Chapter 9
Integrating Learning and Planning

Huaqing Zhang, Ruitong Huang, and Shanghang Zhang

Abstract In this chapter, reinforcement learning is analyzed from the perspective of learning and planning. We initially introduce the concepts of model and model-based methods, with the highlight of advantages on model planning. In order to include the benefits of both model-based and model-free methods, we present the integration architecture combining learning and planning, with detailed illustration on Dyna-Q algorithm. Finally, for the integration of learning and planning, the simulation-based search applications are analyzed.

Keywords Model-based · Model-free · Dyna · Monte Carlo tree search · Temporal difference (TD) search

9.1 Introduction

In reinforcement learning, the agent is allowed to interact with the environment. Information collected during each round of interaction is regarded as the agent's experience and assists the agent to improve its policy. Generally, we refer learning as the policy improvement process based on actual experience during the interactions. As the simplest learning protocol, the direct policy learning is elaborated in Fig. 9.1. The agent initially takes actions in the environment following current policy. Based on the action and current state of the agent, the environment provides feedback of rewards, which let agent evaluate the performance of current policy and explore the policy improvement opportunities. However, direct policy learning is based on the experience of every single step of the action. Due to the uncertainty and randomness

H. Zhang
Google LLC, Mountain View, CA, USA

R. Huang
Borealis AI, Toronto, ON, Canada

S. Zhang (✉)
University of California, Berkeley, CA, USA

H. Dong et al. (eds.), *Deep Reinforcement Learning*,
https://doi.org/10.1007/978-981-15-4095-0_9

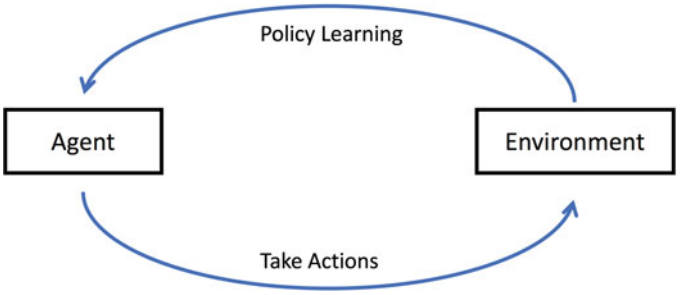

Fig. 9.1 Direct policy learning

of the environment, each experience may have large variance, which affects the learning speed and performance.

In order to improve the learning efficiency, for each episode of policy learning, it is beneficial to accumulate multiple rounds of interactions as the agent's experience. The interactions can be collected by performing roll-outs in the environment, which refer to forming a specific state-action-reward trajectory in the environment based on the current state and policy. In general model-free methods, the agent performs online roll-out in actual environment and combines the interactions for policy learning.

However, when the roll-out is applied online, generating experience with the environment is costly. In industrial scenarios, for example, some states may indicate system failure or engine explosion, which is dangerous to explore in actual situation for policy learning. Furthermore, the roll-out in the real environment has to be done sequentially. The failure on parallel computing results in low sample efficiency and slow learning speed. Therefore, for some scenarios, it is desirable to propose simulated environment and replace the real environment to generate experiences. We regard the roll-out from the simulated environment as planning, which can efficiently generate abundant simulated experiences with parallel computing for policy learning. In order to implement effective simulated environment for planning, the model-based methods are put forward and introduced in Sect. 9.2.

9.2 Model-Based Method

In order to apply planning, the concept of model is implemented between the agent and the environment (Kaiser et al. 2019). As shown in Fig. 9.2, when the agent stays in state S_t and takes actions A_t, the environment provides feedback rewards R_{t+1} and next state S_{t+1} information for the model. Based on the information collected from the experience between the agent and environment, the model representing the relations between S_{t+1} and (S_t, A_t) is called the transition model. The model representing the relations between R_{t+1} and (S_t, A_t) is called the reward model. When

