Reinforcement Learning

Model-Based RL and Planning

Alberto Maria Metelli

14th February 2024

Book References

Richard S. Sutton, Andrew G. Barto
Reinforcement Learning: An Introduction (second edition)
Chapter 8

Outline

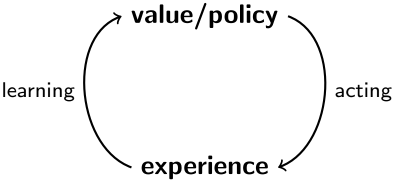
Outline

Model-Free vs Model-Based RL

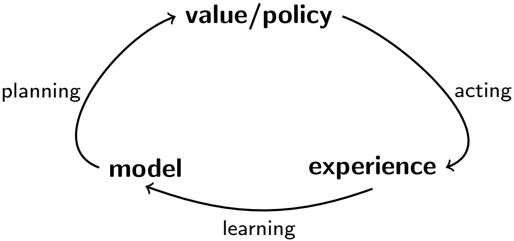
- **Model-free RL**
  - Learn a **value function** v and/or a **policy** π from experience
  - **No** model explicitly represented
- **Model-based RL**
  - Learn a **model** from experience
  - Plan the value function v and/or policy π

Model-Free vs Model-Based RL

Model-Free RL



Model-Based RL



Why learning a Model?

- Disadvantages
  - First learn a **model**, then construct a **value function** v and/or **policy** π
    - **Two** sources of **approximation error**
  - Learn the **value function** v directly
    - **One** source of **approximation error**
- Advantages
  - Model can be learned using **supervised learning** methods
  - Can better represent the **uncertainty**
  - Can be more **sample efficient**

Outline

What is a Model?

- The true **model** is the **joint** reward-next state distribution $\mathcal{P}(r, s'|s, a)$
- We assume conditional independence between next state and reward and deterministic reward

$$\mathcal{P}(r, s'|s, a) = \delta_{r(s,a)}(r) \cdot p(s'|s, a), \qquad \text{where:}$$

  - $p(s'|s, a)$ is the **transition model** of the MDP
  - $r(s, a)$ is the **reward function** of the MDP
- We assume to **know** the state space $\mathcal{S}$ and the action space $\mathcal{A}$
- Consider **parametric** representations of both r and p

$$\begin{aligned} p(s'|s, a) &\approx p_{\boldsymbol{\eta}}(s'|s, a) \\ r(s, a) &\approx r_{\boldsymbol{\eta}}(s, a) \end{aligned}$$

where $\boldsymbol{\eta} \in \mathbb{R}^p$ is a vector of real parameters

Model Learning

- **Goal**: estimate $p_{\boldsymbol{\eta}}$ and $r_{\boldsymbol{\eta}}$ from experience $(S_1, A_1, R_2, S_2, \dots, S_{T-1}, A_{T-1}, R_T, S_T)$
- Can be mapped to a **supervised learning** problem over the dataset

$$\begin{aligned} S_1, A_1 \quad &\rightarrow \quad R_2, S_2 \\ &\vdots \\ S_{T-1}, A_{T-1} \quad &\rightarrow \quad R_T, S_T \end{aligned}$$

- Learning the mapping $s, a \rightarrow r$ is a **regression** problem (as $r(s,a)$ is deterministic)
- Learning the mapping $s, a, \rightarrow s'$ is a **density estimation** problem (as s' is stochastic in general)

Outline

Families of Models

- **Expectation models** (Wan et al., 2019)
  - We treat the problem $s, a, \rightarrow s'$ as a **regression** as well
  - Hoping to learn the **expectation of the next state**

$$\overline{p}_{\boldsymbol{\eta}}(s, a) \quad \approx \quad \mathbb{E}_{S' \sim p(\cdot|s,a)}[S']$$

  - Ok with **deterministic** environments
- **Stochastic models**
  - We treat the problem $s, a \rightarrow s'$ as a **density estimation**

$$p_{\boldsymbol{\eta}}(s'|s, a) \quad \approx \quad p(s'|s, a)$$

  - Ok with also **stochastic environments**

Expectation Models

- Select a **function approximator** $\overline{p}_{\boldsymbol{\eta}}(s,a)$ (e.g., linear models, neural networks, ...)
- Choose a **loss function** $L : \mathbb{R}^p \to \mathbb{R}$ (e.g., **mean square error**)
- Find the parameters $\boldsymbol{\eta}$ minimizing the empirical loss
    - If $\mathcal{S} = \mathbb{R}^k$ and mean square error loss:

$$\widehat{\boldsymbol{\eta}} \in \operatorname*{arg\,min}_{\boldsymbol{\eta} \in \mathbb{R}^p} \frac{1}{T} \sum_{t=1}^{T} \left\| S_{t+1} - \overline{p}_{\boldsymbol{\eta}}(S_t, A_t) \right\|_2^2$$

- Expectation models can have **disadvantages**
    - The **expected next state** might be not informative
    - Ok if the true value function $v_{\mathbf{w}}(s) = \mathbf{w}^{\mathrm{T}} s$ is **linear** in the state

$$\mathbb{E}_{S' \sim p(\cdot|s,a)}[v_{\mathbf{w}}(S')] = \mathbf{w}^{\mathrm{T}} \mathbb{E}_{S' \sim p(\cdot|s,a)}[S'] \quad \approx \quad \mathbf{w}^{\mathrm{T}} \overline{p}_{\boldsymbol{\eta}}(s,a)$$

