Reinforcement learning

Dr. Aissa Boulmerka

The National School of Artificial Intelligence
aissa.boulmerka@ensia.edu.dz

2024-2025

Chapter 7
Planning and Learning with Tabular Methods

Outline

- Models and Planning
- Random Tabular Q-planning
- Dyna Algorithm
- When the Model is inaccurate
- In-depth with changing environments
- Space of reinforcement learning methods

Introduction

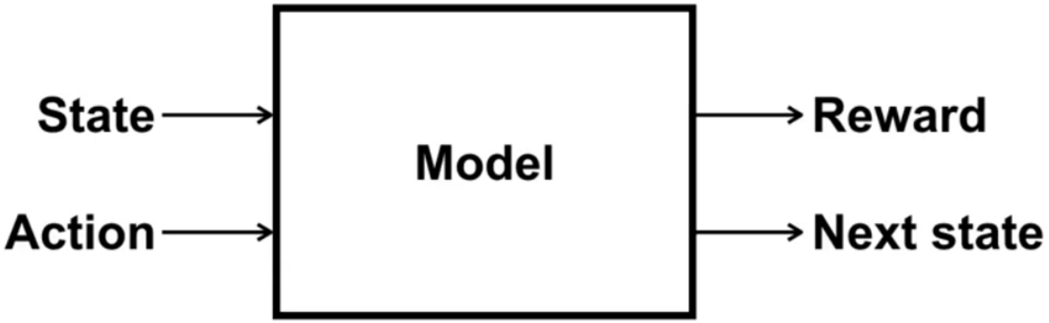
Introduction

- In reinforcement learning, methods can be categorized based on whether they **use a model of the environment or not**.
- The methods that require a model, such as dynamic programming and heuristic search, these methods are called **model-based RL methods**.
- The methods that can be used without a model, such as Monte Carlo and temporal-difference methods are called **model-free RL methods**.
- Model-based methods rely on **planning** as their primary component, while model-free methods primarily rely on **learning**.
- It would be even better if we could obtain an **intermediate method** that can leverage the best of both extremes. In this chapter, we will develop a **unified view** of reinforcement learning methods that require **planning and learning strategies** using the **Dyna architecture**.

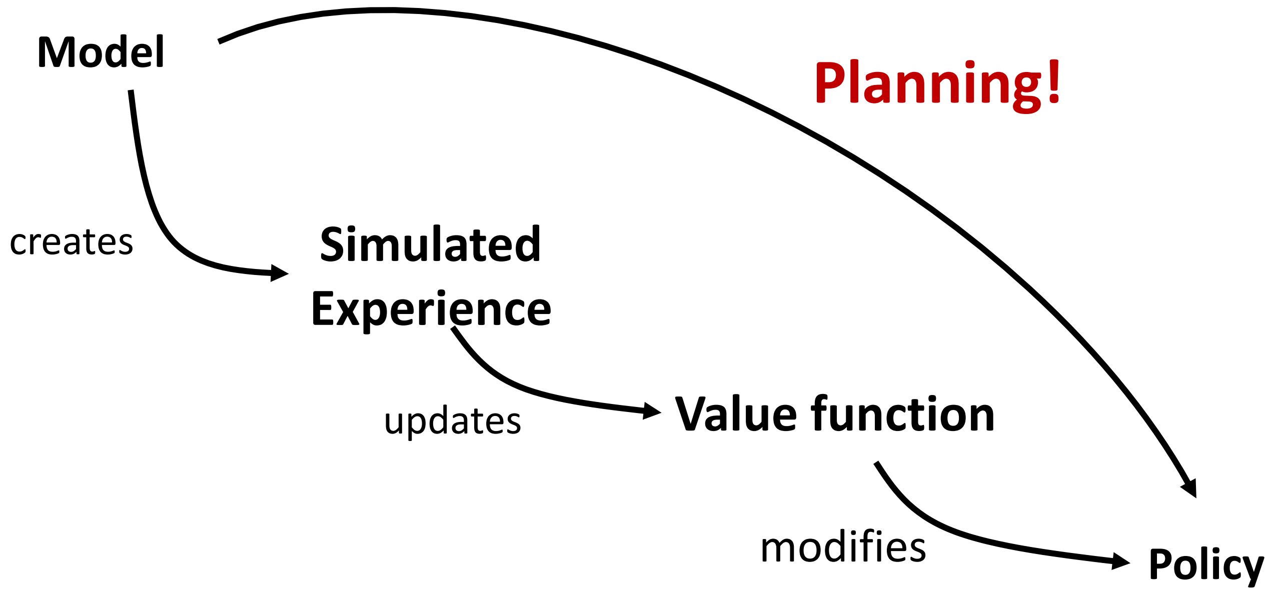
Models and Planning

Models and Planning

- By a **model** of the environment we mean anything that an **agent** can use to **predict** how the environment will respond to its **actions**.
- Given a **state** and an **action**, a **model** produces a **prediction** of the resultant **next state** and **next reward**.
- If the model is **stochastic**, then there are several possible next states and next rewards, each with some **probability** of occurring.



