Reinforcement Learning

Lecture 11

Model-based RL

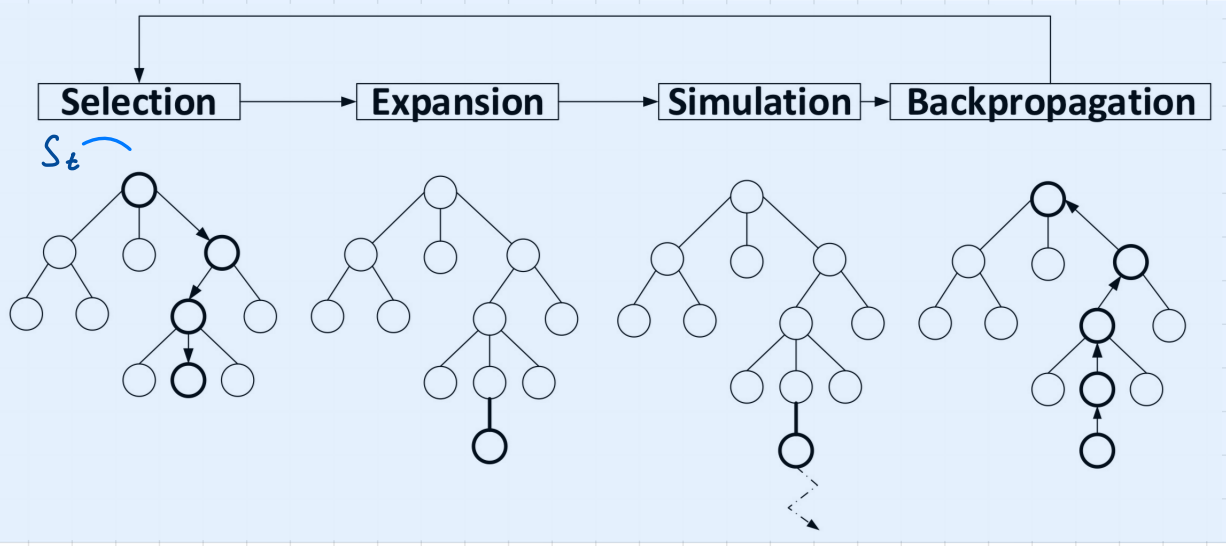
Fitting Tabular Models

Dyna-Q

Robert Peharz

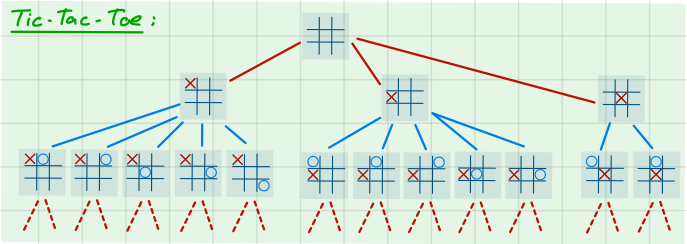
Institute of Theoretical Computer Science
Graz University of Technology
Winter Term 2023/24

Monte Carlo Tree Search



Game Tree

Backward Induction



Tree Policy, Upper Confidence Bound

$$\arg\max_a \; \hat{q}(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$$

Simulation, Rollouts

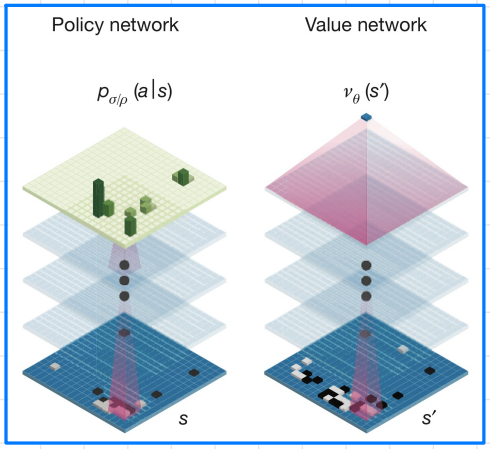
$$\hat{v}(s_{new}) = \frac{1}{N}\sum_{n=1}^{N} G_n$$

