UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3547 Intelligent Agents & Reinforcement Learning

Module 8: Temporal Difference Learning

Course Plan

Module Titles
Module 1 – Introduction to Intelligent Agents
Module 2 – Search
Module 3 – Logical Inference
Module 4 – Planning and Knowledge Representation
Module 5 – Probabilistic Reasoning
Module 6 – Intro to Reinforcement Learning and Finite Markov Decision Processes
Module 7 – Dynamic Programming and Monte Carlo Methods
Module 8 – **Current Focus: Temporal Difference Learning**
Module 9 – Function Approximation for RL
Module 10 – Deep Reinforcement Learning and Policy Gradient Methods
Module 11 – Introduction to Advanced DRL
Module 12 – Presentations (no content)

Learning Outcomes for this Module

- Use Temporal Difference Learning to update value functions on-the-fly
- Apply the TD idea to learning estimates of v, q_π and q_*
- Use n-step Learning to increase learning rates
- Integrate Planning and Learning into a single framework

Topics for this Module

- **8.1** Temporal-Difference Learning
- **8.2** n-step Learning
- **8.3** Integrated Planning and Learning
- **8.4** Resources and Wrap-up

Module 8 – Section 1

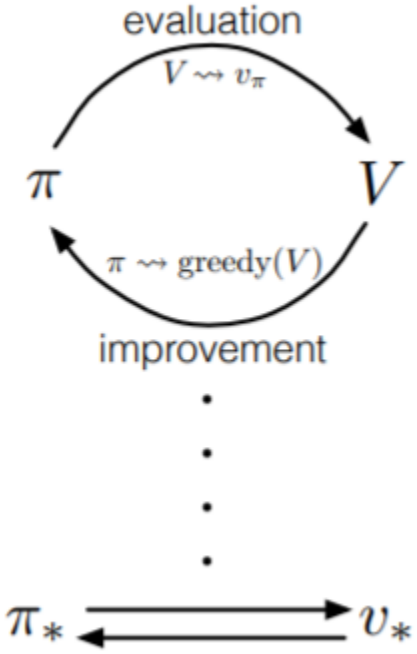
Temporal-Difference Learning

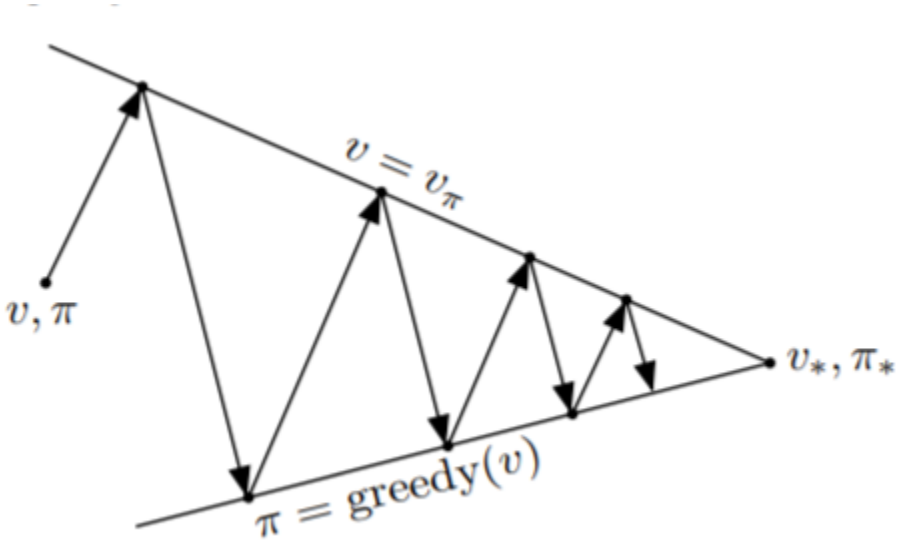
Review of the Four Elements

- **Policy**
  - $\pi(a|s)$
- **Reward Signal**
  - R
  - Total reward: G
- **Value Function**
  - Value of a state: $v_\pi(s) = E_\pi(G_t|S_t = s)$
  - Value of a state-action: $q_\pi(s,a) = E_\pi(G_t|S_t = s, A_t = a)$
- **Model of the Environment**
  - $p(s', r|s, a) = \Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$

Review: Generalized Policy Iteration

- Sutton and Barto call this notion of interacting policy evaluation and improvement **generalized policy iteration**





Source: Sutton & Barto

Review: Monte Carlo Prediction

- In Monte Carlo Prediction, we sampled possible trajectories from a state S_t and updated our estimate of the state's value $V(S_t)$ based on the total reward received along that trajectory:

  $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$ where α is the learning rate

