# Reinforcement Learning

## $n$-step Bootstrapping

Sutton/Barto* Chapter 7

Michael Hahsler, SMU

With figures from Sutton/Barto*

*Sutton and Barto, Reinforcement Learning: An Introduction, 2nd edition, MIT Press, Cambridge, MA, 2018

# Topics of this Course

- Introduction to reinforcement learning
- Markov decision processes
- **Part I: Tabular Methods**
  - Dynamic programming
  - Monte Carlo methods
  - Temporal-difference learning
  - **Multi-step bootstrapping**
  - Planning and learning with tabular methods
- Part II: Approximate Solution Methods
  - Prediction and Control using Approximation
  - Eligibility Traces
  - Policy Gradient Methods
- Part III: Modern RL Methods
  - Deep Reinforcement Learning
  - Current Applications

# Summary of Notation

**General**

| | |
|---|---|
| $X$ | capital letters: random variables |
| $x, p$ | lower-case letters: realizations of random variables or scalar functions |
| $\boldsymbol{w}$ | Bold lower-case letters: real-valued vectors (even if random variables) |
| $\boldsymbol{W}$ | bold capitals: matrices |
| $\alpha$ | Greek letters: parameters (vectors if in bolt) |
| $\Pr\{X = x\}$ | probability that a random variable $X$ takes on the value $x$ |
| $X \sim p$ | random variable $X$ selected from distribution $p(x) = \Pr\{X = x\}$ |
| $\mathbb{E}[X]$ | expectation of a random variable $X$, i.e., $\mathbb{E}[X] = \sum_x p(x)x$ |
| $\text{argmax}_a\, f(a)$ | a value of action $a$ at which $f(a)$ takes its maximal value |

**Value Function**

| | |
|---|---|
| $G_t$ | return (cumulative reward) following time $t$ |
| $G_{t:h}$ | return from $t$ to $h$ (discounted and corrected) |
| $v_\pi(s)$ | value of state $s$ under policy $\pi$ (expected return) |
| $v_*(s)$ | value of state s under the optimal policy |
| $q_\pi(s, a)$ | value of taking action $a$ in state $s$ under policy $\pi$ |
| $q_*(s, a)$ | value of taking action $a$ in state $s$ under the optimal policy |
| $V, V_t$ | array estimates of state-value function $v_\pi$ or $v_*$ |
| $Q, Q_t$ | array estimates of action-value function $q_\pi$ or $v_*$ |

**MDP**

| | |
|---|---|
| $s, s'$ | states |
| $a$ | an action |
| $r$ | a reward |
| $\mathcal{S}$ | set of all (nonterminal) states, $\mathcal{S}^+$ are all states |
| $\mathcal{A}(s)$ | set of all actions available in state $s$ |
| $\gamma$ | discount-rate parameter |
| $t$ | discrete time step |
| $T$ | final time step of an episode (a.k.a. horizon) |
| $A_t$ | random variable for the action at time $t$ |
| $S_t$ | random variable for the state at time $t$ |
| $R_t$ | random variable for the reward at time $t$ |
| $p(s', r \mid s, a)$ | probability of transition to state $s'$ and receiving reward $r$, from state $s$ taking action $a$. |
| $p(s' \mid s, a)$ | probability of transition to state $s'$ fom state $s$ taking action $a$. |
| $r(s, a)$ | expected immediate reward from state $s$ after action $a$. |
| $r(s, a, s')$ | expected immediate reward from state $s$ to $s'$ with action $a$. |
| $\pi(a \mid s)$ | probability of taking action $a$ in state $s$ under stochastic policy $\pi$ |
| $\pi(s)$ | action taken in state $s$ under deterministic policy $\pi$ |

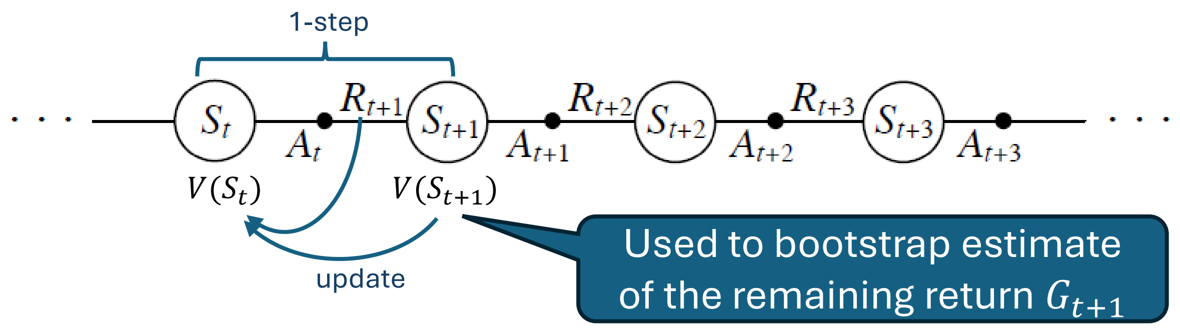
**Temporal Difference Learning**

| | |
|---|---|
| $U_t$ | target for estimate at time $t$ |
| $\delta_t$ | TD error |
| $G_{t:t+n}$ | $n$-step return using rewards from $t + 1$ to $t + n$ and bootstrap the rest |