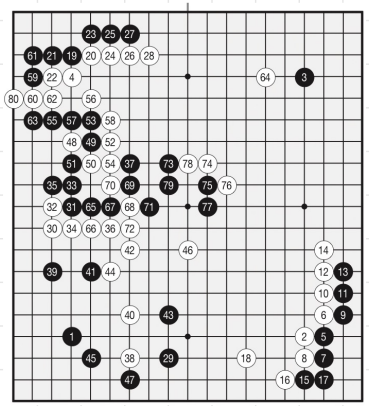
Backpropagation

$$N(s,a) \leftarrow N(s,a) + 1$$

$$\hat{q}(s,a) \leftarrow \hat{q}(s,a) + \frac{1}{N(s,a)}\left(\hat{v}(s_{new}) - \hat{q}(s,a)\right)$$

AlphaGo





- policy network $p_\sigma(a|s)$ trained on human data
- refined policy network $p_\rho(a|s)$ using REINFORCE
- value network $v_\theta(s)$ obtained from $p_\rho(a|s)$

- MCTS tree policy $a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$ $u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$ from p_σ
- value estimate $V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$ fast rollout policy

AlphaGo Zero

- single network $(p, v) = f_\theta(s)$
- MCTS interpreted as policy improvement: $\tilde{\pi}_a \propto N(s,a)^{1/\tau}$ (visit count, temperature)
- outcomes z of selfplay for estimating value function
- $(\boldsymbol{p}, v) = f_\theta(s)$ and $l = (z-v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$

Combining Planning and Learning

Planning vs. Learning

solving Bellman equations

Planning

use known MDP dynamics $p(s', r \mid s, a)$ → v_π, q_π (prediction); → π_* (control)

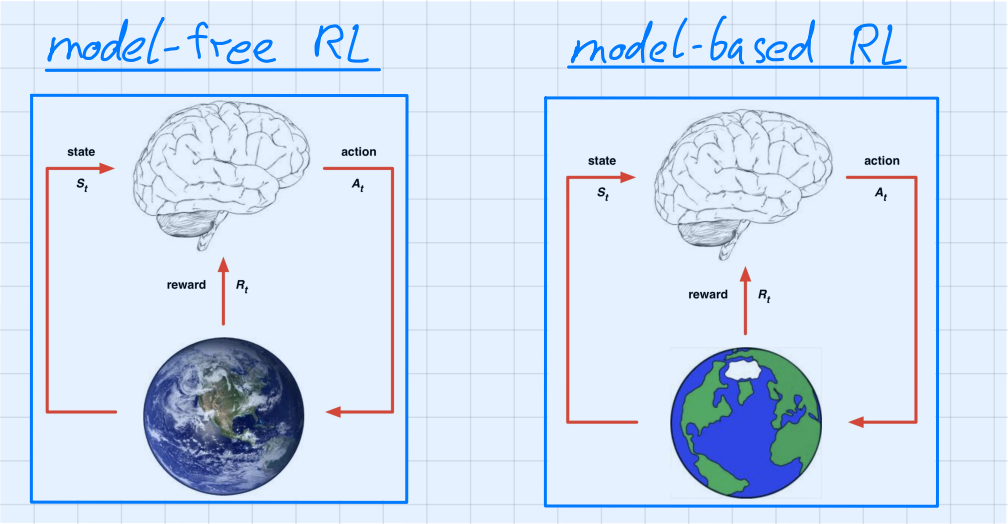
Monte Carlo, Temporal Differences, Policy Gradient

Model-free RL

use only samples from $p(s', r \mid s, a)$ → v_π, q_π (prediction); → π_* (control)

Model-based RL

use samples to (additionally) learn model, "get best of both worlds"



Images: D. Silver

Models

- the environment is governed by MDP dynamics $p(s', r \mid s, a)$
- thus, any representation which captures $p(s', r \mid s, a)$ (exactly or approximately) is a model of the environment/MDP

sample model

- can simulate samples $S', R \sim p(s', r \mid s, a)$
- generates "virtual" experience
- by generating many samples, one can fit a distribution model ⟶

distribution model

- represents conditional pmf/pdf $p(s', r \mid s, a)$
- can be used in dynamic programming
- often also allows sampling ⟵

- Black Jack with fixed dealer policy
- Go with fixed opponent policy
- physical simulation

  → cf. MCTS

- tabular pmf
- generative models, e.g. Bayesian networks, variational auto-encoders, mixture models, probabilistic circuits

