SIT796 Reinforcement Learning

Eligibility Traces

Presented by:
Thommen George Karimpanal
School of Information Technology

Temporal Difference (TD) Learning

DEAKIN UNIVERSITY

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop 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

Combination of MC sampling and bootstrapping

The general name is TD(λ). This just corresponds to $\lambda = 0$

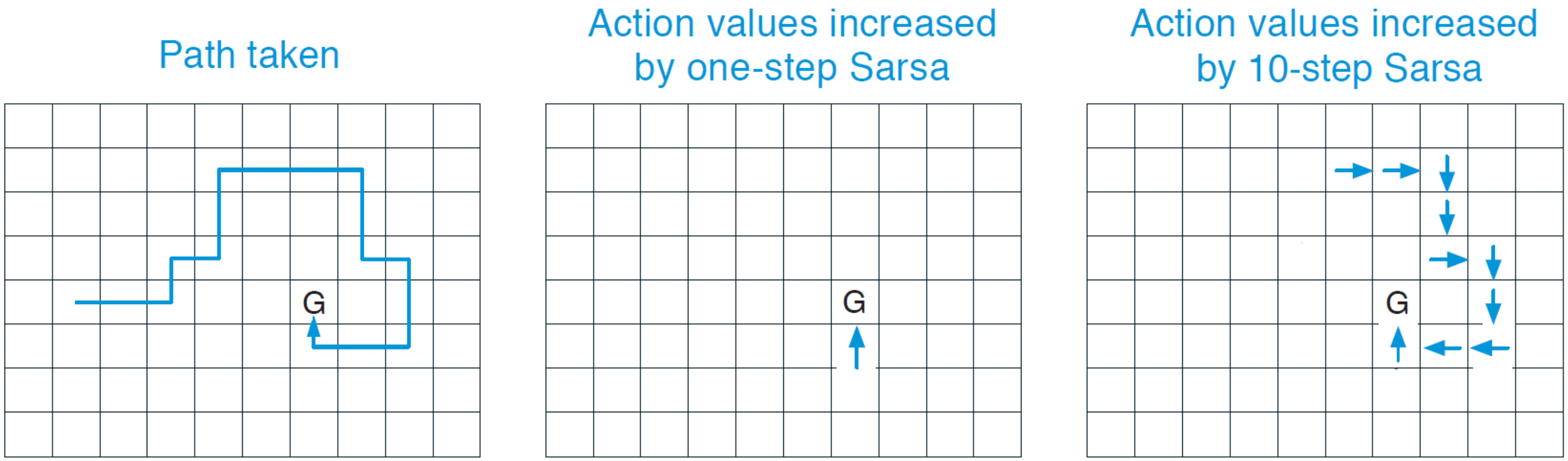
The TD target $R + \gamma V(S')$ is a combination of both samples R and an estimate $\gamma V(S')$

Sutton and Barto (2018)

Example: Gridworld with n-step Sarsa

In Searching for a goal using on-policy control

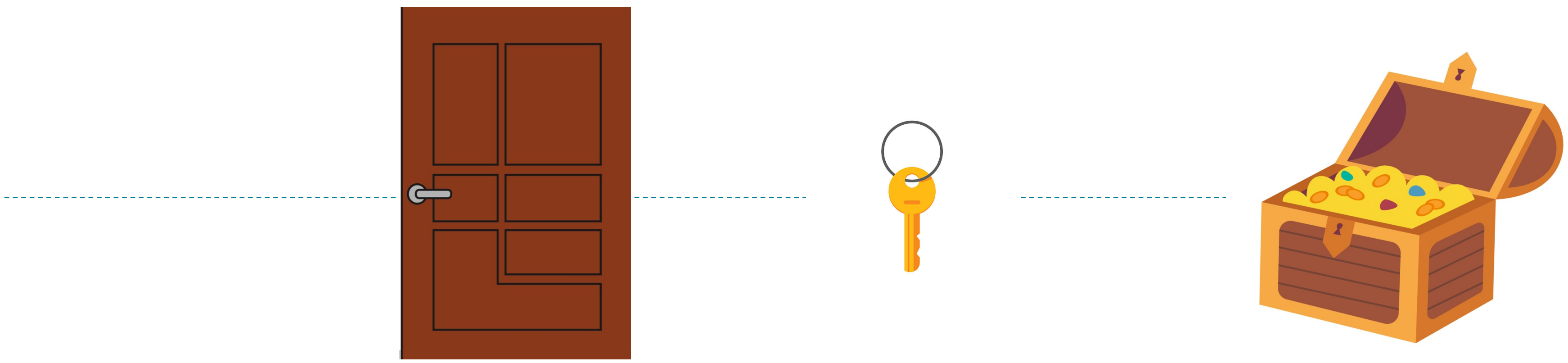
- The path taken which are also all the states that will be backed up using MC
- The state-action's learnt using a one-step Sarsa and 10-step Sarsa
- It is evident that 10 step Sarsa will learn more state-action values.