- This is the best we can do if we don't get any reward until the end of the episode

TD Prediction

- What if we get useful return information before the end of an episode?
- **Temporal-Difference Learning** is a generalization of Monte Carlo to make use of reward information without requiring completion of an episode:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- We can update $V(S_t)$ as early as $t + 1$
- Note that here the update rule for $V(S_t)$ makes use of another estimate $V(S_{t+1})$: this is called **bootstrapping**

TD(0) Prediction Algorithm

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Source: Sutton & Barto

Advantages of TD Prediction Methods

- Incremental/bootstrapping so they don't require full rollouts (like MC does)
- Don't require a model (like DP does)
- Often converge more quickly than MC

Sarsa: On-policy TD Control

- Let's say we are in an online learning situation:
  - The agent selects an action A_t for its state S_t according to its current policy
  - The environment responds by placing the agent in a new state S_{t+1} and possibly handing out a reward R_{t+1}
  - The agent will take the action A_{t+1} at that point, again according to its policy
- We can improve our value function with this experience:
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
  (which we can use to improve our policy)
- Called SARSA because it is dependent on
  $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Sarsa Algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Source: Sutton & Barto

Q-Learning: Off-Policy TD Control

- Maybe there's a $Q(S_{t+1}, a)$ for some a that is larger than the one our policy currently recommends
- Let's use that value to update $Q(S_t, A_t)$ even though it isn't the action we're actually going to take:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- Still converges to q_*

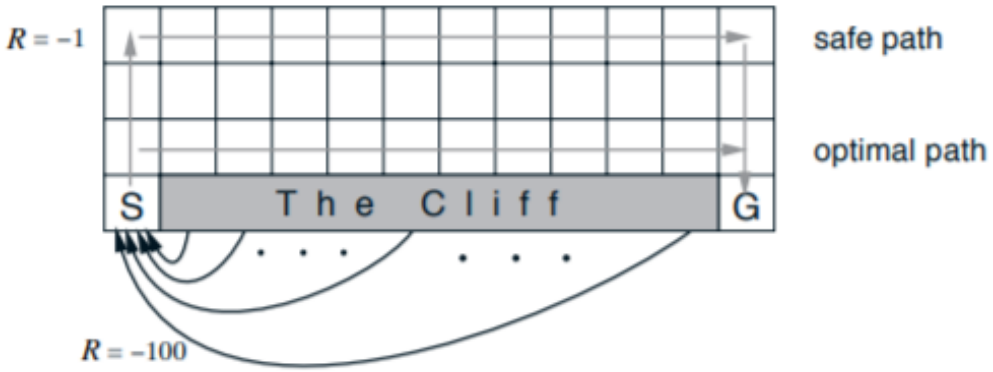
Q-Learning Algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Source: Sutton & Barto

Comparing Sarsa and Q-Learning



Source: Sutton & Barto

Expected Sarsa

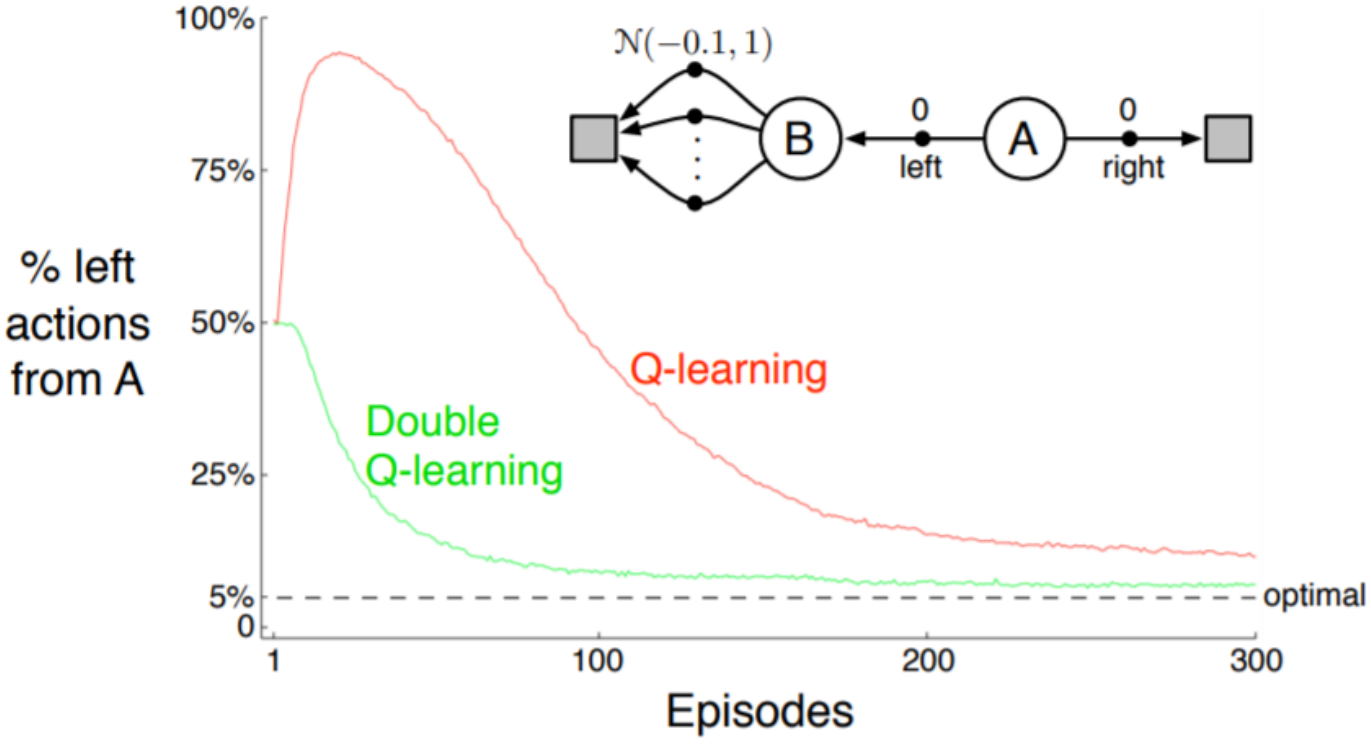
- What if we follow the Q-Learning algorithm but update with the expected value across all the possible next states at each step, rather than just the largest

