DDA4230 Reinforcement learning Model-based methods

Lecture 12

Lecturer: Baoxiang Wang *Scribe: Jing Dong, Shaokui Wei*

1 Goal of this lecture

In this lecture we give an introduction to model-based methods, from both a theoretic perspective and an algorithmic perspective. This introduction is more conceptual and we refer the students who are interested to read more material.
Suggested reading: Chapter 8 of *Reinforcement learning: An introduction*; Chapter 2 of *Reinforcement learning: Theory and algorithms.*

2 Recap: discrete MDPs with known model

For policy evaluation, by the Bellman equation we can obtain the value function as

$$V = (I - \gamma P)^{-1} r \,.$$

To compute V in an iterative way, we use iterative policy evaluation.

Algorithm 1: Iterative policy evaluation

```
Input: Policy $\pi$, threshold $\epsilon > 0$
Output: Value function estimation $V \approx V^*$
Initialize $\Delta > \epsilon$ and $V$ arbitrarily
while $\Delta > \epsilon$ do
    $\Delta = 0$
    for $s \in \mathcal{S}$ do
        $v = V(s)$
        $V(s) = \sum_a \pi(a|s) \sum_{s',r} \mathbb{P}(s', r|s, a) [r + \gamma V(s')]$
        $\Delta = \max(\Delta, |v - V(s)|)$
```

For value optimization, by the Bellman optimality equation it amounts to solve

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & \mathbf{e}^T V \\ \text{subject to} \quad & (I - \gamma P_j)V - r_j \geq 0 \,, \quad j = 1, \ldots, m \,, \end{aligned}$$

where $\mathbf{e}$ is the all-one vector. Alternatively, the fixed-point iteration prescribes the value iteration algorithm

Algorithm 2: Value iteration

```
Input: $\epsilon$
For all states $s \in S$, $V'(s) \leftarrow 0$, $V(s) \leftarrow \infty$
while $\|V - V'\|_\infty > \epsilon$ do
    $V \leftarrow V'$
    For all states $s \in S$, $V'(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')\right]$
$V^* \leftarrow V$ for all $s \in S$
$\pi^* \leftarrow \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s')\right]$ , $\forall\, s \in S$
return $V^*(s),\ \pi^*(s)$ for all $s \in S$
```

The dual of the linear program describes policy optimization

$$\begin{aligned}
\underset{\lambda_1,\ldots,\lambda_m}{\text{maximize}} \quad & \sum_j \lambda_j^T r_j \\
\text{subject to} \quad & \sum_j (I - \gamma P_j^T)\lambda_j = \mathbf{e}\,, \\
& \lambda_j \geq 0, \quad j = 1, \ldots, m\,.
\end{aligned}$$

An iterative approach induces the policy iteration algorithm.

Algorithm 3: Policy iteration

```
Input: $\mathcal{M}, \epsilon$
$\pi \leftarrow$ Randomly choose a policy $\pi \in \Pi$
while true do
    $V^\pi \leftarrow$ POLICY EVALUATION $(\mathcal{M}, \pi, \epsilon)$
    $\pi^* \leftarrow$ POLICY IMPROVEMENT $(\mathcal{M}, V^\pi)$
    if $\pi^*(s) = \pi(s)$ then
        break
    else
        $\pi \leftarrow \pi^*$
$V^* \leftarrow V^\pi$
return $V^*(s),\ \pi^*(s)$ for all $s \in S$
```

3 Model-based MDPs

In this lecture, we take a look at a more abstract sampling model, a generative model, which allows us to study the minimum number of transition we need to observe, to characterizes the sample complexity of estimating the action value function and learning a near optimal policy.

Generative models A *generative model* provides us with a sample $s' \sim P(\cdot \mid s, a)$ upon input of a state action pair (s, a). Let us consider the most naive approach to learning (when we have access to a generative model): suppose we call our simulator N times at each state action pair. Let $\hat{P}$ be our empirical model, defined as follows:

$$\hat{P}(s' \mid s, a) = \frac{\text{count}(s', s, a)}{N} = \frac{\text{count}(s', s, a)}{\text{count}(s, a)}\,,$$

where $\text{count}(s', s, a)$ is the number of times the state-action pairs (s, a) transitions to state s'. As N is the number of calls for each state action pair, the total number of calls to our generative model is $|\mathcal{S}||\mathcal{A}|N$. We can view $\hat{P}$ as a matrix of size $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$.