Models and Planning

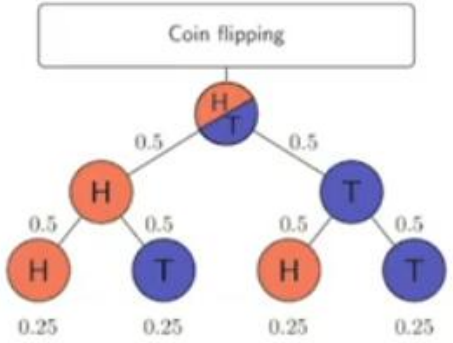


Types of models

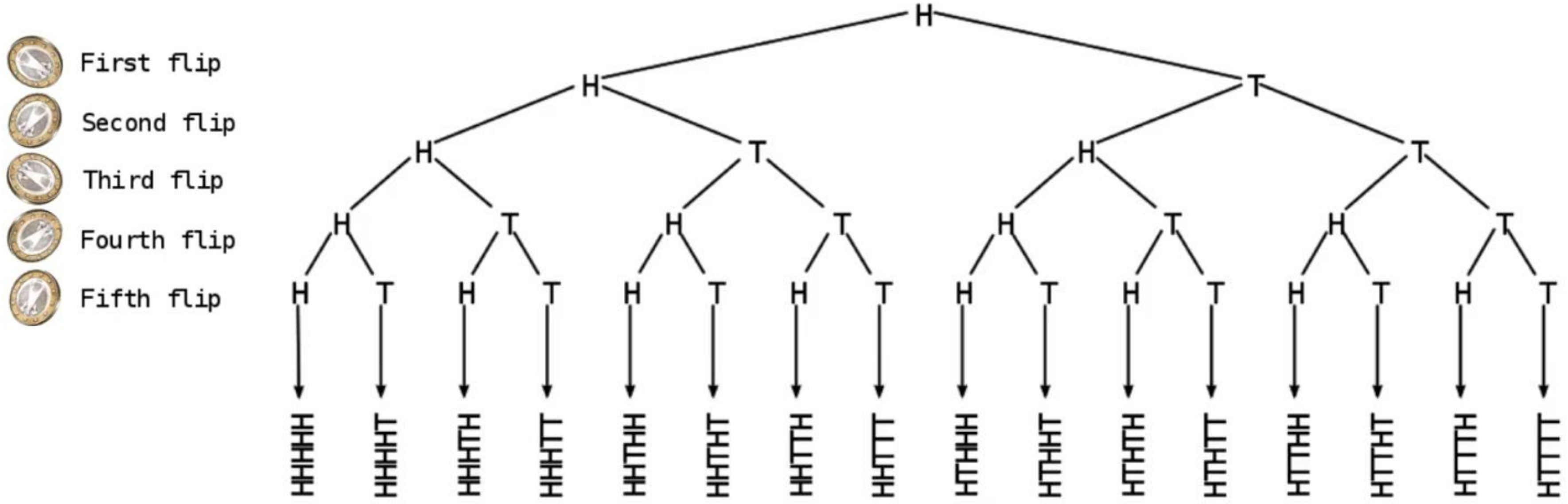
- Some models produce just one of the possibilities, sampled according to the probabilities; these we call **sample models**. For example, a sample model for flipping a coin can generate a random sequence of heads and tails.
- Other models produce a description of all possibilities and their probabilities; these we call **distribution models**.
- Models in reinforcement learning **simulate experiences** by predicting state transitions. A **sample model** provides one possible transition, while a **distribution model** generates all potential transitions with their probabilities.
- Sample models can **be computationally inexpensive** because random outcomes can be produced according to a set of rules.

Sample

Distribution



Distribution models



Outcome:

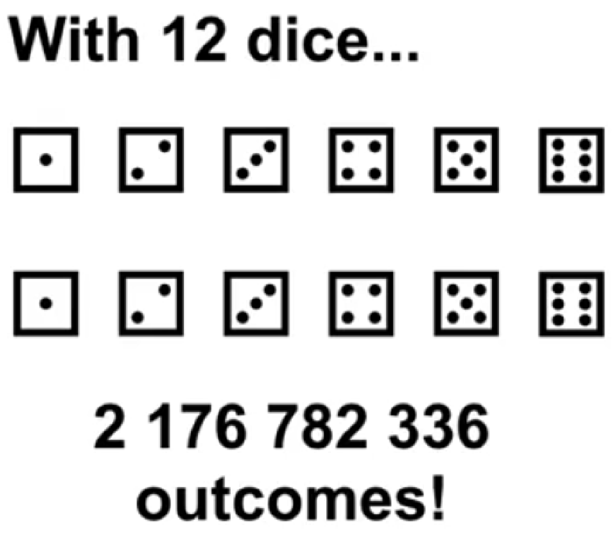
$$p = \frac{1}{32}$$

Comparing Sample and Distribution Models

- Consider the problem of rolling **12 dices**. It is easy to roll 12 dices.

Comparing Sample and Distribution Models

- Modeling the complete **joint distribution** of rolling 12 dice is much more work.
- We have **over 2 billion** possible outcomes to consider.
- But it can be useful that a distribution model produces the **exact probability** of an outcome. For example, we can compute the **expected outcome** directly or **quantify the variability** in outcomes.



Advantages of Sample and Distribution Models

- Sample models require **less memory** (they are more compact than distribution models).
- Distribution models on the other hand can be used to compute the **exact expected outcome** by **summing** over all outcomes **weighted** by their probabilities.
- Sample models can only **approximate** this expected outcome by **averaging many samples** together.
- Knowing the exact probabilities also has the flexibility of **assessing risk**. For example, when a doctor is **prescribing medicine**, they would consider many possible **side effects** and how **likely** they are to occur.

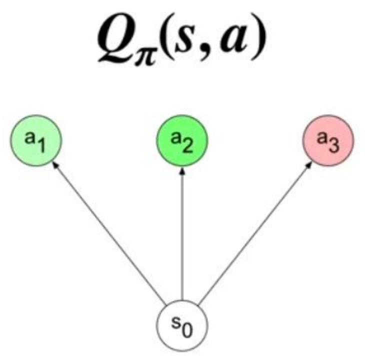
Random Tabular Q-planning

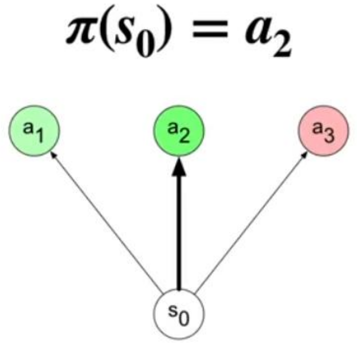
Random Tabular Q-planning

- How one can leverage a model to better inform **decision-making without having to interact with the world**, we call this process **planning with model experience**.
- Learning methods require **only experience as input**, and in many cases they can be applied to simulated experience just as well as to real experience.
- Next slides show a simple example of a **planning method** based on one-step **tabular Q-learning and on random samples** from a sample model.
- This method, which we call ***random-sample one-step tabular Q-planning***, converges to the **optimal policy** for the model under the same conditions that **one-step tabular Q-learning** converges to the optimal policy for the real environment (each state-action pair must be selected an infinite number of times in Step 1, and α must decrease appropriately over time).