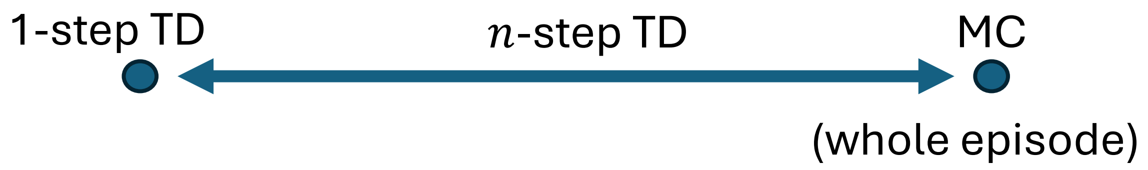
# Policy Evaluation: 1-step TD vs. $n$-step TD vs. MC

**Question**: How far ahead in the sampled episode do we look to estimate the return?

- **MC:** We use the whole episode.
- **1-step TD:** We only observe the **next step** and its reward. The remaining return is **bootstrapped**. This **propagates state value errors**!



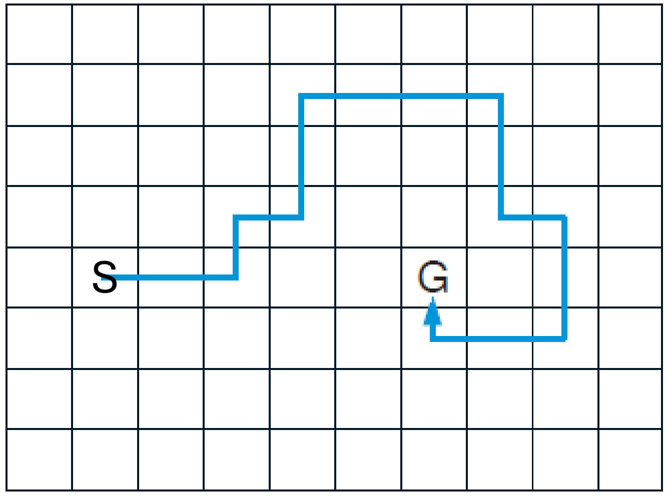
- **$n$-step TD**
  - Looks $n$ time steps ahead and bootstraps the rest. Using more observed rewards should help to **reduce the bootstrapping error.**
  - TD unifies and generalizes MC and one-step TD.



# Why is a Longer Look-ahead Useful?

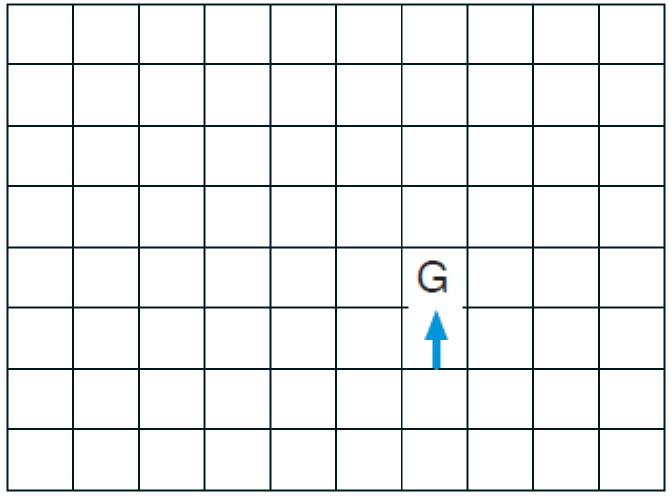
- **Example:** Maze with only one final reward for reaching the goal. We learn from the **first episode**.

Episode (MC)



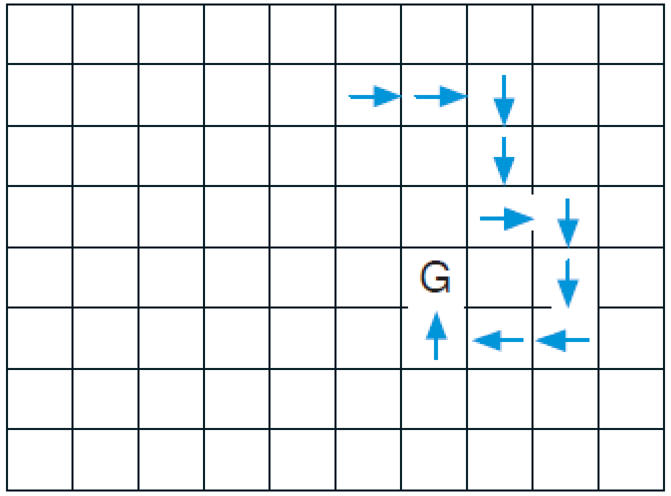
Updates in reverse order all state values on the path using the goal reward.

1-step look ahead



Can only update one state value with the goal reward.