    - Ok if the environment is **deterministic**

Stochastic Models

- Select a **density approximator** $p_{\boldsymbol{\eta}}(s'|s,a)$ (e.g., Gaussian processes, deep belief networks, ...)
- Choose a **loss function** $L : \mathbb{R}^p \to \mathbb{R}$ (e.g., **KL-divergence**)
- Find the parameters $\boldsymbol{\eta}$ minimizing the empirical loss
    - With KL-divergence loss, the problem becomes a **maximum likelihood problem**:

$$\widehat{\boldsymbol{\eta}} \in \arg\max_{\boldsymbol{\eta} \in \mathbb{R}^p} \frac{1}{T} \sum_{t=1}^{T} \log p_{\boldsymbol{\eta}}(S_{t+1}|S_t, A_t)$$

- Models can be **chained** to predict the n-step future state
- ...but the error accumulates

Outline

Table Lookup Model

- When $|\mathcal{S}| < +\infty$ and $|\mathcal{A}| < +\infty$, we can model p and r as tables (Gheshlaghi Azar et al., 2013)
- Count visits $N(s,a) = \sum_{t=1}^{T} \mathbb{1}\{(S_t, A_t) = (s,a)\}$ for every state-action pair

$$\widehat{p}(s'|s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}\{(S_t, A_t, S_{t+1}) = (s,a,s')\}$$

$$\widehat{r}(s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}\{(S_t, A_t) = (s,a)\} R_{t+1}$$

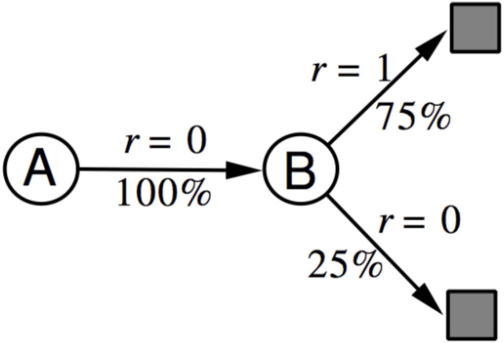
- No explicit parametrization $\boldsymbol{\eta}$
- If $r(s,a)$ is deterministic, one sample is enough for $\widehat{r}(s,a)$

AB Example

- Two states A, B
- one action from each state
- **stochastic** reward
- 8 trajectories of experience $(S_1, R_2, \dots)$

- Table lookup model from experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Linear Expectation Model

- Given a **feature** representation $\mathbf{x} : \mathcal{S} \to \mathbb{R}^p$ (Wan et al., 2019)
- We encode every state $s \in \mathcal{S}$ as $\mathbf{x}(s)$
- Expected next state and reward are **linear** functions

$$\mathbf{x}(s') \quad \approx \quad \mathbf{T}(a)\mathbf{x}(s) \qquad\qquad r(s,a) \quad \approx \quad \boldsymbol{\eta}(a)^{\mathsf{T}}\mathbf{x}(s)$$

  where $\mathbf{T}(a) \in \mathbb{R}^{p \times p}$ and $\boldsymbol{\eta}(a) \in \mathbb{R}^p$
- Can be optimized via **gradient descent** over the mean square error loss

$$\min_{\substack{\mathbf{T}(a)\in\mathbb{R}^{p\times p}\\ \boldsymbol{\eta}(a)\in\mathbb{R}^p \\ a\in\mathcal{A}}} \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}(S_{t+1}) - \mathbf{T}(A_t)\mathbf{x}(S_t)\|_2^2 + \left(R_{t+1} - \boldsymbol{\eta}(A_t)^{\mathsf{T}}\mathbf{x}(S_t)\right)^2$$

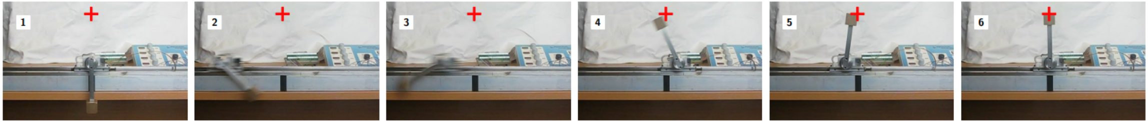
PILCO

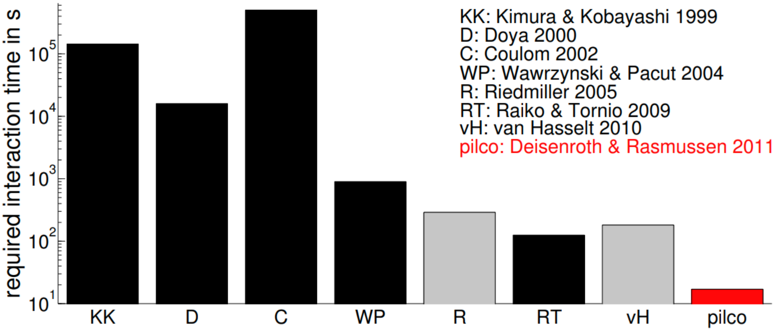
- Based on **Gaussian Processes (GPs)** (Deisenroth and Rasmussen, 2011)

$$p(S_{t+1}|S_t, A_t) = \mathcal{N}(S_{t+1}|\mu_t, \Sigma_t)$$

  - μ_t and Σ_t are fit from data
- Advantages
  - Reduces model bias (GPs model **any** function)
  - Incorporates **uncertainty** into planning (GPs are Bayesian methods)
  - **Analytical** policy gradient computation
- Disadvantages
  - True stochasticity might not be Gaussian
  - GPs are **computationally expensive**

PILCO - Experimental Results





Pictures from (Deisenroth and Rasmussen, 2011)