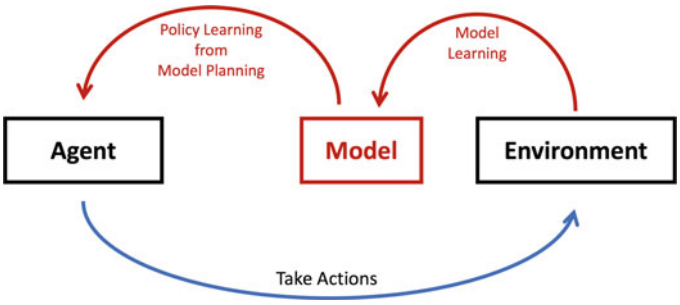

Fig. 9.2 Model-based reinforcement learning methods

the states are not fully represented by the observations, there is also the observation model $\mathcal{M}(O_t|S_t)$ and representation model $\mathcal{M}(S_{t+1}|S_t, A_t, O_{t+1})$ (Hafner et al. 2019), where the O_t is the corresponding observation of the state at time step t. For example, the images captured for representing the object motion are observations, as a function of the underlying dynamics state of the object. However, we will assume the state is fully observed and mainly focus on the transition model and the reward model in the following. The models can be formulated with two functions or distributions $\mathcal{F}_s$ and $\mathcal{F}_r$, to fit the relations, i.e.,

$$S_{t+1} \sim \mathcal{F}_s\left(S_t, A_t\right), \tag{9.1}$$

$$R_{t+1} = \mathcal{F}_r\left(S_t, A_t\right). \tag{9.2}$$

The model learning is a supervised regression learning process, which aims to build a visualized environment and function the same as the original one. Therefore, the model can be applied for planning by the agent with the insights on the true environment and help it perform policy learning.

For different application scenarios, the relations for model learning and policy learning may vary.

- **Direct Learning**: If the agent has already interacted with the environment multiple times following rule-based or expert knowledge, the collected experience can be directly applied for model learning first. When model is well learnt, the agent can regard the model as the environment and interact with it for policy learning.
- **Iterative Learning**: If the model is not well learnt, the model learning and policy learning can be performed iteratively. Based on limited experience from the interactions between the agent and the environment, some but limited insights about the environment can be learnt by the model. With the planning based on weakly learnt simulated model, the agent improves its policy a little and takes actions in real environment so as to provide further experience for model learning. With the number of iterations increasing, both the model learning

and policy learning gradually converge to the optimal results. Therefore, model learning and policy learning are able to assist each other and learn efficiently.

Accordingly, the model-based reinforcement learning establishes a model by learning from the real environment and performs planning with the agent for its policy learning. The advantages for the model learning can be depicted as follows.

- As the planning can be done with the interactions between the agent and the model, the agent does not need to take actions in the real environment for exploration and policy learning. Thus, for some environment that are costly on taking online actions, the model-based method can reduce the training time and guarantee the safety issues during the policy learning. For example, the real-world robot learning tasks require the robot manipulation in practice, like in Qt-opt (Kalashnikov et al. 2018) method, seven robots are collecting real-world samples days and nights for achieving the grasping task. A simulated environment (either learnt or manually engineered) can save large amounts of time and wear and tear of robots.
- When the policy learning is applied between the agent and the simulated model, the learning can be done in parallel. In the distributed system where there exist multiple learners contributing to the policy learning, each learner can interact with one model simulated from the environment. Therefore, the planning of multiple corresponding models can be independent with each other and does not affect the current state of the real environment. The parallelization on policy learning can improve the efficiency and scalability of the learning problem.

Nevertheless, considering the structure of model-based reinforcement learning, the weakness also exists.

- In model-based reinforcement learning, the performance of model learning will affect the policy learning results. For the complicated and dynamic environment scenarios, if the learnt model is unable to simulate the insights of the real environment, the agent actually interacts with the planning of a wrong or inaccurate model, which will further increase the error on the learnt policy.
- If there are some updates of adjustments on the environment, it takes several iterations for the model to learn the changes and further takes time for the agent to adjust the policy. Accordingly, the agent may have long delay to response and adapt to the changes on the online environment, which is not suitable for some applications with real-time requirements.

9.3 Integrated Architectures

Considering the pros and cons of model-free and model-based reinforcement learning methods, it is promising to combine the advantages of both model-based and model-free methods by integrating the learning and planning procedures. For

different applications scenarios, the architectures integrating learning and planning are different.

