Coupling planning and learning

Corrado Possieri

Machine and Reinforcement Learning in Control Applications

Introduction

Planning: model-based.

Learning: model free.

- The hearth of both methods is computing a value function
  - look ahead to future events;
  - compute a backed-up value;
  - update target.
- Can these methods be intermixed?

Model, learning, and planning

- A model is anything the agent can use to predict the behavior of the environment

  distribution models: gives all possibilities and their probability;
  sample models: produce just one of the possibilities.

- Distribution models can be used to generate samples.

$$\text{Model} \longrightarrow \begin{array}{c}\text{Simulated}\\ \text{experience}\end{array} \xrightarrow{\text{Backup}} \text{Values} \longrightarrow \text{Policy}$$

- Learning can be used for planning.

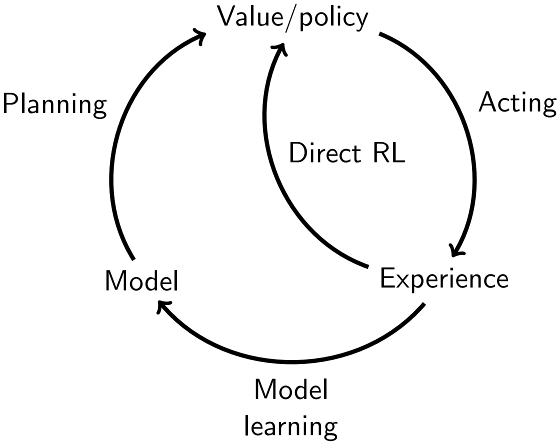
Sample-based planning

- Use the model only to generate samples.
- Sample experience from model

$$
\begin{aligned}
S_{t+1} &\sim p(S_{t+1}|S_t, A_t),\\
R_{t+1} &= r(S_t, A_t).
\end{aligned}
$$

- Apply model-free RL to samples.
- Often more efficient than planning.

Indirect reinforcement learning



Notes on indirect learning

- Indirect methods often make fuller use of a limited amount of experience.
- Direct methods are simpler and are not affected by biases.
- Model-based RL is only as good as the estimated model.
- When the model is inaccurate, planning will compute a suboptimal policy.

How to learn a model

- In deterministic environments
  - in state S_t, take action A_t;
  - observe R_{t+1} and S_{t+1};
  - $p(S_{t+1}|S_t, A_t) \leftarrow 1$, $p(s|S_t, A_t) \leftarrow 0, \forall s \neq S_{t+1}$;
  - $r(S_t, A_t) \leftarrow R_{t+1}$.

- In MDP environments
  - observing the history
    - determining $p(s'|s, a)$ is a *density estimation* problem;
    - determining $r(s, a)$ is a *regression* problem.

Table lookup model

- Assuming $\mathcal{S}$ and $\mathcal{A}$ known
  - for and MDP we need to estimate P and R
    - use empirical samples

$$\hat{P}_{s,s',a} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_{t+1}^k = s', s_t^k = s, a_t^k = a),$$

$$\hat{R}_{s,a} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} r_t^k \mathbf{1}(s_t^k = s, a_t^k = a).$$

Dyna-Q

- Use deterministic model learning
- Use one-step Q-learning as planning method
  - randomly sample from state-action pairs that have been previously experienced;
  - the model returns the last-observed next state and next reward as its prediction.
- Use one-step Q-learning as direct RL method.
- Learning and planning differ only in the source of their experience.

Dyna-Q algorithm

Dyna-Q algorithm

Input: $\alpha > 0$, $\varepsilon > 0$
Output: q_*, π_*

Initialization

$Q(s,a) \leftarrow \text{arbitrary}, \forall a \in \mathcal{A}(s), \forall s \in \mathcal{S}; Q(\text{terminal}, \cdot) \leftarrow 0$
$\text{Model}(s,a) \leftarrow \emptyset, \forall a \in \mathcal{A}(s), \forall s \in \mathcal{S}$

Loop

$S \leftarrow$ current state
$A \leftarrow \varepsilon - \text{greedy}(S, Q)$
take action A and observe R, S'
$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_a Q(S',a) - Q(S,A))$
$\text{Model}(S,A) \leftarrow S', R$
repeat
$(S,A) \leftarrow$random previously experienced pair
$R, S' \leftarrow \text{Model}(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_a Q(S',a) - Q(S,A))$
until next sample is available

Notes on Dyna-Q

- Learns much faster in deterministic environments.
- If the environment changes, it can adapt.
- However, the formerly correct policy may not reveal improvements.
- The planning process is likely to compute a suboptimal policy.
- Exploration/exploitation conflict in a planning context
  - exploration: try to improve the model;
  - exploitation: take the best possible action according to the available model.