10-step look ahead

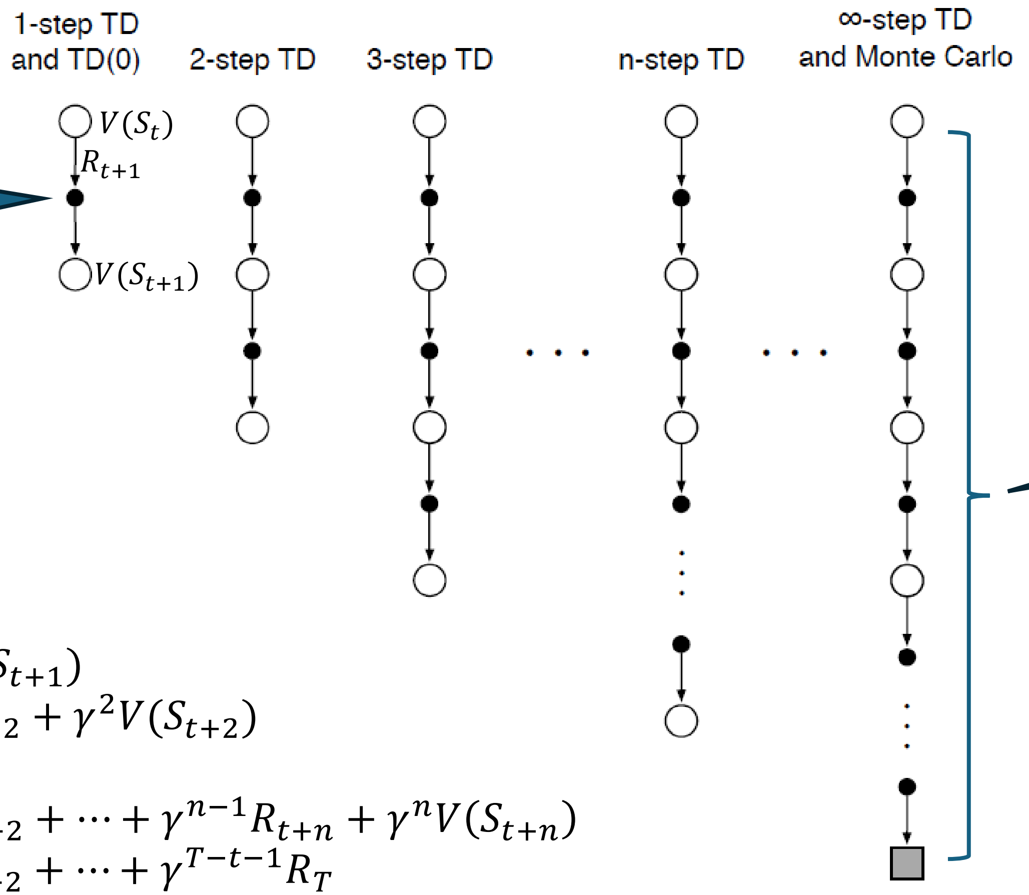


State value can be updated with the goal reward even if it is up to 10 steps away.
Learns faster than 1-step TD.

## $n$-step TD Prediction

Estimate the value function for a given policy.

# $n$-step TD Prediction



Update with reward $R_{t+1}$ + bootstrap the remaining return $G_{t+1}$ by the value of the next state $V(S_{t+1})$.

Update with the whole sequence of observed rewards → no bootstrapping.

