Ch 8. Planning and Learning with Tabular Methods

Outline

Intro

- **Model-based methods** : Dynamic Programming, heuristic search => Rely on **Planning(계획)**
- **Model-free methods** : Monte Carlo, Temporal Difference search => Rely on **Learning (학습)**

These have similarities :

- Both revolve around the computation of value functions(가치 함수를 계산)
- Both look ahead of future events, compute a backed-up value and use it as an update target for an approximate value function(미래 사건을 내다보고, 보강된 가치를 계산)

How model-free and model-based approaches can be combined

1

Models and Planning

Models and Planning

Models

- **Models** is anything the agent can use to predict the environment’s behavior
- Given a state and an action, a model predicts the next state and the next reward.
  - Output the whole probability distribution over the next states and rewards: **distribution models**
  - Produce only one of the possibility (the one with highest probability): **sample models**
- A model can be used to simulate the environment and produce simulated experience

Models and Planning

Planning

- **Planning**: takes a model as input and produces or improves a policy for interacting with the modeled environment.

- **State-space planning**: search through the state space for an optimal policy. Actions cause transitions from state to state, and value functions are computed over states
- **Plan-space planning**: search through the space of plans. Operators transform one plan into another (evolutionary methods, partial-order planning)

Models and Planning

Planning(계획) & Learning(학습)

- **Planning**: uses simulated experience generated by the model
- **Learning**: uses real experience generated by the environment
- In many cases, a learning algorithm can be substituted for the key update step of a planning method, e.g., Q-learning

Random-sample one-step tabular Q-planning

Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

2

Dyna : Integrated Planning, Acting, and Learning

Dyna: Integrating Planning, Acting, and Learning

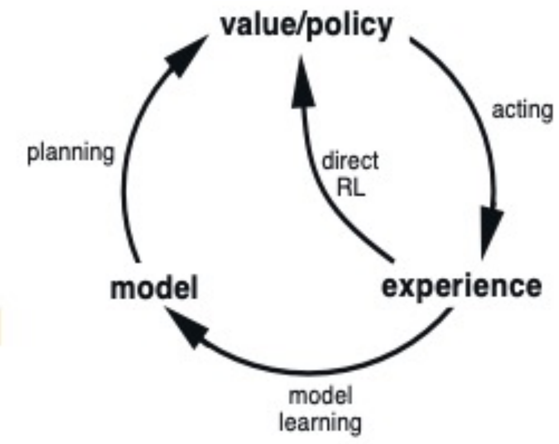
When planning is done online, while interacting with the environment, a number of issues arise.

- New information may change the model (and thus change planning)
- How to divide the computing resources between decision making and model learning?

Dyna-Q: simple architecture integrating the major functions of an online planning agent

Dyna: Integrating Planning, Acting, and Learning

What do we do with real experience



Model-learning: improve the model (to make it more accurately match the environment)

Direct RL: improve the value functions and policy used in the reinforcement learning programs we know

Indirect RL: improve the value functions and policy via the model

Both direct and indirect method have advantages and disadvantages:
- Indirect methods often make fuller use of limited amount of experience
- Direct methods are often simpler and not affected by bias in the design of the model

Dyna: Integrating Planning, Acting, and Learning

Dyna-Q

Dyna-Q includes all of the processes: planning, acting, direct RL and model-learning.

- **Planning**: random sample one-step tabular Q-learning
- **Direct RL**: method is one-step tabular Q-learning
- **Model learning**: assumes the environment is deterministic and

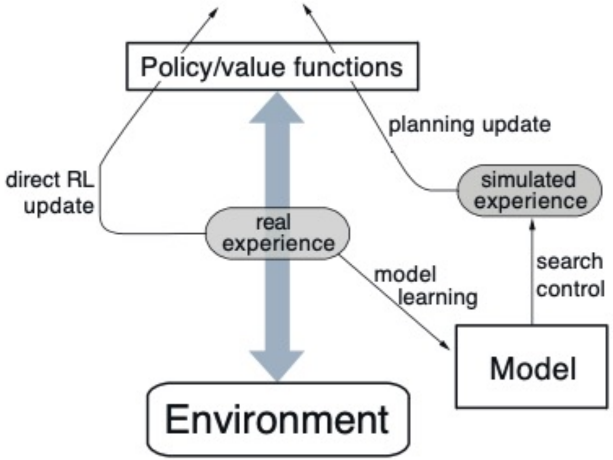
  ->After each transition $S_t, A_t \rightarrow S_{t+1}, R_{t+1}$ the model records in its table entry for S_t, A_t the prediction that S_{t+1}, R_{t+1} will follow.

