Tabular Reinforcement Learning Methods

Adam Page

6 February 2025

Declaration

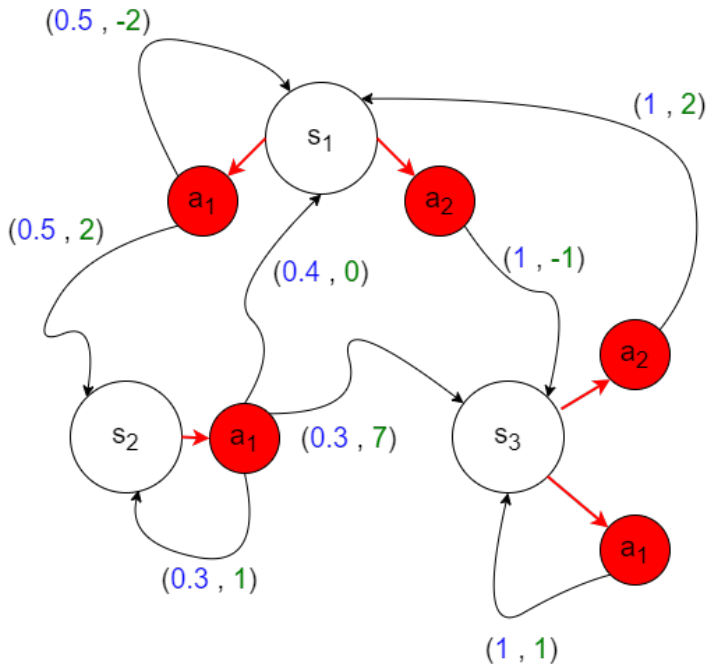
A lot of this info comes from 2 sources:

- Reinforcement Learning: An Introduction - Sutton & Barto, 2nd edition
- David Silver lecture series.

Markov Decision Process

Markov Decision Process: $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- **States:** $\mathcal{S}$
- **Actions:** $\mathcal{A}$
- **Transition Probabilities:** $\mathcal{P}$
- **Reward Function:** $\mathcal{R}$
- **Discount Factor:** γ



All we need is reward

Return - total discounted reward from this time step onwards:

Episodic:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

Continuing:

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

End up with this reccursion:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

State(-Action) Value Functions

Policy, π, gives $\pi(a|s) = \mathbb{P}(a|s)$.
Value of a state:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r \mathbb{P}(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned}$$

State(-Action) Value Functions

Policy, π, gives $\pi(a|s) = \mathbb{P}(a|s)$.
Value of a state:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r \mathbb{P}(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned}$$

Similarly, value of taking an action in this state, and then following π:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \sum_{s', r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \qquad \text{(or } \gamma q_\pi(s', a')\text{)} \end{aligned}$$