Error Bound for Estimated Model

Exercise 1

Let $(\widehat{p}, \widehat{r})$ be estimates of the true transition model and reward function (p, r) such that:

$$\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} D_{\mathsf{TV}}\left(\widehat{p}(\cdot|s,a), p(\cdot|s,a)\right) \le \epsilon_p \quad , \quad \|\widehat{r} - r\|_\infty \le \epsilon_r \quad \text{and} \quad \|r\|_\infty \le R_{\max}.$$

Let v_* be the optimal value function computed with (p, r) and $\widehat{v}_*$ be the optimal value function computed with $(\widehat{p}, \widehat{r})$, then it holds that:

$$\|v_* - \widehat{v}_*\|_\infty \le \frac{\epsilon_r}{1-\gamma} + \frac{\gamma R_{\max}\epsilon_p}{(1-\gamma)^2}.$$

Outline

Planning

- Once we have learned a model $(p_{\boldsymbol{\eta}}, r_{\boldsymbol{\eta}})$...
- ...we can use a **planning** algorithm to solve the MDP
  - Dynamic programming (value iteration, policy iteration)
  - Tree search
  - ...
- **Expensive** if the state/action spaces are large
- **Infeasible** if the state/action spaces are continuous

Sample-Based Planning

- We use the model $(p_{\boldsymbol{\eta}}, r_{\boldsymbol{\eta}})$ to **generate samples only**
- **Simulated experience** from the model

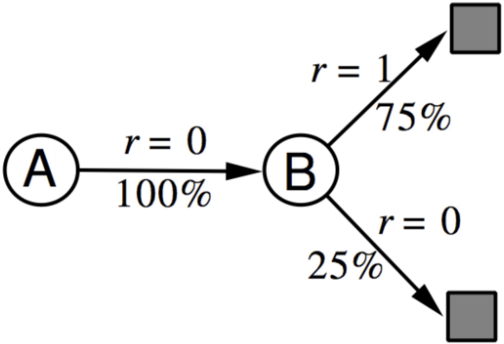
$$S_{t+1} \sim p_{\boldsymbol{\eta}}(\cdot|S_t, A_t)$$
$$R_{t+1} = r_{\boldsymbol{\eta}}(S_t, A_t)$$

- Now, we apply **model-free** RL approaches with the simulated experience
  - MC control
  - SARSA
  - Q-learning

AB Example

- Build the table lookup model from **real** experience
- Apply model-free RL to **simulated** experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



B, 1
B, 0
B, 1
A, 0, B, 1
B, 1
A, 0, B, 1
B, 1
B, 0

- MC learning: $v(A) = 1$, $v(B) = 0.75$

Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Drawbacks of an Inaccurate Model

- The imperfect model has some drawbacks:
  - The policy produced by planning can be **suboptimal**
  - It is the optimal policy of the **approximate** MDP with $(p_{\boldsymbol{\eta}}, r_{\boldsymbol{\eta}})$
  - Model-based RL is only **as good as the estimated model**
- How to cope with them?
  - When the model is wrong, use **model-free** RL
  - reason about the **uncertainty** on $\boldsymbol{\eta}$ (e.g., Bayesian approaches)
  - **Combine** model-based and model-free RL

Real experience (true MDP)

$$S_{t+1} \sim p(\cdot|S_t, A_t)$$
$$R_{t+1} = r(S_t, A_t)$$

Simulated experience (approximated MDP)

$$S_{t+1} \sim p_{\boldsymbol{\eta}}(\cdot|S_t, A_t)$$
$$R_{t+1} = r_{\boldsymbol{\eta}}(S_t, A_t)$$

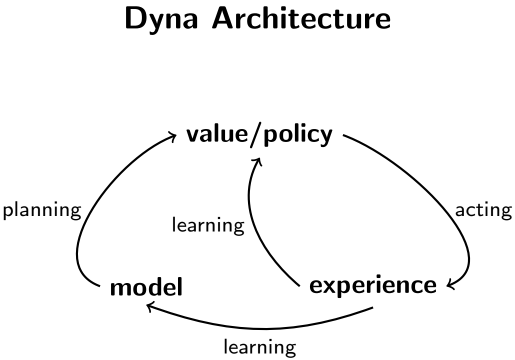
Outline

Combining Model-Free and Model-Based RL

- **Model-free RL**
  - Learn a **value function** v and/or a **policy** π from experience
  - **No** model explicitly represented
- **Model-based RL**
  - Learn a **model** from experience
  - Plan the value function v and/or policy π
- **Dyna**
  - Learn a **model** from experience
  - **Learn and plan** the value function v and/or policy π from real and simulated experience

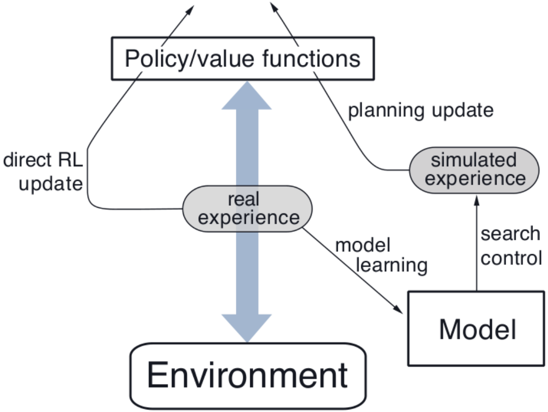
Outline

Dyna Architecture



Tabular Dyna-Q Architecture

- Use **real experience** to (Sutton, 1990, 1991)
  - Learn the $q_*(s,a)$ with tabular Q-learning (**model-free**)
  - Learn the model p and r with table lookup (**model-based**)
- Generate **simulated experience** to
  - Plan for $q_*(s,a)$ with tabular Q-planning



Pictures from (Sutton and Barto, 2018)

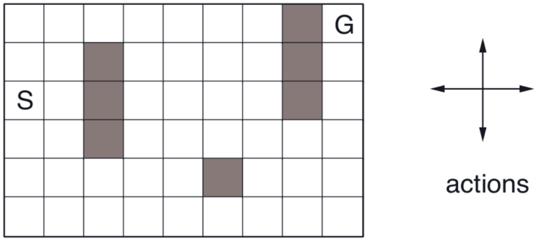
Tabular Dyna-Q with Deterministic Environment

- We need an **exploration policy** that selects the action based on q (e.g., ϵ-greedy, Boltzmann, ...)