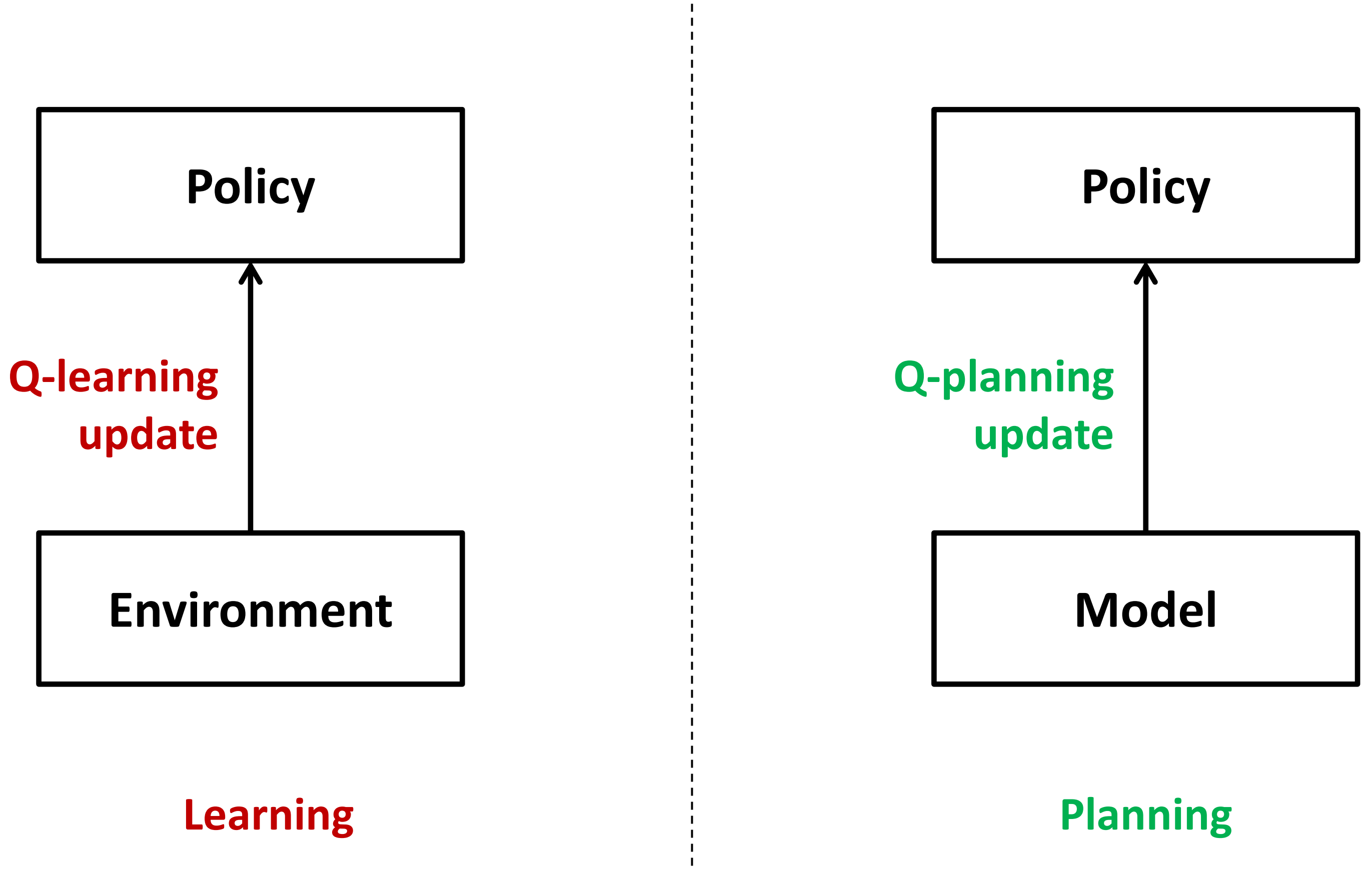
Planning improves policies

Planning





Connection with Q-learning



Random-sample one step tabular Q-planning

1. Sampling:

$$\begin{tikzcd}
\mathcal{S} \arrow[r] & S \arrow[rd] & & S' \\
\mathcal{A} \arrow[r] & A \arrow[r] & \boxed{\text{Model}} \arrow[ru] \arrow[r] & R
\end{tikzcd}$$

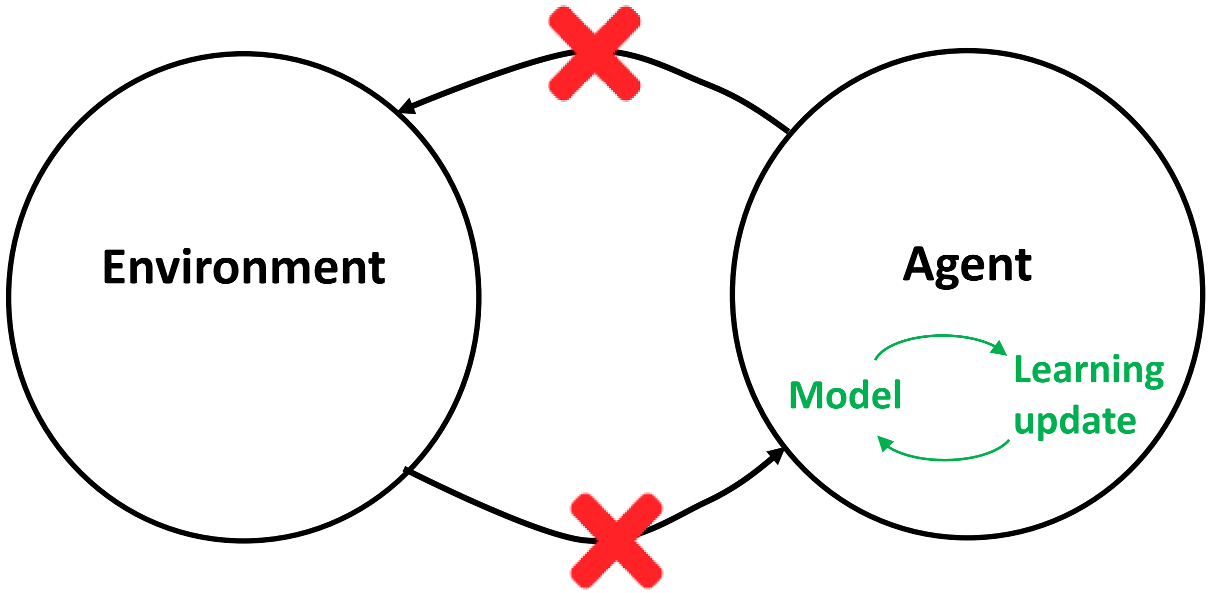
2. Q-learning update:

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A) \right]$$

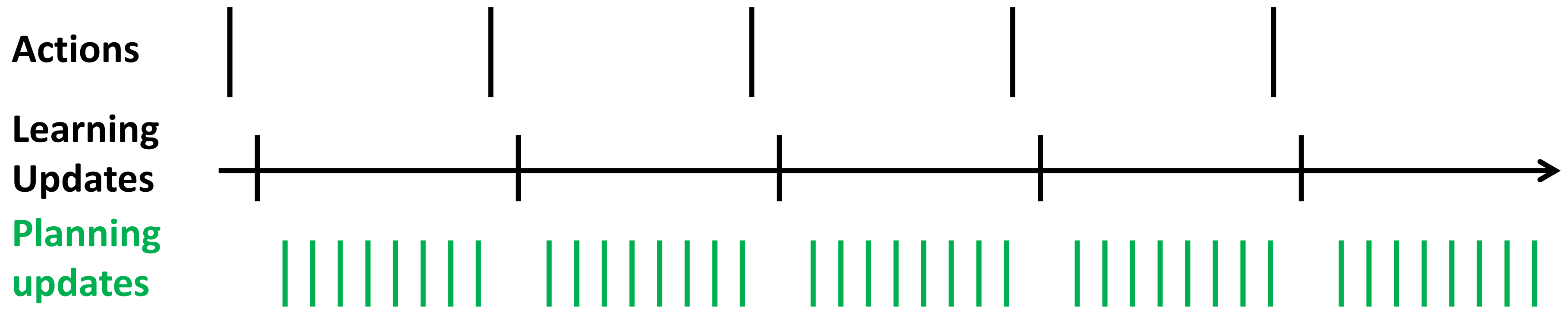
3. Greedy policy improvement:

$$\pi(s) = \operatorname*{argmax}_a Q(s,a)$$

Planning only uses imagined experience



Advantages of Planning

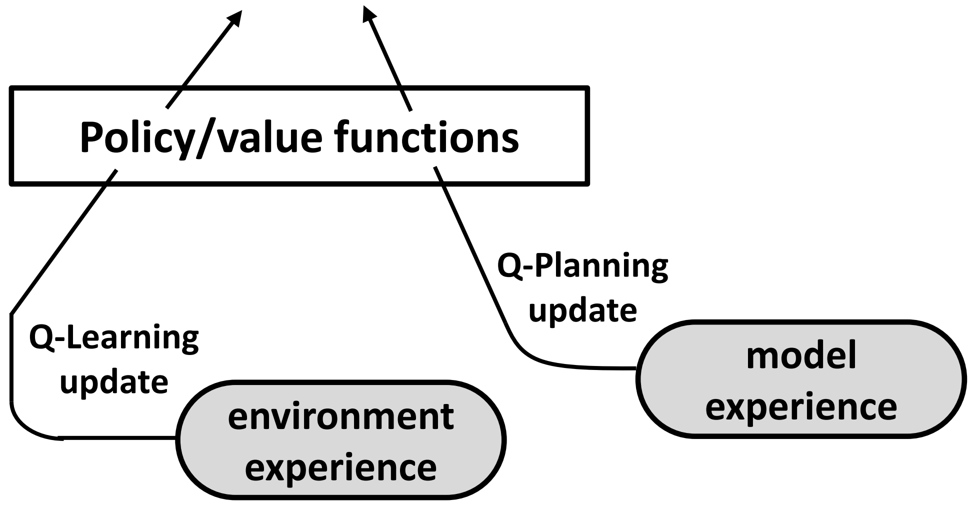


- Imagine that **actions** can only be taken at **specific time points**, but **Learning Updates** can be executed relatively fast.
- This results in some waiting time from after the **Learning Update** and when the next action is taken.
- We can fill in this waiting time with **Planning Updates**.