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma E_\pi(Q(S_{t+1}, A_{t+1})|S_{t+1}) - Q(S_t, A_t)]$$

- Since we get some averaging from the E_π we can safely set α to 1

Maximization Bias

- Greedy algorithms have a tendency to over-value and over-visit states that may pay off poorly on average but occasionally pay well:



Source: Sutton & Barto

Double Learning

- The solution is to use two Q functions and randomly use one for the update and other for the action selection:

Double Q-learning

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

    else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Source: Sutton & Barto

Module 8 – Section 2

n-step Learning

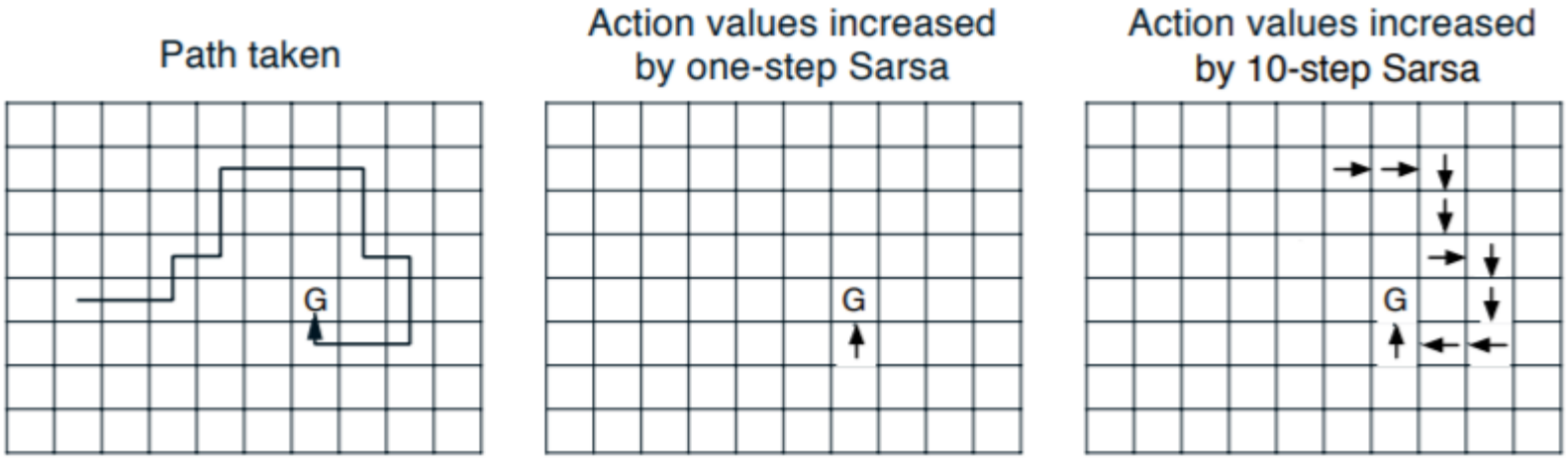
n-step Learning

- Sarsa

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- n-step Sarsa

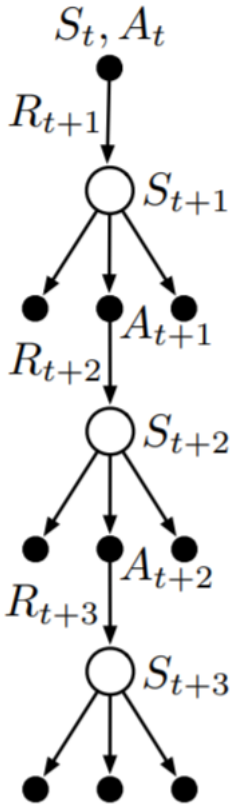
$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$



