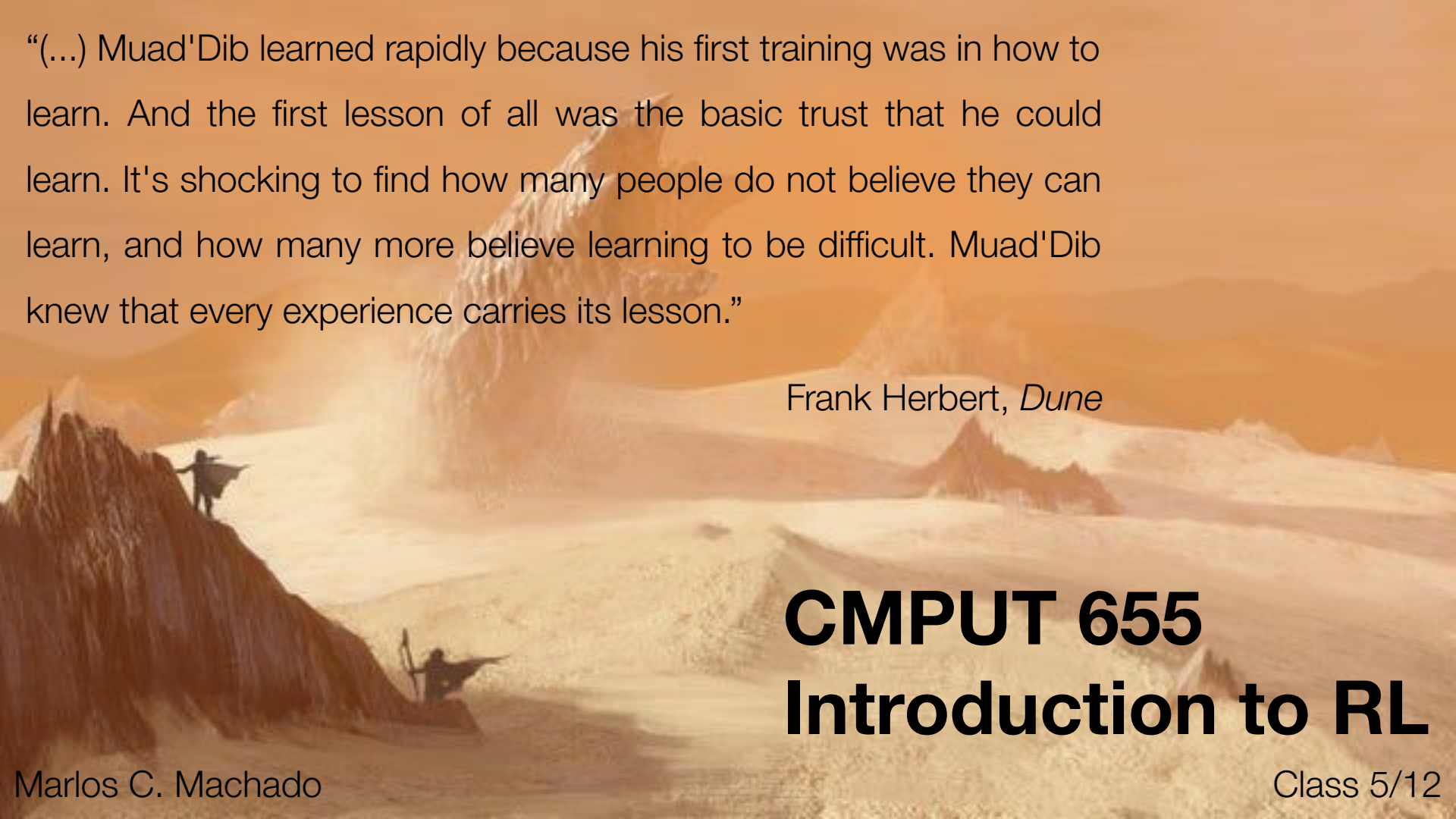

“(...) Muad'Dib learned rapidly because his first training was in how to learn. And the first lesson of all was the basic trust that he could learn. It's shocking to find how many people do not believe they can learn, and how many more believe learning to be difficult. Muad'Dib knew that every experience carries its lesson.”
Frank Herbert, *Dune*
CMPUT 655
Introduction to RL
Marlos C. Machado
Class 5/12

Plan

- Chapter 5: Monte-Carlo Methods
  - Off-Policy Prediction and Control with Importance Sampling
- Chapter 6: Temporal-Difference Learning
  - TD learning
  - Sarsa
  - Q-Learning
  - Expected Sarsa
- Chapter 7: n-Step Bootstrapping
- ~~General value functions~~

Reminder I

You **should be enrolled in the private session** we created in Coursera for CMPUT 365.