**Return Estimate**

1-step TD: $G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1})$

2-step TD: $G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$

…

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

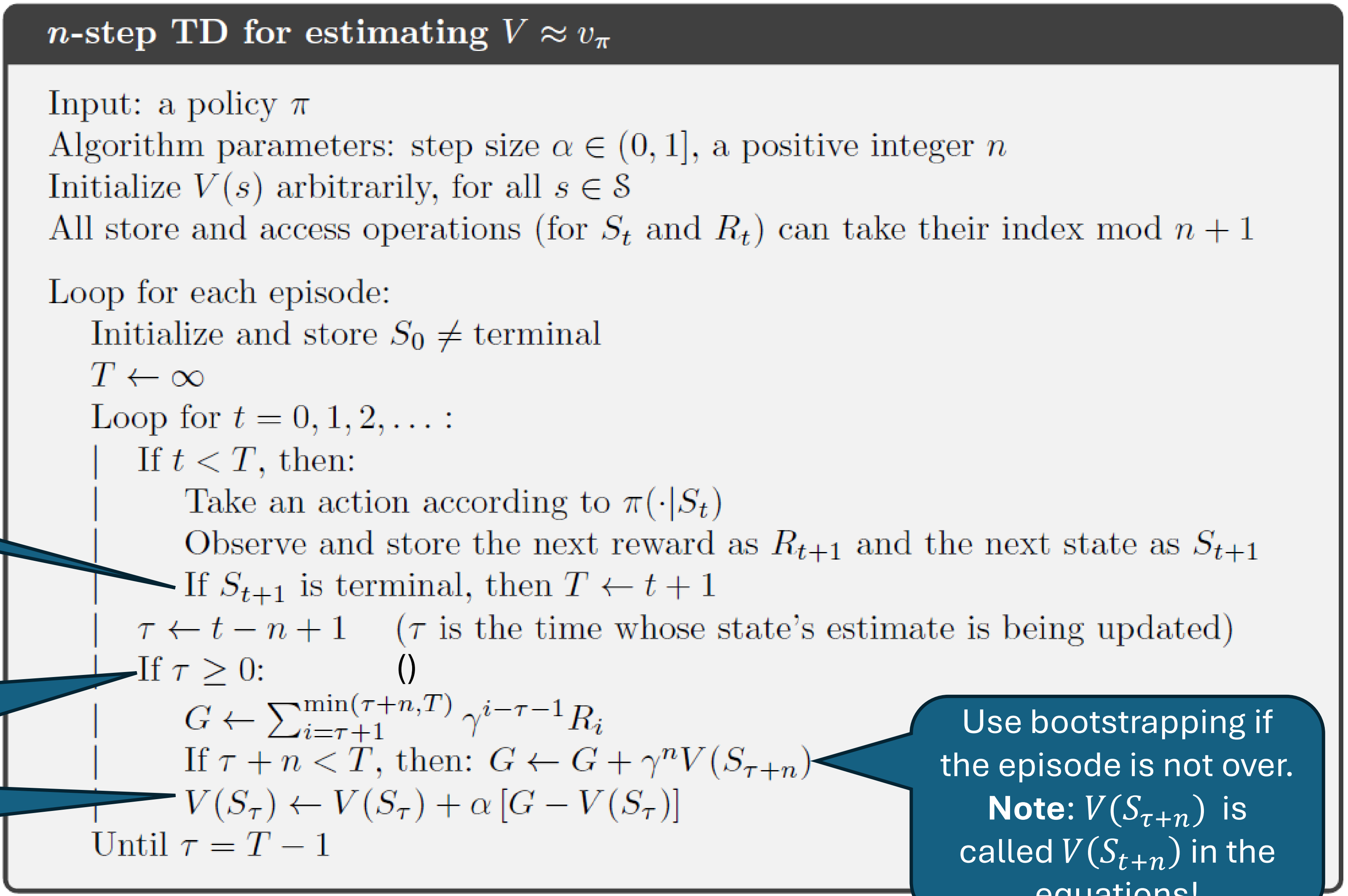
MC: $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$

**Notation**
$G_{t:t+n}$ … the $n$-step return is calculated from the rewards up to $t+n$ and the rest is bootstrapped using the value function.

**Update**: $V(S_t) \leftarrow V(S_t) + \alpha[G_{t:t+n} - V(S_t)]$

The update needs $G_{t:t+n}$ and is $n$ steps delayed!

# Alg. $n$-step TD Prediction



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

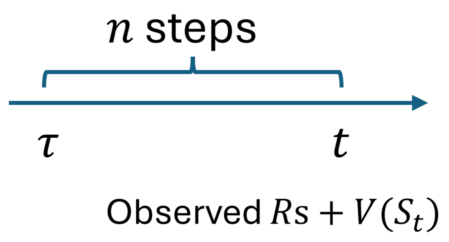
```
Input: a policy $\pi$
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \dots$ :
    |   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 $T \leftarrow t+1$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$
    |       $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
    Until $\tau = T - 1$
```

Detect the end of the episode

Start only when we can update $t \geq n$

The update is delayed $n$ steps

Use bootstrapping if the episode is not over. **Note**: $V(S_{\tau+n})$ is called $V(S_{t+n})$ in the equations!



# Convergence Guarantee

**Error Reduction Property of $n$-step Returns**

The worst error of the expected $n$-step return is guaranteed to be less than or equal to $\gamma^n$ times the worst error of using $V$.

$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \leq \gamma^n \max_s \left| V(s) - v_\pi(s) \right|$$

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

- Observed rewards have in expectation no error.
- The error-prone bootstrap estimate is smaller because it is $n$ time steps away and thus discounted.
- Increasing $n$ adds more observed rewards decreases the impact of the bootstrapping error.

**Note**: This also means that every update is guaranteed to reduce the expected error and moves the estimate closer to the true value function.

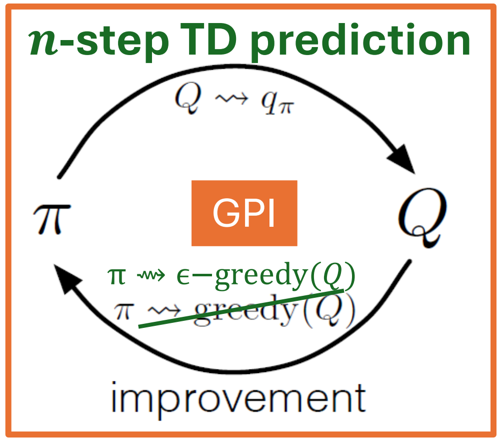
**All $n$-step methods (including 1-step and MC) converge**!

## $n$-step Control: Sarsa

On-Policy learning using $n$-step return estimates.

# $n$-step Sarsa

**Note:** (0) is not the number of steps but means that $\lambda = 0$. We will learn about eligibility traces later.


$n$-step TD prediction
$Q \rightsquigarrow q_\pi$
$\pi$
GPI
$Q$
$\pi \rightsquigarrow \epsilon\text{−greedy}(Q)$
~~$\pi \rightsquigarrow \text{greedy}(Q)$~~
improvement

Original Sarsa is called one-step Sarsa or Sarsa(0)

## $n$-step Sarsa

- **On policy**: The policy needs to keep exploring (e.g., an $\epsilon$-greedy policy)
- Should probably be called Sarrrrr…rsa with $n$ rs.