Initialize $q, \widehat{p}, \widehat{r}$ arbitrarily
loop for each episode
    $S, A \leftarrow$ initial state and action using the exploration policy
    loop for each step of episode
        Take action A, observe reward R, and next state S'
        $q(S,A) \leftarrow q(S,A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} q(S', a') - q(S,A)\right]$
        Update transition model estimate $\widehat{p}(S,A) \leftarrow S'$
        Update reward estimate $\widehat{r}(S,A) \leftarrow R$
        loop for n times
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random previously taken action in S
            $S' \leftarrow \widehat{p}(S,A)$
            $R \leftarrow \widehat{r}(S,A)$
            $q(S,A) \leftarrow q(S,A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} q(S', a') - q(S,A)\right]$
        end loop
    end loop
end loop

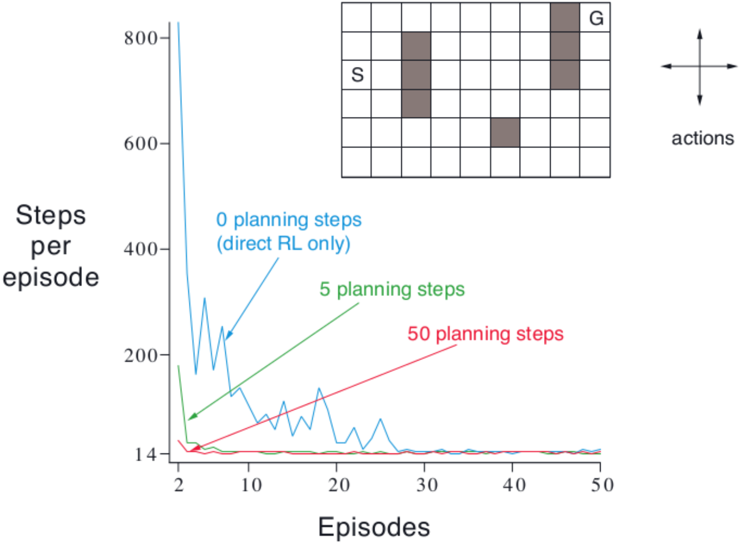
Dyna-Q on a Simple Maze

- Four actions: up, down, right, and left
- Reward: 0 everywhere, $+1$ in the goal
- $\gamma = 0.95$



Pictures from (Sutton and Barto, 2018)

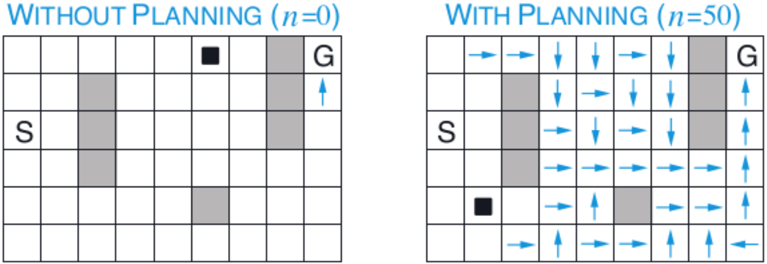
Dyna-Q on a Simple Maze



Pictures from (Sutton and Barto, 2018)

Dyna-Q on a Simple Maze

- Arrow is the greedy action
- No arrow if all actions have the same value



Pictures from (Sutton and Barto, 2018)

What if the Model is Wrong?

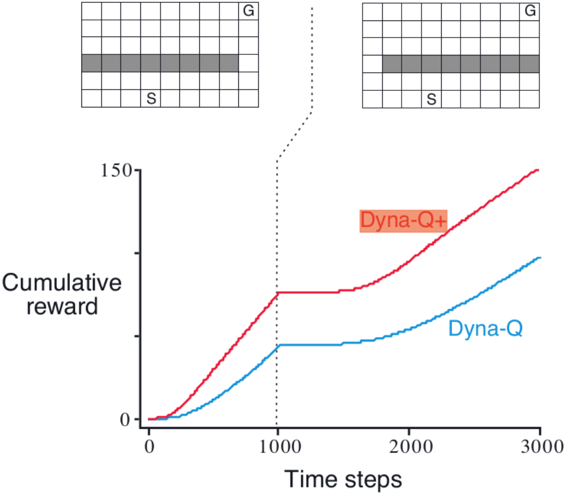
- Possible problems:
  - The environment may be **stochastic**
  - **Too little** real experience
  - **Bad generalization** of the function approximator
- The suboptimal policy computed by planning can lead to the **correction** of the model errors
- This happens when the model is **optimistic**
- Dyna-Q+ favors exploring **less tried** transitions (s, a, r, s')

$$\widehat{r}(s, a) + \kappa\sqrt{\tau(s, a, s')}$$

where $\kappa \geq 0$ and $\tau(s, a)$ is the number of steps elapsed from the last experience of the transition