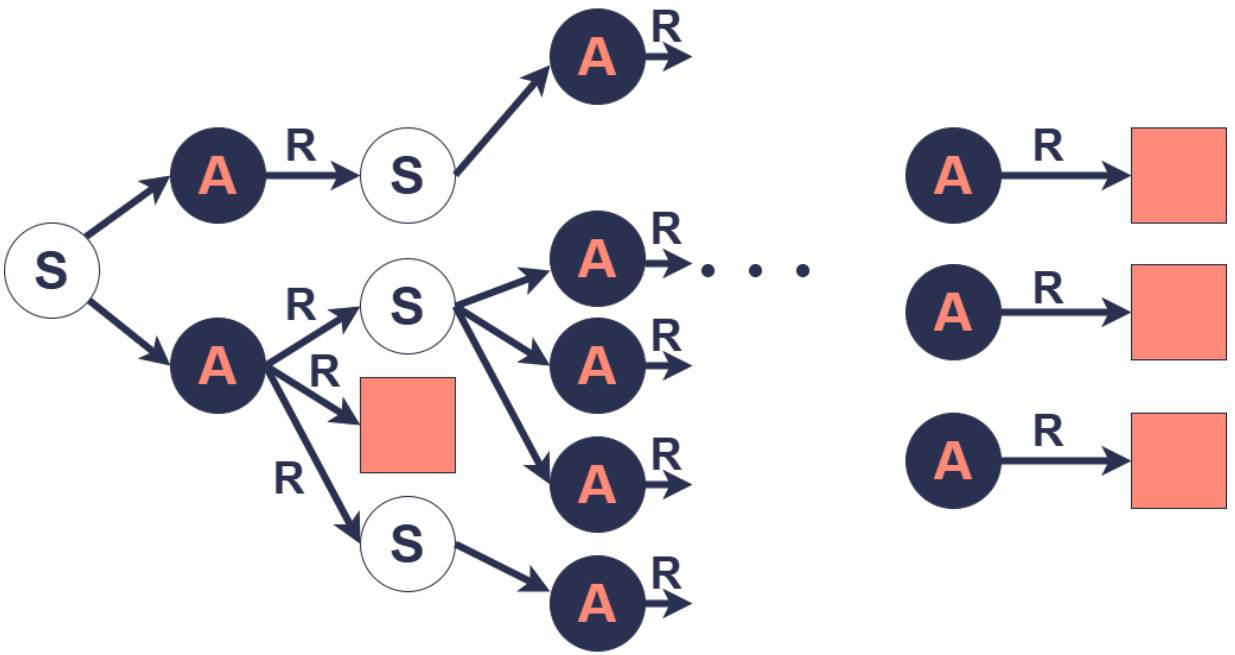
Optimal Value Functions

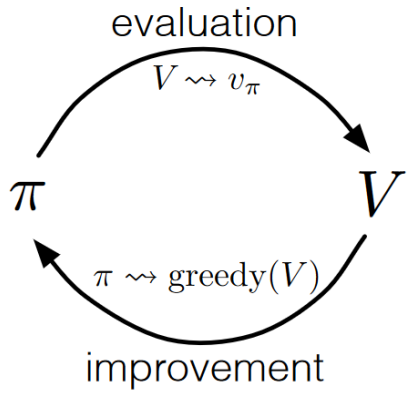
If an optimal policy exists:

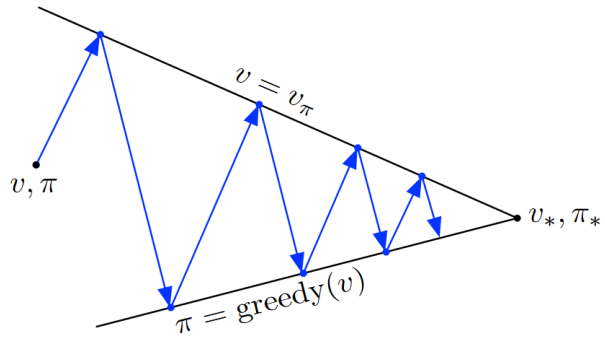
$$v_*(s) = \max_a \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma v_*(s') \right]$$