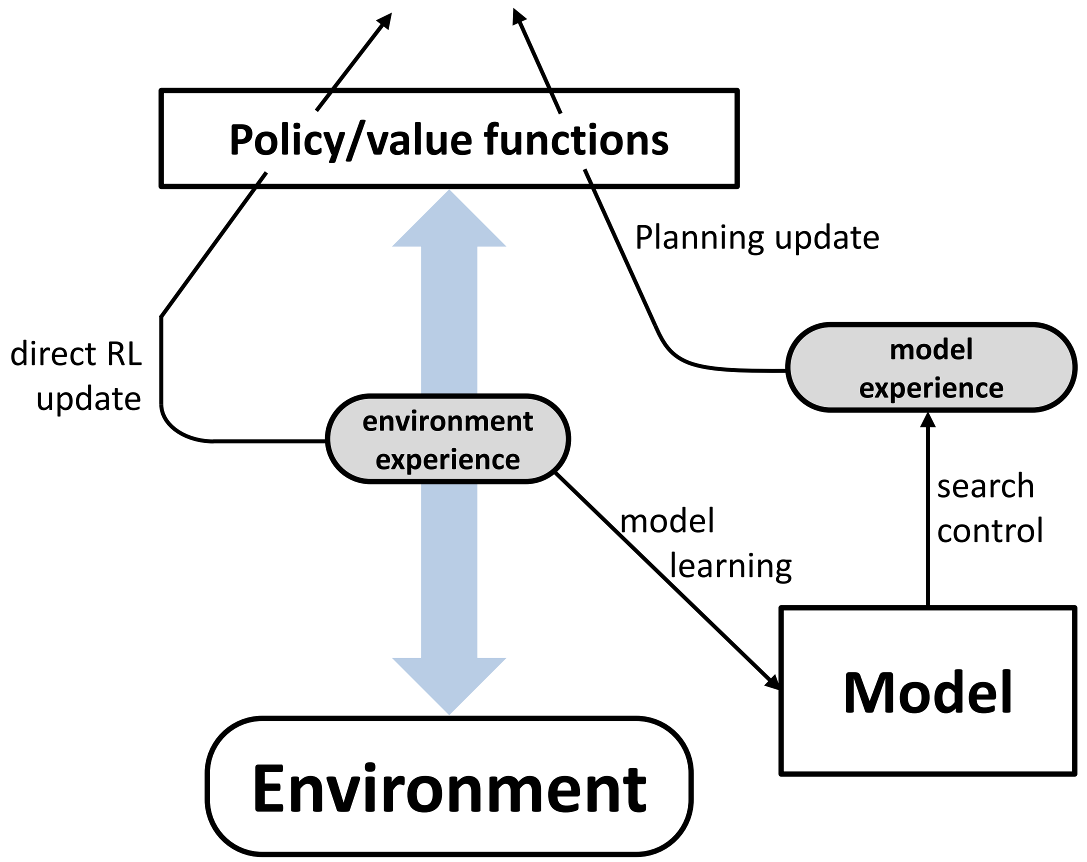
Dyna Algorithm

Combining Q-Learning and Q-Planning

- Q-learning performs updates using **environment experience**.
- Q-planning on the other hand performs updates using **model (simulated) experience** from the model.
- We can combine these two ideas through the **Dyna architecture**.



