17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 15: Model-Based RL

Lecturer: Gokul Swamy
Scribe: Yuemin Mao, Miaosi Dong, Khush Agrawal, Eryn Ma

Model-based RL is a subclass of reinforcement learning where the agent learns a dynamics model of the environment. This model can be used to simulate future states, calculate rewards, and optimize action selection accordingly without ever interacting with the actual environment.

15.1 What's a "model"?

Recall a Markov Decision Process (MDP) that can be defined as:

$$\begin{aligned} M = \{ & S \quad \text{(state space)} \\ & A \quad \text{(action space)} \\ & r \quad \text{(reward function)} \\ & \mathcal{T} \quad \text{(transition dynamics)} \\ & H \quad \text{(horizon)} \\ & \rho_0 \quad \text{(initial state distribution)}\} \end{aligned}$$

The goal of model-based RL is to learn the transition dynamics ($\mathcal{T}$) of the environment. This model can be used to evaluate trajectories without interacting with the actual environment.

1. An MDP includes a *transition function / dynamics* $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, which has $|S|^2|A|$ elements to learn. This is a lot more than learning a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which only has $|S||A|$ elements, increasing the data required.

2. *Generative model* access to an MDP allows an agent to query the "model" at any preferred state-action pair to get predictions of the next state (i.e. samples from the next state distribution). Learning a dynamics model gives us such access to the underlying MDP we're trying to solve.

3. Model-based RL (MBRL) is performing RL in an MDP M' is M but with ground truth T replaced with learned T', which is approximated. Alternatively, it could also involve test time planning (e.g., model predictive control) for finding actions that optimize an objective function inside M' – we will discuss this in detail in a future lecture.

15.2 What makes a good model?

A perfect model would be the one returning predictions exactly the same as the ground truth. However, in the real-world, learning problems are prone to errors. Since we cannot model everything accurately, it is better to give more importance to the regions of the state and action spaces visited by the policy we're evaluating. To summarize, a "good" model is one that allows us to accurately evaluate the performance of a policy: *a policy looks performant in a good model if and only if it is performant in the real world.* We can make this intuition more precise via the following lemma, which tells us that if we've learned a model that can accurately predict state transitions everywhere [1], we will achieve the preceding desiderata.

Lemma 1 (Simulation Lemma, Kearns & Singh, 2002) *Suppose that for all (s, a) in state action spaces, the total variation distance between the learned model and the ground truth is less than ϵ, $\sum_{s'} |T'(s'|s,a) - T(s'|s,a)| \leq \epsilon$. Then for any policy $\pi : \mathcal{S} \to \mathcal{A}$, we have*

$$|J(\pi, M) - J(\pi, M')| \leq \frac{H(H-1)}{2}\epsilon \tag{15.1}$$

Proof: If the following inequality holds for all states, then it also holds in expectation over the initial state distribution, since expectation preserves inequalities. Therefore, in the proof, we bound the value difference pointwise over states, which allows us to derive a bound on the performance difference by averaging these pointwise bounds over the initial states:

$$|V_h^\pi(s, M) - V_h^\pi(s, M')| \leq (H-h)\epsilon + \left|V_{h+1}^\pi(s, M) - V_{h+1}^\pi(s, M')\right| \tag{15.2}$$

We prove Eqn. 15.2 using backwards induction. The following lemmas are used in the proof:

Lemma 2 (Triangle Inequality) $|a - b| = |a - c + c - b| \leq |a - c| + |c - b|$

Lemma 3 (Holder's Inequality) $\sum_x p(x)g(x) =< p, g >\leq \|p\|_1 \cdot \|g\|_\infty$

[1] One can easily modify this proof to add an expectation over any desired state distribution.

Base Case:

$$h = H \Rightarrow LHS = 0 \leq 0 = RHS$$

Inductive Hypothesis: Assume for $h+1$, we have:

$$|V_{h+1}^{\pi}(s, M) - V_{h+1}^{\pi}(s, M')| \leq (H - h - 1)\epsilon.$$

Inductive Step: Expanding the value functions via Bellman equation:

$$\begin{aligned}|V_h^{\pi}(s, M) - V_h^{\pi}(s, M')| = \big|&\mathbb{E}_{a\sim\pi(s), s'\sim T(s,a)}[r(s,a) + V_{h+1}^{\pi}(s', M)] \\ &-\mathbb{E}_{a\sim\pi(s), s'\sim T'(s,a)}[r(s,a) + V_{h+1}^{\pi}(s', M')]\big|\end{aligned}$$

The reward function does not depend on the next states, so the expectation of $r(s, a)$ in both terms are the same.

$$\begin{aligned}= \big|&\mathbb{E}_{a\sim\pi(s), s'\sim T(s,a)}[\cancel{r(s,a)} + V_{h+1}^{\pi}(s', M)] \\ &-\mathbb{E}_{a\sim\pi(s), s'\sim T'(s,a)}[\cancel{r(s,a)} + V_{h+1}^{\pi}(s', M')]\big|\end{aligned}$$

For simplicity, we can upper bound the expectation with the maximum over actions.

$$\leq \max_a \left| \sum_{s'} T(s'|s,a)V_{h+1}^{\pi}(s', M) - T'(s'|s,a)V_{h+1}^{\pi}(s', M') \right|$$

We now proceed by applying Lemma 2.