Sutton and Barto (2018)

Eligibility Traces

It would be useful to extend what learned at t+1 also to previous states, so to accelerate learning.

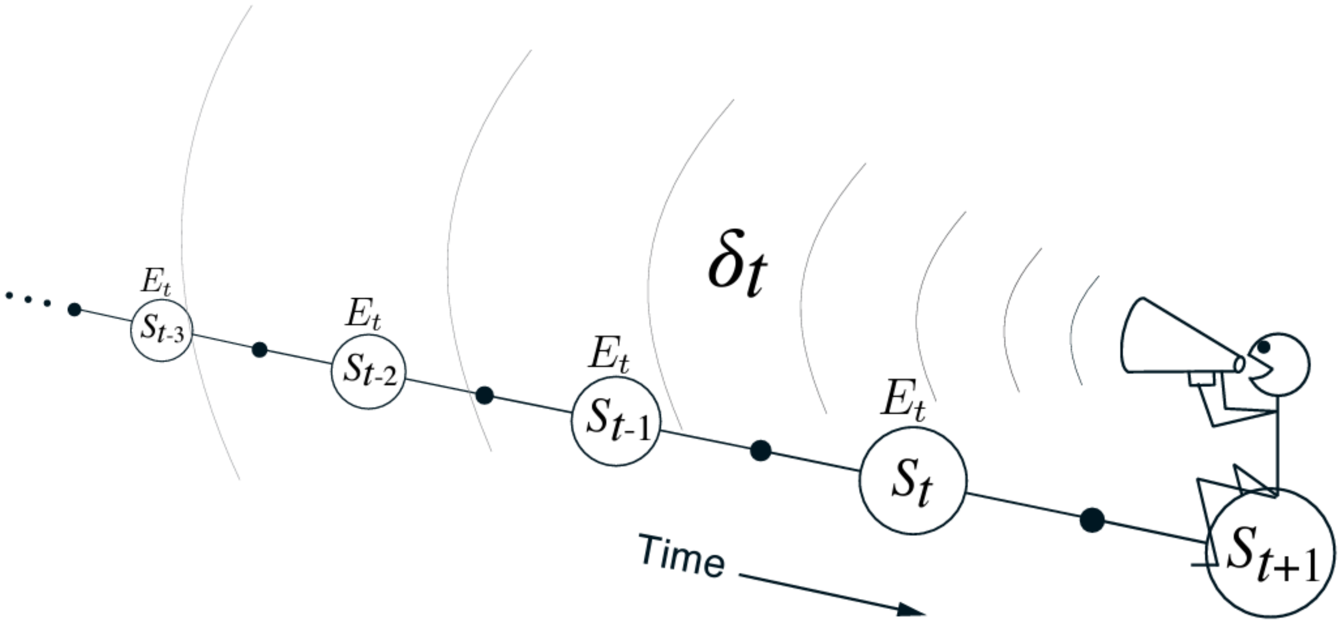


Key idea – when performing the update, instead of updating just the current state, why not also update states that led to that state?

Eg: Key helped unlock the treasure, but door led to the key. So when updating the the value of the 'Key' state, make sure some of the credit goes to the 'Door' state as well

Sutton and Barto (2018)

Eligibility Traces: the backward view



Sutton and Barto (2018)

Eligibility Traces

Eligibility trace values of non-visited traces decay by a factor of λ $$E_t(s) = \gamma\lambda E_{t-1}(s), \qquad \forall s \in \mathcal{S}, s \neq S_t,$$

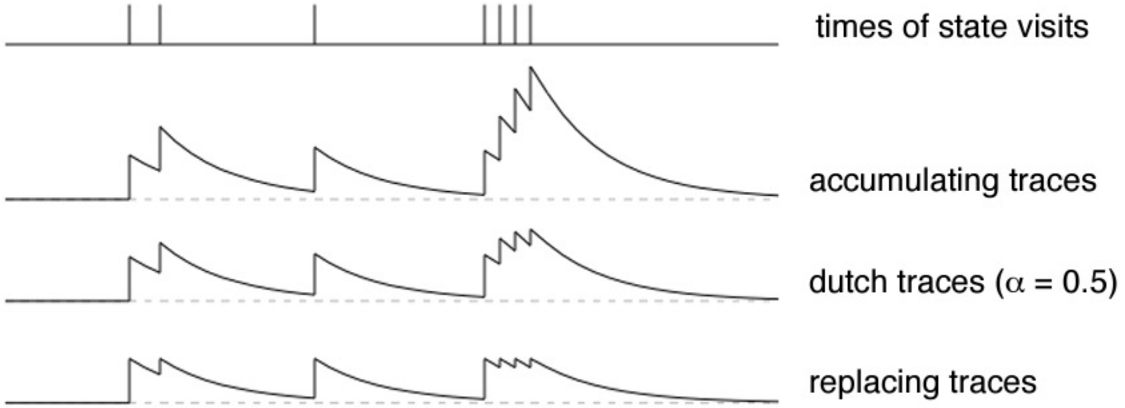
If a state is visited, increase its trace value:

Accummulating traces: $$E_t(S_t) = \gamma\lambda E_{t-1}(S_t) + 1$$

Dutch traces: $$E_t(S_t) = (1-\alpha)\gamma\lambda E_{t-1}(S_t) + 1$$

Replacing traces: $E_t(S_t) = 1$

Eligibility traces essentially serve as a sort of memory of which states are relevant to the current update. The end result is convergence speeds up.



Sutton and Barto (2018)

Learning with Eligibility Traces

Initialize $V(s)$ arbitrarily (but set to 0 if s is terminal)
$V_{\text{old}} \leftarrow 0$
Repeat (for each episode):
  Initialize $E(s) = 0$, for all $s \in \mathcal{S}$
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe reward, R, and next state, S'
    $\Delta \leftarrow V(S) - V_{\text{old}}$
    $V_{\text{old}} \leftarrow V(S')$
    $\delta \leftarrow R + \gamma V(S') - V(S)$
    $E(S) \leftarrow (1 - \alpha)E(S) + 1$
    For all $s \in \mathcal{S}$:
      $V(s) \leftarrow V(s) + \alpha(\delta + \Delta)E(s)$
      $E(s) \leftarrow \gamma\lambda E(s)$
    $V(S) \leftarrow V(S) - \alpha\Delta$
    $S \leftarrow S'$
  until S is terminal

Set $\lambda = 0$, we get TD(0)

$\lambda = 1$ corresponds to Monte-Carlo

Learning with Eligibility Traces: SARSA(λ)

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    Initialize S, A
    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)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $E(S,A) \leftarrow E(S,A) + 1$ (accumulating traces)
        or $E(S,A) \leftarrow (1-\alpha)E(S,A) + 1$ (dutch traces)
        or $E(S,A) \leftarrow 1$ (replacing traces)
        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
            $E(s,a) \leftarrow \gamma \lambda E(s,a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

Learning with Eligibility Traces: Q(λ)

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    Initialize S, A
    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)
        $A^* \leftarrow \arg\max_a Q(S', a)$ (if A' ties for the max, then $A^* \leftarrow A'$)
        $\delta \leftarrow R + \gamma Q(S', A^*) - Q(S,A)$
        $E(S,A) \leftarrow E(S,A) + 1$ (accumulating traces)
        or $E(S,A) \leftarrow (1-\alpha)E(S,A) + 1$ (dutch traces)
        or $E(S,A) \leftarrow 1$ (replacing traces)
        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
            If $A' = A^*$, then $E(s,a) \leftarrow \gamma \lambda E(s,a)$
                else $E(s,a) \leftarrow 0$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

