Chapter 10 – Model Based RL

Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

1. Introduction

The motivation behind the model-based RL is that we want to find a policy which the agent can use to predict how environment will respond to certain actions.

Very often, the process repeated is: Generate data, fit the model on such data and build a planning strategy (LQR, Value Iteration, Policy Iteration).

2. Simulation Lemma

The simulation lemma arises from the fact that we ask ourselves this question:

“Let’s consider I have a driving policy and a self-driving car simulator. If I test such policy in the simulator and in the real world, what’s the difference in terms of policy performance?”

So, we would like to estimate the difference (in terms of performance) of the policy in the simulator and the policy in the true dynamics (in this case we choose the driving car situation).

$$\widehat{V}^{\pi}(s_0) = \mathbb{E}\left[\sum_{h=0}^{\infty} \gamma^h r(s_h, a_h) \mid \pi, \widehat{P}\right]$$

value of policy in the simulator

$$V^{\pi}(s_0) = \mathbb{E}\left[\sum_{h=0}^{\infty} \gamma^h r(s_h, a_h) \mid \pi, P\right]$$

value of policy in the true dynamics

$$\widehat{V}^{\pi}(s_0) - V^{\pi}(s_0) \quad ?$$

$$\widehat{V}^{\pi}(s_0) - V^{\pi}(s_0) = \frac{\gamma}{1-\gamma}\mathbb{E}_{s,a\sim d^{\pi}_{s_0}}\left[\mathbb{E}_{s'\sim \widehat{P}(s,a)}\widehat{V}^{\pi}(s') - \mathbb{E}_{s'\sim P(s,a)}\widehat{V}^{\pi}(s')\right]$$

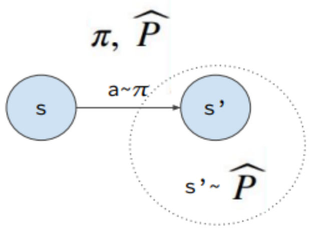
Where, of course, $\mathbb{E}_{s,a\sim d^{\pi}_{s_0}}$ is the distribution of a certain state and a certain action under the policy and the true dynamics, while $\frac{\gamma}{1-\gamma}$ arises from the usual sum of the gammas.

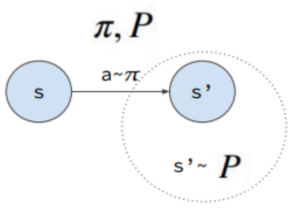
Recalling the properties of distributions, the difference between two distributions P and Q, is computed as:

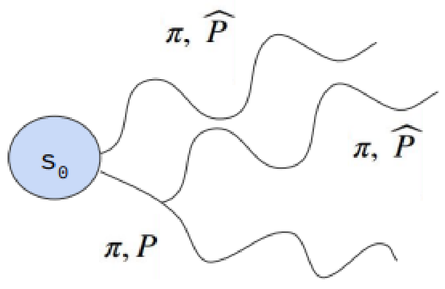
$$\mathbb{E}_{x\sim P}[f(x)] - \mathbb{E}_{x\sim Q}[f(x)]$$

So, that’s why we are able to write that specific object inside the squared parenthesis (which are the model and the true dynamics distributions difference).

This actually means that the difference we want to compute is in the next state, due to the fact that action distribution is the same.







I will take an action, look at the difference between the true dynamics and the simulated dynamics and we assume that starting from the TRUE one, we follow the learned (*simulated*) one once again.

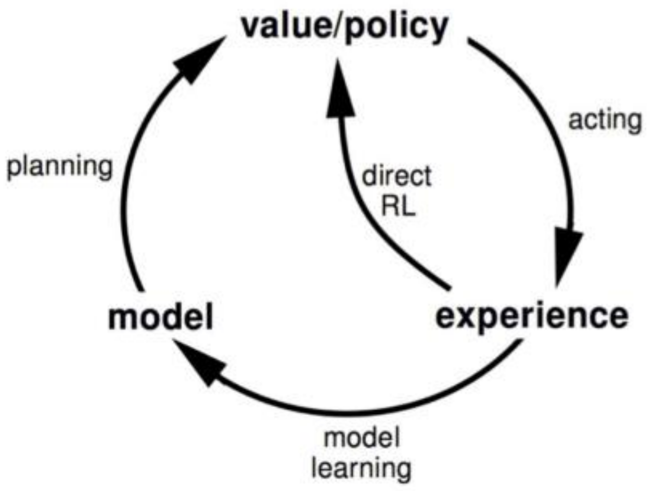
Now, through simple formulation, re-ordering stuff, applying math properties and applying this thing in a recursive way, we can compute the difference between states, which is the formula below the Chapter 2 – Simulation Lemma.