Let $a = \sum_{s'} T(s'|s,a)V_{h+1}^{\pi}(s', M), b = \sum_{s'} T'(s'|s,a)V_{h+1}^{\pi}(s', M'), c = \sum_{s'} T'(s'|s,a)V_{h+1}^{\pi}(s', M)$. Then, we have:

$$\begin{aligned}\leq \max_a &\left| \sum_{s'} T(s'|s,a)V_{h+1}^{\pi}(s', M) - T'(s'|s,a)V_{h+1}^{\pi}(s', M) \right| \\ &+ \left| \sum_{s'} T'(s'|s,a)V_{h+1}^{\pi}(s', M) - T'(s'|s,a)V_{h+1}^{\pi}(s', M') \right|\end{aligned}$$

Next, we apply Lemma 3. Define $p = T(s'|s,a) - T'(s'|s,a), q = V_{h+1}^{\pi}(s', M)$. Then, $|p| \leq \epsilon$ by assumption, and the maximum value of the value function, $|q|$, is at most $H - h$, hence the first term is upper bounded by $\epsilon(H - h)$.

By assumption, the 2nd term is upper bounded by the last step. Intuitively, this is because if we have a bound on the probability of a difference and the maximum amount we can pay per difference, we can in the worst case pay the product of those two quantities.

$$\leq \max_a \left| \underbrace{\sum_{s'} T(s'|s,a)V^{\pi}_{h+1}(s',M) - T'(s'|s,a)V^{\pi}_{h+1}(s',M)}_{\leq \epsilon(H-h)} \right|$$

$$+ \left| \underbrace{\sum_{s'} T'(s'|s,a)V^{\pi}_{h+1}(s',M) - T'(s'|s,a)V^{\pi}_{h+1}(s',M')}_{\leq \text{value of the last step}} \right|$$

We can now expand out the recursion over h to complete the proof:

$$|J(\pi,M) - J(\pi,M')| \leq \sum_{h}^{H} (H-h)\epsilon = \frac{H(H-1)}{2}\epsilon.$$

■

15.3 How do we fit a good model?

Prima facie, one might like the model to be accurate on the current policy's state distribution. However, just doing this is not enough. Once we learn a model, we also run RL steps that optimize the policy using this approximated model. When the model is inaccurate, the RL step might exploit such inaccuracies and lead to poor performance in the real world.

This can work extremely poorly. Similar to approximate policy iteration, and the follow the leader counter-example, the model might switch back and forth repeatedly, with the RL step never finding a good policy. We now sketch such an example on a tree-structured MDP.

$M^\star$ in Figure 15.1 represents the real world. The optimal policy π_E is highlighted in green. Assume the initial policy we start off with goes right twice, as shown by the highlighted red trajectory in M_0 (Figure 15.2). We'll likely be accurate on states seen in the training distribution, causing us to correctly predict the green label of 0 for going right twice. However, we can be arbitrarily bad on states we didn't train on – for example, we can predict a reward of 1 for going right and then left, as we do in M_0. The optimal policy inside M_0 would then exhibit this behavior, effectively exploiting the inaccuracies in the learned model

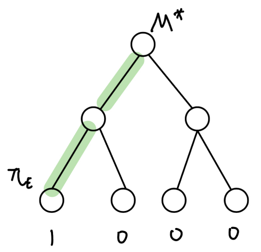

Figure 15.1: The optimal policy π_E goes left twice and recieves reward 1.