SIT796 Reinforcement Learning

Planning, Learning and Dyna

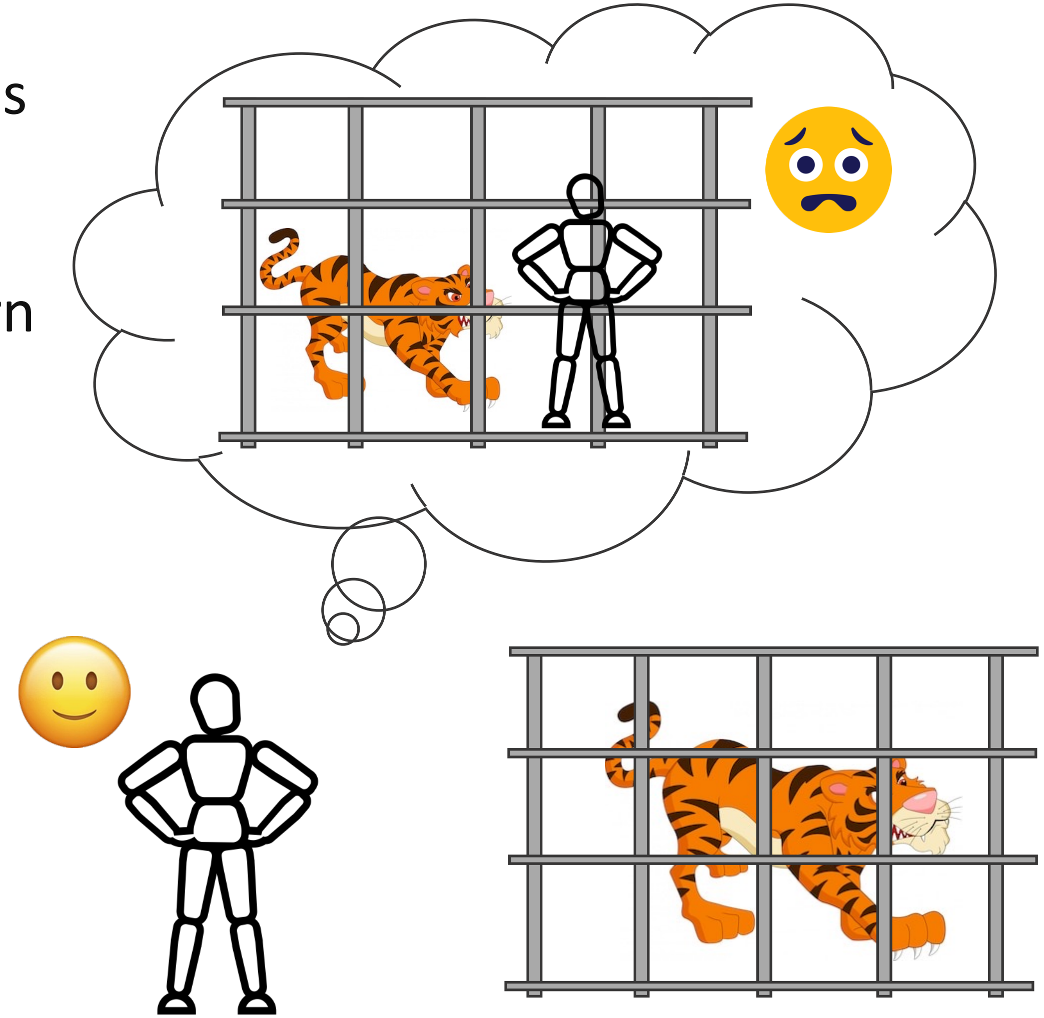
Presented by:
Thommen George Karimpanal
School of Information Technology

Learning with a model

So far we discussed about learning from interactions

We don’t actually need to experience things to learn – we can imagine!

We imagine by building and playing out models of the world, build from interaction data



