CSE6243: Advanced Machine Learning **Fall 2023**

Lecture 24: Imitation Learning

Lecturer: Bo Dai *Scribes: Manthan Joshi, Albert Wilcox*

Note: *LaTeX template courtesy of UC Berkeley EECS Department.*

24.1 Recap

The last four lectures concerned the fundamentals of reinforcement learning (RL). In particular, they covered the following topics.

- **Markov Decision Processes** (MDPs) are a formalism used to reason about agents in decision making settings. A simple deterministic MDP can be specified using a tuple $(\mathcal{S}, \mathcal{A}, f, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the environment dynamics and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function. A key property of MDPs is the Markov property, which states that p and r are only dependent on the current state and action.
- **Policy evaluation** refers to a class of methods whose purpose is to compute expected return under a policy, such as by learning a value function. **Policy optimization** involves using an evaluation in order to improve the policy.
- There are a variety of distinctions to be made between RL algorithms. For example there are **model-free** and **model-based** algorithms, and model-free algorithms can be either **value-based** or **policy-based**. Algorithms can be **on-policy** or **off-policy** and can be tabular or use function approximation. Note that these distinctions need not be mutually exclusive. For example actor-critic methods such as DDPG are both policy-based and value based, and methods like DREAMER have model-based and model-free components.

24.2 New Content

24.2.1 Behavioral Cloning

In reinforcement learning, the goal is for the agent to learn based on interacting with the environment and collecting rewards. In contrast, behavioral cloning addresses a problem setting where we do not have access to a reward function, but do have an offline dataset of states s_i and samples from the optimal policy $a_i \sim \pi^*(a_i|s_i)$. This problem is actually much easier, as we can simply use supervised learning to directly learn actions, avoiding many of the pitfalls associated with RL.

To be more specific, we are given a dataset

$$\mathcal{D} = \left\{ (s_i, a_i^*) \sim \prod_{i=1}^{N} d^{\pi^*}(s_i) \pi(a_i^*|s_i) \right\} \tag{24.1}$$

where $d^{\pi^*}(s_i)$ is the probability density of visiting state s_i when rolling out the optimal policy π^*. With this dataset, we learn an estimator $\hat{\pi}$ according to the the following objective:

$$\hat{\pi} = \arg\max_{\pi} \sum_{i=1}^{N} \log \pi(a_i^*|s_i). \tag{24.2}$$

While this may work well in simple settings, policies learned in this manner tend to suffer from distribution shift. This is a phenomenon where the policy makes a small error, leading it to a state slightly out of its training distribution. In this state, it makes a further error, and this process continues until the policy has completely left its training distribution and acts completely unpredictably. This phenomenon is formalized by Theorem 24.1, which shows that the policy's error grows quadratically in the horizon of a test trajectory.

Theorem 24.1 *Assume that the total variance distance TV between the learned estimator $\hat{\pi}$ and the optimal policy π^* satisfies* $\mathrm{TV}(\hat{\pi}, \pi^*) \le \epsilon$. *Then after H steps, we have*

$$J(\pi^*) - J(\hat{\pi}) \le H^2\epsilon. \tag{24.3}$$

Proof: Let $\pi[t]$ be a sequence of policies or length H, where the first t are optimal and the remainder are from the estimator, so

$$\pi[t] = (\pi_1^*, \ldots, \pi_t^*, \hat{\pi}_{t+1}, \ldots, \hat{\pi}_H). \tag{24.4}$$

We note that with this setup we can recover the learned policy with $\hat{\pi} = \pi[0]$ and the optimal with $\pi^* = \pi[H]$. Using this, we see that

$$\begin{aligned}
J(\pi^*) - J(\hat{\pi}) &= \sum_{t=1}^{H} (J(\pi[t] - J(\pi[t-1])) && (24.5)\\
&= \sum_{t=1}^{H} \mathbb{E}_{s\sim d_t^{\pi^*}} \left[\mathbb{E}_{\pi_t^*}[Q_{t+1}^{\hat{\pi}}] - \mathbb{E}_{\hat{\pi}_t}[Q_{t+1}^{\hat{\pi}}]\right] && (24.6)\\
&= \sum_{t=1}^{H} \mathbb{E}_{s\sim d_t^{\pi^*}} \left[\langle Q_{t+1}^{\hat{\pi}}, \pi_t^* - \hat{\pi}_t\rangle\right] && (24.7)\\
&\le \sum_{t=1}^{H} \mathbb{E}_{s\sim d_t^{\pi^*}} \left[(H-t) \max_{f\in[0,1]} \langle f, \pi_t^* - \hat{\pi}_t\rangle\right] && (24.8)\\
&= \sum_{t=1}^{H} t\epsilon && (24.9)\\
&\sim O(H^2\epsilon) && (24.10)
\end{aligned}$$