  ->If the model is queries with a state-action pair it has seen before, it simply returns the last S_{t+1}, R_{t+1} experienced.

  ->During planning, the Q-learning algorithm randomly samples only from state-actions the model has seen

Dyna: Integrating Planning, Acting, and Learning

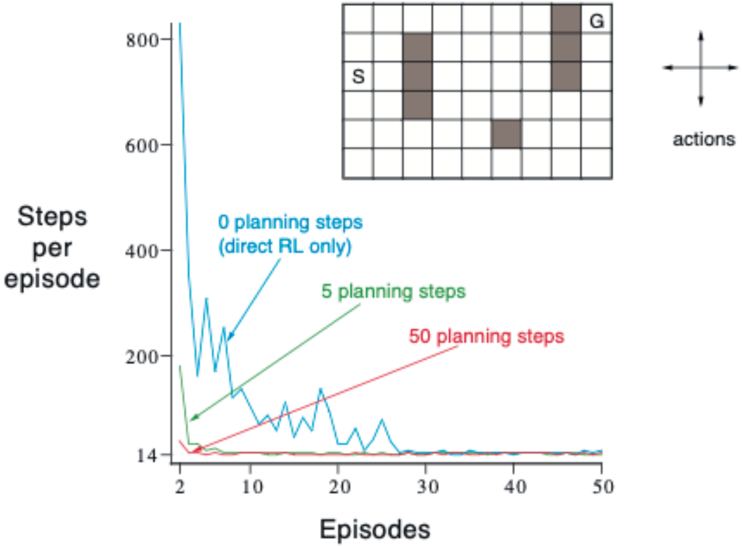
Architecture of Dyna-Q



- **Search control**: process that selects starting states and actions from the simulated experiences
- Planning is achieved by applying RL methods to simulated experience

Dyna: Integrating Planning, Acting, and Learning

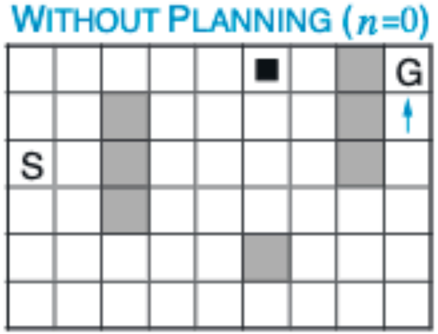
Example : Dyna Maze

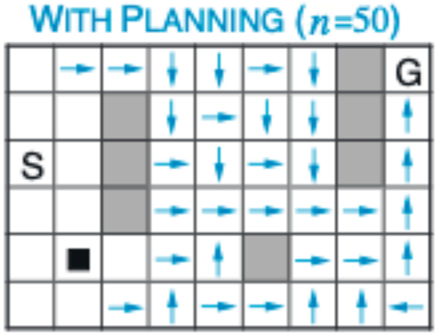


- The task is to travel from S to G as quickly as possible

Dyna: Integrating Planning, Acting, and Learning

Non-planning vs planning : Dyna Maze





- The black square is the agent
- The arrows denote the greedy action for the state (no arrow: all actions have equal value)
- These are the policies learned by a non-planning (left) and a planning (right) agent halfway through the second episode.

Dyna: Integrating Planning, Acting, and Learning

What is going on?

- Intermixing planning and acting: make them both proceed as fast as they can.
- Agent is always reactive, responding with the latest sensory information
- Planning is always done in the background, so is model-learning
- As new information is gained, the model is updated.
- As the model changes, the planning process will gradually compute a different way of behaving to match the newer model.

3

When the Model Is Wrong

When the Model is Wrong

Models may be incorrect because:

- the environment is stochastic and only a limited amount of experience is available
- the model has learned using a function approximation that failed to generalize
- the environment has changed and new behavior has not been observed yet

When the model is incorrect, the planning process is likely to compute a suboptimal policy

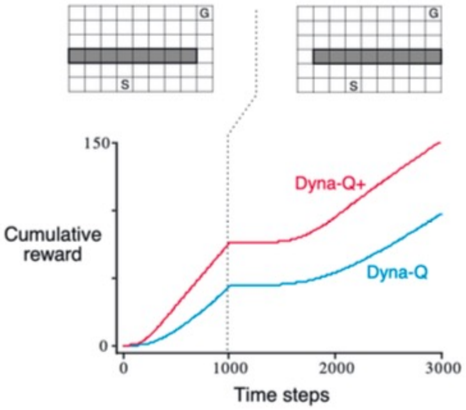
In some cases, following the suboptimal policy computed over a wrong model quickly leads to the discovery and correction of the modeling error.