$$q_*(s,a) = \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

Dynamic Programming: Generalised Policy Iteration







Issues

$$q_*(s,a) = \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

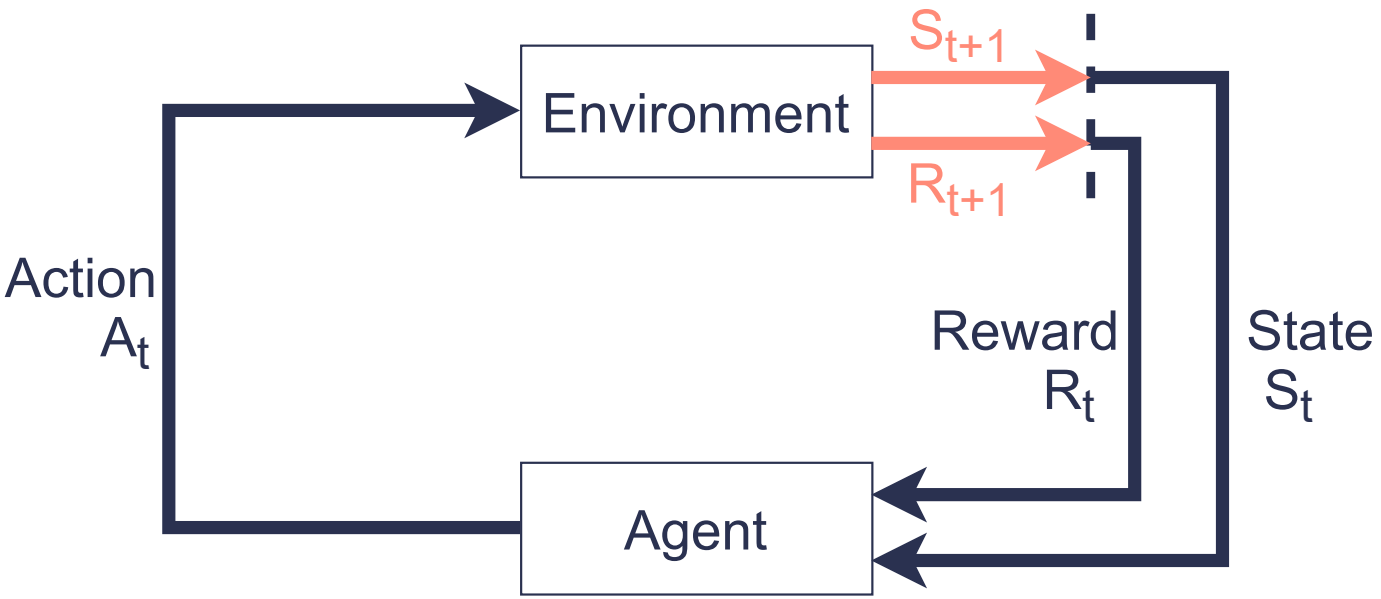
Issues

$$q_*(s,a) = \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

- Misspecified
- Intractable
- Unknown
- Computationally intensive on memory

Agent-Environment Interactions

We model the desired environment as a Markov Decision Process. Learn from simulated experience.



Estimate $Q(S, A) \approx q_*(S, A)$ can be stored in tabular size $\mathcal{S} \times \mathcal{A}$.

Monte Carlo Methods

As the name suggests, Monte Carlo methods solve the RL problem based on averaging sample returns.

As in Generalised Policy Iteration we have 2 questions:

- **Prediction**: What is $v_\pi(s)$?
- **Control**: Starting from π_0, can we find π_*?

Monte Carlo Prediction

Aim: Evaluate $v_\pi(s)$.

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 forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

We can also consider every-visit MC methods.

Convergence

Monte Carlo methods will converge. That is $V(s) \to v_\pi(s)$ as $n_{\text{visits}} \to \infty$.

For first-visit MC, each return is an i.i.d. estimate of $v_\pi(s)$ with finite variance. By law of large numbers the estimates will converge to their expected value.

Each average is an unbiased estimate and the standard deviation of its error falls as $\frac{1}{\sqrt{n_{visits}}}$.

Both first-visit and every-visit MC converge quadratically to $v_\pi(s)$.

Monte Carlo Control

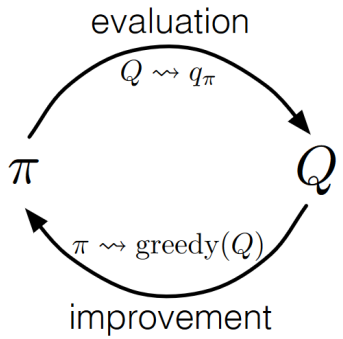
If we have a model, knowing $v(s)$ is sufficient to determine a policy. Look ahead one step and choose the action that leads to the best combination of reward and next state. (i.e., $R(s, a, s') + v(s')$)

With no model available, we now need to evaluate $q_\cdot(s, a)$ in order to suggest a policy.

Luckily for us, this is also needed in order to find an optimal policy, π_*.