■

24.2.2 DAgger

DAgger is an imitation learning algorithm which addresses issues with distribution shift that arise when rolling out behavioral cloned policies. At a high level, it works by rolling out the learned policy and querying the expert policy at each state it encounters. Then, it periodically retrains on the new, expanded dataset. Over time, this expands the distribution of states where the policy is in-distribution and can act predictably. A detailed algorithm is presented in Algorithm 1.

Algorithm 1 DAgger

```
Require: Expert policy $\pi^*$, hyperparameters $\beta_i$.
  Initialize an empty dataset, $\mathcal{D} \leftarrow \varnothing$
  Initialize $\hat{\pi}_1$ to any policy.
  for $i \in \{1, \dots, N\}$ do
    Let $\pi_i \leftarrow \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    Roll out $\pi_i$ for $m$ steps and record visited states. Let $\mathcal{D}_i = \{(s_j, \pi^*(s_j))\}_{j=1}^m$.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$.
    Train $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
  end for
```

Note that the β_is are hyperparameters weighting the demonstrator and learned policies at each iteration, and they should decay from $\beta_1 = 1$ to $\beta_N = 0$.

DAgger has numerous advantages and disadvantages. It does a good job addressing distribution shift, and can lead to some very high-performing trajectories. However, it adds substantial supervisor burden and may in practice lead to lower-quality supervisor actions. For example, it would be very difficult to provide accurate controls in real time to a car which is ignoring the demonstrator's controls and acting according to its own policy since we are used to acting based on how our controls affect the system we are controlling.

24.2.3 Learning from Preferences

In the learning from preferences setting, the learner has access only to a set of expert preferences $\mathcal{D} = \{(s, a^+, a^-)_i\}_{i=1}^m$. There are various ways to address this setting. For example, we can directly optimize a Q function by optimizing Equation 24.11 using a cross entropy loss

$$p(a^+ > a^- | s) = \sigma(Q(s, a^+) - Q(s, a^-)). \tag{24.11}$$

With the Q function learned, we can use it to create a maximum-entropy policy $\pi(a|s) \propto \exp(Q(s, a))$.

24.2.4 Reinforcement Learning from Human Feedback

Reinforcement learning form human feedback (RLHF) refers to a class of methods which use human feedback to learn a reward function for reinforcement learning. They have been applied with great results to large language models, which we'll focus on here. A high-level overview is shown in Figure 24.1.

First, the authors use supervised training to finetune their model on human-written responses and outputs. This is a first step in aligning the model to better handle generative tasks, which humans are more likely to require of it. We note that this is a different process than instruction finetuning Wei et al. (2022), which finetunes the language model by turning problems from publicly available NLP datasets into instruction-following tasks.

Using their supervised fine-tuned model, the authors created a web interface for people on the internet to chat with the chatbot, and the authors recorded the prompts these users gave it. They filtered this down into their prompt dataset. From there they alternated between updating their reward model and using it to fine-tune with RL as discussed below.

In order to learn a reward model, they query the language model using the prompt dataset and record K samples from the model. They then had human experts rank these preferences, leading to $\binom{K}{2}$ comparisons.

