Week 3 of Precept

Catherine Ji

October 2024

1 Introduction

From lecture, we see that **policy gradient** and **behavorial cloning** look quite similar.

2 Behavioral Cloning

Behavioral cloning is a method for learning a policy from expert demonstrations, where the input is expert dataset $\mathcal{D} = \{(s_0, a_0), (s_1, a_1)\}$ and the output is the cloned policy $\pi(a \mid s)$. The schematic of BC is as follows:

1. Collect a dataset of expert demonstrations $\mathcal{D} = \{(s_0, a_0), (s_1, a_1), \ldots, (s_T, a_T)\}$.
2. Train a policy $\pi_\theta(a \mid s)$ to imitate the expert behavior, such that we maximize the likelihood of the expert actions under the learned policy.

Breaking down the second step, we can write the objective of behavioral cloning as:

$$\max_\theta \mathbb{E}_{(s,a)\sim\mathcal{D}} \log \pi_\theta(a \mid s). \tag{1}$$

In practice, we only have access to samples from distribution $\mathcal{D}$ and not the distribution itself. Thus, our true maximized objective is simply maximizing likelihood over our samples:

$$\max_\theta \frac{1}{|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} \log \pi_\theta(a \mid s). \tag{2}$$

which, as you may notice, is similar to the CEM objective! We'll revisit this idea later in the precept. An issue with behavioral cloning is (1) distribution shift and (2) implicit smoothing – BC is typically trained as a Gaussian policy, which may not sufficiently expressive. In addition, learning multimodal policies in high-dimensional spaces is difficult and suffers from the curse of dimensionality.

Note that behavioral cloning is an off-policy algorithm – it learns from a fixed dataset of expert demonstrations and, in its vanilla version, does not require interaction with the environment.

3 Policy Gradient

Policy gradient is a method for learning a policy directly from the environment, without access to expert demonstrations. The policy is parameterized by θ, and we aim to maximize the expected reward of the policy. We write $R(\tau)$ to denote the reward of a trajectory τ:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \tag{3}$$

where $r(s_t, a_t)$ is the reward of taking action a_t in state s_t, and trajectory τ is defined as

$$\tau \triangleq (s_0, a_0, s_1, a_1, \dots, s_T, a_T).$$

Given our **policy gradient objective**, we wish to carry out the following optimization problem:

$$\max_\theta f(\theta) \quad \text{where} \quad f(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)] = \int \pi_\theta(\tau) R(\tau)\, d\tau \tag{4}$$

where π_θ is the policy parameterized by θ, and we are sampling trajectories from this policy. As a brief review of what the $\pi_\theta(\tau)$ object actually look like, invoking the Markov assumption, we can write $\pi_\theta(\tau)$ as the product of the probabilities of each state-action pair:

$$\pi_\theta(\tau) = \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t | s_t).$$

Computing the gradient of Eq. (4) is not trivial, but we can use the **log-derivative trick** to simplify the computation. We can write the gradient of the objective as:

$$\begin{aligned}
\nabla_\theta f(\theta) &= \int (\nabla_\theta \pi_\theta(\tau)) R(\tau)\, d\tau && (5)\\
&= \int \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} R(\tau)\, d\tau && (6)\\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[(\nabla_\theta \log \pi_\theta(\tau)) R(\tau)] && (7)\\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[(\nabla_\theta \log \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t | s_t)) R(\tau)] && (8)\\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)) R(\tau)] && (9)\\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=0}^{T} \gamma^{t'} r(s_{t'}, a_{t'})] && (10)\\
&= \sum_{t=0}^{T} \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(\sum_{t'=0}^{t-1} \gamma^{t'} r(s_{t'}, a_{t'}) + \sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'}))] && (11)\\
&= \sum_{t=0}^{T} \Big[\mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'}) \nabla_\theta \log \pi_\theta(a_t \mid s_t)] && (12)\\
&\qquad + \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t'=0}^{t-1} \gamma^{t'} r(s_{t'}, a_{t'})] \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)] \Big] && (13)\\
&= \sum_{t=0}^{T} \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'}) \nabla_\theta \log \pi_\theta(a_t \mid s_t)] && (14)\\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'}) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)]. && (15)
\end{aligned}$$

Why is the final object more straightforward to estimate than (3)? (Pause for a moment.) The reason is that, similar to the actual objective, we opt to compute *expectations* of functions over a known distribution rather than the functions themselves. Note that in (3), there is no well-defined way to compute the integral over *all* possible trajectories in online settings.

Putting everything together, the policy gradient algorithm **REINFORCE** is as follows:

1. Initialize policy $\pi_\theta(a \mid s)$.
2. Sample a trajectory $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$ from π_θ.
3. Compute the gradient of the expected reward $\nabla_\theta \mathbb{E}_\pi[R(\tau)]$.
4. Update the policy parameters θ in the direction of the gradient $\theta \leftarrow \theta + \eta \nabla_\theta \mathbb{E}_\pi[R(\tau)]$. Repeat 2-4.

Note that REINFORCE is an on-policy algorithm, meaning that it requires samples from the current policy to update the policy.

4 Connection to CEM

Does this estimation of the gradient remind you of anything we've seen before? Indeed, the policy gradient algorithm is quite similar to the cross-entropy method (CEM) we saw in the previous precept. In CEM, we also optimize a function by sampling trajectories and updating the parameters in the direction of the gradient: at the end of the day, the policy is a distribution, just like the distribution $p_\theta(x)$ we considered in CEM.

5 Questions

With someone next to you, write pseudocode to implement CEM.

Answer:

Algorithm 1 Cross-Entropy Method (CEM) for Action Sequence Optimization

```
Require: Initial action probabilities mean_probs of shape (horizon, num_actions), evaluation function evaluate_fn,
    parameters: horizon, num_actions, sample_size, elite_frac, cem_iterations.
1: for $i = 1$ to cem_iterations do
2:     (a) Sample sequences: Draw sample_size action sequences using mean_probs
3:     (b) Evaluate: Compute rewards for each sequence using evaluate_fn
4:     (c) Select top-n_elite: Keep highest-reward sequences
5:     (d) Update best sequence: Store if highest reward so far
6:     (e) Refit probabilities: Update mean_probs based on action frequencies in elite set
7: end for
8: return best_seq, mean_probs
```