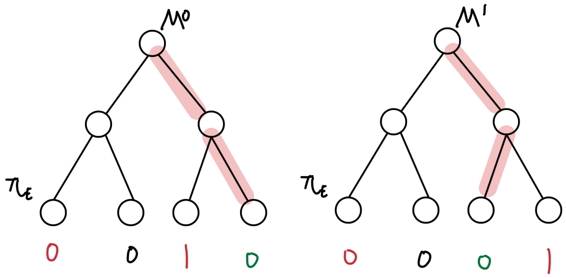

Figure 15.2: Since doing RL exploits the model and can cause it to be queried on states outside of its training distribution, only training on the latest data might lead to a situation where we oscillate between inaccurate models and suboptimal policies. We use a red highlight to visualize the policy used to generate the training data and a green leaf node label to denote we got this reward correct. We use red to denote OOD leaf nodes where we incorrectly predict the reward value. Observe that the optimal policy in M^0 induces M^1 and vice-versa.

that happen in OOD states. Then, based on the new policy, we could learn some new M_1 (Figure 15.2). Again, we're correct in-distribution, but could be overly optimistic on the value we'd receive of going right twice. Observe that the optimal policy in M_1 could induce M_0, so we could just oscillate between these two models and their corresponding optimal policies without learning anything useful. Intuitively, this is because a model trained on a single round of data can be too *optimistic* on unseen states.

Instead, we could aggregate all the past datasets we have observed. This is using a no-regret algorithm over models (specifically, Follow the Regularized) Leader) to deal with the "best response" over policies. This is not explicitly game-solving, but the no-regret principle can help us deal with the distribution shift caused by RL exploiting the model. While $M_{no-regret}$ will accurately predict 0 for the two paths it has seen (visualized in Figure 15.3), since it has never observed any data of optimal policy π_E, it could still underestimate the value of going

Algorithm 1 No-Regret MBRL

```
Initialize some $\pi_0, \hat{T}_0$
for $t = 1, ..., T$ do
    (1) Roll out $\pi_{t-1}$ in $T^*$ to collect $D_t$
    (2) Fit $\hat{T}_t$ on $\bigcup_{\tau=1}^{t} D_\tau$
    (3) Run RL inside $\hat{T}_t$ to compute $\pi_t$
        $\rightarrow \pi_t = \arg\max_{\pi \in \Pi} J(\pi, M_t)$        “best response”
end for
```

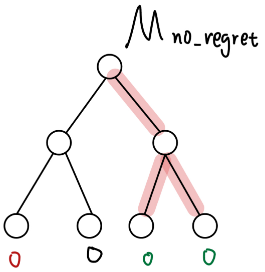

Figure 15.3: A data aggregation data method on the other hand will assign the correct reward values to both the states and not demonstrate an oscillating behavior.

left twice. Intuitively, without seeing data from the expert, we can be overly *pessimistic* about good OOD trajectories. So, while no-regret model fitting guarantees us correct policy evaluation (i.e. we correctly predict both policies get zero reward), it doesn’t guarantee we’ll compute a policy via RL in the model that is anywhere close to the quality of π_E

Now say if we have samples from π_E is that, how could we use them? Even if we fit our model perfectly on the expert data, if our policy distribution is different (i.e., taking the right path instead of left from the top node), our model will still behave poorly on unseen paths. One way to resolve this tension is to combine expert data with our policy distribution, which is essentially DAgger for MBRL, with extra expert data to reduce the exploration burden on the learner. Fitting the model on the *hybrid* distribution leads to Agnostic SysID:

Algorithm 2 Agnostic System Identification (Ross and Bagnell 2012)

```
Initialize some $\pi_0, \hat{T}_0$
for $t = 1, ..., T$ do
    (1) Roll out $\pi_{t-1}$ in $T^*$ to collect $D_t$
    (2) Fit $\hat{T}_t$ on $(\bigcup_{\tau=1}^{t} D_\tau)\left(\frac{1}{2}\right) + \frac{1}{2} D_E$
    (3) Run RL inside $\hat{T}_t$ to compute $\pi_t$
        $\rightarrow \pi_t = \arg\max_{\pi \in \Pi} J(\pi, M_t)$        “best response”
end for
```

Intuitively, *we need to be able to accurately predict the consequences of our actions (both good and bad) to learn a good policy.* Fitting the model on the mixture of expert and learner data helps balance optimism and pessimism without computational complexity. We will further explore this idea of *hybrid* RL in the model-free setting during the following guest lecture.

We can provide some more intuition for why this hybrid model-fitting procedure ensures that learn a model such that *policies look good in the model if and only if they are good in the real world* via the following three-term decomposition:

$$\begin{aligned} J(\pi_E, M) - J(\pi, M) = & \, (J(\pi_E, M) - J(\pi_E, M')) && \rightarrow \text{inaccuracy on } \pi_E \\ & + (J(\pi, M) - J(\pi, M')) && \rightarrow \text{inaccuracy on } \pi \\ & + (J(\pi_E, M') - J(\pi, M')) && \rightarrow \text{planning error} \end{aligned}$$

If the RL solver of our model is perfect, $(J(\pi_E, M') - J(\pi, M') \leq 0$ and the inner term will be non-negative. If we also fit the model well on learner and expert state distributions, $(J(\pi_E, M), J(\pi_E, M')) + ((J(\pi, M) - J(\pi, M'))$ will also be small via the Simulation Lemma. Thus, this idea is sometimes affectionately referred to as the *double simulation lemma*. In the Agnostic SysID paper, Ross & Bagnell use an upper bound on the Simulation Lemma to derive loss functions for the no-regret model fitting procedure on hybrid data (off-policy expert data + on-policy learner data), which boil down to standard MLE on the hybrid data.