Learning with a model

Updating values with interactions with the real world: Learning

-Real world interactions are expensive

- Robots/environment can get damaged

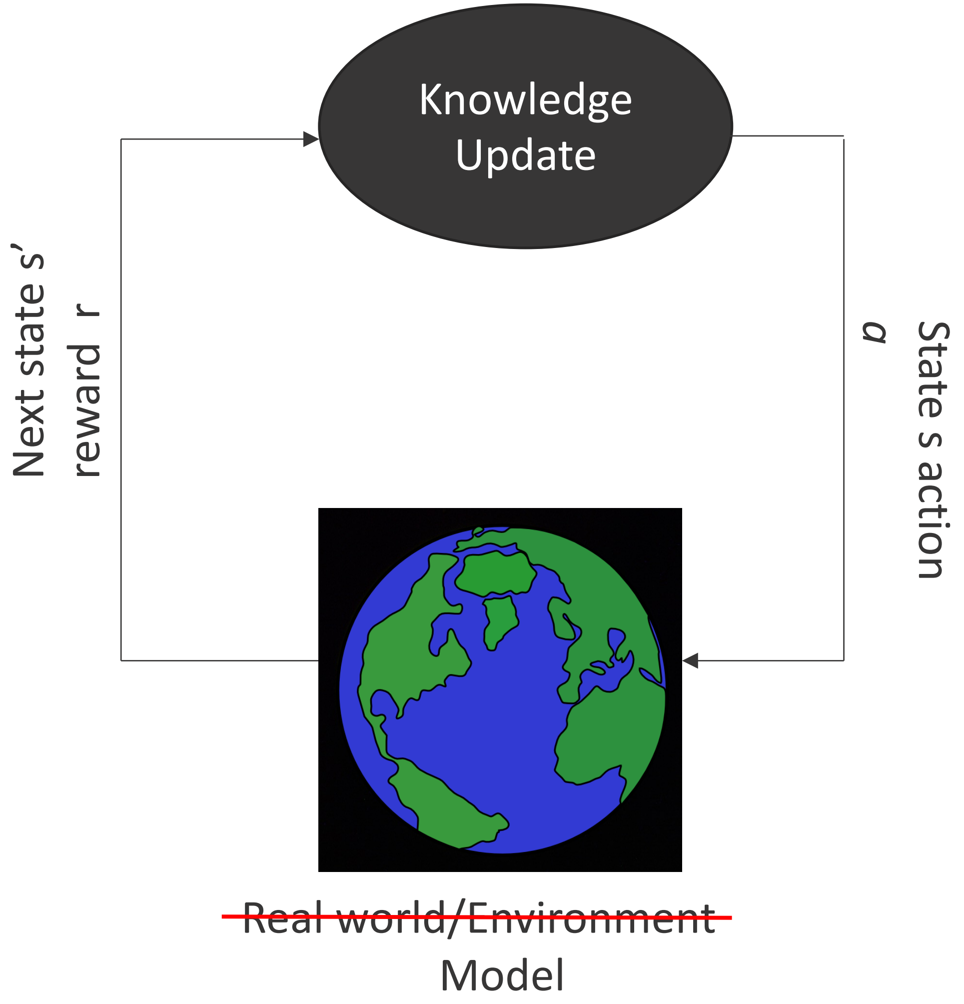
- Experiments can take time

Updating values with interactions with the real world: Planning

-But generally, our model is not perfect

- If it was we would never really need to interact with the real world

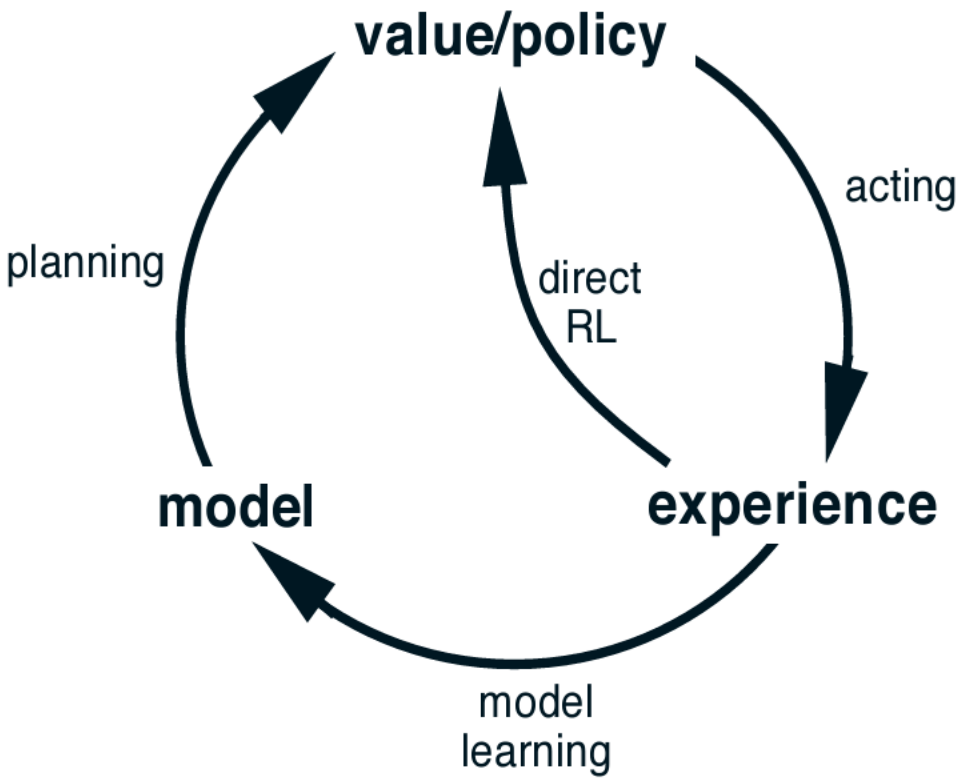
- Models are built using interactions in the first place