Tabular Model

- assume finite $|\mathcal{S}|$ and $|\mathcal{A}|$
- assume simplified MDP representation (rather than $p(s', r \mid s, a)$)
  - $p(s' \mid s, a)$ — $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$ - table — state transition
  - $r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ — $|\mathcal{S}| \times |\mathcal{A}|$ - table — expected reward function
- assume data $\{(S_i, A_i, S'_i, R_i)\}_{i=1}^{N}$ of
  - current state S
  - action A
  - next state S'
  - reward R following S, A

→ e.g., sample S uniformly from $\mathcal{S}$ and A uniformly from $\mathcal{A}$
→ or explore with some policy π

Fitting Tabular Model

- in deterministic environments, i.e. when always the same s' and R follow on S, A, it is sufficient to construct a Lookup Table (LUT):

$$s'[S_i, A_i] \leftarrow S_i' \quad \text{when observing} \quad S_i', S_i, A_i$$

$$r[S_i, A_i] \leftarrow R_i \quad \text{when observing} \quad R_i, S_i, A_i$$

- in general $p(s'|s,a)$ is a conditional Categorical distribution which can be learned with maximum likelihood estimation

- for $r(s,a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ we can use Monte Carlo

Maximum Likelihood for Categorical Distributions

- maximum Likelihood fit of unconditional Categorical distribution:

$$\hat{p}(x) = \frac{\#\, x \text{ in data}}{N} = \frac{\sum_{i=1}^{N} \mathbb{I}[x_i = x]}{N}$$

(empirical frequencies)

E.g. assume data $\{3, 1, 1, 4, 3, 6, 6\}$ of a (biased) die.

Then $\hat{p}(1) = \frac{2}{7}$, $\hat{p}(2) = \frac{0}{7}$, $\hat{p}(3) = \frac{2}{7}$

$\hat{p}(4) = \frac{1}{7}$, $\hat{p}(5) = \frac{0}{7}$, $\hat{p}(6) = \frac{2}{7}$

- one might apply Laplace smoothing, i.e. adding a small pseudo-count:

$$\hat{p}(x) = \frac{\sum_{i=1}^{N} \mathbb{I}[x_i = x] + \varepsilon}{N + \varepsilon M}$$

$\varepsilon \geq 0$, M: number of states

$\longrightarrow$ bias towards uniform distribution

Fitting Tabular Model cont'd

- for $p(s'|s,a)$, we fit one Categorical for each s,a-pair:

$$\hat{p}(s'|s,a) = \frac{N(s',s,a)+\epsilon}{N(s,a)+|\mathcal{S}|\epsilon} = \frac{\sum_{i=1}^{N} \mathbb{I}[S_i'=s', S_i=s, A_i=a]+\epsilon}{\sum_{i=1}^{N} \mathbb{I}[S_i=s, A_i=a]+|\mathcal{S}|\epsilon}$$

avoids division by 0

- $r(s,a)$ estimated via Monte Carlo:

$$\hat{r}(s,a) = \frac{1}{N(s,a)} \sum_{i:\, S_i=s, A_i=a} R_i$$

average all rewards, which have followed whenever $S=s$ and $A=a$

Fitting Tabular Model - Online Version

online (incremental) averages:

$$\bar{x}_N = \frac{1}{N}\sum_{i=1}^{N} x_i = \bar{x}_{N-1} + \frac{1}{N}(x_N - \bar{x}_{N-1})$$

- init $N(s,a) = 0$, $\hat{p}(s'|s,a) = \frac{1}{|\mathcal{S}|}$ (uniform), $\hat{r}(s,a) = 0$ for all s, a
- when seeing s, a, r, s', compute

$$N(s,a) \leftarrow N(s,a) + 1$$

for $\kappa \in \mathcal{S}$:

$$\hat{p}(\kappa|s,a) \leftarrow \hat{p}(\kappa|s,a) + \frac{1}{N(s,a) + |\mathcal{S}|\epsilon}\left[\mathbb{1}[s' = \kappa] - \hat{p}(\kappa|s,a)\right]$$

$$\hat{r}(s,a) \leftarrow \hat{r}(s,a) + \frac{1}{N(s,a)}\left[r - \hat{r}(s,a)\right]$$

Fitting Tabular Model cont'd

- for any (s, a) pair for which $N(s, a) \xrightarrow[N \to \infty]{} \infty$ we have

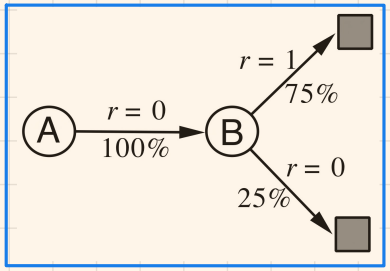
$$\hat{p}(s' \mid s, a) \longrightarrow p(s' \mid s, a)$$
$$\hat{r}(s, a) \longrightarrow r(s, a)$$

Monte Carlo, Law of Large Numbers

- estimated model can be used for dynamic programming, e.g. value iteration, etc.