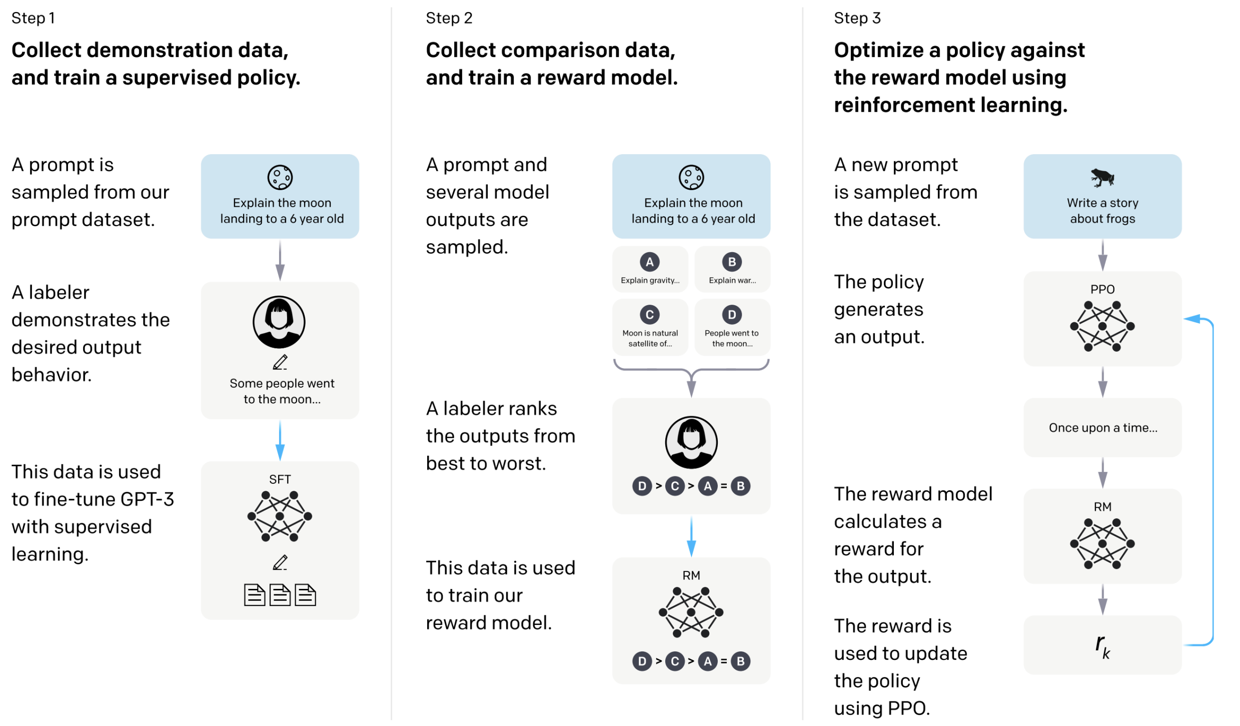
Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A Explain gravity...
B Explain war...
C Moon is natural satellite of...
D People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Figure 24.1: The three steps used in Ouyang et al. (2022) for fine-tuning an LLM using human feedback. First, the authors used supervised fine-tuning on prompts and responses writted by humans. Next, they query human comparisons between model outputs that they use to learn a reward function. Finally, they use this reward function to finetune the model using PPO. Note: this figure is taken from Ouyang et al. (2022).

Finally, they use this to optimize a cross entropy loss

$$\mathcal{L}(\theta) = -\frac{1}{\binom{K}{2}} \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}}\Big[\log(\sigma(r_\theta(x,y^+) - r_\theta(x,y^-)))\Big]. \tag{24.12}$$

After learning a reward function, authors use it to optimize the policy. While a naïve solution would be to simply run PPO out of the box, this leads to regressions because it ends up over-optimizing the learned reward function, which is not perfect. To that end, authors propose the following objective:

$$\begin{aligned} \text{objective}(\phi) = &\mathbb{E}_{(x,y)\sim D_{\pi_\phi^{\text{RL}}}} \left[r_\theta(x,y) - \beta \log\left(\pi_\phi^{\text{RL}}(y|x)/\pi_\phi^{\text{SFT}}(y|x)\right)\right] & (24.13) \\ &+ \gamma \mathbb{E}_{x\sim D_{\text{pretrain}}}[\log(\pi_\phi^{\text{RL}}(x))] & (24.14) \end{aligned}$$

Note that in line 24.13 the authors add a KL divergence term between the RL-trained model and the supervised fine-tuned model, which helps to prevent the learner from overfitting the reward model. They also mix in gradients from pretraining in line 24.14 in order to prevent regressions on the NLP tasks it was trained on.

References

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2022). Finetuned language models are zero-shot learners.