This is the extra action Sarsa need.

- $n$-step return estimate used as the target (redefined using $Q$):

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

- Update rule:

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

# Speedup of $n$-step Sarsa

$Q$-values updated during one episode

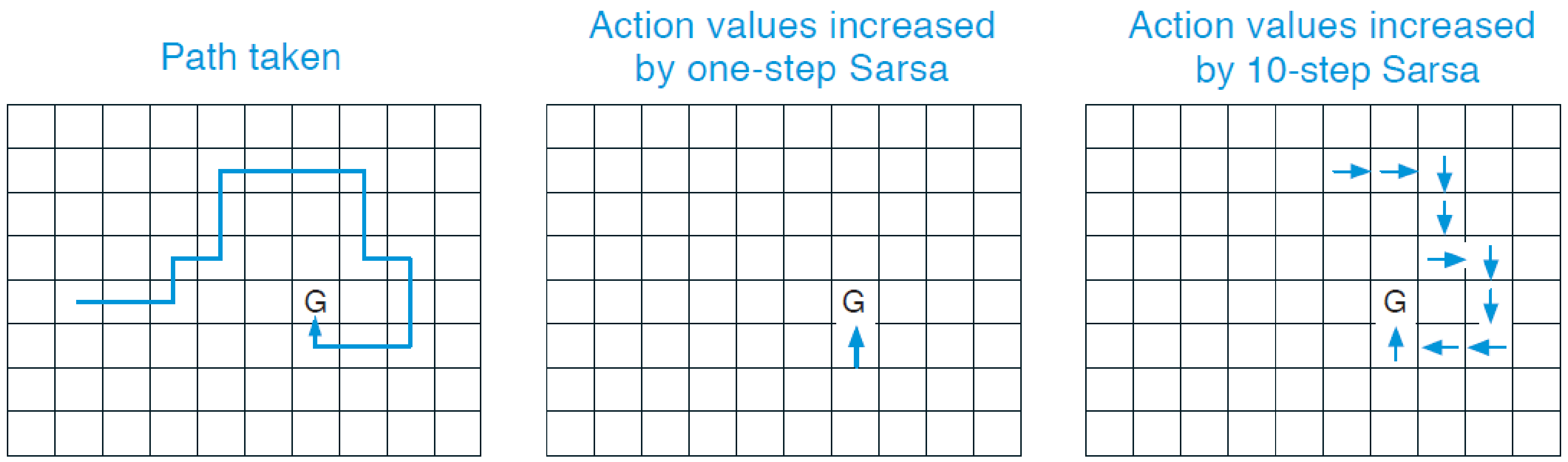
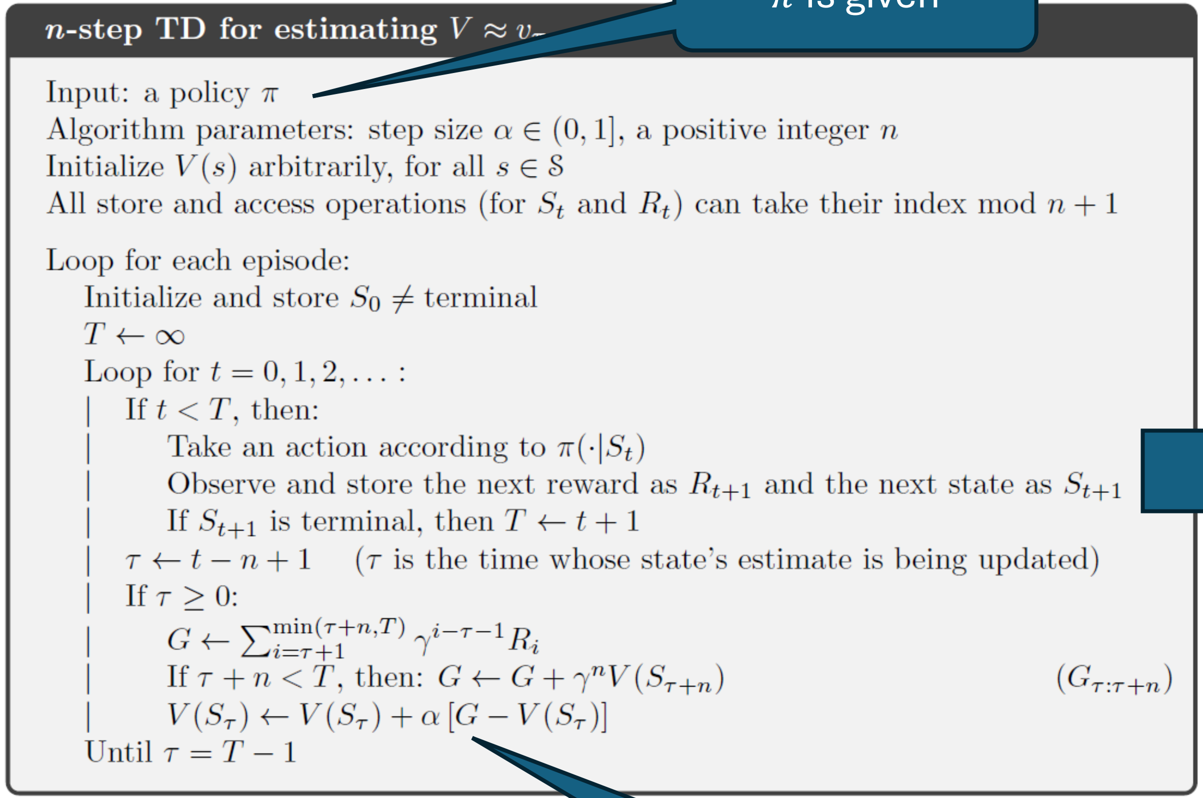