Setting Up Monte Carlo Control

Aim: Find $q_*(s, a)$. Gives us π_*.
We once again look at the GPI framework.



$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*} .$$

Policy Improvement

Policy improvement is done by making the policy greedy with respect to q:

$$\pi(s) \doteq \arg\max_a q(s, a)$$

Construct π_{k+1} as the greedy policy with respect to q_{π_k}. Via the Policy Improvement Theorem (see Rui's notes):

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s, a))) \\ &= \max_a q_{\pi_k}(s, a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

Gives us that MC can be used to give us optimal policies.

Monte Carlo Control

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 $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

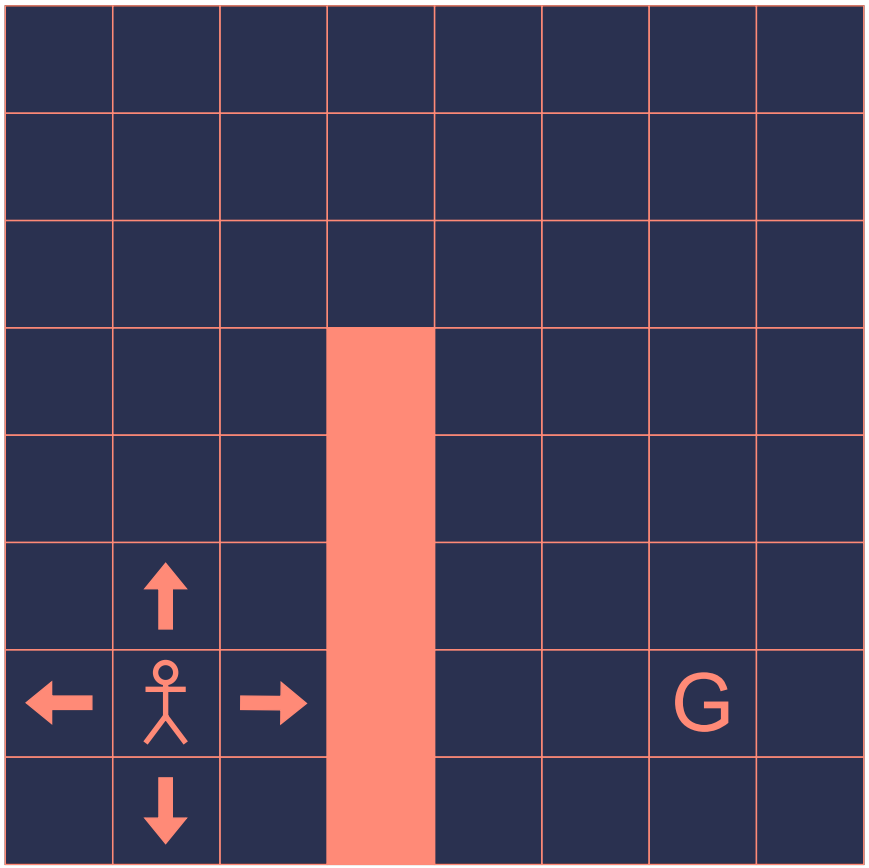
Note On Exploring Starts

$|\mathcal{S} \times \mathcal{A}| \geq |\mathcal{S}|$ so it is feasible that it may take a while to visit all state-action pairs.