Generally, in model-free method, the agent learns policies directly from real experience with the environment. No planning is adopted for policy improvement. In basic model-based method, model learning is firstly applied by the interactions between the agent and the environment. Based on the learnt model, the planning is applied and the policy can be iteratively learnt from the planning experience.

Since the model is implemented between the agent and the environment, for the policy learning of the agent, the experience can be classified into two categories.

- **Real experience**: In real experience, information is sampled directly from the interactions between the agent and the environment. Generally the real experience reflects the correct features of the environment, but it is costly and the states the hard to manually change or recover.
- **Simulated experience**: The simulated experience is generated by the planning of the model. The experience may not precisely provide the true features from the real environment, but the model is easy to manuscript and the model learning tries to minimizes the gap of mistakes.

For policy learning, if both real experience and simulated experience can be combined and considered simultaneously, we can take the advantages of both model-free and basic model-based methods, so as to improve the learning efficiency and accuracy. Therefore, the Dyna architecture is put forward (Sutton 1991). As shown in Fig. 9.3, based on the basic model-based method, during the policy learning, the agent updates the policy not only from the simulated experience with the learnt model, but considering the real experience with the real environment. Thus, in policy learning, the simulated experience can guarantee the quantity required from learning to reduce the variance of learning the features of the environment, while the real experience shows the correct features and dynamic changes from the environment, so as to reduce the bias through learning from the environment.

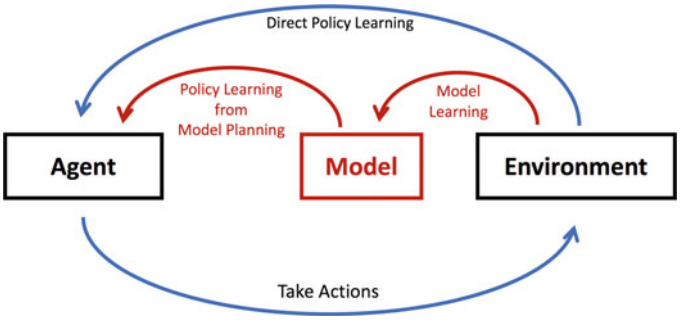

Fig. 9.3 Dyna architecture

Based on the architecture, the Dyna-Q algorithm is put forward and depicted in Algorithm 1. In the Dyna-Q learning, a Q table is established and maintained to instruct the actions of the agent. For each episode of learning, the Q table is learnt and updated from one-step action of the agent in the real environment. Moreover, the simulated model also learns from the real experience, and applies planning to generate n simulated experience for future learning. Accordingly, with the number of episode increasing, the Q table is learnt and converges to the correct results.

Algorithm 1 Dyna-Q

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
Do forever:
(a) $s \leftarrow$ current (non-terminal) state
(b) $a \leftarrow \epsilon$-greedy(s, Q)
(c) Execute action a; Observe resultant reward r, Get next state s'
(d) $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
(e) $Model(s, a) \leftarrow r, s'$
(f) Repeat n times:
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in s
$r, s' \leftarrow Model(s, a)$ random action previously taken in S
$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$

9.4 Simulation-Based Search

In this section, we put more focus on the planning and introduce simulation-based search methods, which initialize at current state and take samples to roll out the trajectory. Accordingly, simulation-based search methods are generally the forward search paradigm using sample-based planning. The concept of forward search and sampling are further shown as follows.

- **Forward search**: Considering the planning of the problem, current state has more significance compared with all other states in Markov decision process (MDP). It is beneficial to view the MDP with finite choices in another perspective, which is a tree structure and the root is the current state. As shown in the Fig. 9.4, the forward search algorithm selects best action from the current state and look ahead for future planning in the branches of the tree structure.
- **Sampling**: When the planning is applied based on the MDP, from the current state, there may exist multiple options for the next states. Thus sampling is required in planning to randomly select the next state and continue to roll out the forward search. The randomness on the next state selection may follow some probabilities or distributions, which reflects the simulation policy adopted by the agent.