Source: Sutton & Barto

n-step Tree Learning

- Why limit the backups to the n-step actual rewards when we can also use the existing Q estimates from those states, weighted by the probability of selecting them under a policy we're exploring?



Source: Sutton & Barto

Module 8 – Section 3

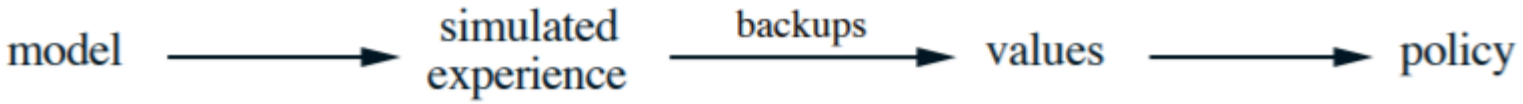
Integrated Planning and Learning

Planning and Learning

- Model-based methods
  - Dynamic Programming
  - Heuristic search
- Model-free methods
  - Monte Carlo
  - Temporal-Difference
- These approaches have a lot in common

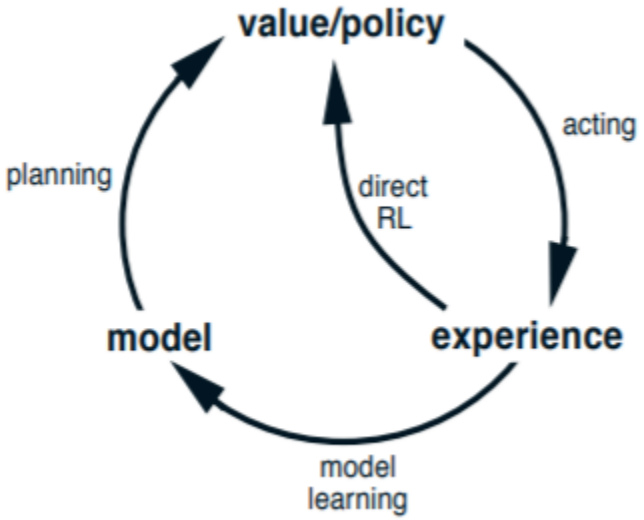
Models and Planning

- Model types
  - Distribution model: Assumed for Dynamic Programming
  - Sample models
- Models allow us to simulate experience
- In RL, the term **planning** means using a model to establish or improve a policy