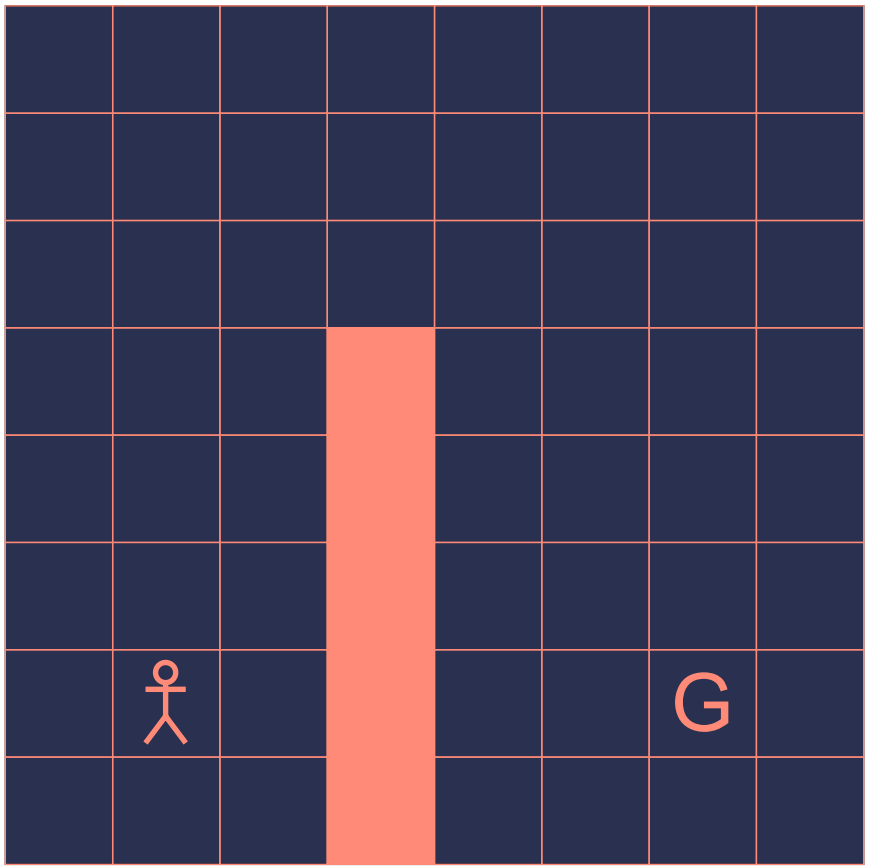
Also note that by following a greedy policy we may never visit certain actions although being the corresponding state quite frequently.

To ease these issues (and guarantee convergence) we enforce *exploration*. This can be done through random starts or having a stochastic policy.

