Lecture 22: Goal-Conditioned RL

Logistics:

- Presentations start on Tuesday. Please see the Ed post to see which order you'll be presenting in. Note that you must use the shared slide deck (talk to us if that won't work). Note that changes to the slide deck will be frozen starting about 15 minutes before lecture, so you won't be able to make changes during class.
- Final project is due on Dean's Date (May 7).
- All homeworks have been graded and returned.

Slides: `https://docs.google.com/presentation/d/1F7Te-r8ua8NjcM4IlhIkZFsqXa04_GKC-F7b0uC2OMU/edit#slide=id.g2cd33dae3b4_0_85`

1 Introduction

Key idea: think about RL and decision making in terms of probabilities.

This will let us develop RL methods that are easier to use, and that allow us to readily make use of ingredients that you've seen in an intro ML course. We'll start by thinking about estimating the probability of reaching states at some point in the future. We'll see how these probabilities can be used to determine the actions for reaching desired goal states. This will give us an effective method for goal-conditioned RL. At the end, we'll talk about some practical extensions of this idea.

2 Goal-Reaching

The main focus of today's lecture will be on learning to reach goals. This is a classic problem dating back to the early days of AI/ML [6, 7, 9]. Goal-reaching has many practical applications:

- Robot locomotion and manipulation [12]
- Autonomous driving
- Chemical synthesis [2, 13]
- Path planning, route optimization
- Games (e.g., build a castle in Minecraft [1, 8]).

Preliminaries. We'll consider a Markov decision process defined by states s_t and actions a_t. The initial state distribution is $p_0(s_0)$ and the dynamics are $p(s' \mid s, a)$, sometimes written as $p(s_{t+1} \mid s_t, a_t)$. The thing we learn is a policy $\pi(a \mid s)$ that takes as input the observation and outputs an action.

Think of this like a classifier, but one that classifiers the correct actions. The tricky thing is that we don't actually know what those correct actions are, so we're going to have to figure out a way of *estimating* which actions actually leed to good outcomes.

The policy interacts with the environment in a sequential manner. It takes as input the observation and it produces an action. That action is executed in the environment, and we get back the next observation. This loop repeats many times.

The overall objective is to learn a policy $\pi(a_t \mid s_t)$ that maximizes the expected cumulative discounted sum of rewards:

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{1}$$

Objective. We will look at the following objective for goal-reaching:

$$\max_{\pi}(1-\gamma)E_{\pi}\left[\sum_{t}\gamma^{t}\mathbb{1}(s_t=g)\right]. \tag{2}$$

The $(1-\gamma)$ constant factor doesn't affect the optimization problem. We include it to make our analysis easier later on. In continuous settings, you never get to a state exactly (this is a measure zero event), so instead we look at the probability that the *next* state is the goal:

$$\max_{\pi}(1-\gamma)E_{\pi}\left[\sum_{t}\gamma^{t}p(s_{t+1}=g\mid s_t,a_t)\right]. \tag{3}$$

Note that this objective is well defined in both continuous and discrete settings, and in both stochastic and deterministic settings. In continuous settings, we will talk about probability *densities*.

Intuition. Intuitively, we can understand this objective as saying that you get a +1 when you're at the goal and zero otherwise. We can think about this as maximizing the total time you spent in the goal. Like the standard RL problem, we'd prefer to get to the goal quickly, which is why we include the discount factor of γ.

