Lecture Notes:

Model-based Reinforcement Learning

Course: Reinforcement Learning,
Bachelor AI, Leiden University

Written by: Thomas Moerland

1 Type of access to the MDP dynamics

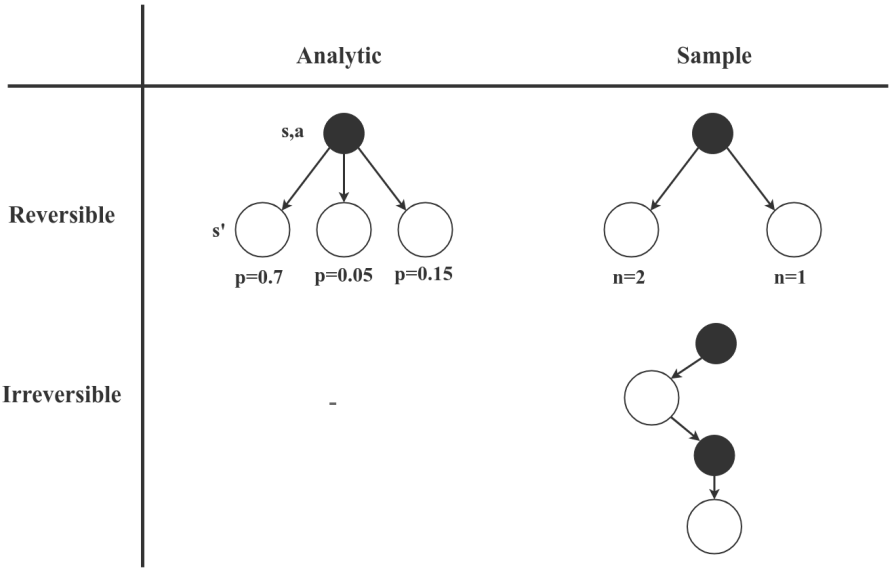
When we are given an MDP problem, the type of access we get to the MDP dynamics may differ. We need to consider: 1) whether our access is reversible or irreversible, and 2) whether we get the full distribution, or only a sample.

1. Reversible (model) versus irreversible (environment) access:
   - A *model* gives *reversible* access to the MDP dynamics $p(s'|s,a)$. We can query it at any state-action.
   - An *environment* gives *irreversible* access to the MDP dynamics $p(s'|s,a)$. After taking an action, we have to move forward to the next state, and make our next query there.
2. Distribution versus sample model:

   For models we additionally need to distinguish between:
   - *Distribution models*, which upon querying provide the full probabilities in $p(s'|s,a)$,
   - *Sample models*, which upon querying only provide a single sample from $p(s'|s,a)$.

   For environments (irreversible access to the MDP dynamics), we in practice always get a sample.

This leads to the following three types of access to the MDP dynamics:



2 Planning versus Learning

Planning and learning can both be used to solve MDP problems. The boundary between both is defined based on two characteristics:

- The type of **access to the environment dynamics**
  - Planning has reversible access to the environment (i.e., a model).
  - (Model-free) reinforcement learning has irreversible access (we need to continue to the next state).
- The **representation of the solution**:
  - Planning methods store a local solution, focusing all effort on the current state.
  - Reinforcement learning methods store a global solution, for each possible state.

The combination of these two aspects leads to the following categorization:

	Local solution	**Global** solution
Reversible dynamics (model)	Planning e.g., MCTS	Borderline/ Model-based RL e.g., DP
Irreversible dynamics (environment)	-	Model-free RL e.g., Q-learning

3 Tabular model learning

We may sample the environment to obtain transitions of the form $\langle s, a, r, s' \rangle$. Define the following counts:

$n(s, a, s')$	Number of times we observed state s' after taking action a in state s
$n(s, a)$	Number of times we took action a in state s
$n(s)$	Number of times we observed state s
$R_{\text{sum}}(s, a, s') = \sum_{i=1}^{n(s,a,s')} r_i(s, a, s')$	Total sum of rewards obtained for transition s, a, s'

We can then easily estimate our dynamics model as

$$\hat{p}(s'|s, a) = \frac{n(s, a, s')}{n(s, a)} = \frac{n(s, a, s')}{\sum_{s'} n(s, a, s')}$$

and estimate our reward model as

$$\hat{r}(s, a, s') = \frac{R_{\text{sum}}(s, a, s')}{n(s, a, s')}.$$

In practice, we need to store two arrays, $n(s, a, s')$ and $R_{\text{sum}}(s, a, s')$, both of size $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$. Algorithm 1 provides full details of tabular model estimation.