Change the Environment during Learning

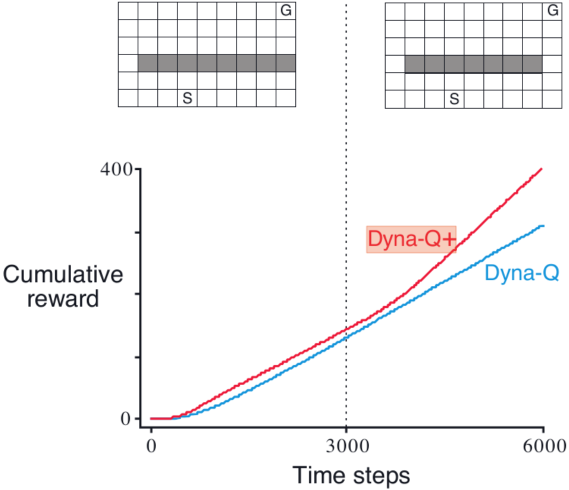
- The changed environment is **harder**



Pictures from (Sutton and Barto, 2018)

Change the Environment during Learning

- The changed environment is **easier**



Pictures from (Sutton and Barto, 2018)

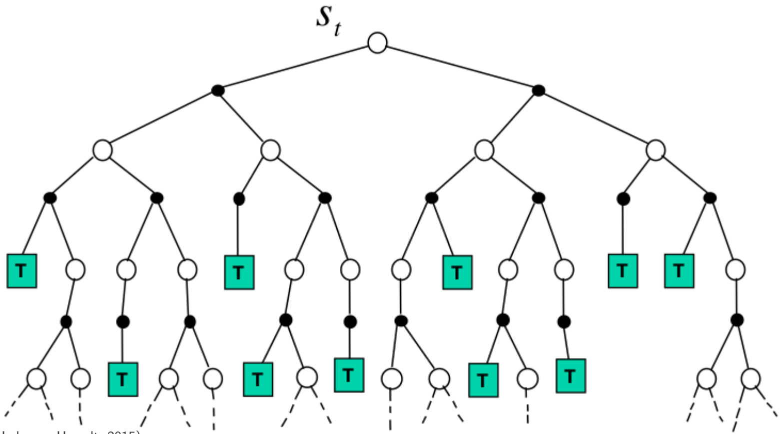
Outline

Planning for Action Selection

- So far, we used planning for improving a **value function** defined over the whole state-action space
- We now consider planning for **selecting the next action** to be executed
- Planning **locally** (for the next action) can be easier than planning for the **global** value function
- But, once we played the action, we need to **re-plan** in the next state

Forward Search

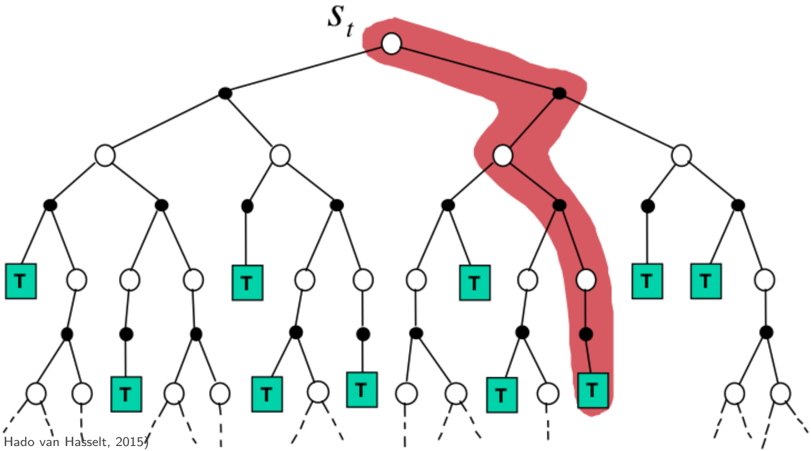
- Build a **search tree** with the current state S_t as root
- The MDP model (p, r) is used to generate node successors and rewards
- Do not solve the whole MDP, only the **sub-MDP** starting from the current state $\rightarrow$ can be easier!



Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Simulation-Based (Forward) Search

- Use forward search with **sample-based planning**
- **Simulate** episodes of experience from the current state
- Do not build the tree, but apply **model-free** RL to find the best action



Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Outline

Prediction via Monte-Carlo Simulation

- **Goal**: estimate the value function $v_\pi(S_t)$ of the **current state** only
- Given a model $(p_{\boldsymbol{\eta}}, r_{\boldsymbol{\eta}})$ and a **simulation policy** π
- Simulate M episodes from the current state S_t

$$\left\{(S_t^i = S_t, A_t^i, R_{t+1}^i, \dots, S_{T-1}^i, A_{T-1}^i, R_T^i, S_T^i)\right\}_{i=1}^M$$

where: $$S_{k+1}^i \sim p_{\boldsymbol{\eta}}(\cdot|S_k^i, A_k^i) \qquad R_{k+1}^i = r_{\boldsymbol{\eta}}(S_k^i, A_k^i)$$

- Estimate $v_\pi(S_t)$ with the **Monte-Carlo returns**

$$\widehat{v}(S_t) = \frac{1}{M}\sum_{i=1}^M G_t^i \qquad \text{where} \qquad G_t^i = \sum_{j=t}^{T-1} \gamma^{j-t} R_{j+1}^i$$

Control via Monte-Carlo Simulation

- **Goal**: find the **best action** to be played in the current state
- Given a model $(p_{\boldsymbol{\eta}}, r_{\boldsymbol{\eta}})$ and a **simulation policy** π
- For each action $a \in \mathcal{A}$, Simulate M episodes from the current state S_t

$$\left\{(S_t^i = S_t, A_t^i = a, R_{t+1}^i, \dots, S_{T-1}^i, A_{T-1}^i, R_T^i, S_T^i)\right\}_{i=1}^N$$

where: $\qquad S_{k+1}^i \sim p_{\boldsymbol{\eta}}(\cdot|S_k^i, A_k^i) \qquad R_{k+1}^i = r_{\boldsymbol{\eta}}(S_k^i, A_k^i)$