Path taken
Action values increased by one-step Sarsa
Action values increased by 10-step Sarsa
G
G
G

**Figure 7.4:** Gridworld example of the speedup of policy learning due to the use of $n$-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and $n$-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the $n$-step method strengthens the last $n$ actions of the sequence, so that much more is learned from the one episode.

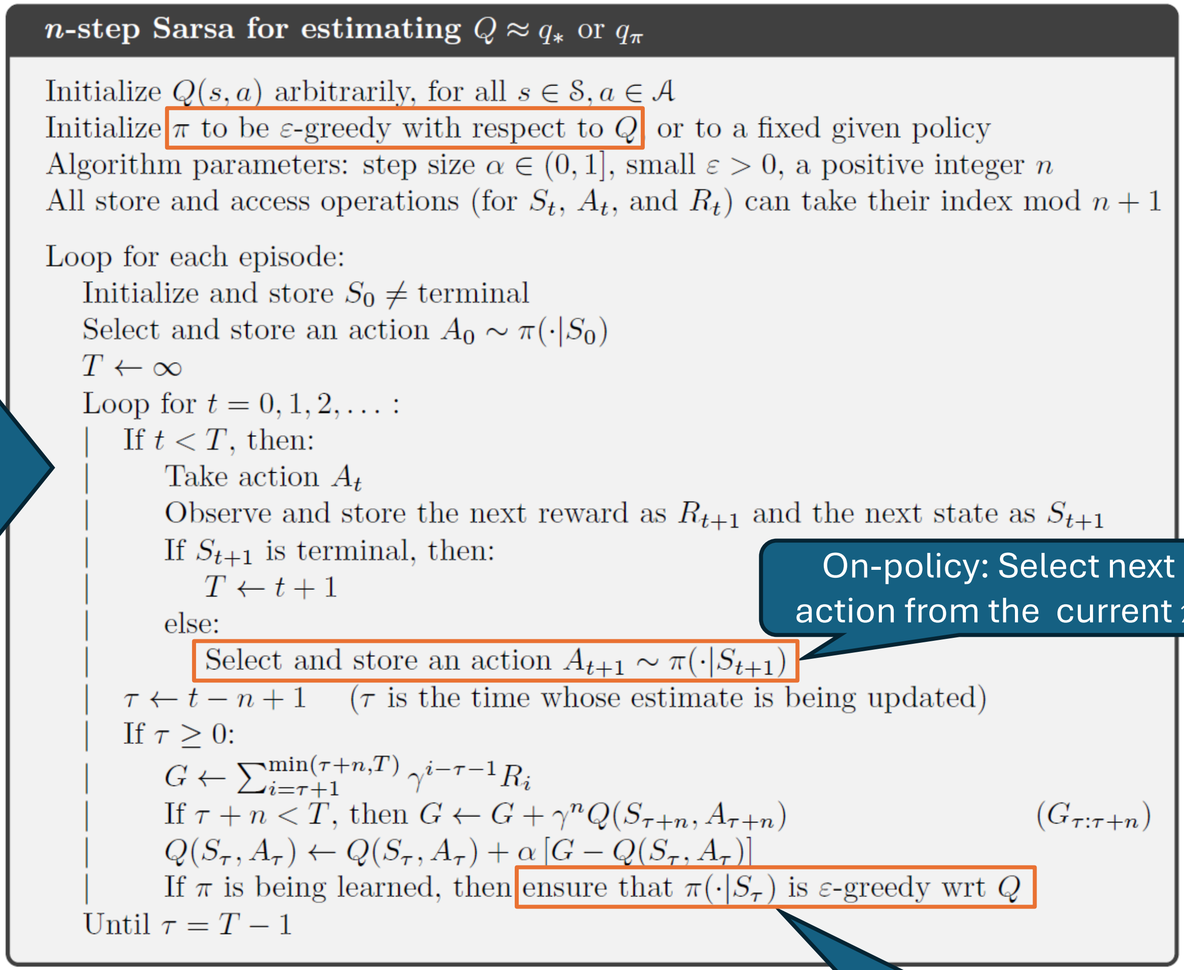
# Alg. From DT prediction to $n$-step Sarsa Control

$\pi$ is given



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

Input: a policy $\pi$
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 $T \leftarrow t+1$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$          $(G_{\tau:\tau+n})$
    |       $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
    Until $\tau = T - 1$
```

Update $V$