Dyna: Integrated Planning, Acting, and Learning



Dyna: Integrated Planning, Acting, and Learning

Tabular Dyna-Q

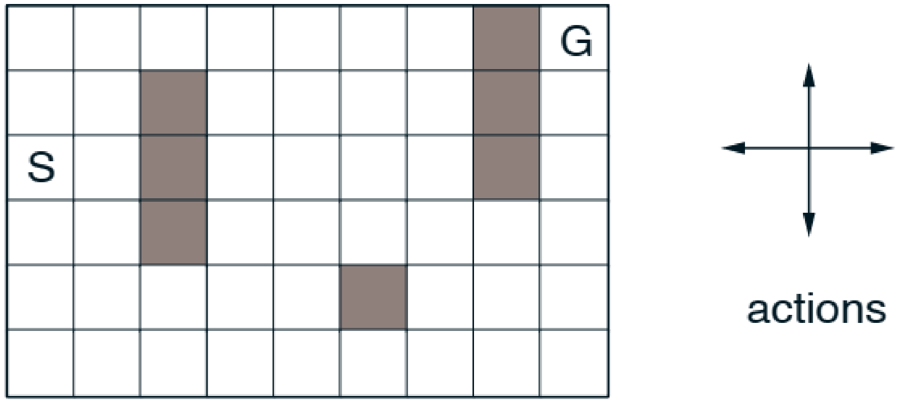
```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S, Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    (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 $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```

Dyna: Integrated Planning, Acting, and Learning

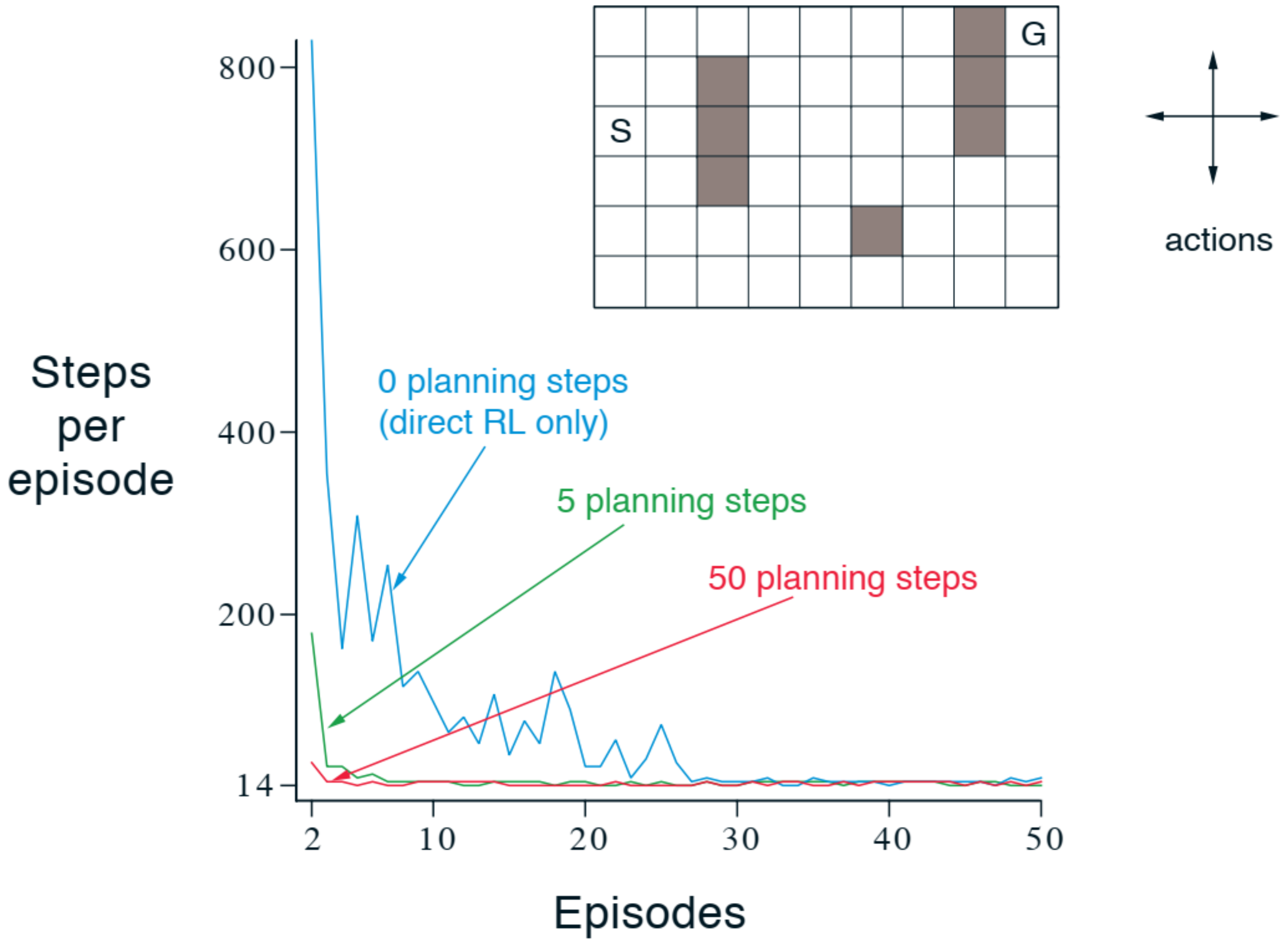
- In Dyna-Q, the acting, model-learning, and direct RL processes require **little computation**, and we assume they consume **just a fraction of the time**.
- The remaining time in each step can be devoted to the **planning process**, which is inherently **computation-intensive**.
- Let us assume that there is time in each step, after acting, model-learning, and direct RL, to complete n iterations (Steps 1–3) of the Q-planning algorithm.
- In the pseudocode algorithm for Dyna-Q, $Model(s, a)$ denotes the contents of the (predicted next state and reward) for state–action pair (s, a).
- Direct reinforcement learning, model-learning, and planning are implemented by steps (d), (e), and (f), respectively. If (e) and (f) were omitted, the remaining algorithm would be **one-step tabular Q-learning**.

Example: Dyna Maze

- Consider the following **simple maze**. In each of the 47 states there are four actions, **up, down, right, and left**, which take the agent deterministically to the corresponding neighboring states, except when movement is blocked by an obstacle or the edge of the maze, in which case the agent remains where it is.
- Reward is **0 on all transitions**, except those into the **goal state, it is $+1$**.
- After reaching the **goal state (G)**, the agent returns to the **start state (S)** to begin a new episode. This is a **discounted, episodic task with $\gamma = 0.95$**.

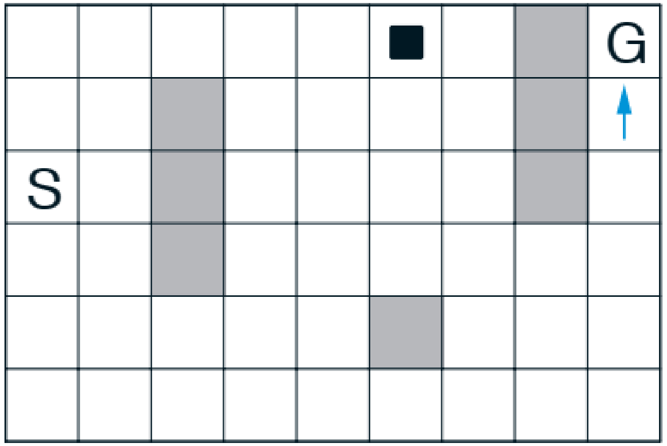


Example: Dyna Maze



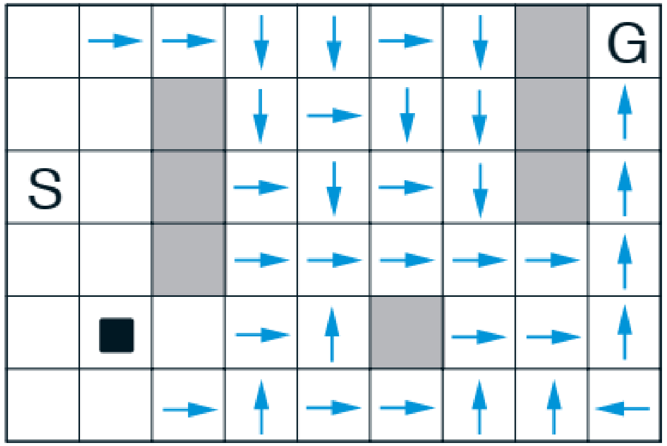
Example: Dyna Maze

WITHOUT PLANNING (n=0)



- Each episode adds only **one additional step** to the policy, and so only one step (the last) has been learned so far.

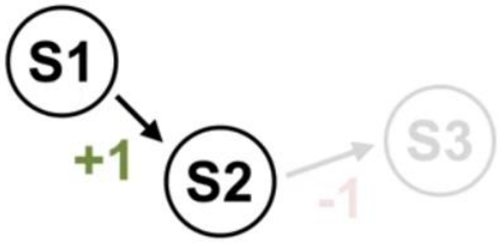
WITH PLANNING (n=50)