Connection with RL. This objective looks like the standard RL problem with a reward function

$$r_g(s,a)=(1-\gamma)p(s'=g\mid s,a). \tag{4}$$

Pros. The nice thing about this problem formulation is that the rewards are defined directly in terms of the data. Ordinarily, RL problems are defined in terms of scalar reward functions. However, actually constructing these reward functions is really hard (see slide). Experts in RL (i.e., grad students) spend dozens of hours trying to figure out how to tune all the parameters in here. Designing the reward function requires knowledge of math and coding, which means that many potential users can't use these tools. Also, by defining this objective in terms of probabilities, it will have close connections with probabilistic modeling that you've seen in other areas of ML.

```
def compute_reward(observation):
    objPos = obs[3:6]
    (rightFinger, leftFinger) = (self._get_site_pos('rightEndEffector'),
                                 self._get_site_pos('leftEndEffector'))
    fingerCOM = (rightFinger + leftFinger) / 2
    heightTarget = self.heightTarget
    placingGoal = self._target_pos
    reachDist = np.linalg.norm(objPos - fingerCOM)
    placingDist = np.linalg.norm(objPos[:2] - placingGoal[:-1])
    def reachReward():
        reachRew = -reachDist
        reachDistxy = np.linalg.norm(objPos[:-1] - fingerCOM[:-1])
        zRew = np.linalg.norm(fingerCOM[-1] - self.init_fingerCOM[-1])
        if reachDistxy < 0.06:
            reachRew = -reachDist
        else:
            reachRew = -reachDistxy - zRew
        if reachDist < 0.05:
            reachRew = -reachDist + max(actions[-1], 0) / 50
        return (reachRew, reachDist)
    def pickCompletionCriteria():
        tolerance = 0.01
        if objPos[2] >= heightTarget - tolerance:
            return True
        else:
            return False
    if pickCompletionCriteria():
        self.pickCompleted = True
    def objDropped():
        return objPos[2] < self.objHeight + 0.005 and placingDist \
            > 0.02 and reachDist > 0.02
    def placeCompletionCriteria():
        if abs(objPos[0] - placingGoal[0]) < 0.05 and abs(objPos[1]
                - placingGoal[1]) < 0.05 and objPos[2] < self.objHeight  0.05:
            return True
        else:
            return False
// Yu et al. "Meta-world: A benchmark and evaluation for multi-task and meta
reinforcement learning." CoRL, 2020
```

Figure 1: Designing reward functions is challenging

Cons. Eq. 4 shows how we've reduced this problem of probability matching or goal-reaching into a problem of RL. However, the RL problem itself is very hard. And, applying it to this reward requires estimating this reward function, which would require fitting a density model to the dynamics, which is challenging in high-dimensional settings. So, we're going to need a way of estimating and optimizing the objective in Eq. 3 that doesn't require estimating densities over high-dimensional objects. Perhaps surprisingly, this is doable, and not too hard!

3 Contrastive RL [4]

The key idea will be to estimate *relative* likelihoods, as given to us by a classifier.

Discounted state occupancy measure. To start, let's return to the objective in Eq. 3. The reason for including the $1-\gamma$ term there was so that we can think about this as one big probability:

$$p^\pi(s_{t+}) \triangleq (1-\gamma) E_\pi \left[\sum_t \gamma^t p(s_{t+1} = g \mid s_t, a_t) \right]. \tag{5}$$

This is the probability of reaching state s_{t+} at some point in the future. It is often referred to as a the *discounted state occupancy measure* [10, 11]. Imagine that you close your eyes and open them after a random number of time steps. You want to know which state the agent will be at. This object encodes this probability, assuming that you open your eyes with a geometric distribution. Formally, this probability is referred to as the discounted state occupancy measure. We will sometimes think about the conditional version of this, $p(s_{t+} \mid s, a)$, which tells us the states we'll visit at some point starting at a particular state s and action a.

Now, if we could estimate this conditional occupancy measure, we could use it to directly figure out the right actions for reaching a goal:

$$\arg\max_a p^\pi(s_{t+} = g \mid s, a), \qquad \max_\pi \mathbb{E}_{\pi(a|s,g)}[p^\pi(s_{t+} = g \mid s, a)]. \tag{6}$$

However, fitting this density directly is challenging [5], so we're going to need another way.