I **cannot** use marks from the public repository for your course marks.

You **need** to **check**, **every time**, if you are in the private session and if you are submitting quizzes and assignments to the private section.

The deadlines in the public session **do not align** with the deadlines in Coursera.

If you have any questions or concerns, **talk with the TAs** or email us `cmput655@ualberta.ca`.

Reminder II

- You just submitted 2 practice quizzes and 2 programming assignments:
  - Week 2 of Sample-based Learning Methods: TD Learning Methods for Prediction.
  - Week 3 of Sample-based Learning Methods: TD Learning Methods for Control.
- Next week only 1 practice quiz and programming assignment:
  - Week 4 of Sample-based Learning Methods: Planning, Learning, and Acting.
- **The project proposal is due in 2 weeks!**
  As well as, *for now*, 3 weeks of Coursera content: Prediction and Control with Function Approximation.

Please, interrupt me at any time!

https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Monte-Carlo Methods

Learning with exploration

- We stopped after *On-policy first-visit MC control (for ε-soft policies).*
- … but how can we learn about the optimal policy while behaving according to an exploratory policy? We need to behave non-optimally in order to explore 🤔.
- So far we have been *on-policy*, which is a compromise: we learn about a near-optimal policy, not the optimal one.
- But what if we had two policies? We use one for exploration but we learn about another one, which would be the optimal policy?

Behaviour policy

Target policy

That's off-policy learning!

Pros and cons of off-policy learning

Pros

- It is more general.
- It is more powerful.
- It can benefit from external data
  - and other additional use cases.

Cons

- It is more complicated.
- It has much more variance.
  - Thus it can be much slower to learn.
- It can be unstable.

Check Example 5.5 in the textbook about Infinite Variance

What's the actual issue?

Let π denote the target policy, and let b denote the behaviour policy.

We want to estimate $\mathbb{E}_{\pi}[G_t]$, but what we can actually directly estimate is $\mathbb{E}_{b}[G_t]$.

In other words, $\mathbb{E}[G_t \mid S_t = s] = v_b(s)$.

Importance Sampling

A general technique for estimating expected values under one distribution given samples from another. It is based on re-weighting the probabilities of an event.

Importance Sampling

In RL, the probability of a trajectory is:

$$\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&\quad = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&\quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}$$

Importance Sampling

In RL, the probability of a trajectory is:

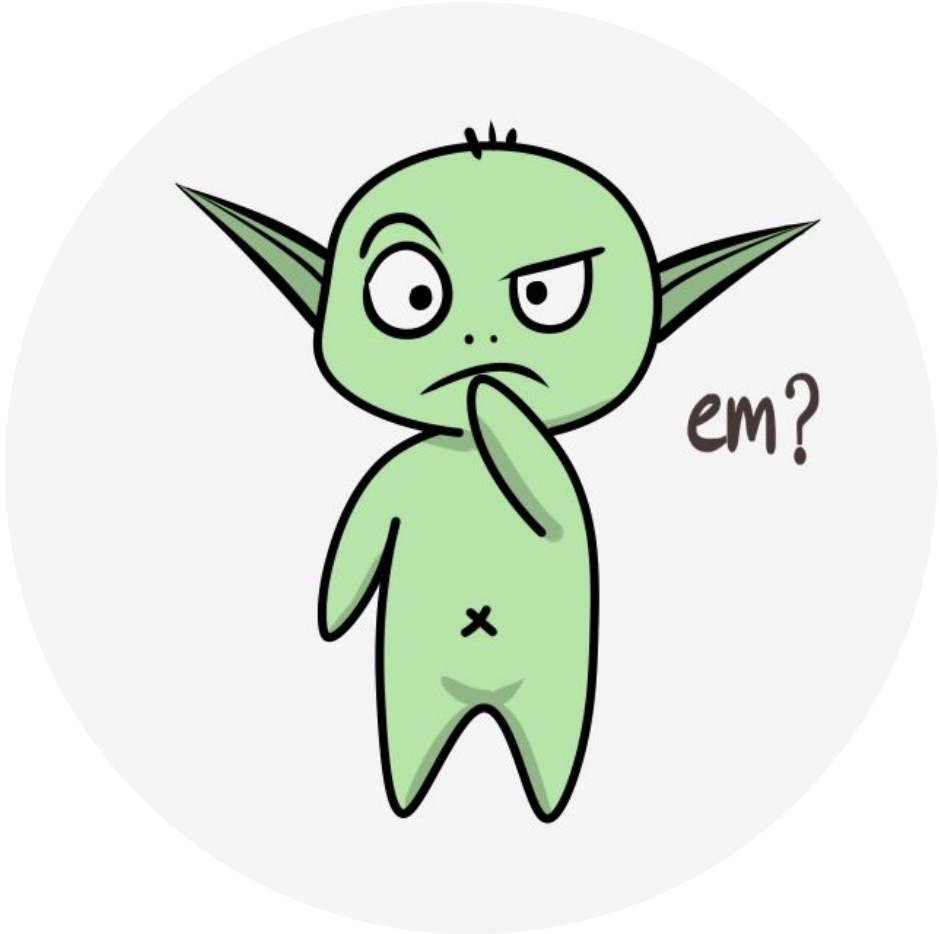
$$\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&\quad = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&\quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}$$