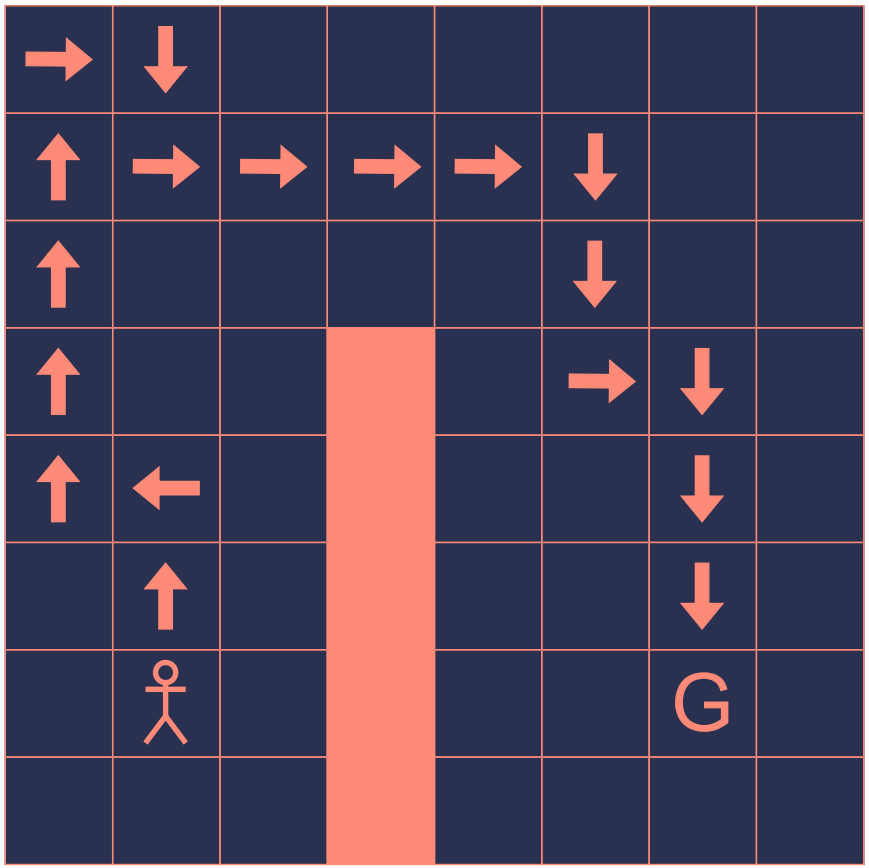
Gridworld



Starting Information



Monte Carlo RL Gridworld Agent



Extras

- MC control without exploring starts
- Prediction via importance sampling
- Incremental Implementable
- Discounting aware importance sampling
- Per-decision importance sampling

Intuition

$$q_*(s,a) = \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

Intuition

$$q_*(s,a) = \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

- MC-RL: $\approx$ Returns(S, A)

Intuition

$$q_*(s,a) = \sum_{s',r} \mathbb{P}(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

- MC-RL: $\approx$ Returns(S, A)
- Why wait?

Temporal-Difference Learning

- TD methods learn directly from experience without knowledge of the environment's dynamics. Similar to MC Methods.
- TD methods update estimates somewhat based on other learned estimates, without waiting for final outcome (bootstrapping). Similar to DP.

Targets and Errors

MC and TD use experience to solve the prediction problem. For MC we wait until the return, G_t has been evaluated and then use G_t as a target for current estimate $V(S_t)$. The update step could instead be written as:

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha \underbrace{[G_t - V(S_t)]}_{error} \\ &\leftarrow (1-\alpha)\, V(S_t) + \alpha G_t \end{aligned}$$

where α is some step-size.

TD Prediction

Using the earlier intuition, TD methods need only wait until the next time step. At time $t+1$ the agent immediately forms a target and updates using the reward R_{t+1} and the current estimate of $V(S_{t+1})$.
A simple TD method update:

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

with this update being made every time step.

- MC Target: G_t
- TD Target: $R_{t+1} + \gamma V(S_{t+1})$

TD Error

Difference between estimated value of S_t and better estimate, know as TD Error:

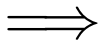
$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

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

Backup Diagrams



Why use TD?

- vs DP: TD methods don't need a model of the environment.
- vs MC: TD can be implemented in online, incremental settings.
- For a fixed π we have that TD converges to v_π.

On- and Off- Policy

Before we can start to control need to discuss '*bootstrapping*' methods.

When looking at the TD update, specifically $R_{t+1} + \gamma V(S_{t+1})$, we base our update on an already existing estimate. As we are estimating the state value , $V(S)$, we do not need to consider the action that causes this value to occur.

For control, we need action value estimates, $Q(S_{t+1}, A_{t+1})$, in order to define a policy. Choice of action may be an issue. If $A_{t+1} \sim \pi(S_{t+1})$ we call this **on-policy**. The alternative is **off-policy**.

On-Policy TD Control

Once again, we return to GPI.

Action values TD update:

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

Let $A_{t+1} \sim \pi(S_{t+1})$. On-policy control algorithms we continually estimate q_π, and at the same time update π toward greediness with respect to q_π.

Each episode has multiple versions of the following quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ leading to the following algorithm ...

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 $Q(terminal, \cdot) = 0$

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

Off-Policy TD Control

As we are trying to find π_*, why not directly estimate q_* rather than q_π.

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

By acting greedy within our update the learned action value function Q directly approximates q_*. This is independent of the policy being followed.

Doing this simplifies the analysis of the algorithm and allowed for early convergence proofs.