The generative model setting is a reasonable abstraction to understand the statistical limit, without having to directly address exploration. We define $\hat{\mathcal{M}}$ to be the empirical MDP that is identical to the original $\mathcal{M}$, except that it uses $\hat{P}$ instead of P for the transition model. When clear from context, we drop the subscript on $\mathcal{M}$ on the values, action values. We let $\hat{V}^\pi$, $\hat{Q}^\pi$, $\hat{Q}^*$, and $\hat{\pi}^*$ denote the value function, state-action value function, optimal state-action value, and optimal policy in $\hat{\mathcal{M}}$ respectively.

A natural, key question here is then

Do we require an accurate model of the world in order to find a near optimal policy?

3.1 A naive model-based approach

Since P has $|\mathcal{S}|^2|\mathcal{A}|$ parameters, let us start with a naive approach, which estimates P accurately and then use our accurate model $\hat{P}$ for planning.

Lemma 1 (Simulation Lemma) *for all π we have that*

$$Q^\pi - \hat{Q}^\pi = \gamma(I - \gamma\hat{P}^\pi)^{-1}(P - \hat{P})V^\pi\,.$$

Proof: We start with showing $\hat{Q}^\pi = (I - \gamma\hat{P}^\pi)^{-1}r$, by showing that $(I - \gamma\hat{P}^\pi)$ is invertible. Notice that, to show a matrix $A \in \mathbb{R}^{d\times d}$ is invertible, it suffices to show that $\forall x \in \mathbb{R}^d \backslash \{0\}$, $\|Ax\| > 0$.

$$\begin{aligned}
\|(I - \gamma\hat{P}^\pi)x\|_\infty &= \|x - \gamma\hat{P}^\pi x\|_\infty \\
&\geq \|x\|_\infty - \gamma\|\hat{P}^\pi x\|_\infty \\
&\geq \|x\|_\infty - \gamma\|x\|_\infty \\
&= (1-\gamma)\|x\|_\infty > 0\,.
\end{aligned}$$

The first inequality is by triangle inequality for norms, the second inequality is due to the fact that each element of $\hat{P}^\pi x$ is an average of x. The last strict inequality is because that $\gamma < 1, x \neq 0$.

By similar arguments, we obtain that $r = (I - \gamma P^\pi)Q^\pi$. Therefore,

$$\begin{aligned}
Q^\pi - \hat{Q}^\pi &= Q^\pi - (I - \gamma\hat{P}^\pi)^{-1}r \\
&= Q^\pi - (I - \gamma\hat{P}^\pi)^{-1}(I - \gamma P^\pi)Q^\pi \\
&= Q^\pi(I - (I - \gamma\hat{P}^\pi)^{-1}(I - \gamma P^\pi)) \\
&= (I - \gamma\hat{P}^\pi)^{-1}((I - \gamma\hat{P}^\pi) - (I - \gamma P^\pi))Q^\pi \\
&= \gamma(I - \gamma\hat{P}^\pi)^{-1}(P^\pi - \hat{P}^\pi)Q^\pi \\
&= \gamma(I - \gamma\hat{P}^\pi)^{-1}(P - \hat{P})V^\pi
\end{aligned}$$

as we desired. □

Lemma 2 *For any policy π, MDP $\mathcal{M}$, and vector $v \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, we have $\|(I-\gamma P^\pi)^{-1}v\|_\infty \leq \|v\|_\infty/(1-\gamma)$.*

Proof: Notice that

$$\begin{aligned} v &= (I-\gamma P^\pi)(I-\gamma P^\pi)^{-1}v \\ &= (I-\gamma P^\pi)w\,, \end{aligned}$$

where we denote $w = (I-\gamma P^\pi)^{-1}v$. Then,

$$\begin{aligned} \|v\|_\infty &= \|(I-\gamma P^\pi)w\| \\ &\geq \|w\|_\infty - \gamma\|P^\pi w\|_\infty \\ &\geq \|w\|_\infty - \gamma\|w\|_\infty\,, \end{aligned}$$

where the final inequality follows since $P^\pi w$ is an average of the elements of w by the definition of P^π so that $\|P^\pi w\|_\infty \leq \|w\|_\infty$. □

Before we move on to the error bounds for the naive model based approach, let us establish a concentration bound for discrete distributions. We omit the proof here and students that are interested in it can find the proof in standard probability textbooks.