- Estimate $q_\pi(S_t, a)$ with the **Monte-Carlo returns**

$$\widehat{q}(S_t, a) = \frac{1}{N}\sum_{i=1}^N G_t^i \qquad \text{where} \qquad G_t^i = \sum_{j=t}^{T-1} \gamma^{j-t} R_{j+1}^i$$

- Select the action maximizing the estimated $\widehat{q}(S_t, a)$

$$A_t \in \arg\max_{a \in \mathcal{A}} \widehat{q}(S_t, a)$$

Outline

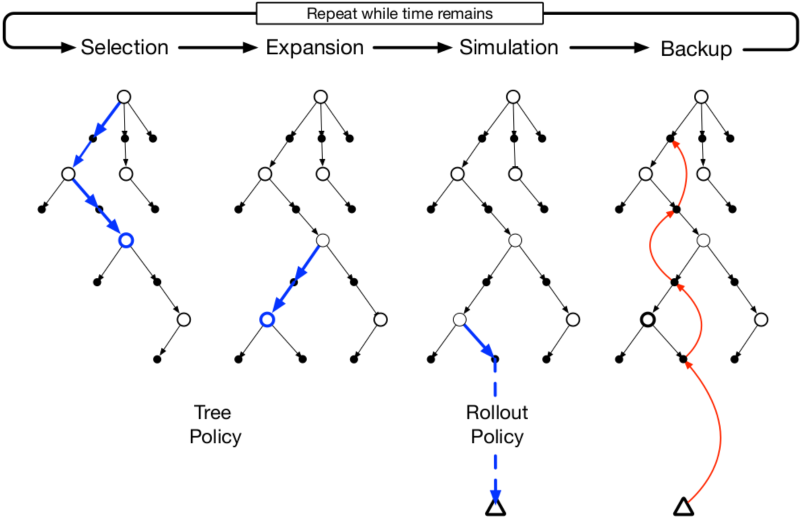
Monte Carlo Tree-Search

- Incrementally build a **search tree** with the visited states, actions, and estimated $q_*(S_t, a)$ from the current state S_t (Browne et al., 2012)
- Repeat the **four** steps (until budget is expired)
  1. **Selection**: from root reach a node, choosing actions with the **tree policy**
  2. **Expand**: one (or more) child nodes are added to expand the tree, according to the available actions
  3. **Simulation**: a simulation is run from the new node(s), using the **rollout (or default) policy**
  4. **Backup**: $\widehat{q}(s, a)$ of the path are updated based on the return of the simulation
- Select the action maximizing the estimate in the root node

$$A_t \in \operatorname*{arg\,max}_{a \in \mathcal{A}} \widehat{q}(S_t, a)$$

- Under certain conditions, **converges** to $q_*(S_t, a)$

Monte Carlo Tree-Search



- $\bullet$ = random node (action)
- $\circ$ = decision node (state)

Pictures from (Sutton and Barto, 2018)

Monte Carlo Tree-Search - Policies

- The **tree policy** balances **exploration** and **exploitation**
  - ϵ-greedy on the estimated $\widehat{q}(s, a)$
  - UCB policy $\rightarrow$ UCT (Upper Confidence Tree) (Coquelin and Munos, 2007)

$$\widehat{a} \in \arg\max_{a \in \mathcal{A}} \widehat{q}(s, a) + \sqrt{\frac{\alpha \log N(s)}{N(s, a)}}$$

  - It **improves** during execution
- The **rollout (or default) policy** is fixed
  - e.g., random uniform policy over $\mathcal{A}$

MCTS Algorithm

Create root node (S_t)
Initialize $(S_t).N \leftarrow 0$
Initialize $(S_t).V \leftarrow 0$
loop within computational budget
    $(S) \leftarrow \textsc{TreePolicy}((S_t))$
    $\Delta \leftarrow \textsc{RolloutPolicy}(S, H, \gamma)$
    $\textsc{Backup}((S), \Delta, \gamma)$
end loop
return $\textsc{BestChild}((S_t))$

procedure $\textsc{BestChild}((S))$
    for (A) child of (S) **do**
        Compute Q-value $Q(A) \leftarrow (A).W/(A).N$
        Compute the bonus $B(A) \leftarrow \sqrt{\alpha \log((S).N)/((A).N)}$
    end for
    return $(\arg\max_{a \in \mathcal{A}} \{Q(a) + B(a)\})$
end procedure

- State-nodes are denoted with (S) where S is the state
  - $(S).V$ is the rollout return from node (S)
  - $(S).N$ is the number of updates to node (S)
  - $(S).R$ is the immediate reward obtained in the transition that has S as next state
- Action-nodes are denoted with (A) where A is the action
  - $(A).W$ is the sum of the returns from node (A)
  - $(A).N$ is the number of updates to node (A)
  - Thus $\widehat{q}(S, A) = (A).W/(A).N$