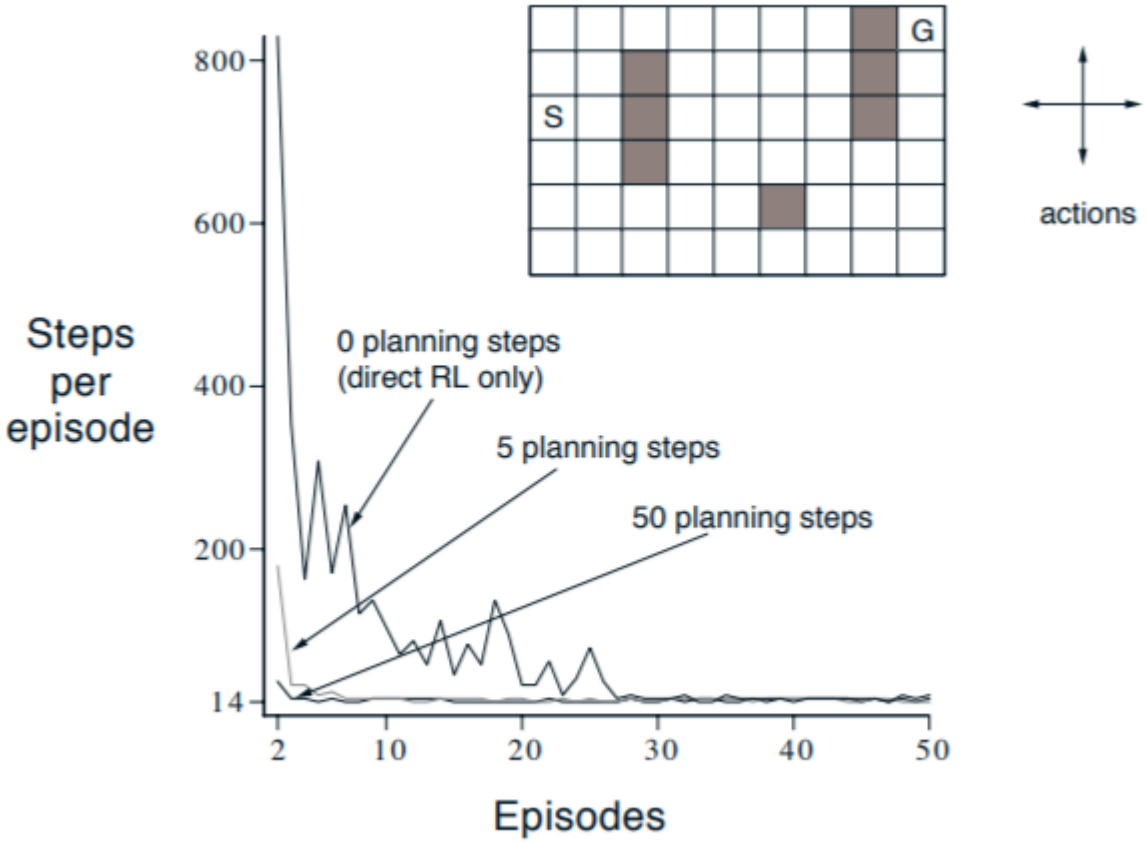
Dyna-Q

- Why not learn $p(s, a)$ from experience if we don't have it?
- Build a table of experience: a list or distribution of (next state, reward) pairs observed for each (state, action) taken while acting in the real environment
- Then combine it with Q-learning



Source: Sutton & Barto

Dyna-Q



Source: Sutton & Barto

Tabular Dyna-Q Algorithm

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

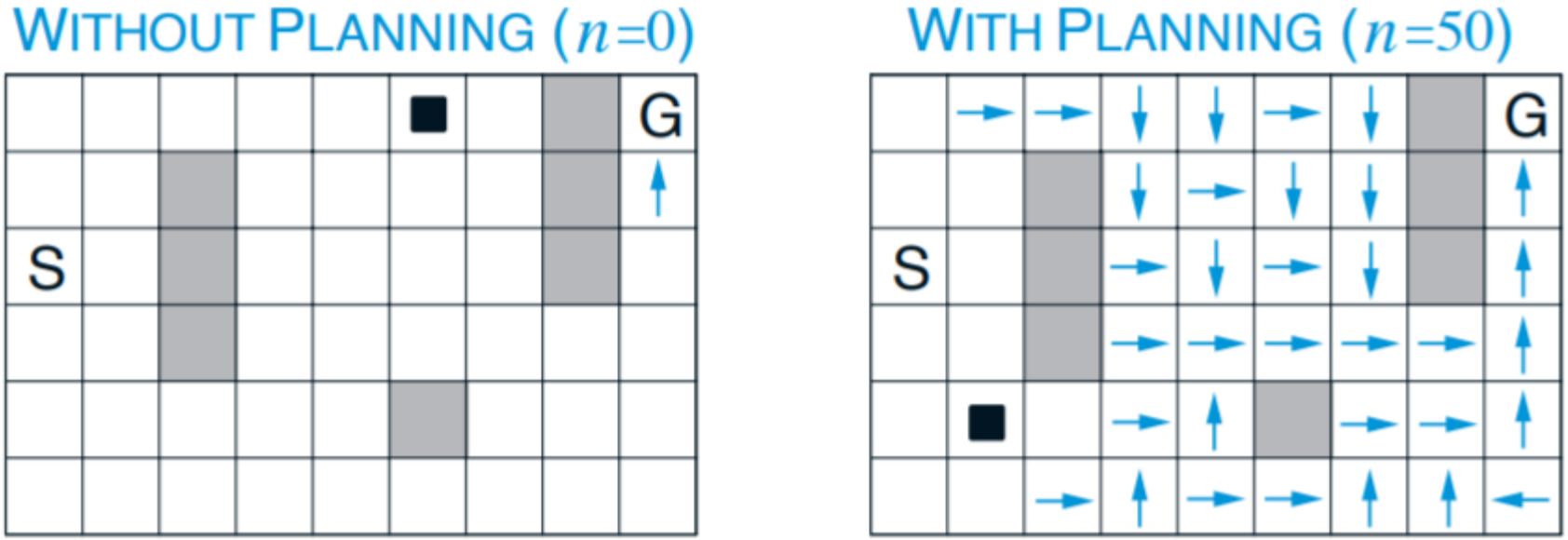
$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