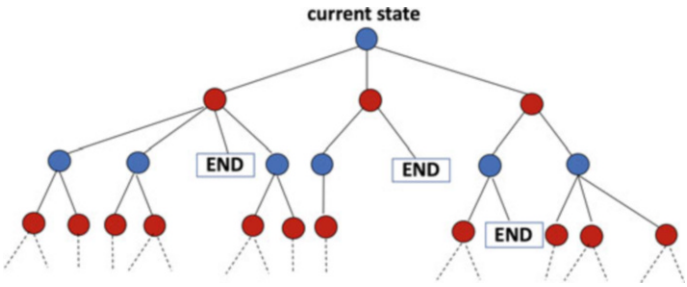

Fig. 9.4 Forward search

During the simulation-based search, the simulation policy is considered to instruct the planning direction. The simulation policy relates to the learning policy and assists on efficient planning to reflect the correct evaluations of the current policy of the agent.

For the rest of the section, we introduce different kinds of simulation-based search methods and combine with learning for problem-solving.

9.4.1 Simple Monte Carlo Search

When both the model $\mathcal{M}$ and policy π are fixed and initially provided, the simple Monte Carlo search can be applied to evaluate the performance of the action and update the learnt policy based on the experience. As shown in Algorithm 2, for each action a, $a \in \mathcal{A}$ applied on current state S_t, following the simulated policy π, K trajectories can be formulated and the total revenue from each trajectory is denoted as G_t^k. Based on the recorded trajectories, the performance by taking action A_t is evaluated as $Q(S_t, A_t)$. And based on the Q value of all actions, the optimal one is selected and the next state is determined.

Algorithm 2 Simple Monte Carlo search

Provided the model $\mathcal{M}$ and simulation policy π
for each action $a \in \mathcal{A}$ **do**
  for each episode $k \in \{1, 2, \ldots, K\}$ **do**
    Following the model $\mathcal{M}$ and simulation policy π, roll out in the environment started from current state S_t
    Record the trajectory as $\{S_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, R_{t+2}^k, \ldots S_T^k\}$
  end for
  Evaluate actions by mean return. $Q(S_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t^k$
end for
The learnt policy is to select current action with maximum Q value $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$

9.4.2 Monte Carlo Tree Search

One clear drawback of simple Monte Carlo search is that the simulation policy π is prefixed, and, thus, never leverages the new information collected during planning. Monte Carlo Tree Search (MCTS) (Browne et al. 2012) is designed to overcome this drawback. In particular, MCTS proposes to maintain a search tree for keeping the collected information and improve the simulation policy gradually.

As shown in Algorithm 3, after sampling a trajectory from the current state S_t, MCTS updates the Q value for all the visited state and action pair (s, a) along the trajectory, similarly by the average reward of all the trajectories starting from (s, a). Then the simulation policy π on the nodes in the search tree is updated accordingly based on the new Q value in the tree. One example of updating π can be similar to the Q-learning, where at any node (state) s, π picks the optimal action based on the current Q with ϵ uniform exploration. When the simulation reaches a new state that is currently not in the tree, π sticks to the default policy like uniform exploration. The first new state in the trajectory will then be added into the search tree.[1] Such evaluation and policy improvement is repeated for every simulation until the simulation budget is reached. Lastly, the agent selects the action with maximum Q value at the current state S_t.

Algorithm 3 Monte Carlo tree search

Provided the model $\mathcal{M}$
Initialize simulation policy π
for each action $a \in \mathcal{A}$ **do**
  for each episode $k \in \{1, 2, \ldots, K\}$ **do**
    Following the model $\mathcal{M}$ and simulation policy π, roll out in the environment started from current state S_t
    Record the trajectory as $\{S_t, a, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \ldots S_T\}$
    Update the Q value of every $(S_i, A_i), i = t, \ldots, T$ by mean return starting from (S_i, A_i) with $A_t = a$
    Update the simulation policy π according to the current Q values
  end for
end for
Output the action with maximum Q value at the current state, $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$

9.4.3 TD Search

Apart from the Monte Carlo (MC) search methods, the temporal difference (TD) search can also be considered (Silver et al. 2012). Compared with the MC methods, the TD search does not require to roll out a trajectory to evaluate and update the

[1] Another option can be adding all the new nodes in the trajectory into the search tree.

current policy. Instead, in TD search, for each step of simulation, the policy is updated and instructs to select the action for the next state.