Dyna-Q+

- Keep track for each state–action pair of how many time steps have elapsed since last visit.
- The more time that has elapsed, the more is like that the model is incorrect.
- Encourage behavior that tests long-untried actions
  - the modeled reward for a transition is r;
  - the transition has not been tried in τ time steps;
  - planning assumes that the reward is $r + \kappa\sqrt{\tau}$;
  - an alternative is to select action as that maximizing

$$Q(S_t, a) + \kappa\sqrt{\tau(S_t, a)}.$$

Prioritized sweeping

- Planning is more efficient if simulated transitions and updates are focused on particular state–action pairs.
- If simulated transitions are generated uniformly, then many wasteful updates are made.
- In general, we want to work back from any state whose value has changed
  - the predecessor pairs of those that have changed are more likely to also change;
  - prioritize the updates according to a measure of their urgency;
  - needs an inverse model.

Prioritized sweeping algorithm

Prioritized sweeping algorithm

Input: $\alpha > 0$, $\varepsilon > 0$, threshold $\theta > 0$
Output: q_*, π_*

Initialization

$Q(s, a) \leftarrow$ arbitrary, $\forall a \in \mathcal{A}(s), \forall s \in \mathcal{S}; Q(\text{terminal}, \cdot) \leftarrow 0$
Model$(s, a) \leftarrow \emptyset, \forall a \in \mathcal{A}(s), \forall s \in \mathcal{S}$
PQueue $\leftarrow \emptyset$

Loop

$S \leftarrow$ current state
$A \leftarrow \varepsilon$-greedy(S, Q)
take action A and observe R, S'
Model$(S, A) \leftarrow S', R$
$P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$
if $P > \theta$ **then**
    insert S, A into PQueue with priority P
while PQueue $\neq \emptyset$ **do**
    $(S, A) \leftarrow$ first(PQueue)
    $R, S' \leftarrow$ Model(S, A)
    $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$
    for all $\bar{S}, \bar{A}$ predicted to lead to S **do**
        $\bar{R}, \bar{S}' \leftarrow$ Model$(\bar{S}, \bar{A})$
        $\bar{P} \leftarrow |\bar{R} + \gamma \max_a Q(\bar{S}', a) - Q(\bar{S}, \bar{A})|$
        if $\bar{P} > \theta$ **then**
            insert $\bar{S}, \bar{A}$ into PQueue with priority $\bar{P}$

Trajectory sampling

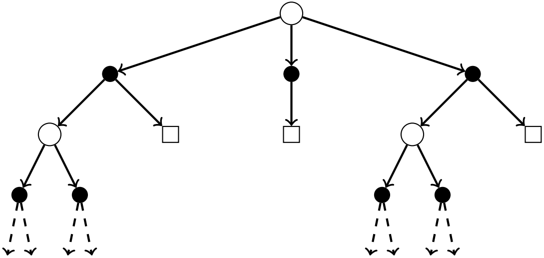
- Distribute updates according to the on-policy distribution
  - distribution observed when following the current policy;
  - simulate individual trajectories and perform updates at the state encountered along the way.
- States actually visited are updated more often.
- Uninteresting parts of the space are ignored.
- This is the same that happens in real time DP
  - can find a policy that is optimal on the relevant states without visiting every state infinitely often.

Planning at decision time

- Planning executed at current state.
- The values and policy are specific to the current state.
- The values and policy created are discarded after being used.
- Useful in applications in which fast responses are not required

Heuristic search

- Go through the tree of possible continuations.
- Use a model of the sub-MDP starting from *now*.
- Focused on state/actions that immediately follow.
- Build a search tree with S_t at its root.



Rollout algorithms

- Heuristic search guided by MC simulation.
- Simulate episodes from now with the model.
- Average returns of simulated trajectories that start with each action and then follow rollout policy
  - simulate K episodes following first action a and then policy π

$$S_t, a, R^k_{t+1}, S^k_{t+1}, A^k_{t+1}, R^k_{t+2}, \ldots, R^k_T, S^k_T;$$

  - evaluate actions by mean return

$$Q(S_t, a) = \frac{1}{K} \sum_{k=1}^{K} G^k_t;$$

  - take action that maximize Q

$$A_t = \arg\max_a Q(S_t, a).$$

Monte Carlo tree search

- Use rollout method by modifying policy.
- Record the values of Q in the search tree.
- In the tree, we pick actions to maximize Q (*e.g.*, ε-greedy).
- Outside the tree use a default policy.
- MC control applied to simulated experience.
- Expand the part of the tree that looks promising.
- MC can be substituted by TD.

Monte Carlo tree search logic