```
$n$-step Sarsa for estimating $Q \approx q_*$ or $q_\pi$

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$ or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take action $A_t$
    |       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:
    |           $T \leftarrow t+1$
    |       else:
    |           Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ is the time whose 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$          $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$
```

On-policy: Select next action from the current $\pi$

Update $Q$ and $\pi$

## $n$-step Off-Policy Control

Add importance sampling ratios to prediction and control.

# $n$-step Off-Policy Control

- **Goal**: Learn a deterministic target policy $\pi$ using an exploring (soft) behavioral policy $b$.
- **Method**: Use importance sampling ratio as a weight for updates

$$V_{t+1}(S_t) = V(S_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - V(S_t)]$$

where the weight is

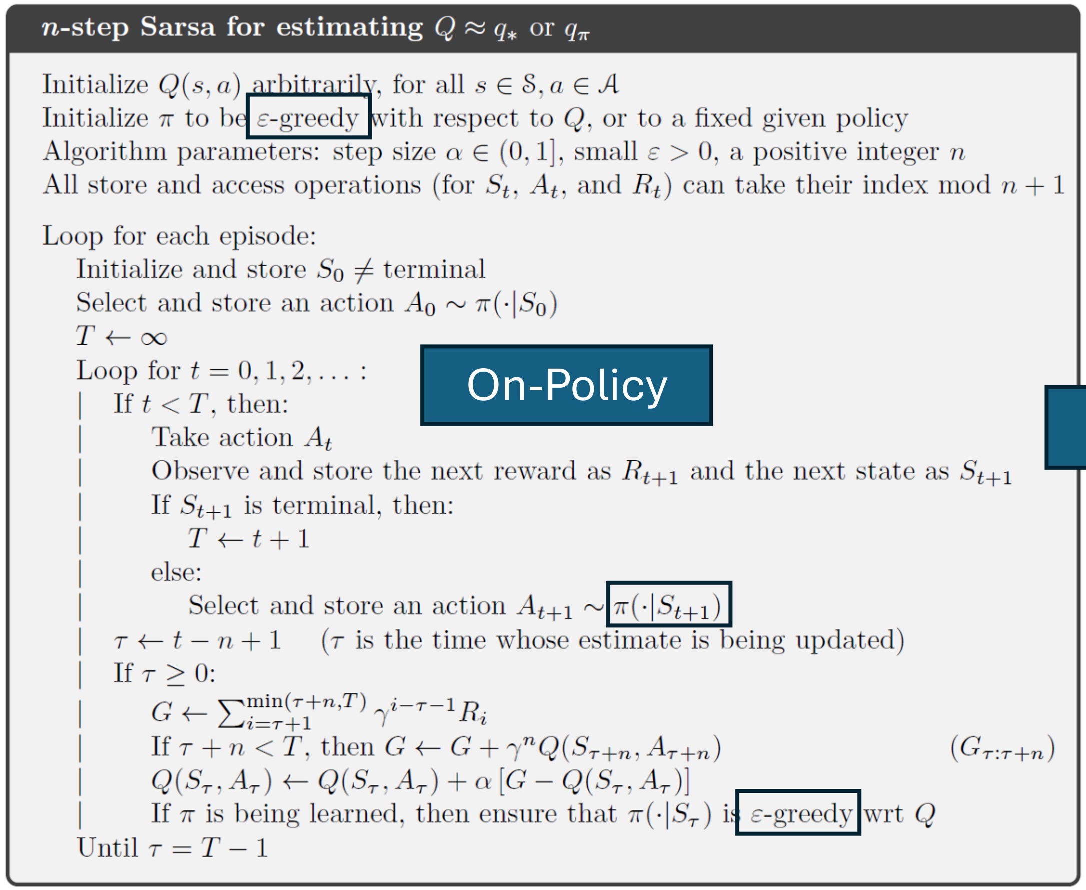
$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-t)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

How likely is $\pi$ to choose an action compared to $b$?

- Can be used in $n$-step TD for prediction or in $n$-step Sarsa for control.

- **Sample efficiency issue**: Can only learn (i.e., $\rho > 0$) from data where the sampled actions for the $n$-steps are compatible with $\pi$.
  Updating the behavior policy $b = \epsilon\text{–greedy}(\pi)$ with a small $\epsilon$ helps.

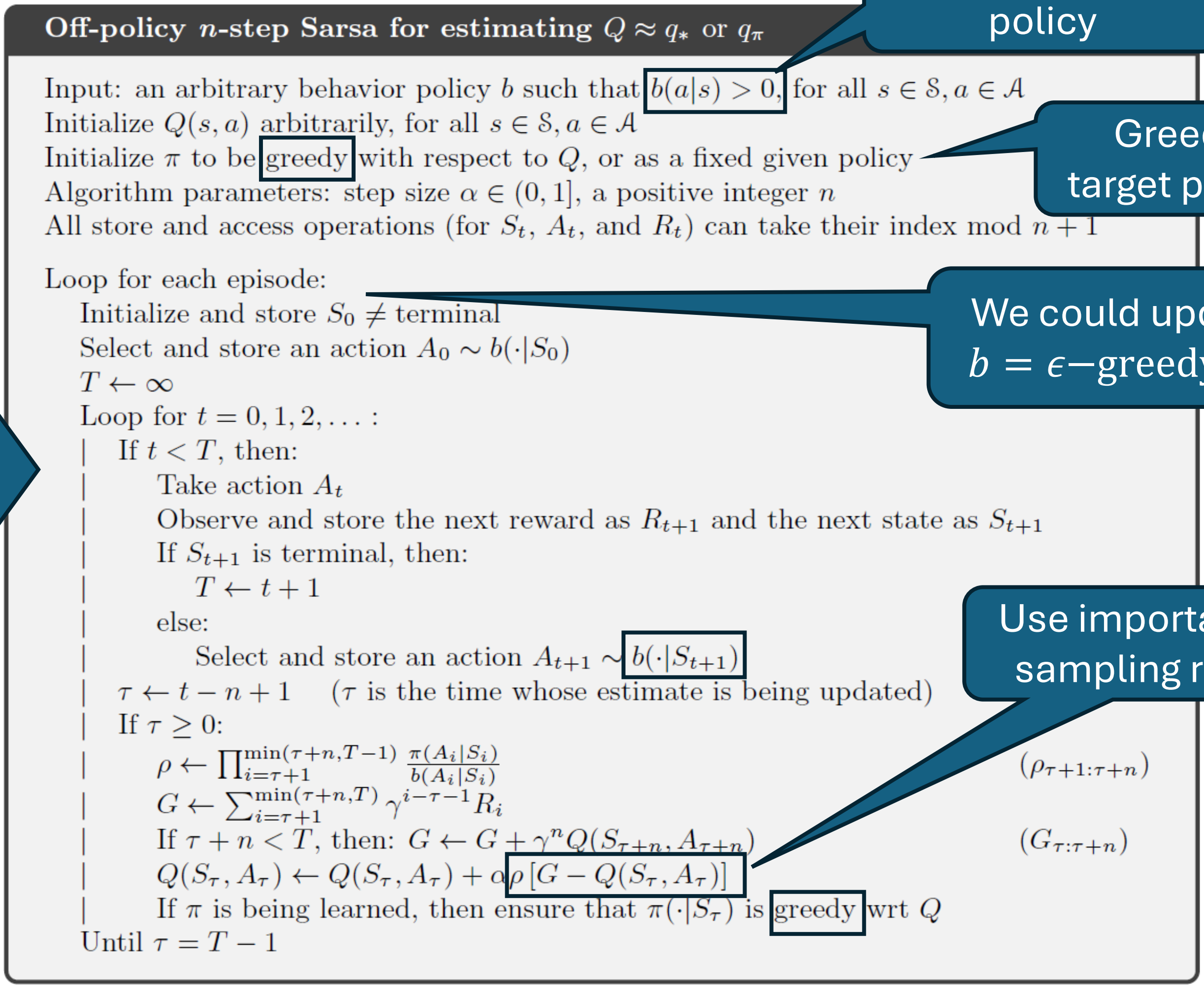
# Alg. Comparing On-Policy with Off-Policy



**$n$-step Sarsa for estimating $Q \approx q_*$ or $q_\pi$**