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

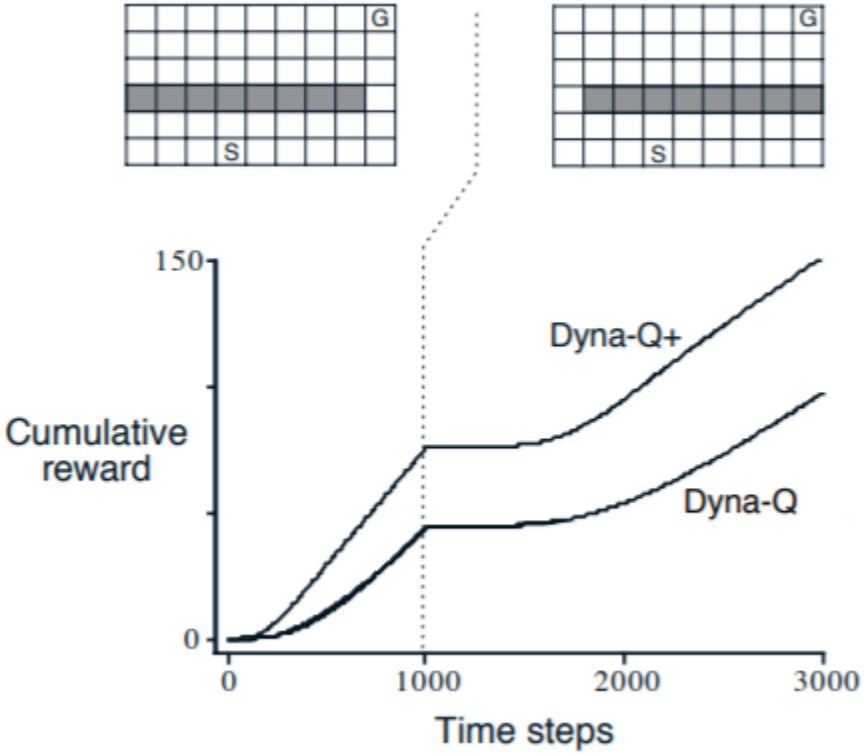
Source: Sutton & Barto

How Planning Accelerates Learning



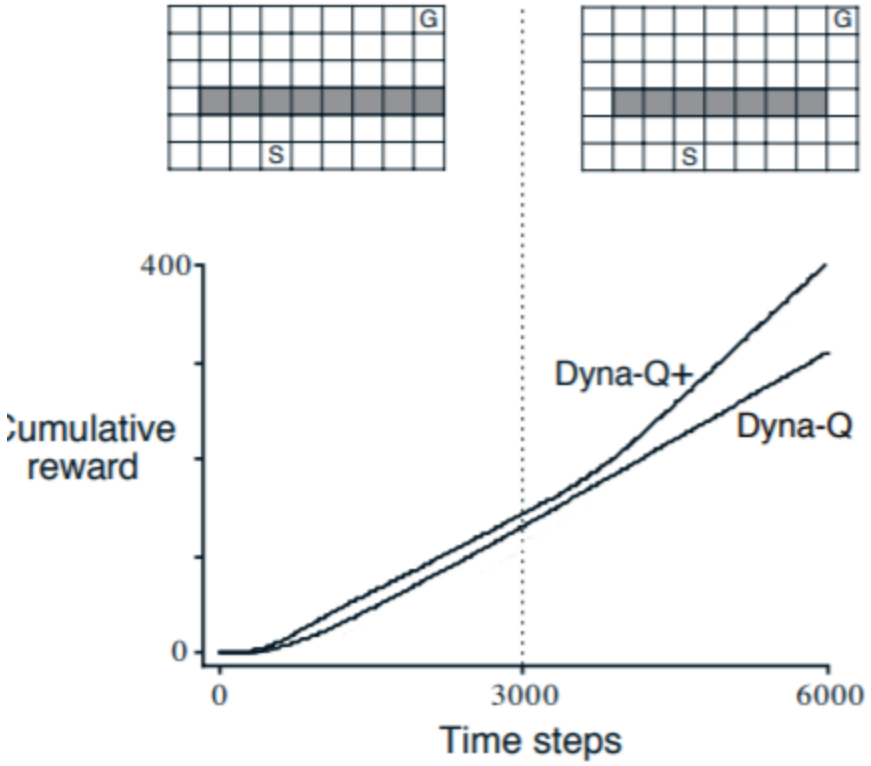
Source: Sutton & Barto

What if the Learned Model Is Now Wrong?



Source: Sutton & Barto

What if the Situation has Improved?