Now, since we know that the difference between two distributions of $f(x)$ is $\leq$ then the superior of $f(x)$ (which we know it's $\frac{1}{1-\gamma}$) multiplied by the two distributions, we can derive the final formula:

$$\leq \frac{1}{(1-\gamma)^2}\mathbb{E}_{s,a\sim d^{\pi}_{s_0}} \left\| \widehat{P}(\cdot\,|\,s,a) - P(\cdot\,|\,s,a) \right\|_1$$

Here, $\frac{1}{(1-\gamma)^2}$ comes from the fact that we had already $\frac{\gamma}{1-\gamma}$, but since the V is at maximum $\frac{1}{1-\gamma}$, then we rewrite the first term and multiply by the superior (which is, again, $\frac{1}{1-\gamma}$).

3. Model planning



How can we actually fit a model? We can collect N-data-points and estimate it as follows:

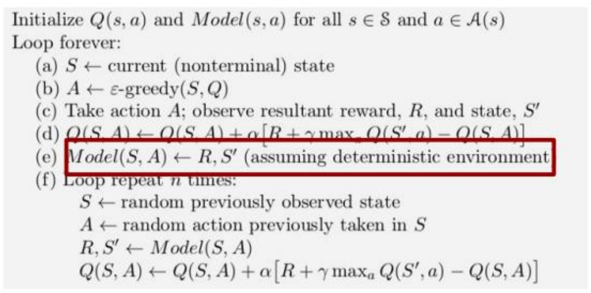
$$\widehat{P}(s'|s,a) = \frac{\sum_{i=1}^{N} \mathbf{1}\{s'_i = s'\}}{N}$$

At infinity, this converges to true P.

Once we fit the model, we have to do planning. This is easily done through Value Iteration, Policy Iteration, LQR, Q-Planning or Monte-Carlo Tree Search.

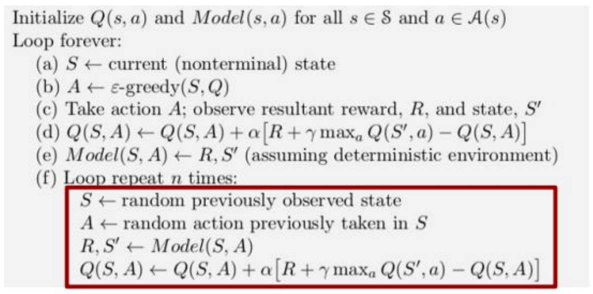
3.1 Dyna-Q

In Dyna-Q, we exactly do model fitting and model planning, by simply following the Q-Learning algorithm for planning.



```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S,Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
```

Model Fitting



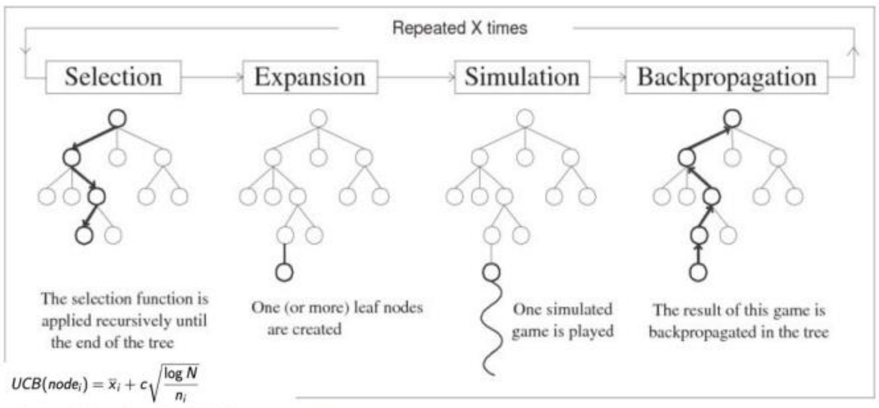
```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S,Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
```

Q-planning

3.2 Roll-out planning & MCTS

When we have to do roll-out, we can also do it through planning. Namely, we estimate only value of actions for current state and given policy (taking the highest value action).

When we deal with decision-time planning, we talk about MCTS, which is like a rollout algorithm but instead of discarding estimated values, we actually accumulate them.



Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until the end of the tree
One (or more) leaf nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

$$UCB(node_i) = \bar{x}_i + c\sqrt{\frac{\log N}{n_i}}$$

$\bar{x}_i$: mean node value; n_i: #visits of node i; N #visits parent;

In selection, we select a route until I try some actions I have never tried.

In backprop, we propagate all information up to the tree.

Multiple algorithms have been very successful with this method, such as AlphaGo, MuZero & PlaNet along with the inclusion of random perturbations, randomization in general which seemed to improve the results.