Recall from Lecture 5

1: A, 0, B, 0
2: B, 1
3: B, 1
4: B, 0
5: B, 1
6: B, 1
7: B, 1
8: B, 1



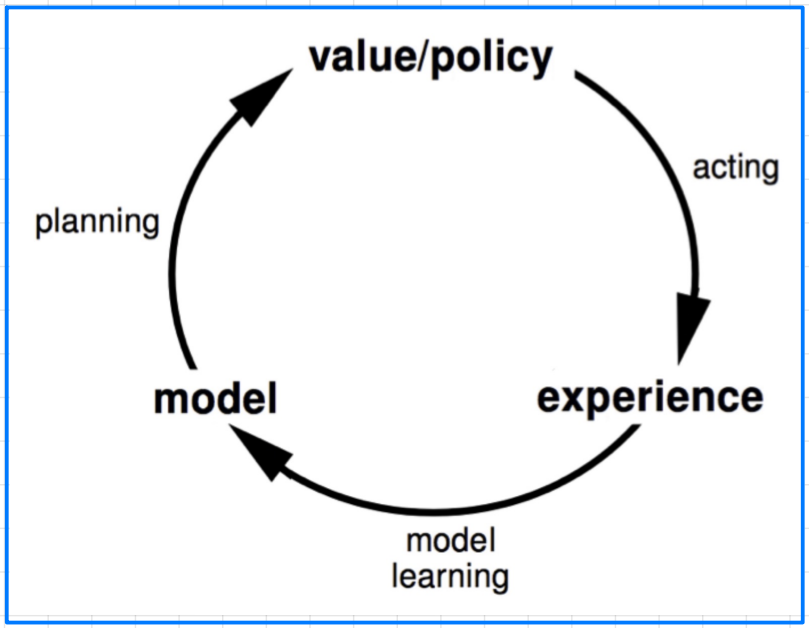
TD equivalent to maximum likelihood fit, followed by dynamic programming

- thus, TD can be seen as a "shortcut"
- however, models can be blended with model-free RL e.g. using Dyna-framework

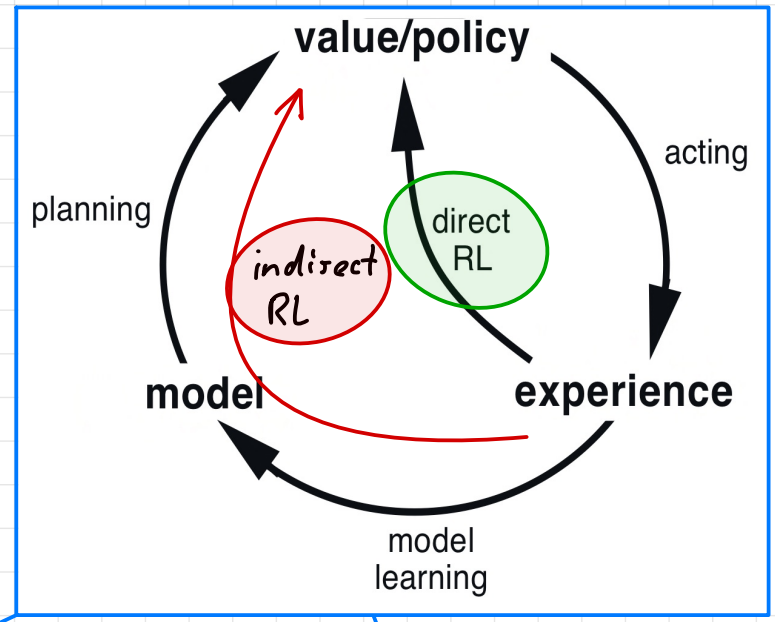
Dyna Framework

[Sutton, 1991]