Relative Likelihoods. The key idea behind contrastive RL is to estimate the *relative* discounted state occupancy measure:

$$\frac{p^\pi(s_{t+} \mid s, a)}{p(s_{t+})}. \tag{7}$$

Note that adding the term in the denominator doesn't change the dependence on actions. Namely,

$$\arg\max_a \frac{p^\pi(s_{t+} \mid s, a)}{p(s_{t+})} = \arg\max_a p^\pi(s_{t+} \mid s, a). \tag{8}$$

Estimating probability ratios is easier than estimating probabilities. Mathematically, the underlying reason is that we no longer need to estimate the partition function. How do you estimate probability ratios?

Classifiers! The problem of estimating probability ratios is a problem of classification: guessing whether an instance of a random variable came from one distribution or another. Let's see how this works in our setting, where we want to distinguish $p^\pi(s_{t+} \mid s, a)$ from $p(s_{t+})$.

We will learn the classifier using positive examples $p^\pi(s_{t+} \mid s, a)p(s, a)$ and negative examples $p(s_{t+})p(s, a)$. The classifier $C_\theta(s, a, s_{t+}) \in (0, 1)$ takes as input the current state and action and a future state, and guesses whether the future state is a real future state or a random future state. We will train this classifier using the standard binary cross entropy loss:

$$\max_\theta \mathbb{E}_{p^\pi(s_{t+}|s,a)p(s,a)}[\log C_\theta(s, a, s_{t+})] + \mathbb{E}_{p^\pi(s_{t+})p(s,a)}[\log(1 - C_\theta(s, a, s_{t+}))]. \tag{9}$$

What is the Bayes' optimal classifier? The optimal classifier satisfies:

$$\frac{C_\theta(s, a, s_{t+})}{1 - C_\theta(s, a, s_{t+})} = \frac{p^\pi(s_{t+} \mid s, a)}{p(s_{t+})}. \tag{10}$$

Thus, we can directly use this classifier to estimate the desired probability ratio.

3.1 Complete Algorithm

1. Start with data $\{(s_0, a_0, s_1, a_1, \cdots)\}$.
2. Fit the classifier $C_\theta(s, a, s_{t+})$ to these data.
3. Extract the policy $\pi(a \mid s, g) = \arg\max_a C_\theta(s, a, s_{t+} = g)$.
4. Optionally, collect some data and go back to step 2.

Does this work? [slide]

3.2 Geometric perspectives

Previously, we thought about the classifier $C_\theta(s, a, s_{t+})$ as a black box. We now look at a particular parametrization of this classifier, one that remains universal yet can improve efficiency and reveals surprising structure in learned representations.

We will use the following parametrization:

$$C_\theta(s, a, s_{t+}) = \sigma(-\|\phi(s, a) - \psi(s_{t+})\|_2^2), \tag{11}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. With this parametrization we have

$$\log \frac{C_\theta(s, a, s_{t+})}{1 - C_\theta(s, a, s_{t+})} = \|\phi(s, a) - \psi(s_{t+})\|_2^2. \tag{12}$$

Actor Loss. The actor loss[1] is equivalent to choosing the action that *moves* the representation $\phi(s, a)$ closest towards the goal representation [slide].

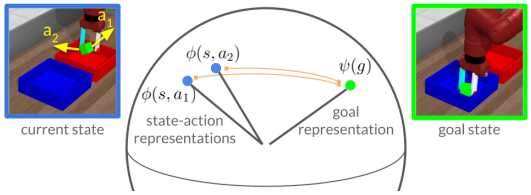

Figure 2: Action selection.

In control, we often talk about greedy planning: naïvely taking actions that are pointed at the goal. This strategy can be myopic, failing in settings where there are obstacles. Greedy planning in the original observation space is a bad idea. However, the results above show that greedy planning in the representation space is a great idea.

Critic objective. We can also use these representations to re-interpret the classifier objective. We can think about the objective as assigning representations to each state and state-action pair. The first term in Eq. 9 pulls together representations of state-action pairs with real future states. The second term pushes away representations of states that occur in different trajectories. Thus, the resulting representation space is one where states that occur nearby in time have similar representations. If it's difficult to navigate from one state to another, then these two states have distant representations.