MCTS Algorithm

procedure $\textsc{TreePolicy}((S))$
    while S is non-terminal **do**
        $(A) \leftarrow \textsc{BestChild}((S))$
        $S' \leftarrow p(S, A)$
        if (S') is a child of (A) **then**
            $(S').R \leftarrow r(S, A)$
            $(S) \leftarrow (S')$
        else
            return $\textsc{Expand}(S', (A))$
        end if
    end while
    return (S')
end procedure

procedure $\textsc{Expand}(S',(A))$
    Create node (S') as a child of (A)
    Initialize the value $(S').V \leftarrow 0$
    Initialize the count $(S').N \leftarrow 0$
    for action $A' \in \mathcal{A}$ **do**
        Create node (A') as a child of (S')
        Initialize $(A').W \leftarrow 0$
        Initialize $(A').N \leftarrow 0$
    end for
    return (S')
end procedure

MCTS Algorithm

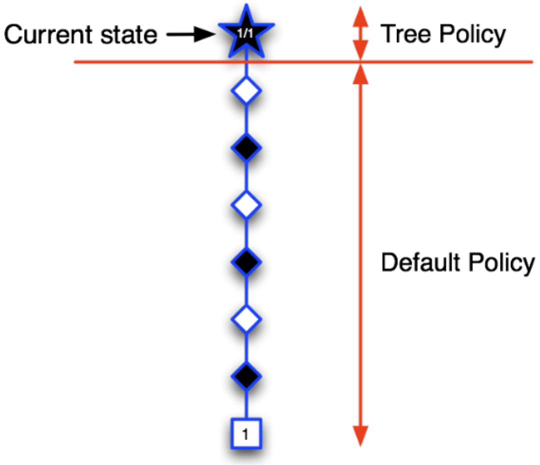
procedure ROLLOUTPOLICY(S,H,γ)
  $\Delta \leftarrow 0$
  $t \leftarrow 0$
  while S is non-terminal **and** $t < H$ **do**
    Choose A uniformly at random
    $\Delta \leftarrow \gamma^t r(S, A)$
    $S \leftarrow p(S, A)$
    $t \leftarrow t + 1$
  end while
  return Δ
end procedure

procedure BACKUP((S), Δ, γ)
  $(S).V \leftarrow \Delta$
  $(S).N \leftarrow (S).N + 1$
  while (S) is not root **do**
    $\Delta \leftarrow (S).R + \gamma\Delta$
    Get (A) parent of (S)
    $(A).N \leftarrow (A).N + 1$
    $(A).W \leftarrow (A).W + \Delta$
    Get (S) parent of (A)
    $(S).N \leftarrow (S).N + 1$
  end while
end procedure

Example

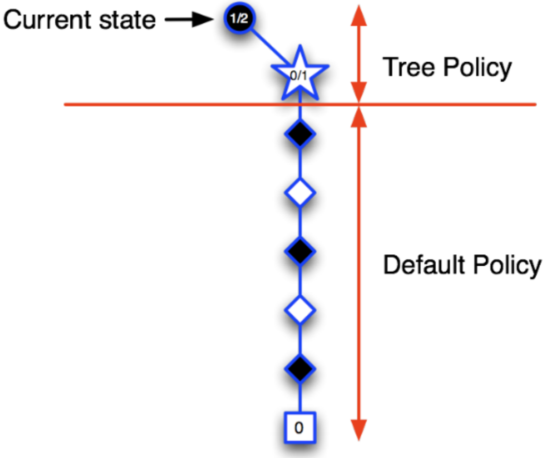
- 2 actions
- Reward in terminal state only
- **Greedy** tree policy
- **Random** rollout policy

Example - 1



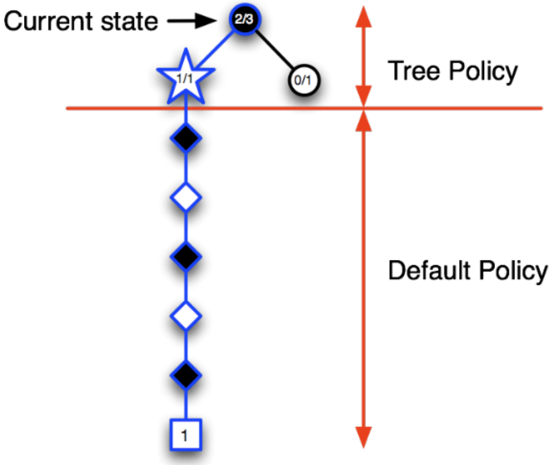
Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Example - 2



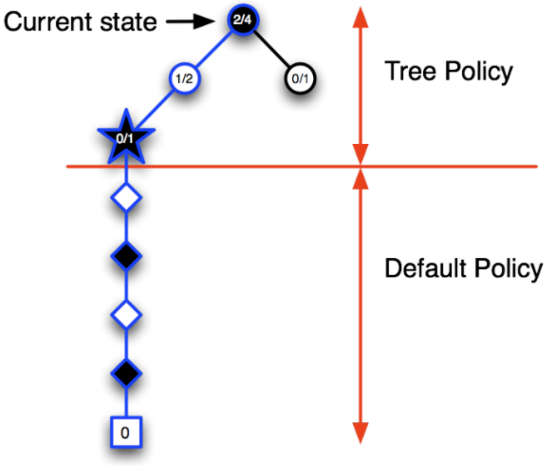
Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Example - 3



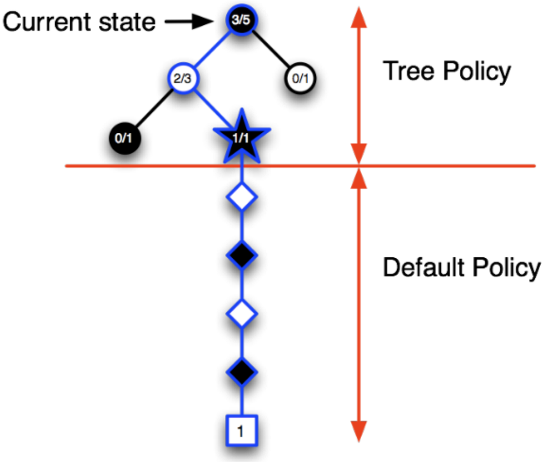
Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Example - 4



Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Example - 5



Pictures from (Silver, 2015; Hado van Hasselt, 2015)

Advantages of Monte Carlo Tree-Search

- States are evaluated **dynamically**, unlike DP
- Uses sampling to cope with large state spaces
- Works for **black-box** models
- Computationally efficient, anytime, parallelizable
- Can be applied to **games**

Drawbacks of Monte Carlo Tree-Search

- Actions must be **finite** ($|\mathcal{A}| < +\infty$), otherwise we keep exploring actions
- Environment **can be stochastic and continuous**, but the number of next states must be finite, otherwise infinite branching factor (Jonsson et al., 2020)

$$|\{s' \in \mathcal{S} \,:\, p(s'|s,a) > 0\}| \leq B < +\infty \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A}$$

- How to cope with these problems?

Outline

Open Loop Planning

- Plan over the **sequence of actions** as opposed to plan over the **policies** (mapping from states to actions) (Bubeck and Munos, 2010)
- We optimize the **open-loop** objective

$$q_{\mathsf{OL}}(S_t, a) = \max_{a_{0:\infty} \in \mathcal{A}^\infty} q(S_t, a_{1:\infty}) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t r(S_t, a_t) | a_0 = a\right]$$

  where $a_{0:\infty} = (a, a_1, \dots)$ are selected in advance
- Instead, RL optimizes the **closed-loop** objective

$$q_*(S_t, a) = \max_{\pi : \mathcal{S} \to \mathcal{A}} q_\pi(S_t, a) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t r(S_t, A_t) | A_0 = a, A_t = \pi(S_t)\right]$$

Open Loop Planning

- **Computationally intensive**: search in the space of sequences of length H: $|\mathcal{A}|^H$ sequences
- **Optimal** for deterministic environments
- Can be used with **stochastic environments** but ...
  - ... forced to play the same action at time t, regardless the state...
  - ... so **suboptimal** for stochastic environments

$$q_{\mathsf{OL}}(S_t, a) \leq q_*(S_t, a)$$

- Still requires **finite** actions
- Using upper confidence bounds $\rightarrow$ **Open Loop Optimistic Planning** (Bubeck and Munos, 2010)

Outline

Progressive Widening

- If we have **continuous actions** ($|\mathcal{A}| = +\infty$), we need to **discretize**
- **Fixed discretization**: choose a finite set of k actions

$$\mathcal{A}_k = \{a_1, \dots, a_n\} \subset \mathcal{A}$$

  - Never vanishing approximation error
  - Can be optimally selected with environment regularities are present
- **Progressive Widening (PW)**: adapt the discretization through time (Chaslot et al., 2008)

$$k(t) = \lceil C t^{\alpha} \rceil \qquad \text{for some } \alpha \in (0,1) \qquad \mathcal{A}_{k(t)} = \{a_1, \dots, a_{k(t)}\} \subset \mathcal{A}$$

  - The action set **grows** though time
  - PW does not work in stochastic environments with **infinite possible next states** $\rightarrow$ **Double Progressive Widening** (Couëtoux et al., 2011)

Error Bound for Open Loop Planning

Exercise 2

Consider the optimal value function:

$$v_*(s) = \max_{\pi:\mathcal{S}\to\mathcal{A}} v_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t r(S_t, A_t) | S_0 = s\right]$$

and the open-loop value function:

$$v_{\mathsf{OL}}(s) = \max_{a_{0:\infty}\in\mathcal{A}^\infty} \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t r(S_t, a_t) | S_0 = s\right]$$

where $a_{0:\infty} = (a_0, a_1, \dots)$. Prove that:

$$\|v_* - v_{\mathsf{OL}}\|_\infty \leq \frac{2\gamma R_{\max}}{(1-\gamma)^2}\left(1 - \min_{s,a\in\mathcal{S}\times\mathcal{A}} \max_{s'\in\mathcal{S}} p(s'|s,a)\right)$$

Note that when p is deterministic, we have **zero error**!

References I

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

S. Bubeck and R. Munos. Open loop optimistic planning. In *COLT 2010-The 23rd Conference on Learning Theory*, 2010.

G. M. J. Chaslot, M. H. Winands, H. J. v. d. Herik, J. W. Uiterwijk, and B. Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, 2007.

A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, pages 433–445. Springer, 2011.

M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 465–472. Omnipress, 2011.

M. Gheshlaghi Azar, R. Munos, and H. J. Kappen. Minimax pac bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91(3):325–349, 2013.

M. H. Hado van Hasselt, Diana Borsa. Reinforcement learning lecture series 2021. `https://deepmind.com/learning-resources/reinforcement-learning-series-2021`, 2015.

A. Jonsson, E. Kaufmann, P. Ménard, O. Darwiche Domingues, E. Leurent, and M. Valko. Planning in markov decision processes with gap-dependent sample complexity. *Advances in Neural Information Processing Systems*, 33:1253–1263, 2020.

C. Nota and P. S. Thomas. Is the policy gradient a gradient? In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 939–947, 2020.

D. Silver. Introduction to reinforcement learning. `https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver`, 2015.

References II

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

R. S. Sutton. Planning by incremental dynamic programming. In *Machine learning proceedings 1991*, pages 353–357. Elsevier, 1991.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Y. Wan, M. Zaheer, A. White, M. White, and R. S. Sutton. Planning with expectation models. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3649–3655, 2019.