The TD search is applied in Dyna-2 algorithm (Silver et al. 2008), as depicted in Algorithm 4. In Dyna-2, the agent stores two sets of weights, denoted as long-term memory and short-term memory. Applying TD learning, the weights of short-term memory is updated with the simulated experience. The learnt $\overline{Q}$ with the weights of short-term memory further assists the agent to take actions in real environment. The higher-level TD learning is applied to update the weights of long-term memory. Finally, the learnt Q with weights θ is the learnt policy for the agent.

Algorithm 4 Dyna-2

function LEARNING
  Initialize $\mathcal{F}_s$ and $\mathcal{F}_r$
  $\theta \leftarrow 0$ # Initialize the weights of long-term memory
  loop
    $s \leftarrow S_0$
    $\overline{\theta} \leftarrow 0$ # Initialize the weights of short-term memory
    $z \leftarrow 0$ # Initialize eligibility trace
    SEARCH(s)
    $a \leftarrow \pi(s; \overline{Q})$ # Choose action based on polity related with $\overline{Q}$
    while s is not terminal **do**
      Execute a, observe reward r and next state s'
      $(\mathcal{F}_s, \mathcal{F}_r) \leftarrow UpdateModel(s, a, r, s')$
      SEARCH(s')
      $a' \leftarrow \pi(s'; \overline{Q})$ # Choose action applied in the next state s'
      $\delta \leftarrow r + Q(s', a') - Q(s, a)$ # Calculate TD-error
      $\theta \leftarrow \theta + \alpha(s, a)\delta z$ # Update weights of long-term memory
      $z \leftarrow \lambda z + \phi$ # Update eligibility trace
      $s \leftarrow s', a \leftarrow a'$
    end while
  end loop
end function

function SEARCH(s)
  while time available **do**
    $\overline{z} \leftarrow 0$ # Clear eligibility trace
    $a \leftarrow \overline{\pi}(s; \overline{Q})$ # Choose action based on polity related with $\overline{Q}$
    while s is not terminal **do**
      $s' \leftarrow \mathcal{F}_s(s, a)$ # Sample transition
      $r \leftarrow \mathcal{F}_r(s, a)$ # Sample reward
      $a' \leftarrow \overline{\pi}(s'; \overline{Q})$
      $\overline{\delta} \leftarrow R + \overline{Q}(s', a') - \overline{Q}(s, a)$ # Calculate TD-error
      $\overline{\theta} \leftarrow \overline{\theta} + \overline{\alpha}(s, a)\overline{\delta}\overline{z}$ # Update weights of short-term memory
      $\overline{z} \leftarrow \overline{\lambda}\overline{z} + \overline{\phi}$ # Update eligibility trance in short-term memory
      $s \leftarrow s', a \leftarrow a'$
    end while
  end while
end function

Compared with the MC methods, as the policy is updated for each step, TD search generally is more efficient. However, due to the frequent update, the search trends to reduce the variance but increase the bias for problem-solving.

References

Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intel AI Games 4(1):1–43

Hafner D, Lillicrap T, Ba J, Norouzi M (2019) Dream to control: learning behaviors by latent imagination. Preprint. arXiv:191201603

Kaiser L, Babaeizadeh M, Milos P, Osinski B, Campbell RH, Czechowski K, Erhan D, Finn C, Kozakowski P, Levine S, Mohiuddin A, Sepassi R, Tucker G, Michalewski H (2019) Model-based reinforcement learning for Atari. Preprint. arXiv:1903.00374

Kalashnikov D, Irpan A, Pastor P, Ibarz J, Herzog A, Jang E, Quillen D, Holly E, Kalakrishnan M, Vanhoucke V, et al (2018) Qt-opt: scalable deep reinforcement learning for vision-based robotic manipulation. Preprint. arXiv:180610293

Silver D, Sutton RS, Müller M (2008) Sample-based learning and search with permanent and transient memories. In: Proceedings of the 25th international conference on machine learning. ACM, New York, pp 968–975

Silver D, Sutton RS, Müller M (2012) Temporal-difference search in computer go. Mach Learn 87(2):183–219

Sutton RS (1991) Dyna, an integrated architecture for learning, planning, and reacting. ACM Sigart Bull 2(4):160–163