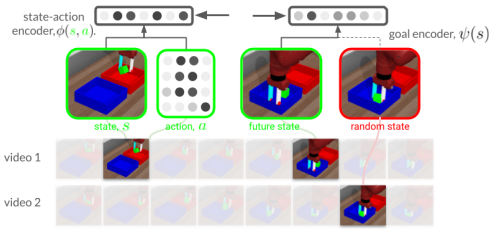

Figure 3: Critic objective.

We can also think about these representations as a sort of model of the world, one where $\phi(s, a)$ tells you the *representation* of a future state, rather than what the raw future state will look like.

[1] We've silently added an extra $\log(\cdot)$ here, but this doesn't change the optimum policy because the log function is monotone increasing.

Representation learning perspective. Thus, we can think about this algorithm in two ways. On the one hand, it's an algorithm for estimating probability ratios. These probability ratios represent the Q-function, so estimating them will tell us what the right actions are. One the other hand, it's a method for learning representations, for learning a (quasi) metric embedding. Prior methods had typically used different methods to solve these two pieces, using one method to acquire representations and another to find good policies with these representations. Here, we see that we can use a single method to simultaneously do both.

Interpolation. Finally, given the intuition developed above, one might guess that simply drawing a line from one representation to another provides a way of doing planning. Perhaps surprisingly, this works well in practice and is theoretically justified [3]. [slide]

4 Fully General RL Problems

Up to now we have been considering goal-reaching problems. However, certain practical problems of interest are defined by behaviors that are not uniquely identified by one goal. For example, maybe we want to pick up the object in front of us, regardless of what object that is. Maybe we want to a particular goal on the other side of the room, but do it while making as little sound as possible. Practically, in some settings you have not just one desired goal state, but many. Or, maybe you have some examples of good outcomes and some examples of bad outcomes. Or, maybe you have a set of states labeled with rewards (i.e., the fully-general RL problems). We'll see how these same estimated probabilities can be used to address any of these problems.

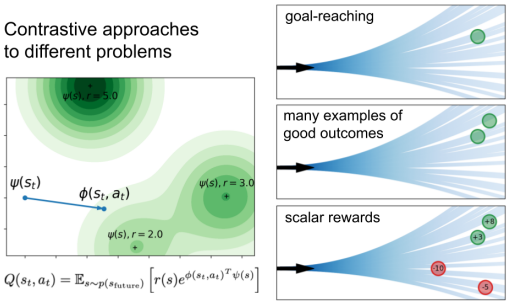

Figure 4: Estimating the probability of future events lets you solve many different types of control problems.

While the math above focused on the probability of reaching a single goal, we can use those same estimated probabilities to estimate the likelihoods of other sorts of events: multiple good outcomes, bad outcomes, outcomes with varying levels of goodness (i.e., rewards).

Let's see how this works. Let's say that we have a set of reward-labeled states $\{(s, r(s)\}$, and let's assume that the distribution over these states is the same as the $p(s_{t+})$ distribution used for contrastive learning. For simplicitly, we'll assume that the rewards depend only on the current state. Then, after learning the contrastive representations, we can estimate the Q function for a task as

$$Q(s,a) = \mathbb{E}_{p(s_{t+}|s,a)}[r(s)] \tag{13}$$

$$= \mathbb{E}_{p(s_{t+})}\left[\frac{p(s_{t+} \mid s,a)p(s_{t+})}{r}(s)\right] \tag{14}$$

$$\approx \mathbb{E}_{p(s_{t+})}\left[e^{\phi(s,a)^T\psi(s_{t+})}r(s)\right] \tag{15}$$

$$\approx \frac{1}{k}\sum_{s,r} e^{\phi(s,a)\psi(s)}r. \tag{16}$$

Note that this last expression looks like kernel smoothing. However, whereas kernel methods typically *assume* that the data are structured such that the kernel makes sense, we're learning the representations so that the kernel makes sense.