- This can happen when the model is optimistic: predicting greater reward or better state transitions than are actually possible
- The planned policy attempts to exploit these opportunities.

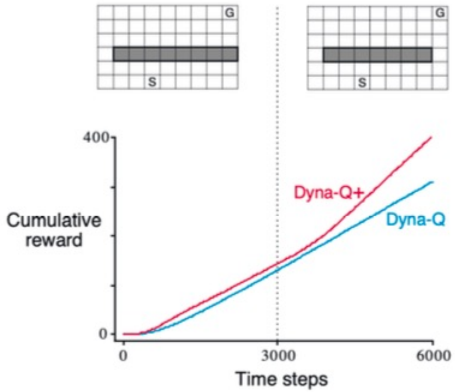
Dyna: Integrating Planning, Acting, and Learning

Example : Blocking Maze and Shortcut Maze

Blocking Maze



Shortcut Maze



Blocking Maze : When the environment changes(the barrier shifts a little)
Shortcut Maze : When the environment changes to offer a better alternative without changing the availability of the learned path

Dyna: Integrating Planning, Acting, and Learning

Dyna-Q+

Exploration-exploitation tradeoff:

- Want the agent to explore to find changes in the environment
- But not so much that the performance is greatly degraded.

Dyna-Q+: Simple heuristic approach

- Keeps track for each state-action pair of **how many time steps have elapsed** since last tried in a real interaction
- A special **bonus reward** is added to simulated experience involving these actions.
- If the transition has not been tried for over τ steps, then planning updates are done as if the transition produced a reward of $R + k\sqrt{\tau}$ for some small κ.

4 Prioritized Sweeping

Priortized Sweeping

Can we do better than uniform sampling?

- For simulated transitions, state-action pair selected uniformly at random from all previously experience… Usually not the best.
- Focus on particular state-action pairs!
- Prioritize state-action pairs according to a measure of their urgency

Backward-focusing of planning computations: in general, we want to work back not just from goal states but from any state whose value have changed.

Prioritized sweeping: A queue is maintained for every state-action pair whose value would change if updated, prioritized by the size of the change.

5 Trajectory Sampling

Trajectory Sampling

1. Exhaustive sweep: classical approach from DP: perform sweeps through the entire state space, updating each state once per sweep

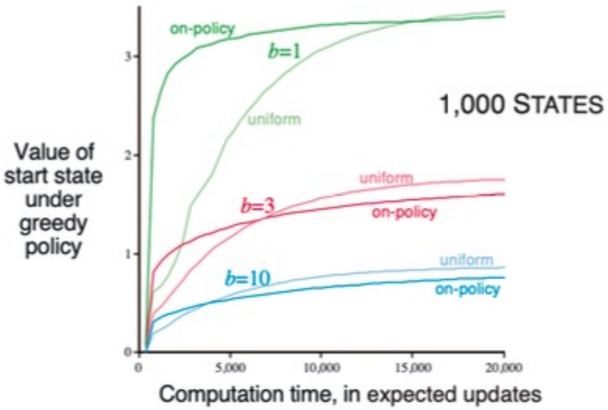
2. Sample from the state space according to some distribution

- Uniform sample: Dyna-Q
- **Trajectory sampling**: On-policy distribution following the current policy
- Sample state transitions and rewards are given by the model, and sample actions are given by the current policy.
- Simulate explicit individual trajectories and perform update at the state encountered along the way → trajectory sampling

Trajectory Sampling

On-Policy Distribution

- Good: it causes vast, uninteresting parts of the space to be ignored
- Bad: it may update the same parts over and over



- b: branching factor, results averaged over 200 sample tasks with 1000 states

6

Real-time Dynamic Programming

Real-time Dynamic Programming

RTDP: on-policy trajectory-sampling version of value-iteration

Updates the values of states visited in actual or simulated trajectories by means of expected tabular value-iteration updates.