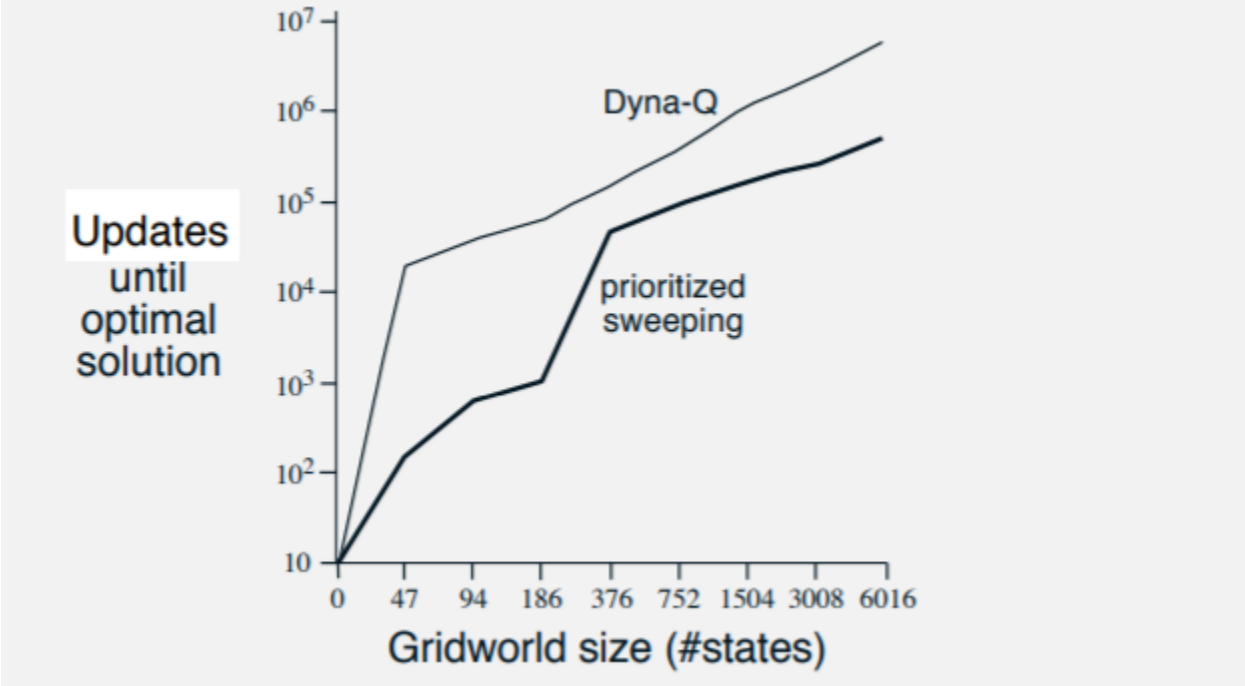


Source: Sutton & Barto

Prioritized Sweeping

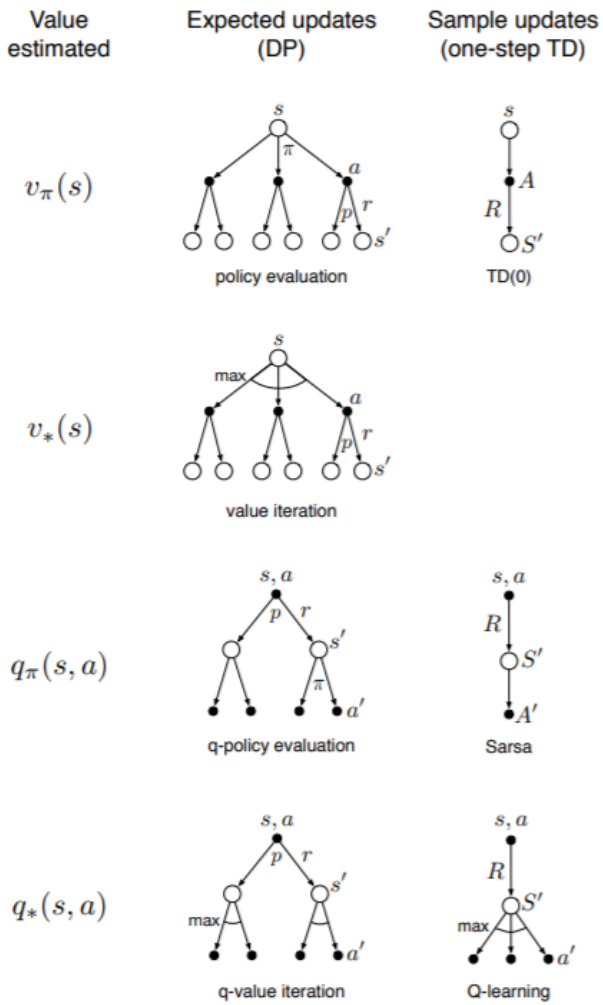
- We'd like to focus our updates where they'll do the most good
- The notion here is to maintain an update priority queue based on the idea that state-actions whose Q have changed a lot are most likely to change π at the states leading into them