Learning + Planning



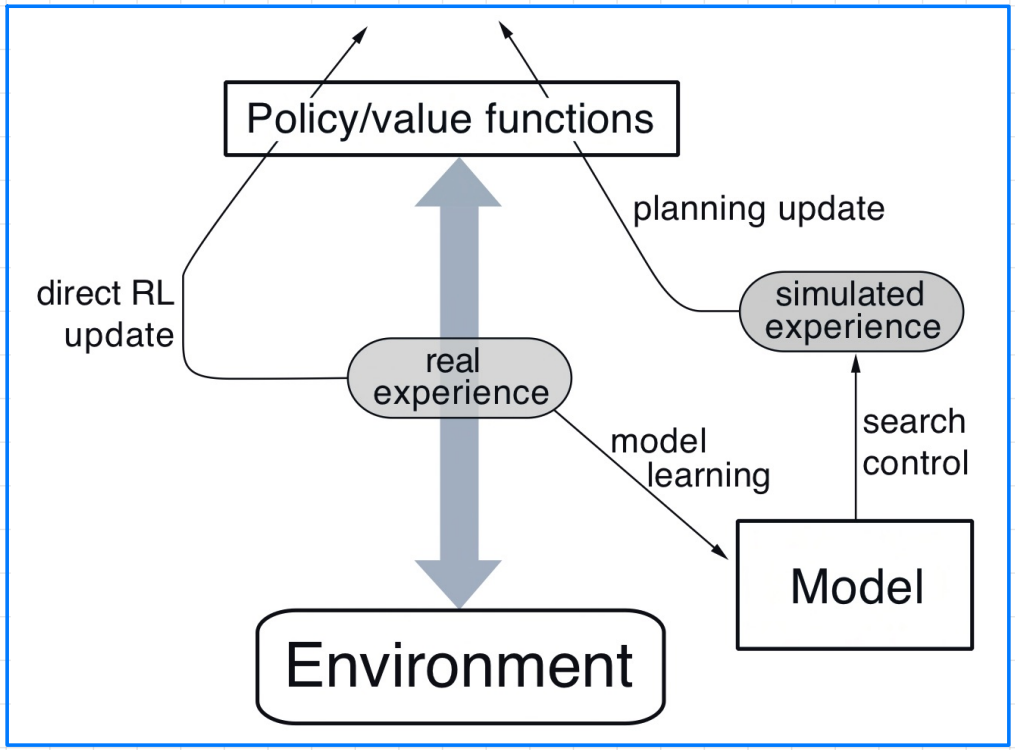
Dyna (from dynamic programming)



The main idea of Dyna is the old, commonsense idea that planning is 'trying things in your head,' using an internal model of the world (Craik, 1943; Dennett, 1978; Sutton & Barto, 1981). This suggests the existence of a more primitive process for trying things *not* in your head, but through direct interaction with the world. *Reinforcement learning* is the name we use for this more primitive, direct kind of trying, and Dyna is the extension of reinforcement learning to include a learned world model.

- model-free RL = direct RL
  - simple
  - not affected by model bias
- model-based RL = indirect RL
  - more data-efficient

Dyna Framework: Schematic



- real experience gathered by interacting with enviroment
  - → used for direct RL (Sarsa, Q-learning, Policy Gradient, etc.)
  - → also used for model learning (plug in learning algorithm)
- model is used to generate simulated experience
  - → used for planning / indirect RL

Dyna-Q

(Q-learning, tabular case, deterministic environment)

initialize $q(s,a)$ and models (LUTs) $s'[s,a]$, $r[s,a]$

initialize S

repeat // for each episode

$A \leftarrow \varepsilon\text{-greedy}(q(s,\cdot))$

take action A, observe S', R

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

// standard Q-learning, direct RL

$s'[S,A] = S'$

$r[S,A] = R$

// modeling (simple in this case)