Q-Learning

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

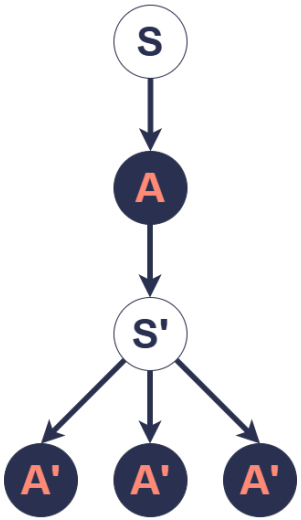
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

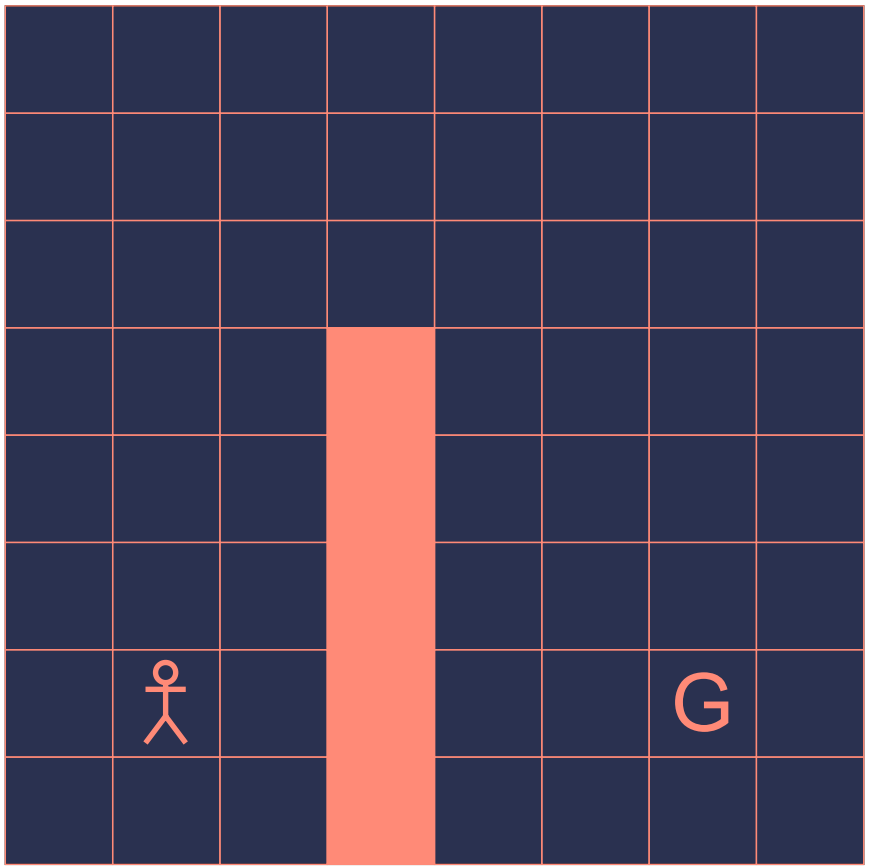
Backup diagrams of TD Learners

Sarsa