As a special case where there is one good state $r(s) = \mathbb{1}(s = s^*)$, the Q function is $e^{\phi(s,a)^T\psi(s^*)}$, so we recover the original goal-reaching setting. In the special case where we have a collection of good states, the Q function is the $\frac{1}{k}\sum_{s^*} e^{\phi(s,a)^T\psi(s^*)}$. And, in a setting where we have a collection of good states s^* ($r = +1$) and bad states s^- ($r = -1$), the Q function is $\frac{1}{k}\sum_{s^*} e^{\phi(s,a)^T\psi(s^*)} - \sum_{s^-} e^{\phi(s,a)^T\psi(s^-)}$. [slide]

5 Outlook

1. Contrastive RL is a way of estimating the probability ratios for visiting states in the future. This enables goal reaching and other tasks.
2. Because the problem is defined in terms of data, it avoids reward engineering.
3. The method simultaneously learns representations and policies, thus making it scalable to high-dimensional problems.

Connections with generative AI Broadly, the aim is to learn how to do *anything*. In the same way that LLMs can talk about anything and generative image models can paint a picture of anything, we want RL methods that can do anything.

In fact, there's a very close connection between generative AI and RL. [slide] Let's think about a generative image models. In a deep neural network, you go through a bunch of layers, performing iterative computation. For today's image models, this involves iteratively removing noise from the image.

RL, too, is a generative process, also involving an iterative process. The key difference is that some parts of the computation aren't differentiable – they come from interacting with the world, rather than from

One last note. Conceptually, we often schematize ML into supervised learning, unsupervised learning, and RL. However, contrastive RL highlights how these are blurry boundaries. It is simultaneously an RL method (it learns a policy), a supervised learning method (it's solving a classification problem) and an unsupervised method (there are no human labels for good/bad actions).

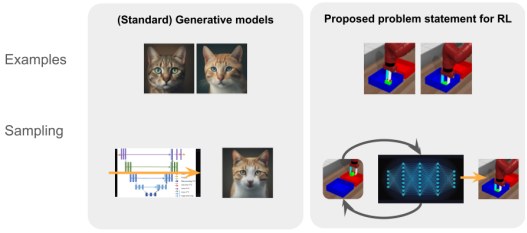

Figure 5: RL as generative AI.

References

[1] Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. (2022). Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

[2] Beeler, C., Subramanian, S. G., Bellinger, C., Crowley, M., and Tamblyn, I. (2023). ChemgymRL: An interactive framework for reinforcement learning for digital chemistry. In *NeurIPS 2023 AI for Science Workshop*.

[3] Eysenbach, B., Myers, V., Salakhutdinov, R., and Levine, S. (2024). Inference via interpolation: Contrastive representations provably enable planning and inference. *arXiv preprint arXiv:2403.04082*.

[4] Eysenbach, B., Zhang, T., Levine, S., and Salakhutdinov, R. R. (2022). Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35:35603–35620.

[5] Janner, M., Mordatch, I., and Levine, S. (2020). gamma-models: Generative temporal difference learning for infinite-horizon prediction. *Advances in Neural Information Processing Systems*, 33:1724–1735.

[6] Kaelbling, L. P. (1993). Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer.

[7] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64.

[8] Milani, S., Kanervisto, A., Ramanauskas, K., Schulhoff, S., Houghton, B., and Shah, R. (2024). Bedd: The minerl basalt evaluation and demonstrations dataset for training and benchmarking agents that solve fuzzy tasks. *Advances in Neural Information Processing Systems*, 36.

[9] Newell, A. and Simon, H. A. (1961). Gps, a program that simulates human thought.

[10] Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[11] Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039.

[12] Team, O. M., Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Xu, C., Luo, J., et al. (2023). Octo: An open-source generalist robot policy.

[13] Zhou, Z., Li, X., and Zare, R. N. (2017). Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, 3(12):1337–1344.