- Only one step is learned during the first episode, but here during the second episode **an extensive policy has been developed** that by the end of the episode will reach almost back to the start state.

When the Model is Inaccurate?

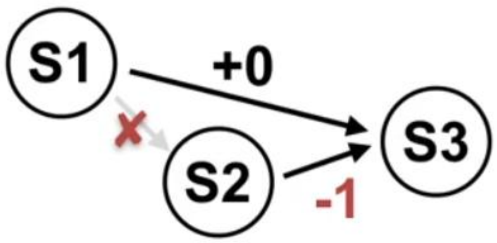
How model can be inaccurate?



$S1 \rightarrow S2, +1$

$S2 \rightarrow ??$

Incomplete model



$S1 \rightarrow S2, +1$ ✗

$S2 \rightarrow S3, -1$

Changing environment

Exploration and exploitation for model accuracy

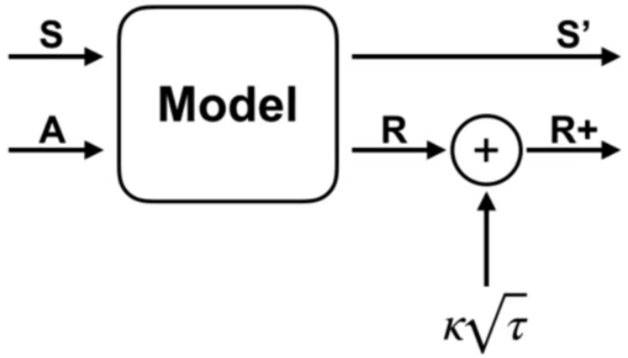
- The general problem here is another version of the **conflict** between **exploration and exploitation**.
- In a planning context, **exploration** means trying actions that **improve** the model, whereas **exploitation** means behaving in the **optimal way** given the current model.
- We want the agent to **explore** to find **changes** in the environment, but not so much that **performance** is greatly degraded.
- As in the earlier **exploration/exploitation conflict**, there probably is **no solution** that is both perfect and practical, but simple **heuristics** are often effective.
- The **Dyna-Q+ agent** that did solve the **shortcut maze** uses one such heuristic.

Dyna-Q+

- Dyna-Q+ tracks elapsed time since each **state-action** pair was **last tried**. Greater **elapsed time** suggests a higher chance the model of the pair is **incorrect**.
- A **"bonus reward"** is given for **simulated experiences** with long-untried actions.

$$\boldsymbol{New\ reward = r + \kappa\sqrt{\tau}}$$

- where $\boldsymbol{r}$ is the **actual reward**, $\boldsymbol{\tau}$ is **the time steps since transition was last tried**, and $\boldsymbol{\kappa}$ is a **small constant**.
- This encourages the agent to test all **accessible state transitions** and discover **new action sequences**, despite the cost.



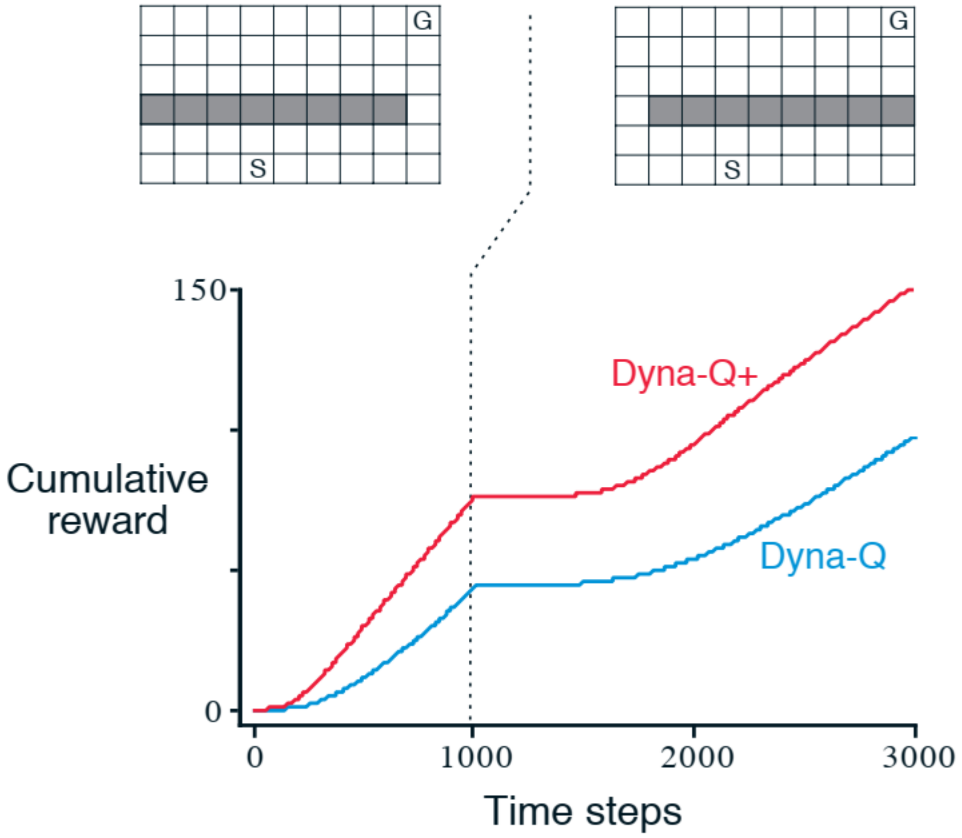
Dyna-Q vs Dyna-Q+ in a changing environment

Example: Shortcut Maze

- During the **first half of the experiment**, Dyna-Q and Dyna-Q+ perform similarly. In this case, Dyna-Q+'s **increased exploration** helps it develop a better policy more quickly.