Inverse model In specific case, we may also be interested in the *reverse model*, a distribution that specifies which states and actions have previously led to a specific next state:

$$\hat{p}(s, a|s') = \frac{n(s, a, s')}{n(s')} = \frac{n(s, a, s')}{\sum_{s,a} n(s, a, s')}$$

Algorithm 1: Tabular model update pseudo-code. PS = prioritized sweeping.

Input: Maximum number of timesteps T.
Initialization: Initialize $n(s, a, s') = 0$ and $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
repeat T **times**

Observe $\langle s, a, r, s' \rangle$	`/* Observe new transition */`
$n(s, a, s') \leftarrow n(s, a, s') + 1$	`/* Update transition counts */`
$R_{sum}(s, a, s') \leftarrow R_{sum}(s, a, s') + r$	`/* Update reward sums */`
$\hat{p}(s'|s, a) \leftarrow \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}$	`/* Estimate transition function */`
$\hat{r}(s, a, s') \leftarrow \frac{R_{sum}(s,a,s')}{n(s,a,s')}$	`/* Estimate reward function */`
$\hat{p}(s, a|s') \leftarrow \frac{n(s,a,s')}{\sum_{s,a} n(s,a,s')}$	`/* Reverse model (only for PS) */`

end

4 Dyna

Dyna is a specific model-based RL algorithms, in which we 1) learn a model (Algorithm 1), and subsequently use this model to make additional one-step planning updates to our value function. Algorithm 2 shows the Dyna approach for Q-learning with ϵ-greedy exploration.

Algorithm 2: *Dyna* Q-learning with ϵ-greedy exploration.

Input: Number of planning updates K, exploration parameter $\epsilon \in (0, 1]$, learning rate $\alpha \in (0, 1]$, discount parameter $\gamma \in [0, 1]$, maximum number of timesteps T.

Initialization: Initialize $\hat{Q}(s, a) = 0$, $n(s, a, s') = 0$, $R_{sum}(s, a, s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
- **repeat** K **times**
  - $s \leftarrow$ random previously observed state — `/* State to plan on */`
  - $a \leftarrow$ previously taken action in state s — `/* Planning action */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
- **end**

end

5 Prioritized Sweeping

The main insight of prioritized sweeping is that we may update our value function more efficiently, by identifying which states are most promising for updating. When the $\hat{Q}(s,a)$ estimate of a particular state-action changes a lot, then it is likely that the state-action that lead to state s also likely need an update. A full algorithm is provided in Alg. 3

Algorithm 3: *Prioritized sweeping* (Q-learning with ϵ-greedy exploration).

Input: Number of planning updates K, exploration parameter $\epsilon \in (0,1]$, learning rate $\alpha \in (0,1]$, discount parameter $\gamma \in [0,1]$, maximum number of timesteps T, priority threshold θ.

Initialization: Initialize $\hat{Q}(s,a) = 0$, $n(s,a,s') = 0$, $R_{sum}(s,a,s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.

for $t = 1...T$ **do**
- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.1) */`
- $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)|$ — `/* Compute priority p */`
- **if p**$> \theta$ **then**
  - Insert (s, a) into PQ with priority **p** — `/* State-action needs update */`
- **end**
- **/// Start sampling from PQ to perform updates**
- **repeat** K **times**
  - $s, a \leftarrow$ pop highest priority from PQ — `/* Sample` PQ`, break when empty */`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ — `/* Update Q-table */`
  - **/// Loop over all state action that may lead to state** s
  - **for** *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
    - $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ — `/* Get reward from model */`
    - $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_a \hat{Q}(s, a) - \hat{Q}(\bar{s}, \bar{a})|$ — `/* Compute priority p */`
    - **if p**$> \theta$ **then**
      - Insert (s, a) into PQ with priority **p** — `/* State-action needs update */`
    - **end**
  - **end**
- **end**

end