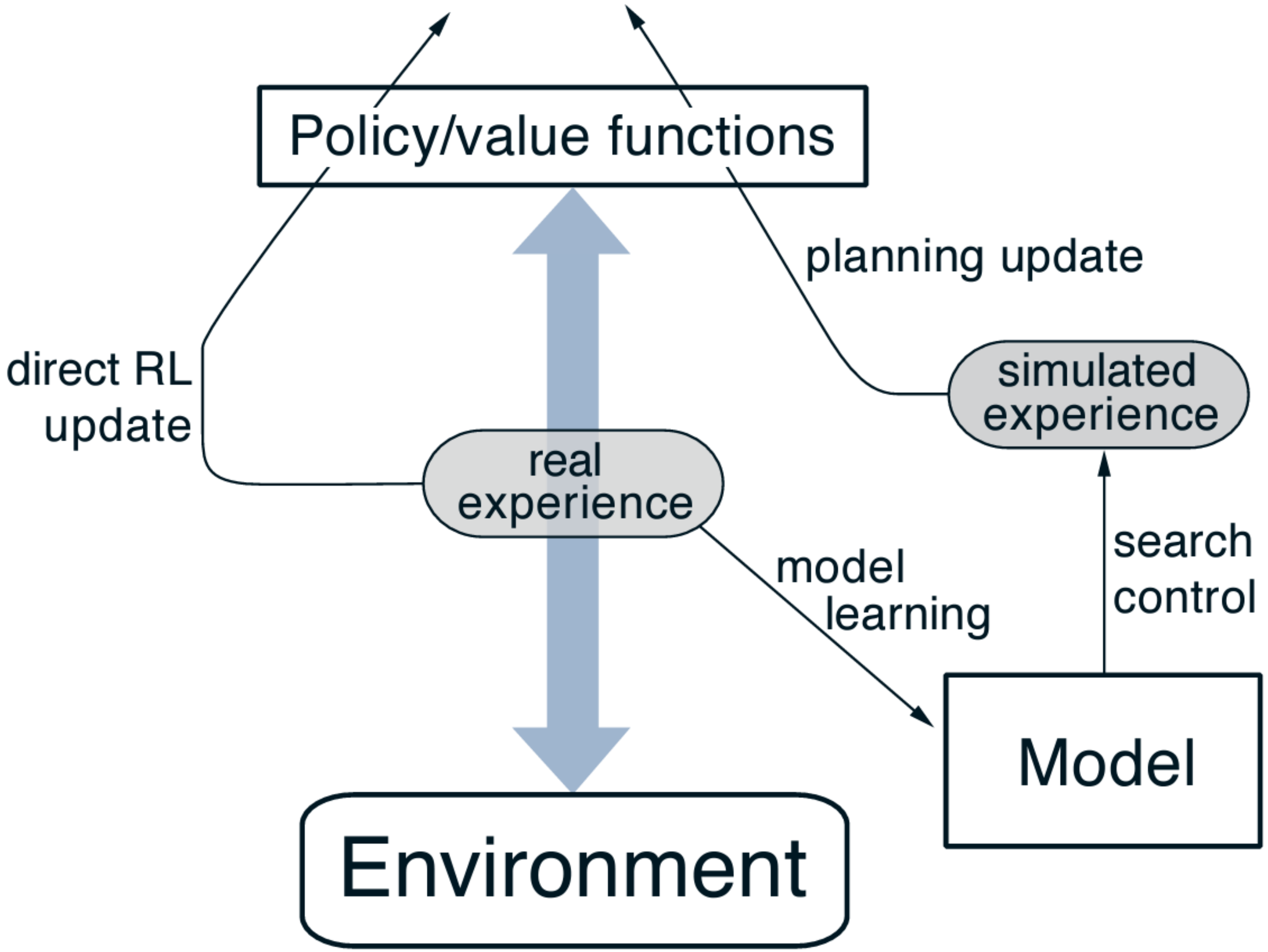
Integrating Planning and Learning

Both planning and direct RL aims to update the value/policy

So both processes can be interrupted if needed



DYNA architecture



DYNA

DEAKIN UNIVERSITY

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \epsilon$-greedy(S, Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