$S \leftarrow S'$

repeat n times // indirect RL

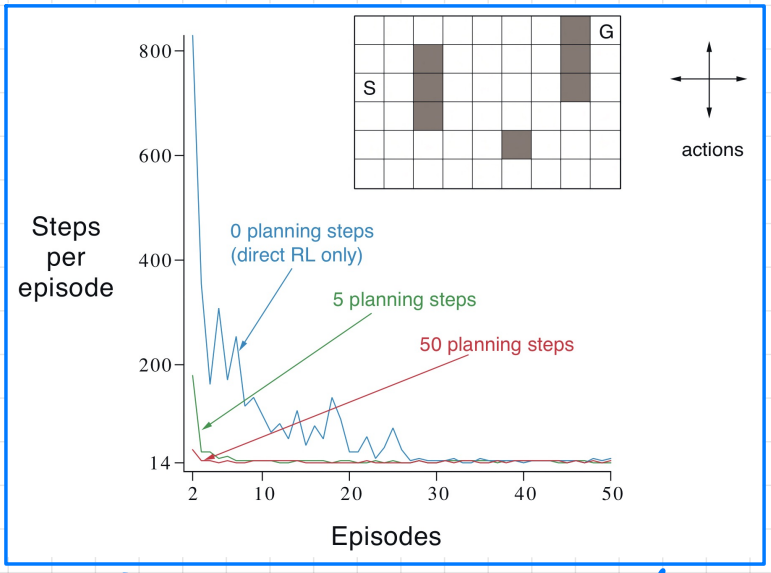
$\tilde{S}, \tilde{A} \leftarrow$ random previously observed state-action pair

$\tilde{R}, \tilde{S} \leftarrow r[\tilde{S},\tilde{A}],\ s'[\tilde{S},\tilde{A}]$

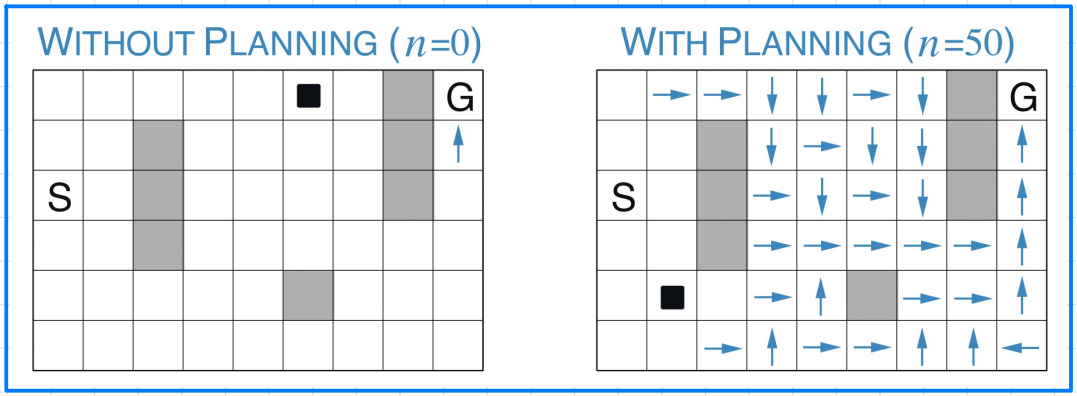
$q(\tilde{S},\tilde{A}) \leftarrow q(\tilde{S},\tilde{A}) + \alpha\left[R + \gamma \max_a q(\tilde{S}',a) - q(\tilde{S},\tilde{A})\right]$

Gridworld Maze

- $\gamma = 0.95$, $\alpha = 0.1$, $\varepsilon = 0.1$
- 0 reward, but 1 reward in goal state