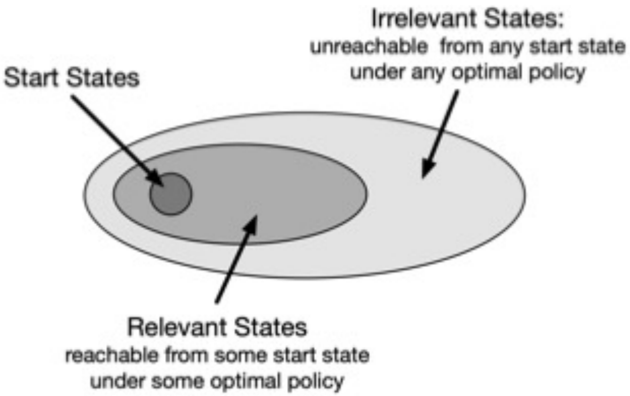
Example of an **asynchronous** DP algorithm

- Asynchronous algorithms do not do full sweeps, they update state values in any order.
- In RTDP, update order is dictated by the order states are visited in real or simulated trajectories

Trajectory Sampling

RTDP Policy Evaluation

- Trajectories start from a designated set of start states
- On-policy trajectory sampling allows to focus on useful states



Trajectory Sampling

RTDP Policy Improvement

- There might be states that cannot be reached by any of the optimal policies from any of the start states.
- No need to specify optimal actions for those irrelevant states
- Only need an optimal partial policy!

But…

- Finding such an optimal partial policy can require visiting all state-action pairs, even those turning out to be irrelevant in the end

However,

- or certain problems, RTDP is guaranteed to find an optimal policy **without visiting** every state infinitely often, **or even without visiting** some states at all.

7

Planning at Decision Time

Planning at Decision Time

Background planning

- In Dyna (or DP), planning gradually improves a policy or value function on the basis of simulated experience from a model
- Well before an action is selected for any current state S_t planning has played a part in improving the way to select the action for many states.
- Planning here is not focused on the current state

Decision-time planning

- Begin and complete planning after visiting each state S_t to produce A_t
- Planning when only state-values are available
- At each state we select an action based on the values of model-predicted next states for each action
- Can also look much deeper than one step ahead
- Planning focuses on a particular state.

Planning at Decision Time

Decision-time planning

Even when planning is done at decision time,

- can still view it as proceeding from simulated experience to updates and values, and ultimately to a policy.
- It's just that now the values and policy are specific to the current state.

In general, we may want to do a bit of both:

- Focus planning on the current state
- store the results of planning so as to help when we return to the same state after.

Decision-time planning is best when fast responses are not required.

8 Heuristic Search

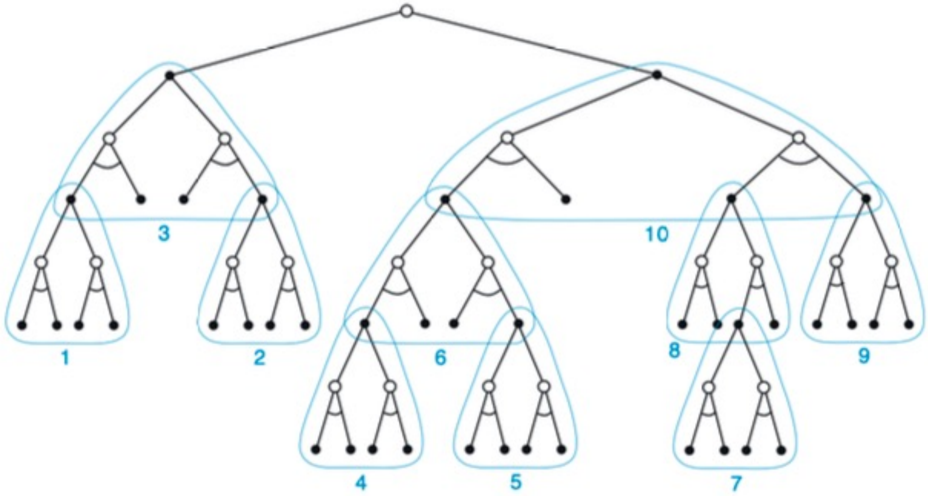
Heuristic Search

- For each state encountered, a large tree of possible continuations is considered.
- The approximate value function is applied to the leaf nodes and then backed up toward the current state at the root.
- The best is chosen as the current action.
- All other backed-up values are discarded.