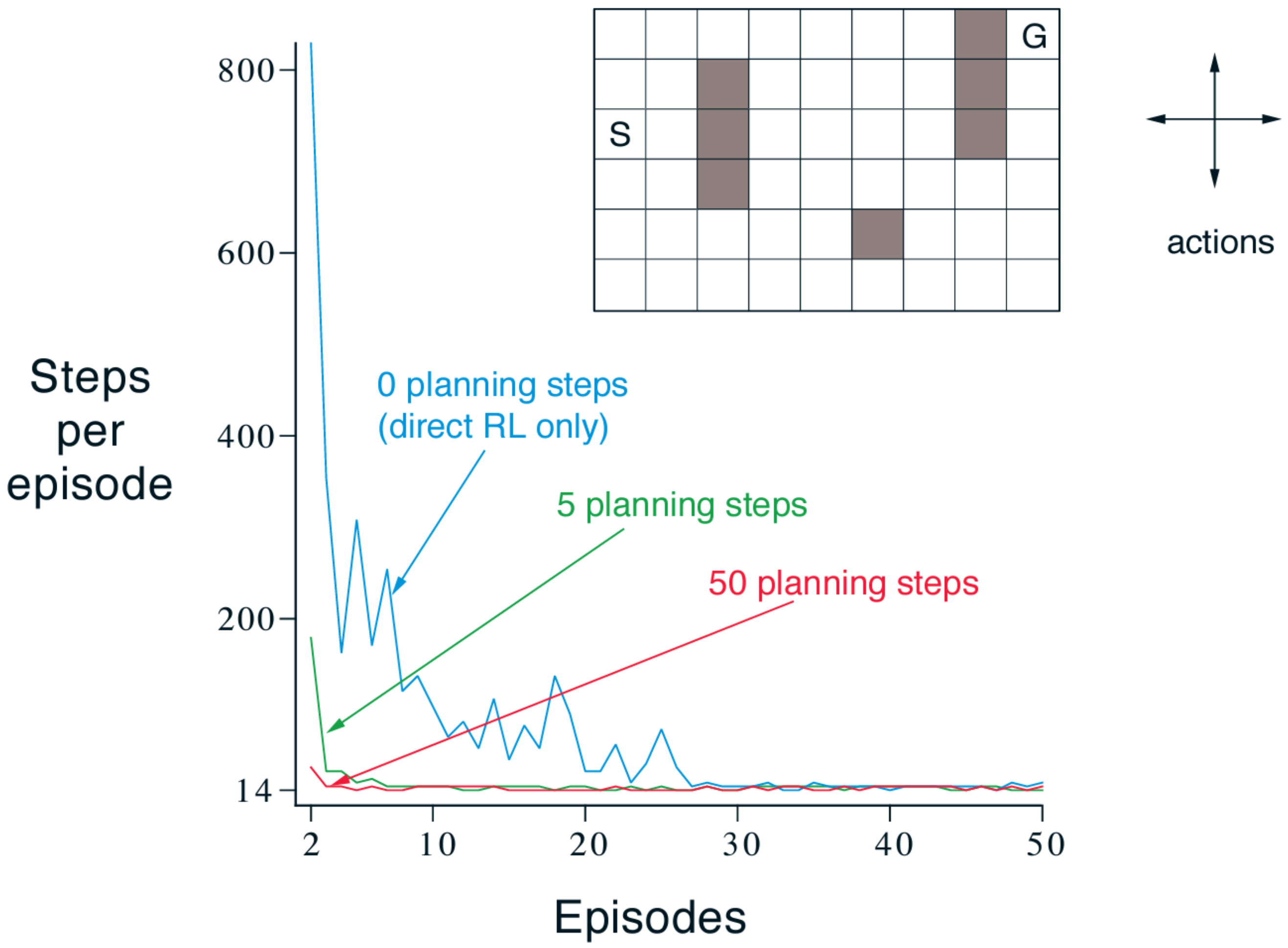
Learning

Model update

Planning

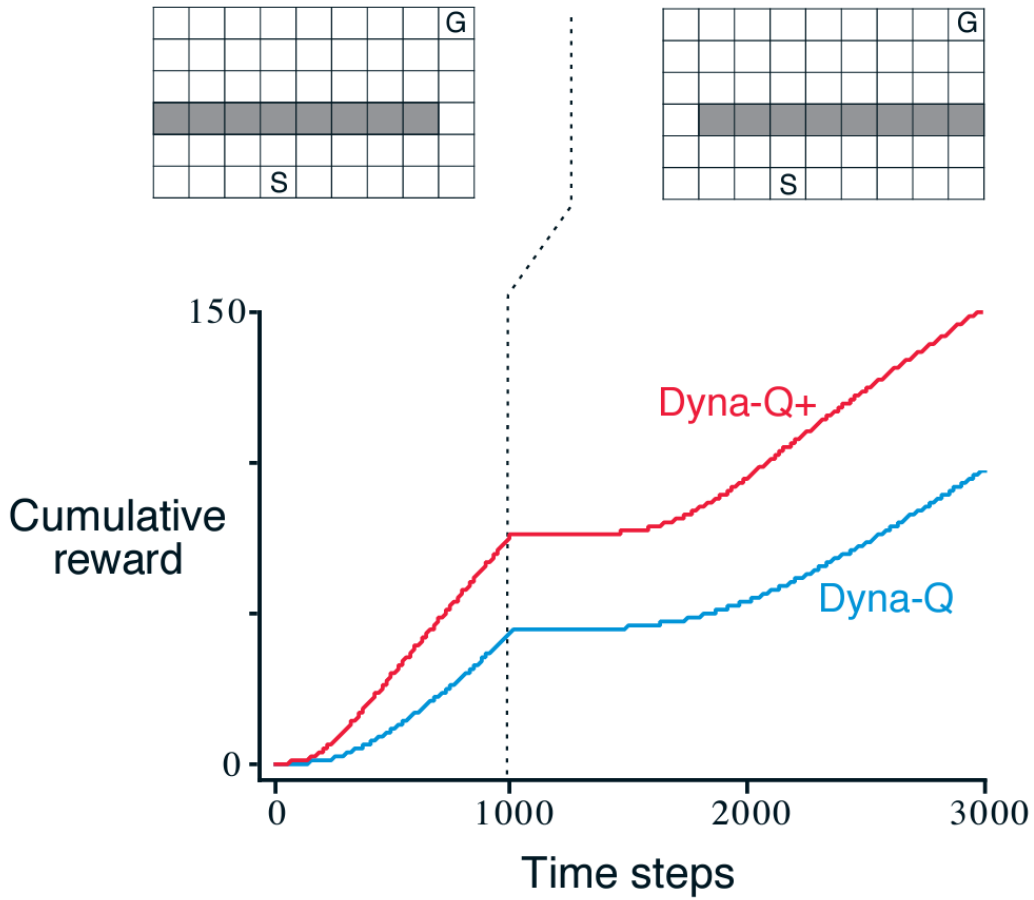
The model could be trained using any known method (eg: supervised learning)