Prioritized Sweeping Improvement Over Dyna-Q



Source: Sutton & Barto

Expected vs. Sample Updates



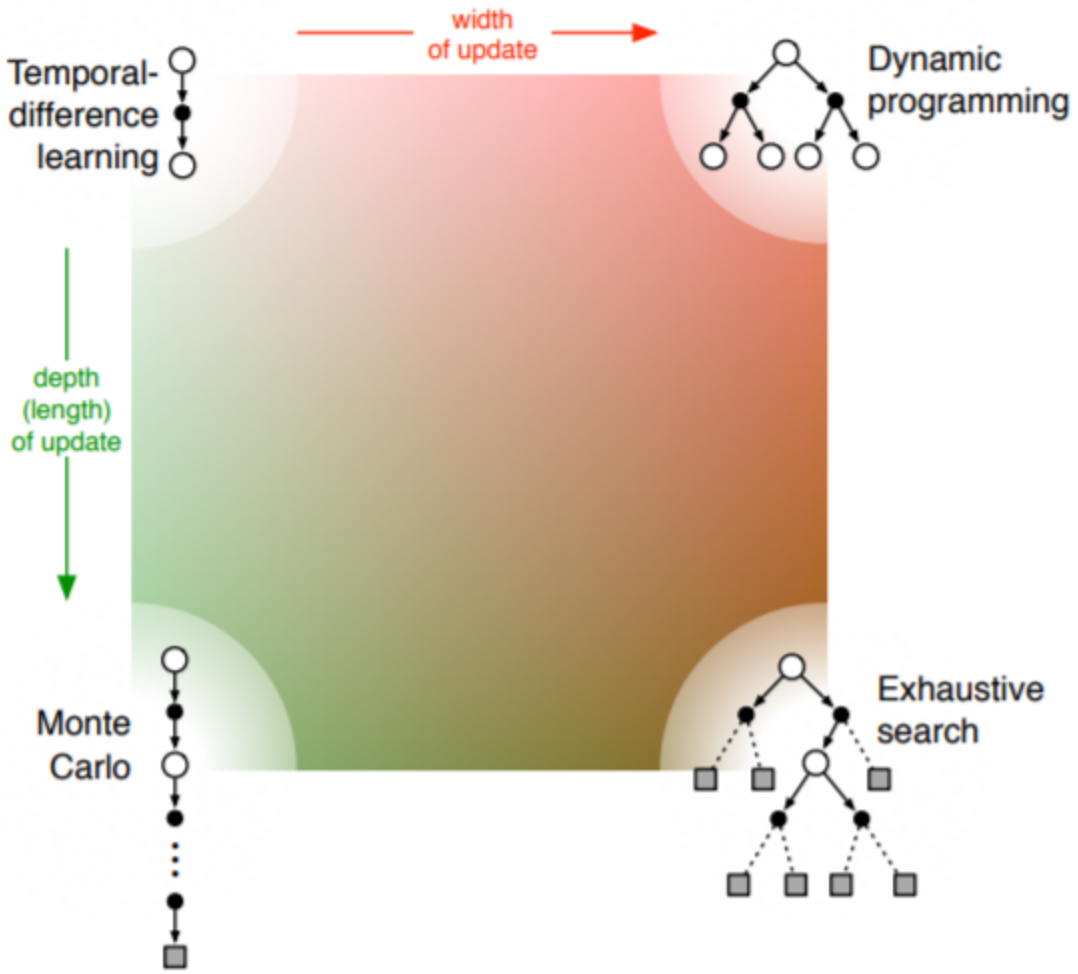
Source: Sutton & Barto

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 8 – Section 4

Resources and Wrap-up

Summary



Source: Sutton & Barto

Resources

- Sutton & Barto. Reinforcement Learning, 2nd Ed. MIT Press. 2018.
- Richard Sutton's website: http://incompleteideas.net/
- https://gym.openai.com/
- Pumperla & Ferguson. Deep Learning and the Game of Go, Chap. 9. Manning. 2019.
- Double Learning: https://www.scirp.org/pdf/jdaip_2016101714072270.pdf

Next Week

- If the state space is large we can't possibly work with it as a table, so: Can we learn to approximate the value functions using models we know from Machine Learning?
- Spoiler: Yes!

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies