SIT796 Reinforcement Learning

Recap

Presented by:
Thommen George Karimpanal
School of Information Technology

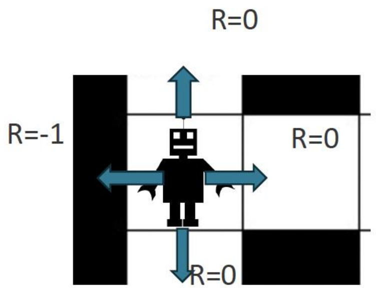
Week 1

Introduction & History,

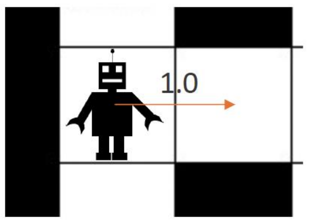
RL formulation: States, Actions, Rewards, Transition function, value function, action-value function

Formally, the RL problem is formulated as a Markov Decision Process (MDP)

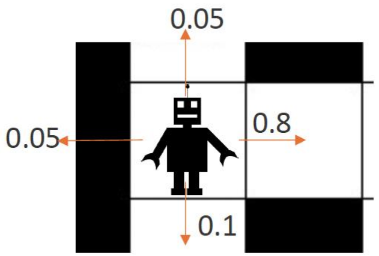
- An MDP is a tuple $M=\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R}\}$
  - $\mathcal{S}$ - The set of possible states
  - $\mathcal{A}$ - The set of actions the agent can take
  - $\mathcal{T}$ - transition probabilities
  - γ - The discount rate or the discount factor.
  - $\mathcal{R}$ - A reward distribution function conditioned.



Reward function



Deterministic transition



Probabilistic transition

Sutton and Barto (2018)

Week 2

Psychological aspects

Multiarmed Bandits

Initialise: for $a = 1$ to k

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

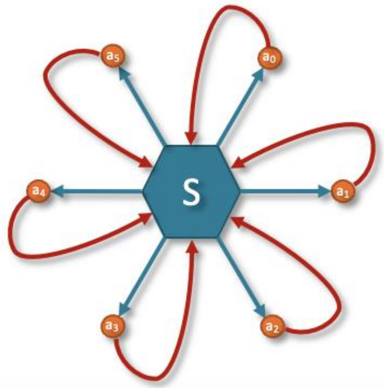
Loop forever:

$$A \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \underset{a}{\operatorname{random}}\, Q(a) & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow \mathrm{N(A)} + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$



Types of MDPs

Dynamic Programming

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determinin
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum$
    $\Delta \leftarrow \max(\Delta, |v - V($
until $\Delta < \theta$

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
     $\Delta \leftarrow 0$
     Loop for each $s \in \mathcal{S}$:
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the a

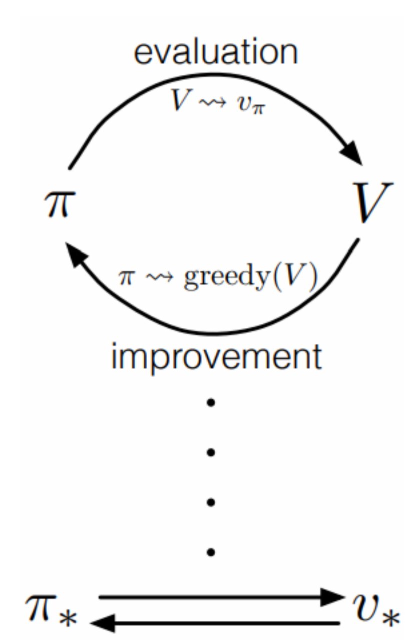
')]
lse
d $\pi \approx \pi$

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$



Week 4

Monte-Carlo Methods

Monte-Carlo Prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop foreve

Generat

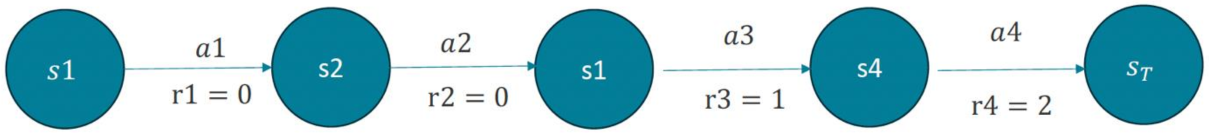
$G \leftarrow 0$

Loop for

$G \leftarrow$

Unle

3 episodes, $\gamma = 1$



G_1=0+0+1+2

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Monte-Carlo Control

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \text{argmax}_a\, Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

(on/off policy – importance sampling)

Pros & Cons

DP	Monte-Carlo
Transition and reward model required	No model needed
Requires full sweep of the state/state-action space	Requires full trajectories of experience
Convergence guaranteed	Converges only if states/state-actions are visited enough number of times

In reality, we don't have perfect models, and we don't need full trajectories to learn

– we learn on-the-go

Source: https://www.batterseachessclub.org.uk/10-benefits-of-playing-chess/

SIT796 Reinforcement Learning

Temporal Difference (TD) Learning

Presented by:
Thommen George Karimpanal
School of Information Technology

DEAKIN UNIVERSITY

Temporal Difference (TD) Learning

A combination of Dynamic Programming and Monte Carlo methods:

- Like Monte Carlo it learns from experience and doesn't need a model
- Like Dynamic Programming it updates estimates using other learned estimates

Temporal difference

Related to time

Learning happens as time progresses - can update values without having to wait till the end of the episode

Sutton and Barto (2018)

Temporal Difference (TD) Learning

In Monte-Carlo, we wait till the end of the episode and update values accordingly:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{G_t}_{\text{This is the target}} - V(S_t)\Big]$$

In TD, we take one step (and experience one reward) and still try to update the value:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{This is the target for TD}} - V(S_t)\Big]$$

Sutton and Barto (2018)

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)

TD – error

$$V(S) \leftarrow V(S) + \alpha\Big[\overbrace{R + \gamma V(S')}^{\text{1 step of experience + belief about the future}} - \overbrace{V(S)}^{\text{Belief about the goodness of the present state}}\Big]$$

$$\underbrace{\Big[R + \gamma V(S') - V(S)\Big]}_{\text{TD error } \delta}$$

Depends on the reward and next state

This is the error TD methods aim to minimise

Sutton and Barto (2018)

TD vs MC vs DP

Clearly TD (Like MC) is better than DP if you do not have a model

- Most real world problems we do not have a complete or even a partial model – making DP impossible
- DP is also highly computationally intensive.

TD is also obviously better over MC in online environments

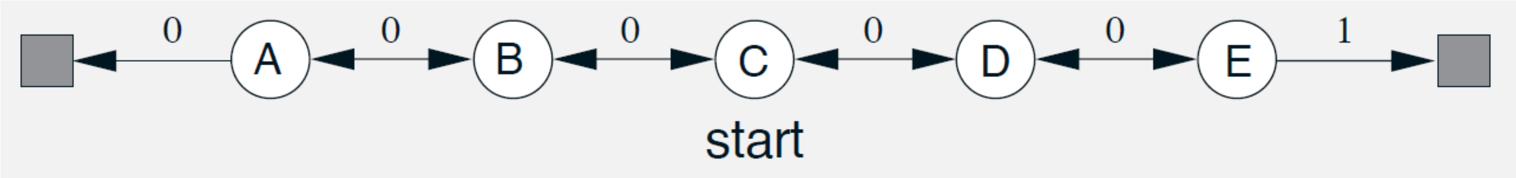
- MC methods must wait until the end of the episode to be updated – problematic in long or continuing tasks.
- If exploration has occurred, then many updates can not be made.
- Whereas, TD you can update more often (each step) – doesn't matter how long the episode is.
- TD is less susceptible to issues around exploratory actions – only affects the step when the exploration was taken.

TD methods have been proven to converge when the policy is fixed.

- Shown for table-based methods in MDPs

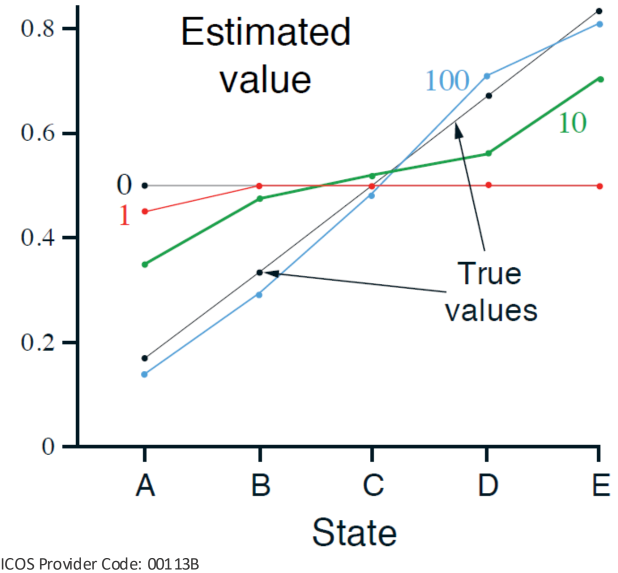
But which is faster (MC or TD)?

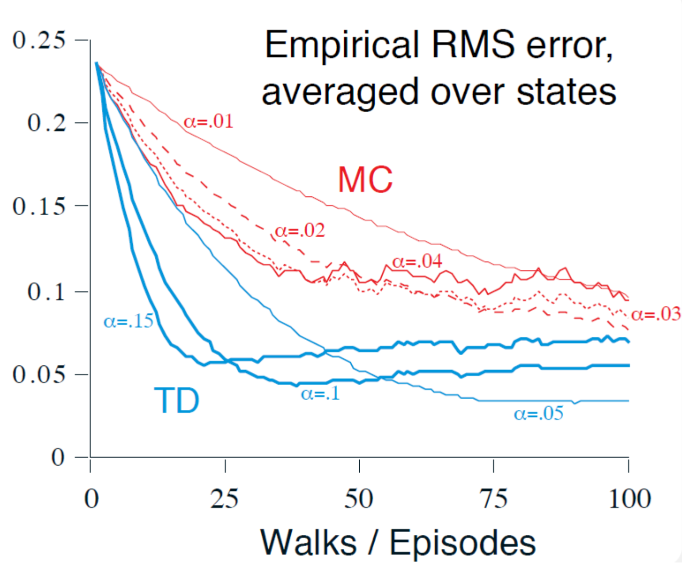
Example



A simple MDP where you start at C and move left or right until you hit a terminal state.

- Provided policy is just a 50/50 random split between left or right actions.
- TD Prediction aims to learn the true value for each state
- In this example TD is always better than MC.





TD makes more efficient use of data

Sutton and Barto (2018)

TD Control

TD for Control

- On-Policy – where our behaviour policy is the same policy we are learning (SARSA)
- Off-Policy – where we have a separate behaviour policy from the target policy we are attempting to learn (Q learning)

SARSA: On-Policy TD Control (2)

Uses quintuple ($S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$). Hence the name SARSA

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Convergence of SARSA is guaranteed if all state action pairs are guaranteed to be sampled an infinite number of times

Q-Learning: Off-Policy TD Control

DEAKIN UNIVERSITY

The off-policy equivalent to SARSA is known as Q-Learning (Watkins, PhD thesis 1989).

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop 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

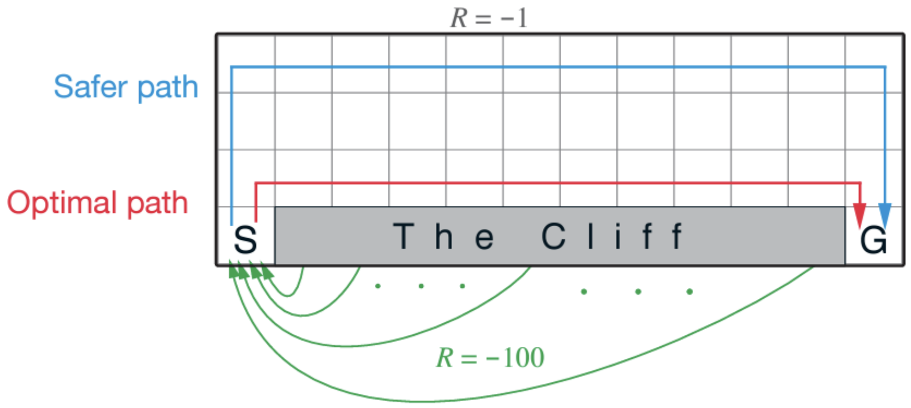
Notice the subtle difference to SARSA – the learning target and the way the actions are selected is different

SARSA vs Q-learning

Both SARSA and Q learning implemented with ε-greedy exploration ($\varepsilon = 0.1$)

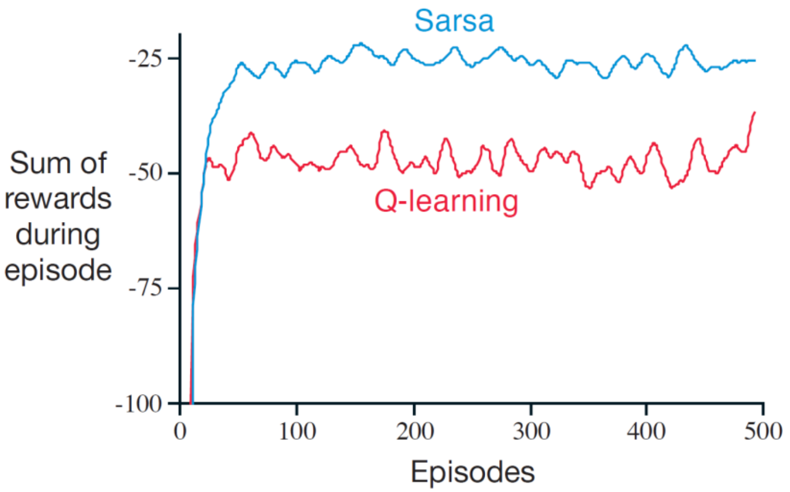
Q learning target only "cares" about the best action

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



Even if there is some non-zero probability of falling into the cliff, Q learning still prefers the risky (but optimal) path

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



However, SARSA is on-policy – so it accounts for the action-selection and prefers the safer path

Note: if ε was reduced to 0 overtime then both algorithms will converge to an optimal "cliff edge" policy

Sutton and Barto (2018)

Expected SARSA

Expected Sarsa updates its value based on the expected reward – incorporating how likely an action is to be taken.

- It constructs the learning target based on the probability of each action – doesn't rely on maxQ(s',a') like Q learning, nor does it rely on the taken action.
- So it can act as either On-policy or Off-policy.

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

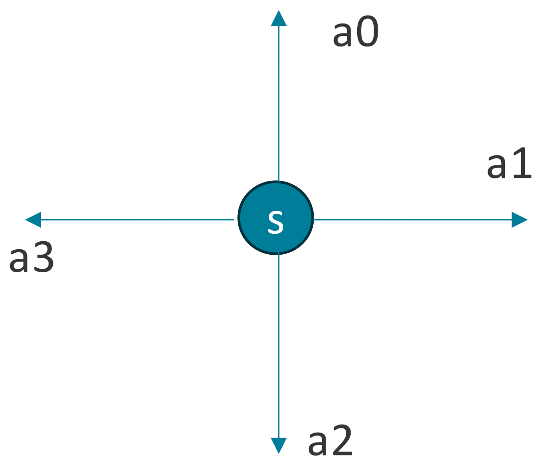
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a | S_{t+1}) \, Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

SIT796 Reinforcement Learning

Maximisation bias and double Q Learning

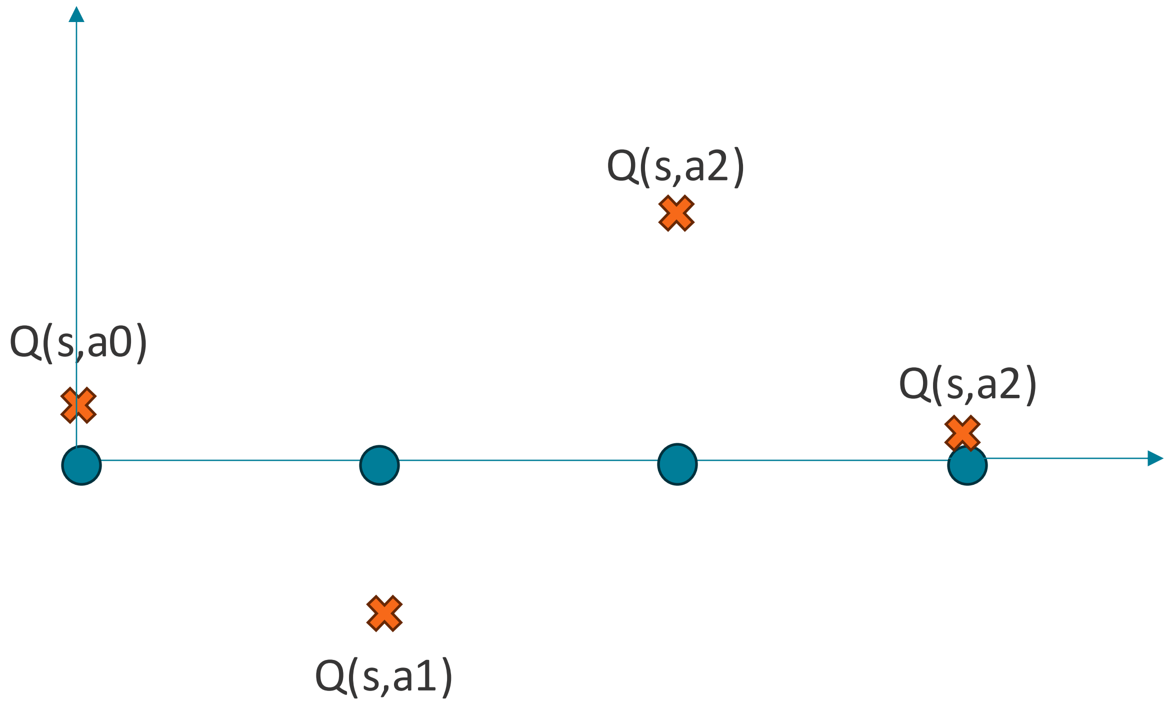
Presented by:
Thommen George Karimpanal
School of Information Technology

Maximization bias



True Q values = 0

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



$max_{a'} Q_{true}(s, a')$ = 0

$max_{a'} Q_{est}(s, a')$ > 0

This positive bias is called maximization bias

Double Q-Learning

Proposed as a solution to biased learning

Maintain two values for each action ($Q_1(a)$ and $Q_2(a)$)

We then randomly choose one of these values to decide which action we are going to use to select the maximum action

But then update the value of the other Q value for that action.

Can be proved that this addresses the maximization bias issue

Double Q-Learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy 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:

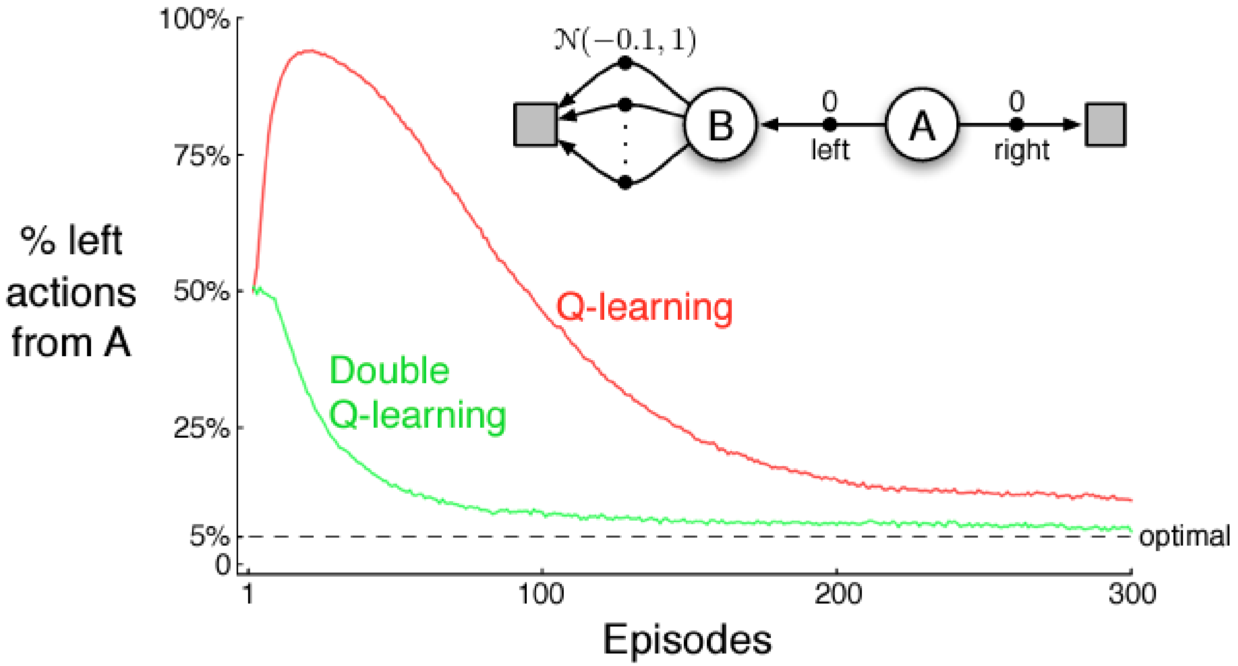
$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. No. 1. 2016.

Such "double" versions also exist for SARSA and expected SARSA

Double Q-Learning



n-step TD learning

The TD target relies partially on samples (experience) and partially on bootstrapping estimates

So far, we only looked at 1 step of experience. But in general, we can roll out multiple steps (n-steps)

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

n-step Sarsa (TD On-policy Control)

This bootstrapping idea can easily be extended to On-Policy Control (and Expected Sarsa)

- Whereas, in n-step TD we add all the rewards along with the final state estimate. In Sarsa we add all the rewards with the final *state-action* value estimate.

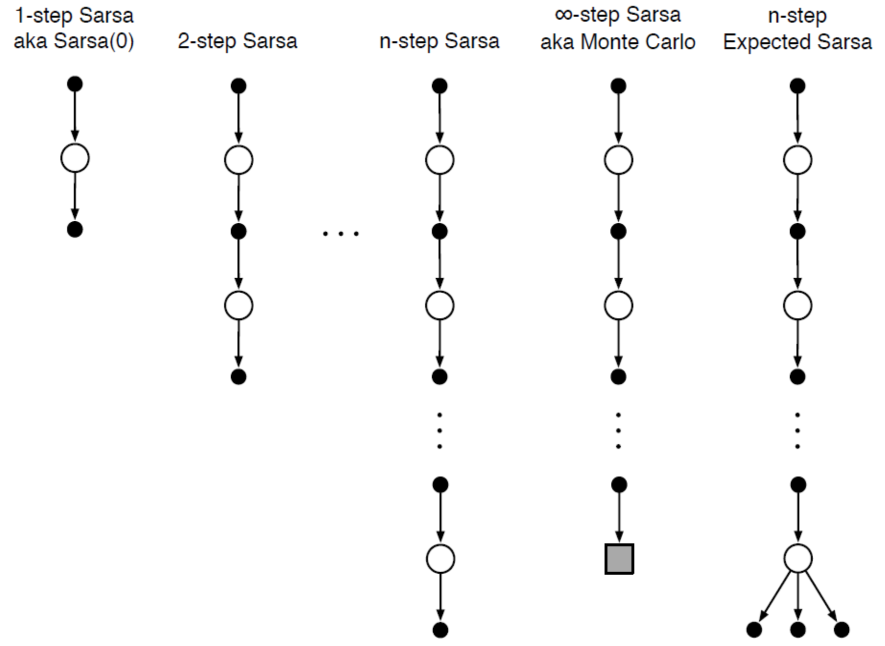
$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

- In Expected Sarsa we do the same except in the last step we use a weighted sum of the estimates of possible actions.

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n})$$

- Where $\bar{V}_t(s)$ is the *expected approximate value* of state s.
- This is calculate using the estimated action values of all states from s weighted by the probability of their being selected using policy π

$$\bar{V}_t(s) = \sum_a \pi(a|s)\, Q_t(s,a)$$

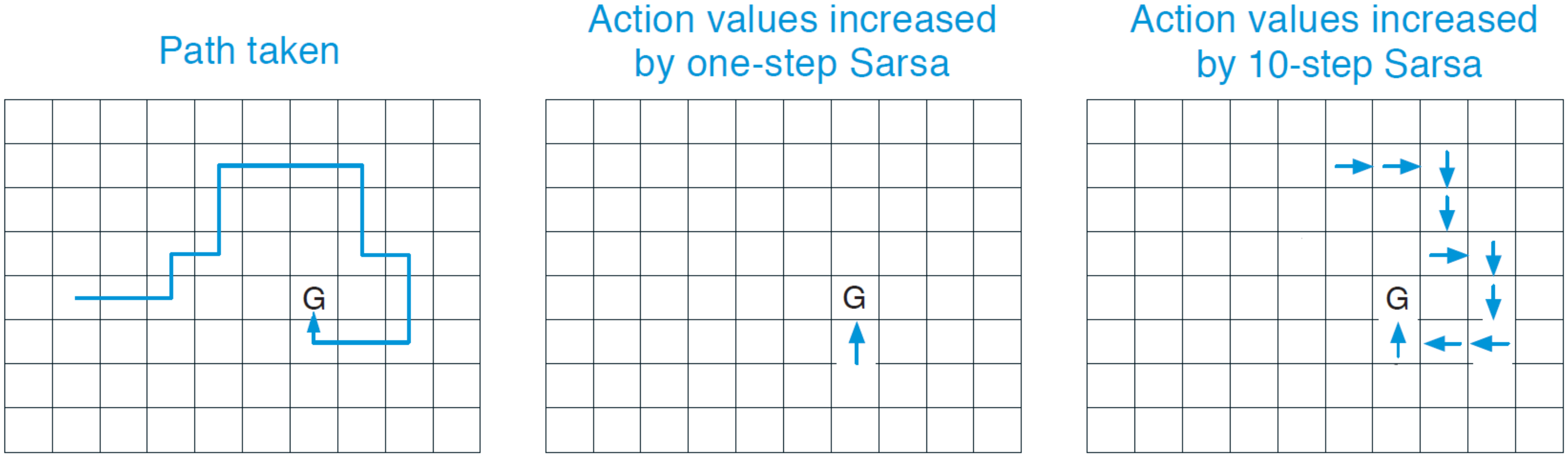


Sutton and Barto (2018)

Example: Gridworld with n-step Sarsa

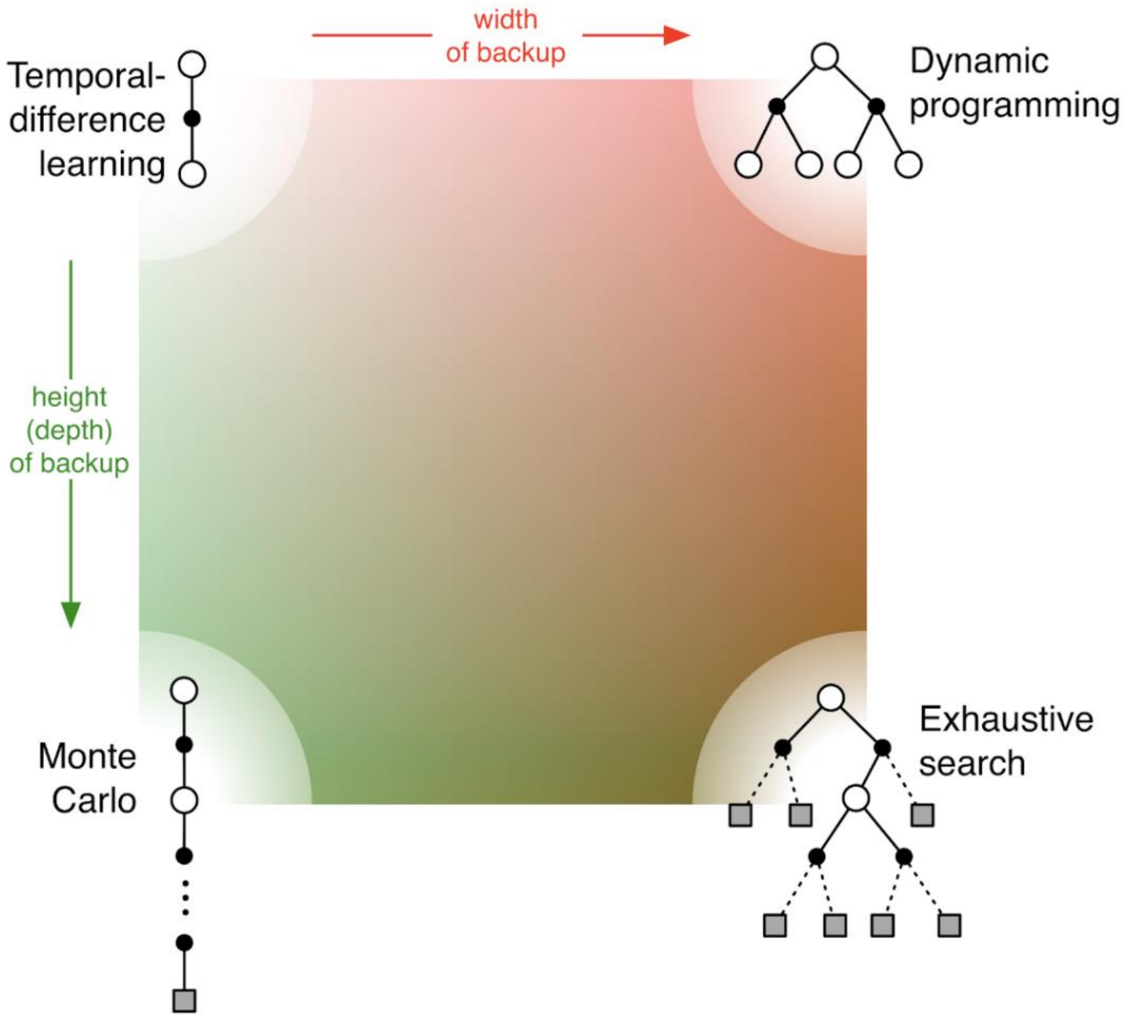
In Searching for a goal using on-policy control

- The path taken which are also all the states that will be backed up using MC
- The state-actions 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)

Putting it all together



Sutton and Barto (2018)

Readings

This lecture focused on methods for solving MDPs using Temporal difference.

- Future topics will look into advanced topics in using Temporal difference learning.
- Ensure you understand what was discussed here before doing the following topics

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

- *Chapter 6: Temporal-Difference Learning*
- http://incompleteideas.net/book/RLbook2020.pdf
- Lecture content has been borrowed from the above mentioned book