seen so far

Bandit methods:

1. Greedy: $\arg\max_a \hat{r}(a)$.
2. Epsilon greedy
3. Softmax: $\pi(a) = \textsc{SoftMax}(\hat{r}(a)) = \frac{e^{\frac{1}{\beta}\hat{r}(a)}}{\sum_{a' \in \mathcal{A}} e^{\frac{1}{\beta}\hat{r}(a')}}$.
4. UCB: $\arg\max_a \hat{r}(a) + \sqrt{\frac{2\log(t)}{N_a(t)}}$

RL methods:

- DQN: epsilon greedy
- MaxEnt RL: Softmax
- DDPG, TD3: add noise to the actions
- SVG: directly learn a stochastic policy.

3 Parameter Space Noise [2, 5]

Most of the deep RL exploration methods that we've seen can be interpreted as adding noise to the *actions*. However, this means that the exploration isn't time-correlated (i.e., wouldn't explore the line well). How can we get time-correlated exploration?

2 papers from 2017 that proposed almost the same idea: instead of adding noise to the actions, add noise to the policy parameters.

3.1 DDPG.

Simply add noise to the policy network when collecting data: $\pi_{\theta+\epsilon}(a \mid s)$.

- Should the noise be resampled at each time step?
- Is it OK that we're collecting data from a policy that is slightly different from the one we're learning?
- Would this policy perform well on the Chain task?
- At convergence, will the policy continue to explore?

3.2 DQN.

Act greedily w.r.t. noisy version of the Q network:

$$\arg\max_a Q_{\theta+\epsilon}(s,a). \tag{1}$$

3.3 Policy gradient methods

First attempt: Collect data using $\pi_{\theta+\epsilon}(a \mid s)$, and then compute the policy gradient $R(\tau)\nabla_\theta \log \pi_\theta(\tau)$.

Second attempt:

$$\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma)}[\mathbb{E}_{\pi_{\theta+\epsilon}(\tau)}[R(\tau)]]. \tag{2}$$

Compute the gradient:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma)}[\mathbb{E}_{\pi_{\theta+\epsilon}(\tau)}[R(\tau)]] &= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma)}[\nabla_\theta \mathbb{E}_{\pi_{\theta+\epsilon}(\tau)}[R(\tau)]] &(3)\\ &= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma)}[\mathbb{E}_{\pi_{\theta+\epsilon}(\tau)}[R(\tau)\nabla_\theta \log \pi_{\theta+\epsilon}(\tau)]]. &(4)\end{aligned}$$

Q: At convergence, will the policy continue to explore?

4 Bootstrap DQN [4]

One of the challenges with adding noise to the policy or Q-network parameters is that it's unclear how much noise to add: noise in parameter space doesn't necessarily correspond to meaningful noise in function space. A second challenge (at least for DDPG and DQN) is that the degree of exploration doesn't necessarily decrease throughout the learning process. A third challenge, related to the other two, is that the "drive" for exploration is somewhat heuristic, and not directly tied to *learning*.

Our next method will lift these limitations. The key idea is to capture uncertainty about what the future returns are, and to act in accordance with this uncertainty.

4.1 Thompson Sampling [6]

Thompson sampling is an exploration strategy usually targeted at bandit problems, but we'll see how it can be extended to full RL problems. The key idea is to sample actions with probability proportional to the likelihood that those actions are optimal.

In the RL context, when we only have observed data $\mathcal{D}$, we have uncertainty over what the true reward function and dynamics function are. Let $\mathcal{M}$ be a random variable denoting the MDP (i.e., the dynamics and reward function), and let $p(\mathcal{M} \mid \mathcal{D})$ denote our distribution over MDPs given the data observed so far. Given this uncertainty, which policy should we use? Intuitively, if a policy π is not optimal for any MDP $\mathcal{M}$, then we probably shouldn't use it. If a policy is optimal for many MDPs that have high likelihood, then that's a good policy to try. Formally, we might explore by sampling a policy with probability proportional to the likelihood that that policy is optimal:

$$p(\pi \mid \mathcal{D}) = \int \mathbb{1}(\mathbb{E}[R \mid \pi, \mathcal{M}] = \max_{\pi'} \mathbb{E}[R \mid \pi, \mathcal{M}]) p(\mathcal{M} \mid \mathcal{D}) d\mathcal{M}. \tag{5}$$

While this integral is very hard to estimate, sampling from the corresponding distribution is easy, because it corresponds to the pushforward measure of $p(\mathcal{M} \mid \mathcal{D})$ by the function $f : \mathcal{M} \mapsto \pi$. Thus we can sampling form this by first sampling $\mathcal{M} \sim p(\mathcal{M} \mid \mathcal{D})$, computing the optimal policy, and then doing an episode of exploration with that policy. After collecting this trajectory, we would store it in our dataset $\mathcal{D}$, and this would update our distribution $p(\mathcal{M} \mid \mathcal{D})$.

4.2 Bootstrap DQN

The Bootstrap DQN algorithm can be seen as a certain approximation to this algorithm. Rather than modeling uncertainty over the MDP itself, this algorithm models uncertainty over the Q values. That is, given that we have seen limited data, we have uncertainty over what the true Q function is.

Abstractly, this algorithm works as follows:

1. Sample $Q \sim p(Q \mid \mathcal{D})$
2. Collect an episode of experience by acting greedily w.r.t. Q: $\arg\max_a Q(s, a)$
3. Store that episode of experience in your replay buffer, and update $p(Q \mid \mathcal{D})$.

To make this algorithm more concrete, we need to think about how we are going to estimate $p(Q \mid \mathcal{D})$, a distribution over Q *functions*. The main challenge is that the space of Q function is enormous – even in the tabular setting, this corresponds to a distribution over matrices of size $|\mathcal{S}| \times |\mathcal{A}|$.

The key idea used in Bootstrap DQN is to represent this distribution as an ensemble of learned Q networks:

$$p(Q \mid \mathcal{D}) = \frac{1}{k} \sum_{i=1}^{k} \delta(Q = Q_i). \tag{6}$$

That is, we learn k Q networks, each with the same architecture but different weights. The most important difference is that these weights are randomly initialized. In theory, you're supposed to update each member of the ensemble on a different subset of the data; in practice this doesn't matter much.

Intuitively, you can see that these Q networks should converge to all be the same after you've collected enough data and done enough gradient updates. This is a desirable property – it means that the degree of exploration decreases over time, eventually going to zero.

Extension: learning to adapt to any model in the ensemble [1].

References

[1] Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. (2018). Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pages 617–629. PMLR.

[2] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017). Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

[3] Knuth, D. E. (1976). The computer as master mind. *Journal of Recreational Mathematics*, 9(1):1–6.

[4] Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29.

[5] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

[6] Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.