Lemma 3 (Concentration for discrete distributions) *Let z be a discrete random variable that takes values in $\{1, ..., d\}$, distributed according to q. We write q as a vector where $\overrightarrow{q} = (\mathbb{P}(z=j))_{j=1}^d$. Assuming we have N i.i.d samples, and that our empirical estimate of $\overrightarrow{q}$ is $(\hat{q})_j = \sum_{i=1}^N \{z_i = j\}/N$. We have that $\forall \epsilon > 0$,*

$$\mathbb{P}(\|\hat{q} - \overrightarrow{q}\|_2 \geq 1/\sqrt{N} + \epsilon) \leq \exp(-N\epsilon^2)\,,$$

which implies that,

$$\mathbb{P}(\|\hat{q} - \overrightarrow{q}\|_1 \geq \sqrt{d}(1/\sqrt{N} + \epsilon)) \leq \exp(-N\epsilon^2)\,.$$

Proposition 4 (Model accuracy) *The transition model is error bounded as*

$$\max_{s,a} \|P(\cdot|s,a) - \hat{P}(\cdot|s,a)\|_1 \leq |\mathcal{S}||\mathcal{A}|\sqrt{\frac{|\mathcal{S}|\log(1/\delta)}{N}}\,,$$

with probability at least $1-\delta$, where N is the number of samples used to estimate $\hat{P}(\cdot \mid s,a)$.

Proof: Using Proposition 3, we have

$$\mathbb{P}(\|P(\cdot \mid s,a) - \hat{P}(\cdot \mid s,a)\|_1 \geq \sqrt{S}(\frac{1}{\sqrt{N}} + \sqrt{\frac{\log(1/\delta)}{N}}) \leq \exp(-N\epsilon^2)$$

by setting $\epsilon = \sqrt{\log(1/\delta)/N}$. Then,

$$\mathbb{P}(\|P(\cdot \mid s,a) - \hat{P}(\cdot|s,a)\|_1 \geq \sqrt{\frac{|S|\log(1/\delta)}{N}}) \leq \exp(-N\log(1/\delta)/N) = \delta\,.$$

Therefore, we have $\|P(\cdot|s,a) - \hat{P}(\cdot|s,a)\|_1 \leq \sqrt{\frac{|S|\log(1/\delta)}{N}}$ with probability at least $1-\delta$. Using the union bound, we have $\max_{s,a} \|P(\cdot|s,a) - \hat{P}(\cdot|s,a)\|_1 \leq |S||A|\sqrt{\frac{|S|\log(1/\delta)}{N}}$ with probability at least $1-\delta$. □

Combining these statements, we obtain the sample complexity sufficient to learn $\hat{Q}^\pi$ that is within ϵ of Q^π for arbitrary π.

4 Model-based planning and learning

By a model of the environment we mean anything that an agent can use to predict how the environment will respond to its actions. Given a state and an action, a model produces a prediction of the resultant next state and next reward. If the model is stochastic, then there are several possible next states and next rewards, each with some probability of occurring. Some models produce a description of all possibilities and their probabilities; these we call distribution models. Other models produce just one of the possibilities, sampled according to the probabilities; these we call sample models. In either case, models can be used to simulate the environment and produce simulated experience.

The word planning is used in several different ways in different fields. We use the term to refer to any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment. State-space planning, the canonical planning approach, is viewed primarily as a search through the state space for an optimal policy or an optimal path to a goal. Actions cause transitions from state to state, and value functions are computed over states. All state-space planning methods share a common structure, a structure that is also present learning methods. There are two basic ideas: (1) all state-space planning methods involve computing value functions as a key intermediate step toward improving the policy, and (2) they compute value functions by updates or backup operations applied to simulated experience. This common structure can be diagrammed as follows:

$$\begin{tikzcd}
\text{model} \arrow[r] & \begin{array}{c}\text{Simulated}\\\text{Experience}\end{array} \arrow[r, "\text{backups}"] & \text{values} \arrow[r] & \text{policy}\,.
\end{tikzcd}$$

For example, dynamic programming methods (Algorithm 2, LN11) clearly fit this structure: they make sweeps through the space of states, generating for each state the distribution of possible transitions. Each distribution is then used to compute a backed-up value (update target) and update the state's estimated value

The heart of both learning and planning methods is the estimation of value functions by backing-up update operations. The difference is that whereas planning uses simulated experience generated by a model, learning methods use real experience generated by the environment (for which, we will be discussing for the second half of the semester). Learning methods require only experience as input, and in many cases they can be applied to simulated experience just as well as to real experience.