```
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\boxed{\varepsilon\text{-greedy}}$ with respect to $Q$, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take action $A_t$
    |       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:
    |           $T \leftarrow t+1$
    |       else:
    |           Select and store an action $A_{t+1} \sim \boxed{\pi(\cdot|S_{t+1})}$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ is the time whose 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$        $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\boxed{\varepsilon\text{-greedy}}$ wrt $Q$
    Until $\tau = T - 1$
```

On-Policy



**Off-policy $n$-step Sarsa for estimating $Q \approx q_*$ or $q_\pi$**

```
Input: an arbitrary behavior policy $b$ such that $\boxed{b(a|s) > 0}$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\boxed{\text{greedy}}$ with respect to $Q$, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim b(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take action $A_t$
    |       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:
    |           $T \leftarrow t+1$
    |       else:
    |           Select and store an action $A_{t+1} \sim \boxed{b(\cdot|S_{t+1})}$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ is the time whose estimate is being updated)
    |   If $\tau \geq 0$:
    |       $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$        $(\rho_{\tau+1:\tau+n})$
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$        $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \boxed{\rho \left[G - Q(S_\tau, A_\tau)\right]}$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\boxed{\text{greedy}}$ wrt $Q$
    Until $\tau = T - 1$
```

Add behavior policy

Greedy target policy

We could update $b = \epsilon\text{−greedy}(\pi)$

Use importance sampling ratio

# What you Need to Know

- $n$-step methods let us choose the amount of bootstrapping from 1-step to no bootstrapping (MC).

- Advantage:
  - **Learning**: Looking more steps ahead speeds up learning, especially for sparse rewards.
  - **Error reduction**: Significantly reduces the bootstrapping error.
  - An intermediate amount of bootstrapping typically **performs better** than the one-step and MC extremes. $n$ is application-dependent and can be tuned.

- Drawback:
  - **Delay**: Updates are delayed $n$-step since future events (rewards and states) are needed.
  - **Computation**: More computation per time step and higher memory needs than one-step methods.

**Eligibility traces** address the delay and computation drawbacks. We will cover these methods later.