- After a **shortcut** is introduced on the right side of the wall, **Dyna-Q+** quickly identifies and uses this shortcut following the **environment changes**.
- In contrast, **Dyna-Q fails** to find the shortcut within the given time and would take a **very long time** to **discover** it.
- The **persistent** and **systematic** exploration by Dyna-Q+ proves to be **crucial** in this environment.

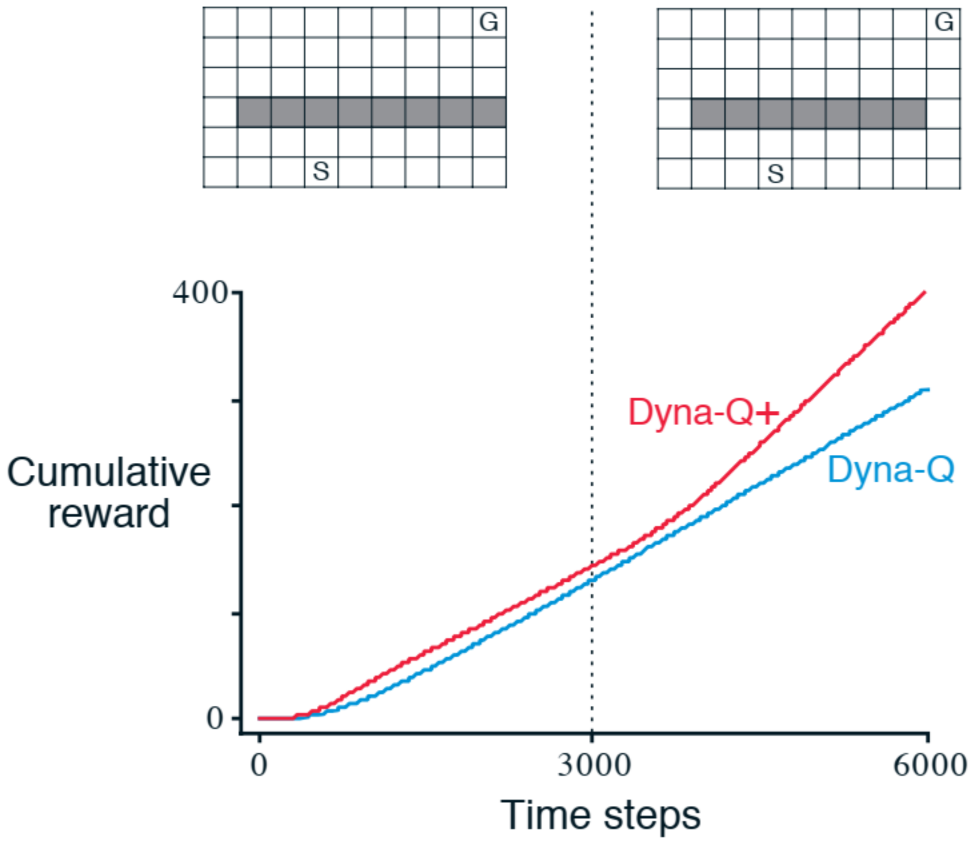
Dyna-Q vs Dyna-Q+ in a changing environment

- Average performance of Dyna agents on a **blocking task**.
- The left environment was used for the first **1000 steps**, the right environment for the rest.
- Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration.



Dyna-Q vs Dyna-Q+ in a changing environment

- Average performance of **Dyna agents** on a shortcut task.
- The left environment was used for the **first 3000 steps**, the right environment for the rest.



Space of Reinforcement Learning Methods

Space of reinforcement learning methods

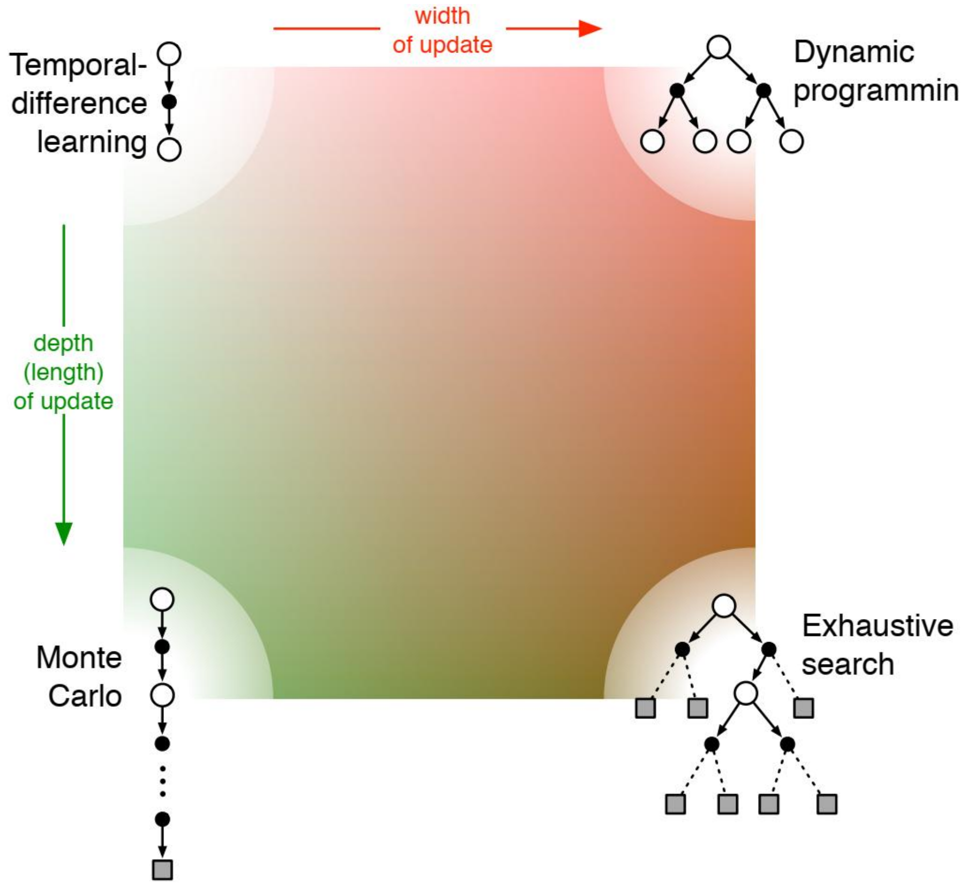
- All of the methods we have explored so far in this course have **three key ideas** in common:

  1) **Approximate value function:** they all seek to estimate value functions.
  2) **Approximate policy:** they all operate by backing up values along actual or possible state trajectories.
  3) **Generalized Policy Iteration GPI :** they all follow the general strategy of GPI i.e. they continually try to improve each on the basis of the other.

- Two of the most important **dimensions** along which the methods vary are shown in the following figure.

Space of reinforcement learning methods

A slice through the space of reinforcement learning methods, highlighting the two of the most important dimensions explored: the depth and width of the updates.



Other dimensions

- In addition to the three dimensions just discussed, we have identified a number of other dimensions :
  - **The definition of return:** (episodic or continuing, discounted or undiscounted)
  - **Action values vs. state values vs. after state.**
  - **Action selection/exploration.**
  - **Synchronous vs. asynchronous**
  - **Real vs. simulated**
  - **Location of updates**
  - **Timing of updates**
  - **Memory for updates**

Thank you!
Q/A