the relative prob. of the traj. under the target and behavior policies (the IS ratio) is:

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

We require *coverage*: b(a|s) > 0 when π(a|s) > 0

The IS ratio does not depend on the MDP, that is, on p(s', r | s, a)!



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

The solution

The ratio $\rho_{t:T-1}$ transforms the returns to have the right expected value:

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s).$$

Ordinary importance sampling:

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}.$$

Set of all time steps in which state s is visited.

Weighted importance sampling:

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

Incremental update (Weighted IS)

We want to form the estimate

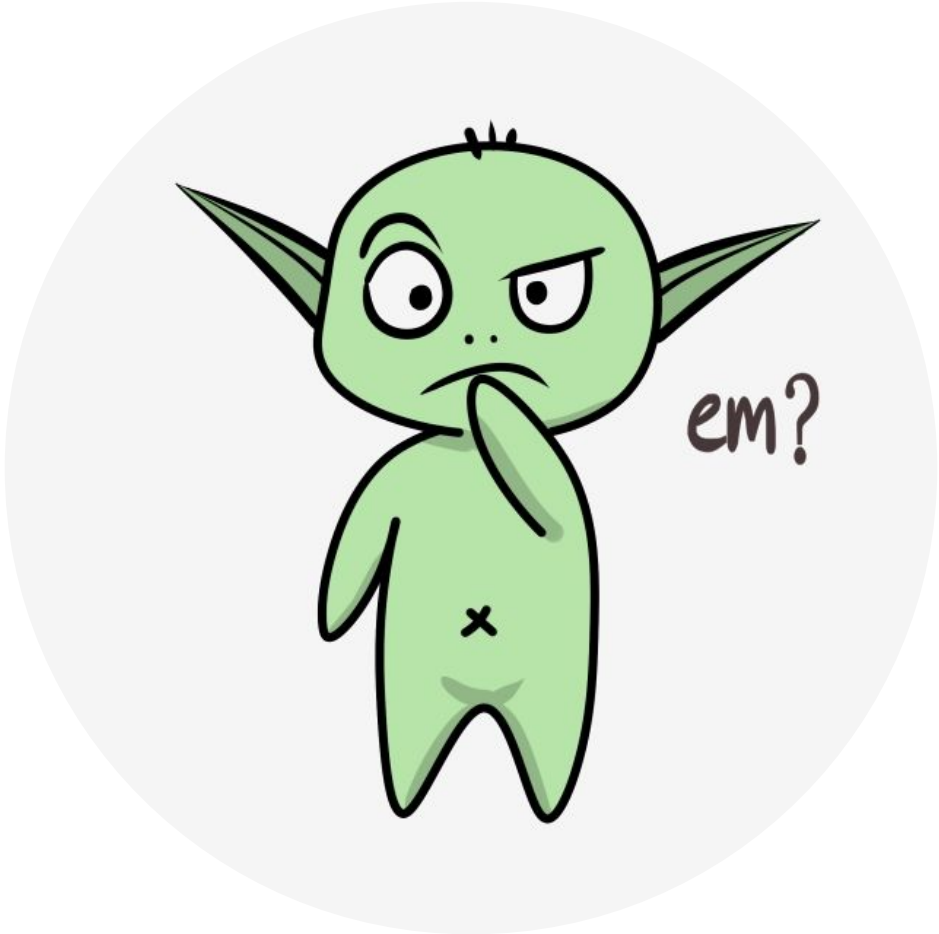
$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \qquad n \geq 2 \qquad\qquad W_i = \rho_{t_i:T(t_i)-1}$$