15.4 How do we scale this idea?

To address the key challenge that modeling pixels is hard, we can do things in latent space by using sequential VAEs like Dreamer [1]. The training process of Dreamer is illustrated as Figure 15.4. First, an encoder maps sensory inputs x_t to stochastic representations z_t. Then, a sequence model with recurrent state h_t predicts the sequence of these representations given past actions a_{t-1}. The concatenation of h_t and z_t forms the model state from which

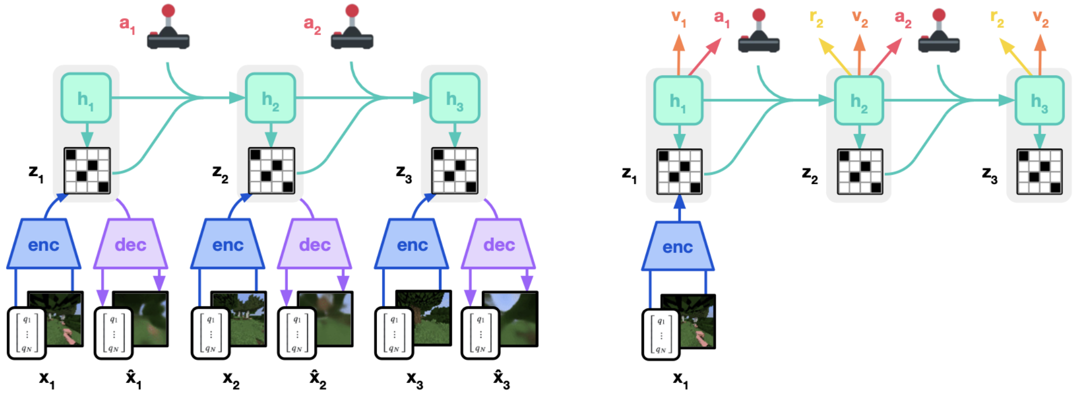

Figure 15.4: Training Process of Dreamer

we predict rewards r_t and episode continuation flags $c_t \in \{0, 1\}$ and reconstruct the inputs to ensure informative representations:

Sequence Model	$h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$
Encoder	$z_t \sim q_\phi(z_t \vert x_t, h_t)$
Dynamic Prediction	$\hat{z}_t \sim p_\phi(\hat{z}_t \vert h_t)$
Reward predictor:	$\hat{r}_t \sim p_\phi(\hat{r}_t \vert h_t, z_t)$
Continue predictor:	$\hat{c}_t \sim p_\phi(\hat{c}_t \vert h_t, z_t)$
Decoder	$\hat{x}_t \sim p_\phi(\hat{x}_t \vert z_t, h_t)$
Dynamic Prediction	$\hat{z}_t \sim p_\phi(\hat{z}_t \vert h_t)$

Question 1: Why we need a decoder?

- Without a decoder, the learned latent representations are at risk of collapsing to trivial or degenerate solutions—such as constant vectors—since there is no explicit constraint enforcing the retention of meaningful information from the original observations. In such a scenario, the latent state may become uninformative. Essentially, the decoder acts as a *grounding* mechanism, ensuring that the latent representation remains coupled with the actual observations.

Question 2: Why do we need alternative objectives beyond a decoder?

- While decoders are commonly used to encourage latent representations to retain input information, they can be inefficient for decision-making tasks. For example, reconstructing the entire input image may force the model to encode irrelevant details—such as leaves blowing in the wind—that have no impact on the task, such as driving. This leads to representations that are perceptually rich but behaviorally redundant.

- To obtain minimal sufficient representations for reinforcement learning, alternative objectives are needed. One effective approach is to replace the decoder with an inverse dynamics model, which predicts the action taken between two latent states [2]. This encourages the model to retain only information that is influenced by agent behavior, naturally filtering out task-irrelevant factors, leading to more minimal representations.

15.5 *

References

[1] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[2] Alex Lamb, Riashat Islam, Yonathan Efroni, Aniket Didolkar, Dipendra Misra, Dylan Foster, Lekan Molu, Rajan Chari, Akshay Krishnamurthy, and John Langford. Guaranteed discovery of control-endogenous latent states with multi-step inverse models, 2022.