4.1 Dyna: Integrated planning, acting, and learning

When planning is done online, while interacting with the environment, a number of interesting issues arise. New information gained from the interaction may change the model and thereby interact with planning. It may be desirable to customize the planning process in some way to the states or decisions currently under consideration, or expected in the near future. If decision making and model learning are both computation-intensive processes, then the available computational resources may need to be divided between them. To begin exploring these issues, in this section we present Dyna-Q, a simple architecture integrating the major functions needed in an online planning agent.

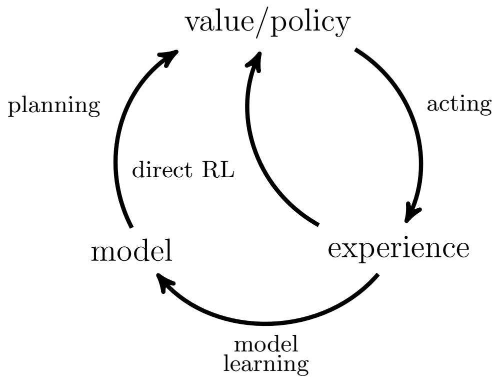

Figure 1: Relationships between experience, model, values, and policy

Within a planning agent, there are at least two roles for real experience: it can be used to improve the model (to make it more accurately match the real environment) and it can be used to directly improve the value function and policy using the kinds of planning value/policy reinforcement learning methods. The former we call model learning, and the latter we call direct reinforcement learning (direct RL). The possible relationships between experience, model, values, and policy are summarized in Diagram 1. Note how experience can improve value functions and policies either directly or indirectly via the model. It is the latter, which is sometimes called indirect reinforcement learning, that is involved in planning.

Both direct and indirect methods have advantages and disadvantages. Indirect methods often make fuller use of a limited amount of experience and thus achieve a better policy with fewer environmental interactions. On the other hand, direct methods are much simpler and are not affected by biases in the design of the model.

Dyna-Q includes all of the processes shown in the diagram above - planning, acting, model learning, and direct RL - all occurring continually. The planning method is the random-sample one-step tabular Q-planning method. The direct RL method is one-step tabular Q-learning. The model-learning method is also table-based and assumes the environment is deterministic.

The overall architecture of Dyna agents, of which the Dyna-Q algorithm is one example, is shown in Figure 2. The central column represents the basic interaction between agent and environment, giving rise to a trajectory of real experience. The arrow on the left of

Algorithm 4: Dyna-Q

```
Input: $\varepsilon$, learning rate $\alpha$, planning steps $n$
Initialize $s_0$, $Q(s,a)$, and Model$(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
for $t = 0, 1, 2, \cdots$ do
    $a_t = \varepsilon$-greedy$(s_t, Q)$
    Take action $s_t$ and observe reward, $r_t$ and state $s_{t+1}$
    $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
    Model$(s_t, a_t) = r_t, s_{t+1}$
    for $\tau = 0, 1, \cdots, n-1$ do
        Choose $s$ from previously observed states randomly
        Choose $a$ from previously taken actions in $s$ randomly
        $r, s' =$ Model$(s, a)$
        $Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_a Q(s', a) - Q(s,a))$
```

the figure represents direct reinforcement learning operating on real experience to improve the value function and the policy. On the right are model-based processes. The model is learned from real experience and gives rise to simulated experience. We use the term search control to refer to the process that selects the starting states and actions for the simulated experiences generated by the model. Finally, planning is achieved by applying reinforcement learning methods to the simulated experiences just as if they had really happened. Typically, as in Dyna-Q, the same reinforcement learning method is used both for learning from real experience and for planning from simulated experience. The reinforcement learning method is thus the "final common path" for both learning and planning. Learning and planning are deeply integrated in the sense that they share almost all the same machinery, differing only in the source of their experience.

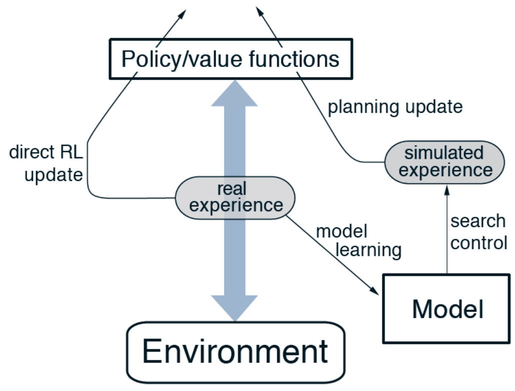

Figure 2: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

Acknowledgement

This lecture notes partially use material from *Reinforcement learning: An introduction*, *Reinforcement learning: Theory and algorithms*, and *CS234: Reinforcement learning* from Stanford.

The majority of the midterm exam of DDA4230 covers up to this lecture (LN12).