DYNA architecture



More planning steps means faster learning

When the model is wrong: Blocking Maze

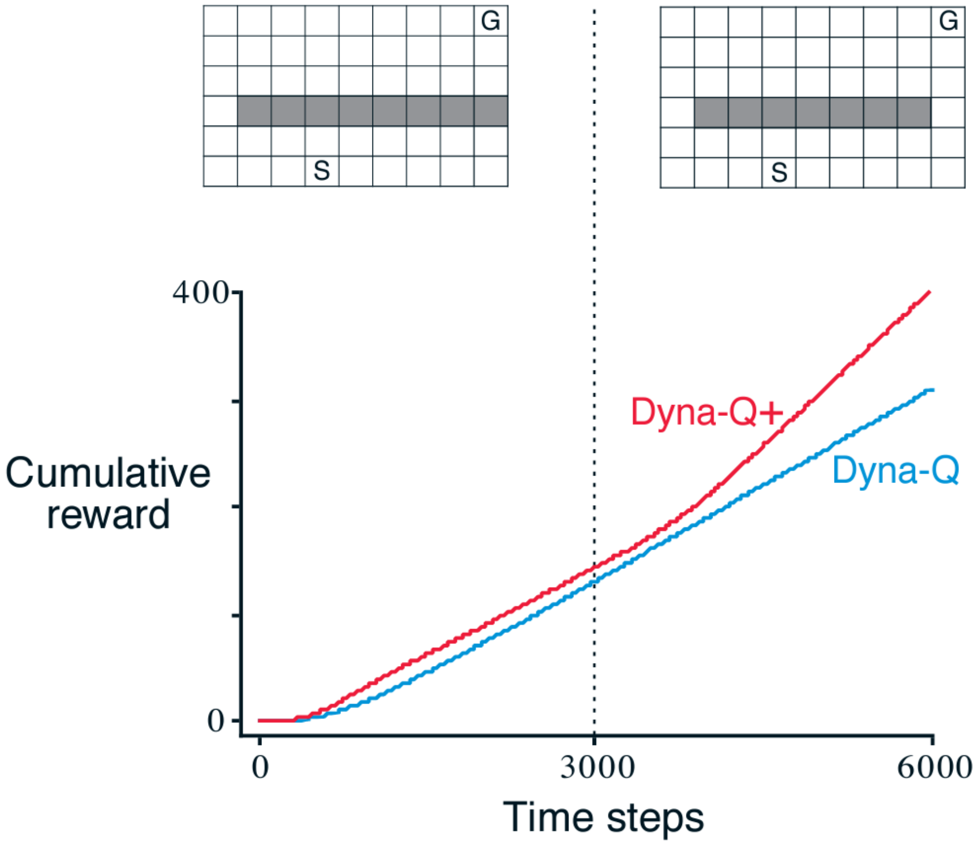


During the latter half of learning, the model makes optimistic predictions, which never come true

After sometime, the correct path is found

Providing a dedicated exploration bonus (as in Dyna-Q+) is useful

When the model is wrong: Shortcut Maze



Dyna-Q tends to get stuck and fails to improve – because the model's predictions are correct and the agent still reaches the goal

Dyna-Q+ eventually discovers the shortcut

Prioritized Sweeping

Similar to Dyna, but experiences are prioritized

$$Q(s,a) \leftarrow Q(s,a) + \alpha\underbrace{[r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)]}_{\text{Temporal difference (TD) error}}$$

α : learning rate

γ : discount factor

Higher error experiences are prioritized

Prioritized Sweeping

Initialize $Q(s, a)$, $Model(s, a)$, for all s, a, and $PQueue$ to empty
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S, Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Model(S, A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
(g) Repeat n times, while $PQueue$ is not empty:
$S, A \leftarrow first(PQueue)$
$R, S' \leftarrow Model(S, A)$
$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
$\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
$P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.
if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

Model update

Prioritization

Planning

Update priority queue

Readings

This lecture focused on eligibility traces and DYNA.

- Future topics will look at function approximation techniques and alternatives to value based approaches

For more detailed information see Sutton and Barto (2018) Reinforcement Learning: An Introduction (Version 2)

- *Chapter 12 and Chapter 8*
- http://incompleteideas.net/book/RLbook2020.pdf

Other Readings - Sutton and Barto (1998) Reinforcement Learning: An Introduction (Version 1)

- *Chapter 7: Eligibility Traces*
- http://incompleteideas.net/book/first/ebook
- *Note: this book is now primarily obsolete but some parts are worth knowing as they are commonly used*