Lecture 15: Model-Based RL for Actor-Critic Methods

Logistics:

- HW6 released. This HW and the remaining two (7, 8) are both worth 50%. The aim is that you spend the other portion of the time on the project
- Final project. Find a partner as soon as possible. Talk with us after class if you still need a partner. Post on Ed if you still need a partner. Come to OH if you still need a partner. The first part of the project is a project proposal, which is due on Monday Apr 1. This counts for very little of the grade (5%), but is useful for getting feedback before submitting the final project. Do make use of office hours for discussing project ideas. Both Mengdi and I are happy to steer projects towards one that we'd find more exciting/aligned with our research.

1 Story

- Learning to fit models to the world is one of the oldest problems in science. E.g., predicting where stars will be on different nights.
- Early 1600s, Galileo and others discovered Jupiter's moons.
- In turned out that, by fitting an accurate model to the moons, you could reason backwards to infer you longitude in earth. This was a really important problem for navigation.
- On land, this was very effective. After a survey of France using this technique, French king complained that the astronomers had taken more territory from him than his enemies.
- At sea, it was very difficult to actually see Jupiter's moons. Solution ended up to be building very accurate clocks.
- In the mid 1900s, there was a lot of interest in fitting models with a small number of parameters. This was referred to as *system identification*.
- Today, using clocks actually requires a model of the world. Because of relativistic effects, satellite clocks gain about 38 microseconds per day (goal is to be less than 10ns / day). And, simply making the clocks tick more slowly doesn't work, because satellites are often in elliptical orbits and so the rate of change isn't constant throughout a day. Read more.

Where are we going? The second half of the semester will be a mix of advanced topics:

- Building effective algorithms
  - Practical considerations for implementing actor critic methods (last week)
  - How models can help
  - How to do exploration?
- Different problem settings:
  - Inverse RL
  - RLHF
  - Offline RL
  - multi-agent RL (alpha star)
- Student presentations!

2 Review: How to learn a model

- Input is (s, a), output is s'.
- Note that this can be a stochastic function – when you feed in the same state and action, you may get different next states. This stochasticity is important for two reasons:
  - The environment may truly be stochastic. E.g., hard to predict where bar-goers will amble at 1am (aleatoric uncertainty)
  - In new scenarios, we just haven't learned how things work. E.g., driving in a new country. (epistemic uncertainty)
- Effective MBRL methods, like PETS, capture both sorts of uncertainty. E.g., in PETS, we sample $i \sim \{0, 1, \cdots, k-1\}$ and $a \sim p_i(s' \mid s, a)$.
  - $i \sim$ – this captures epistemic uncertainty. There are multiple plausible models/hypotheses for how the world works, and we don't yet have enough data to weed out all the possible hypotheses.
  - $a \sim$ – this captures aleatoric uncertainty, randomness innate in how the world works.

  Check out work on Bayesian Deep Learning! This video tutorial is good; COS 513 is good too.
- Last time we discussed how to use this model for planning – using many rollouts to evaluate whether a given sequence of actions was good.

Today. How can we use models for actor-critic RL? These methods will be more scalable and faster than methods that search over sequences of actions. They will also reveal that the line between model-based and model-free methods is blurry.

Modern actor-critic methods have three components: an actor, a critic, and a replay buffer. Today, we'll see how each of these components can be improved using a model. These ideas are usually studied separately.

- Sec. 3: Augmenting the replay buffer
- Sec. 5: Improving the actor update
- Sec. 4: Improving the critic update

3 Dyna [7]

Let's start by recalling the rough template for a model-based RL algorithm from last time:

1. Initialize a random policy
2. Collect data from that policy – $\{(s, a, r, s')\}$
3. Fit a model to that data – $\max_\theta \mathbb{E}[\log p_\theta(s' \mid s, a)]$
4. Use that model for RL (somehow)
5. What step is missing here?
6. Repeat – go back and collect more data

Last time, the way that we used the model was to perform planning – optimizing over *sequences* of states, rolling out the model for many time steps. One of the key challenges with doing this, however, is that of compounding errors (see Fig. 1). Ideally, we'd have a way of using a model for just short horizon rollouts. As you might recall, for actor-critic methods, the critic is trained on individual transitions, and the actor is trained just on states (no model/transitions required). So, it seems like combining models with actor critic RL is a good idea.

To do this, we'll maintain two replay buffers, one of real data and one of synthetic data. We'll fill the first buffer with data sampled from the environment and the second buffer with data sampled from the learned model. Of course, the learned model will just be trained on the first buffer. The actor and the critic will be trained on data from both buffers:

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

Initialize policy π_ϕ, predictive model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs **do**
  Train model p_θ on $\mathcal{D}_{\text{env}}$ via maximum likelihood
  for E steps **do**
    Take action in environment according to π_ϕ; add to $\mathcal{D}_{\text{env}}$
    for M model rollouts **do**
      Sample s_t uniformly from $\mathcal{D}_{\text{env}}$
      Perform k-step model rollout starting from s_t using policy π_ϕ; add to $\mathcal{D}_{\text{model}}$
    for G gradient updates **do**
      Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$

Figure 2: Algorithm credit [5]

Note that the model is only used for short-horizon rollouts, thus avoiding the problem of compounding errors. This general idea is typically referred to as Dyna [7].

- Q: Does Dyna result in changes to the actor update and/or the critic update? (A: It always results in more data for training the critic. It only results in changes to the actor update when $k \geq 2$.)

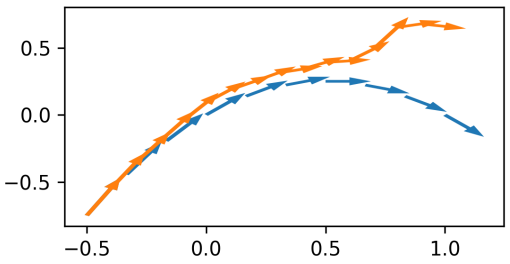

Figure 1: Rolling out a model for many time steps can result in compounding errors

3.1 Practical Considerations

We usually use MBRL in setings where we have limited data – the hope, of course, is that we can use the model to "make up" for the limited data.

- Q: What can go wrong when we fit a large parametric model to limited data? (Overfitting)
- Q: How can we detect overfitting? (validation set)
- Q: How can we combat overfitting (e.g., early stopping, dropout, input noise, weight decay)

To gain some more intuition here, I want you to consider the special case where $k = 1$ and we massively, massively overfit the data. That is, at every state, we correctly predict the next state with no errors. What happens if you apply a Dyna algorithm in this case?

Answer: You recover model-free RL! This highlights one of the connections between model-based and model-free RL. In actuality, they live on a continuous spectrum. And, when models work, it must be because they are underfitting the data; they are finding structure/patterns in the data and generating new combinations of those patterns. If they were just regurgitating the training data, then they'd be exactly the same as model-free RL. This reasoning suggests that models will be most useful when your model is parametrized in a way that exploits things that are known about the environment. For example, if you expect the state to vary slowly, you might parametrize your model as $s_{t+1} = s_t + f_\theta(s_t, a_t)$. Similarly, if you expect that your model is equivariant, you can exploit that. This reasoning also highlights how we can think about the replay buffer itself as a form of *nonparametric model*. Finally, like we mentioned last time, we expect that models will be most useful in settings where they can be trained (or pretrained) on additional data that is not available for doing RL (e.g., it lacks rewards)

In a certain special case with linear function approximation, you can show that Dyna is equivalent to a form of Q-learning [8].

4 Improving the Critic Update

We next look at how we can improve the critic update using a model. We'll focus on a technique known as model-value expansion [2], though there are many related techniques that go by slightly different names. The key idea is to get a more accurate estimate of the TD target for use in the critic loss. Recall that the TD target y_t is supposed to be an estimate of the cumulative discounted return starting from a particular state and action; it is treated as the "label" for training $Q(s_t, a_t)$. Previously, we had estimated the TD target as $y_t = r(s_t, a_t) + \gamma Q(s_t, a_t)$. In effect, we're combining the reward at the current step plus our estimate for the reward at the next step. But what if we looked at the rewards at more future steps before plugging in our Q-value estimate? This would look like the following:

$$y_t = \sum_{\Delta=0}^{H-1} \gamma^{\Delta} r(s_{t+\Delta}, a_{t+\Delta}) + \gamma^H V(s_{t+H}). \tag{1}$$

This looks like the idea of n-step returns that we had discussed before. Previously, we had talked about how this sort of estimator is biased because we're looking at actions sampled from a different policy. However, if we learn the dynamics model, then we can compute these n-step returns by sampling actions from the *current* policy, thus removing the bias issue:

$$y_t = \sum_{\Delta=0}^{H-1} \gamma^{\Delta} \hat{r}(\hat{s}_{t+\Delta}, \hat{a}_{t+\Delta}) + \gamma^H V(\hat{s}_{t+H}). \tag{2}$$

I've used the " ˆ " notation to indicate that these are estimated quantities. Note that we're estimating both the future states and the reward function (i.e., we have to learn the reward function). Typically, estimating the reward function is pretty easy, so we'll focus the discussion on learning the dynamics.[1]

This MVE idea modifies the TD target, which gets plugged into the usual critic loss. The actor loss remains the same. Like with the methods above, we'll iteratively collect data from the real environment to add to the replay buffer, update the model, and then update the actor and critic before repeating the loop.

Note that there is a tradeoff involved in selecting k. A longer model rollout will make the estimated TD target less reliant on value function but is more subject to compounding errors. Some work has looked at dynamically choosing k based on the model epistemic uncertainty [1].

5 Improving the Actor Update

In actor critic methods, our aim is to update the actor to select return-maximizing actions. Typically, we estimate the return with the Q function at the current time step:

$$\max_{\pi} \mathbb{E}_{a \sim \pi(a|s)}[Q(s, a)]. \tag{3}$$

However, if we have a model, we can use the same idea as in Sec. 4 to come up with a better estimate of the future returns:

$$\max_{\pi} \mathcal{L}(s_t; \pi) = \mathbb{E}_{\substack{a_t \sim \pi(a_t|s_t) \\ s_{t+1} \sim \hat{p}(s_{t+1}|s_t, a_t) \\ a_{t+2} \sim \pi(a_{t+2}|s_{t+1}) \\ s_{t+2} \sim \hat{p}(s_{t+2}|s_{t+1}, a_{t+1}) \\ \cdots}} \left[\sum_{\Delta=0}^{H-1} \gamma^{\Delta} \hat{r}(s_{t+\Delta}, a_{t+\Delta}) + \gamma^H V(s_{t+H}) \right]. \tag{4}$$

Note that the value function is weighted by γ^H, so that errors in the value function end up mattering less.

Taking the gradient of this objective involves backpropagating through multiple sampling steps. This is sometimes referred to as *backprop through time* (BPTT):

[1] You can treat the reward function as part of the dynamics by appending the rewards to the observations as a $d + 1^{\text{th}}$ coordinate; the new reward function is just the last coordinate of the observation.

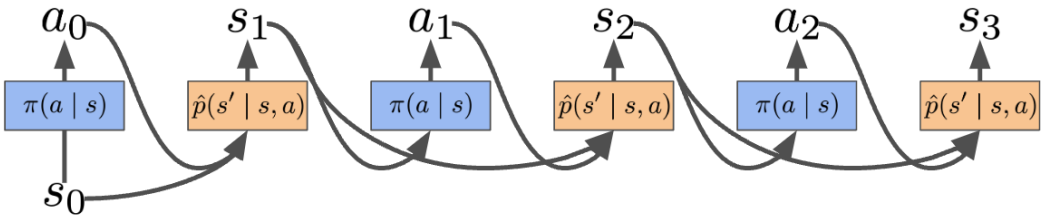

Figure 3: Backprop through time.

When backproping through $r(s_0, a_0)$, we only have to backprop through a single neural network. However, when backproping through (say) $r(s_2, a_2)$ or $V(s_3)$, we have to backprop through many networks. This can result in gradients getting very very large and ill-behaved [6]. Thus, a medium value of k tends to work best. We call this method of updating the actor $SVG(k)$. Note that SVG(0) is equivalent to DDPG but with a stochastic actor.

You might note that BPTT looks like training an RNN or LSTM. However, what makes it especially challenging in the RL setting is that we can't choose to make $p(s' \mid s, a)$ a simple function – it's predetermined by the environment. However, some RL methods look at learning representations that do evolve according to a simple function. These methods have successfully gotten BPTT to work effectively for longer horizons [3, 4].

6 Summary

- Models can be effective. Handling uncertainty is important
- Models can be used directly for action selection (last time), or as part of an actor-critic algorithm (today).
- On Thursday, we'll have a guest lecture from Yuandong Tian about tbd.
- Following week we'll explore connections between RL and probabilistic inference. We'll see how this allows us to do better exploration, and how to do mind-reading – how to guess what reward functions an agent is trying to maximize.

References

[1] Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. (2018). Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31.

[2] Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. (2018). Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*.

[3] Ghugare, R., Bharadhwaj, H., Eysenbach, B., Levine, S., and Salakhutdinov, R. (2022). Simplifying model-based rl: learning representations, latent-space models, and policies with one objective. *arXiv preprint arXiv:2209.08466*.

[4] Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

[5] Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32.

[6] Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2018). Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR.

[7] Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier.

[8] Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. P. (2012). Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*.