An extension of the idea of a greedy policy beyond a single step to obtain better action selections

- Effectiveness of heuristic search is due to its search tree being tightly **focused on the states and actions that might immediately follow the current state**.
- Tradeoff here is that deeper search leads to more computation

Heuristic Search



- Heuristic search as a sequence of one-step updates (in blue), backing up values from leaf to root. The ordering here is selective depth-first search.

9 Rollout Algorithm

Rollout Algorithm

- Decision-time planning algorithms based on Monte Carlo control applied to simulated trajectories.
- Trajectories all begin at the current environment state.
- Estimate action values for a given policy by averaging the returns of many simulated trajectories

- Unlike MC algorithm, the goal here is not to estimate a complete optimal action-value function.
- Only care about the current state and one given policy, called the **rollout policy**.
- The aim of a rollout algorithm is to **improve upon the rollout policy, not to find an optimal policy**

Rollout Algorithm

Rollout Algorithms in Practice

- May run many trials in parallel on separate processes because Monte Carlo trials are independent of one another
- Can truncate the trajectories short of complete episodes and correct the truncated returns by the means of a stored evaluation function
- Monitor the Monte Carlo simulations and prune away actions that are unlikely to be the best

10

Monte Carlo Tree Search

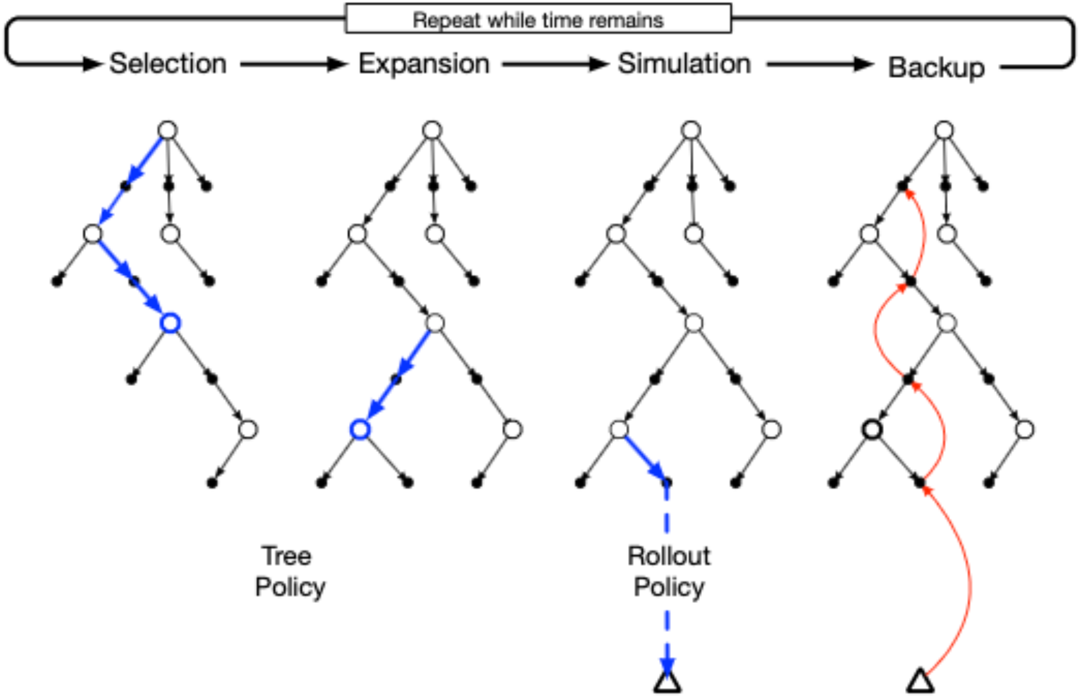
Monte Carlo Tree Search(MCTS)

- Decision-time planning
- Essentially a rollout algorithm, but enhanced with a means of **accumulating** value estimates from the Monte Carlo simulations
- Successively direct simulations towards more highly-rewarding trajectories
- **Used in AlphaGo!**

Monte Carlo Tree Search(MCTS)

- MCTS is executed after encountering each new state to select the action
- Each execution is an iterative process that simulates many trajectories starting from the current state
- **The core idea** of MCTS is to successively focus multiple simulations starting at the current state by extending the initial portions of trajectories that have received high evaluations from earlier simulations
- Actions in the simulated trajectories are generated using a simple policy, also called a rollout policy

MCTS Illustration



Thank You