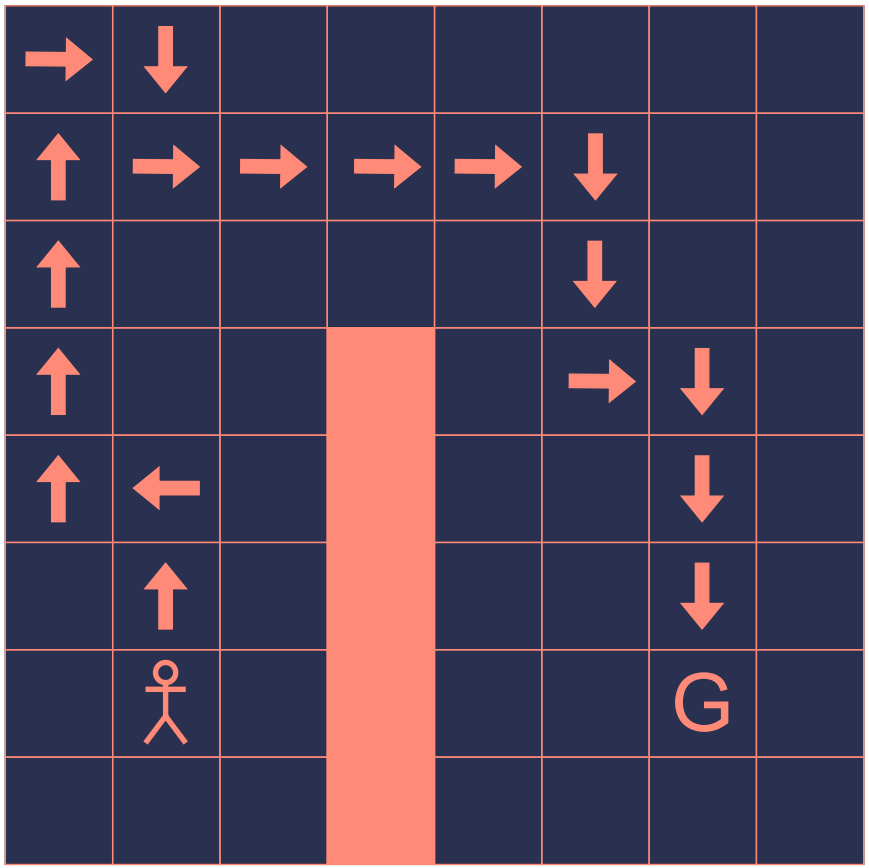
Q-learning



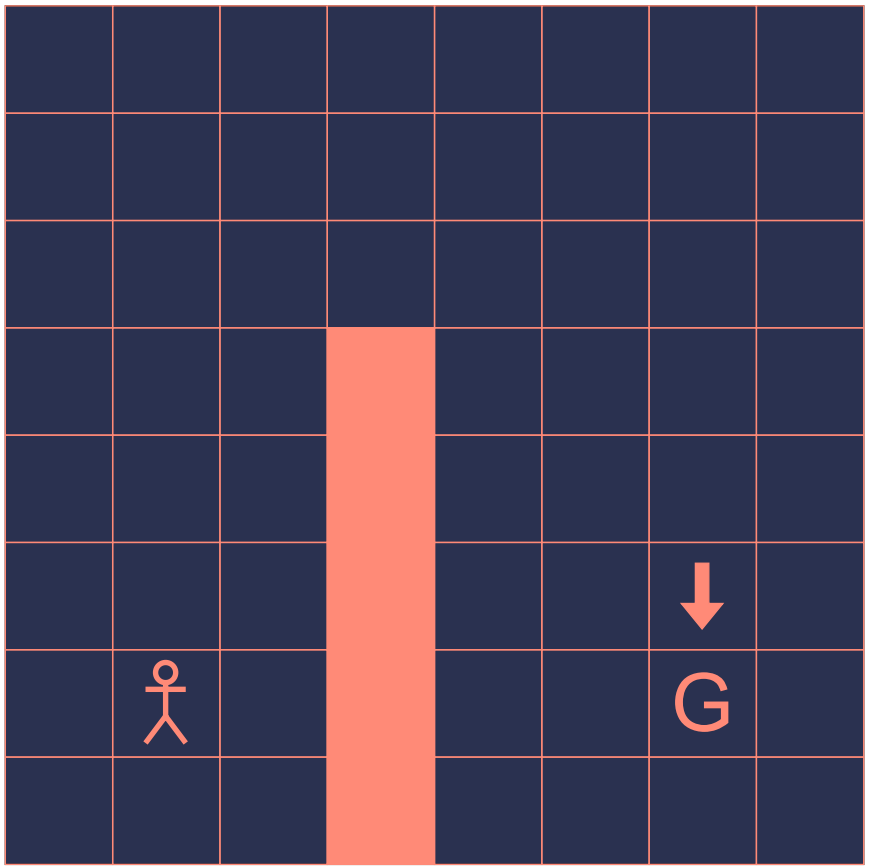
Gridworld



Monte Carlo Gridworld Revisited



TD Gridworld Agent



Extras

- Batch MC and Batch TD.
- Expected Sarsa.
- Double learning to reduce on bias.
- n-step Bootstrapping