performance over episodes

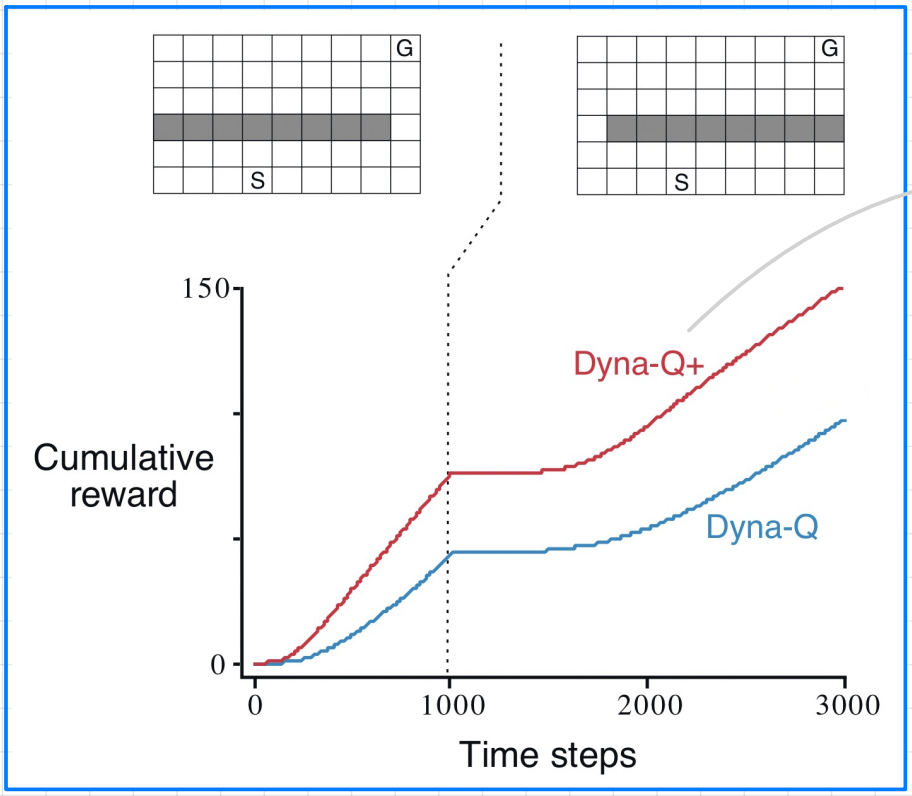


policy found in middle of 2nd episode

- note similarity with experience replay (e.g. Deep-Q learning)

When the Model is Wrong

(and when the situation becomes worse)

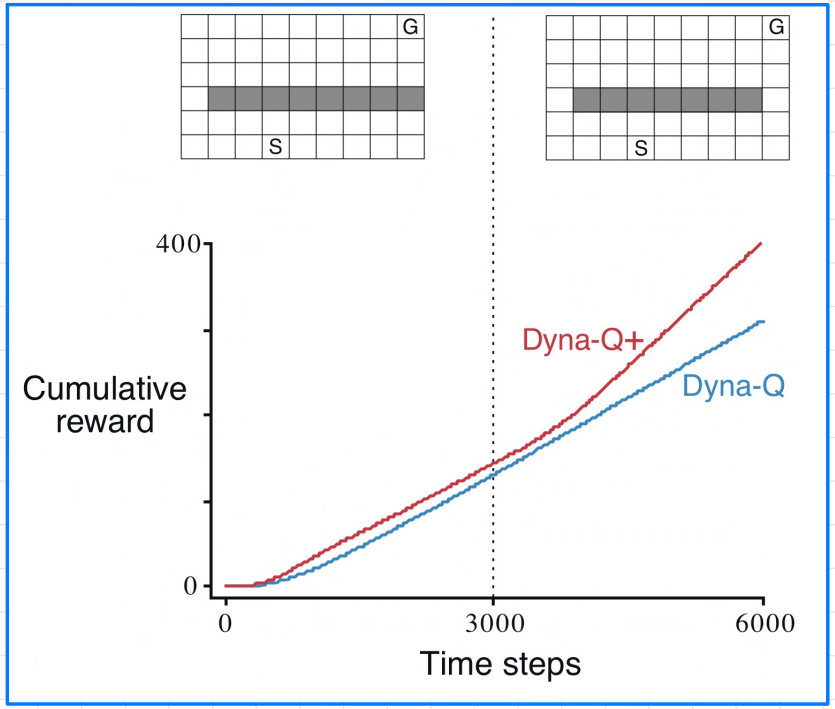


explained in the following

- assume grid world with "quick path" to the goal
- after some time, quick path closes and a longer path opens
- model will adapt and after some time find the new path

When the Model is Wrong

(and when the situation becomes better)



- assume a long path to the goal
- after some time, an additional short path opens
- however, the model's experience discourages exploration in this area
- finding the shortcut will take long (via ε-greedy)
- instance of exploration-exploitation trade-off
- can we incorporate exploration in the model?

Dyna-Q+

- modification of Dyna-Q to stimulate exploration
- for each state-action pair s,a, keep track of number of elapsed time steps τ since it was tried the last time
- add a "bonus reward" of $\kappa\sqrt{\tau}$, where $\kappa > 0$ is a small constant
- "computational curiosity"
- Principle: Optimism in face of Uncertainty
- the longer back the last visit, the more likely the state-action pair will be tried again

compare with tree policy (MCTS)

$$\arg\max_a \hat{q}(s,a) + C\sqrt{\frac{\log N(s)}{N(s,a)}}$$