The update rule for V_n is

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n}\Big[G_n - V_n\Big], \qquad n \geq 1,$$

and

$$C_{n+1} \doteq C_n + W_{n+1}$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Putting Everything Together for Prediction

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

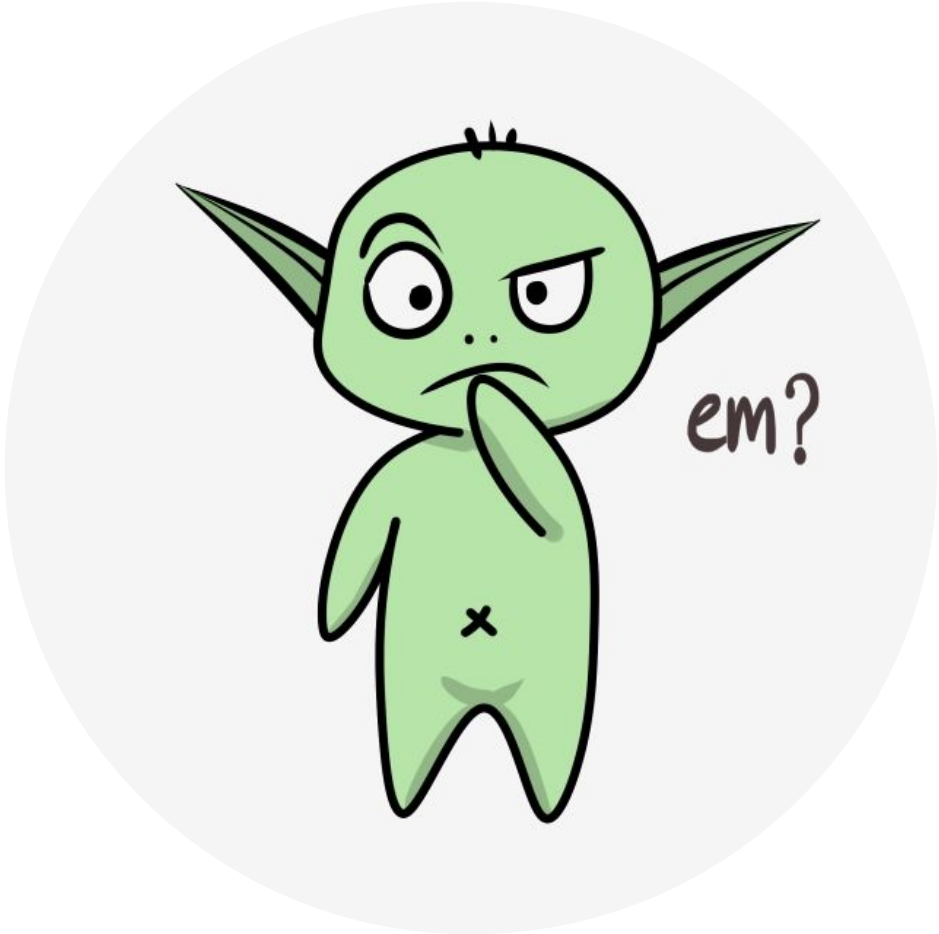
Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of π
    Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Putting Everything Together for Control

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \in \mathbb{R}$ (arbitrarily)
$C(s,a) \leftarrow 0$
$\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
$b \leftarrow$ any soft policy
Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
$W \leftarrow W \frac{1}{b(A_t|S_t)}$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Interlude

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

A flavour of RL: Bandits (Chapter 2)

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

But what does that mean?
What is this sequential decision-making problem we are trying to solve?
What does solution mean here?

A problem formulation: MDPs (Chapter 3)

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

- What about the solution?

A first solution: Dynamic Programming (Chapter 4)

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

- What about the solution?
  - Dynamic programming! ← We need to know p(s', r | s, a) and it can be computationally expensive to solve the system of linear equations.

Our first learning algorithm: Monte Carlo Methods (Chapter 5)

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

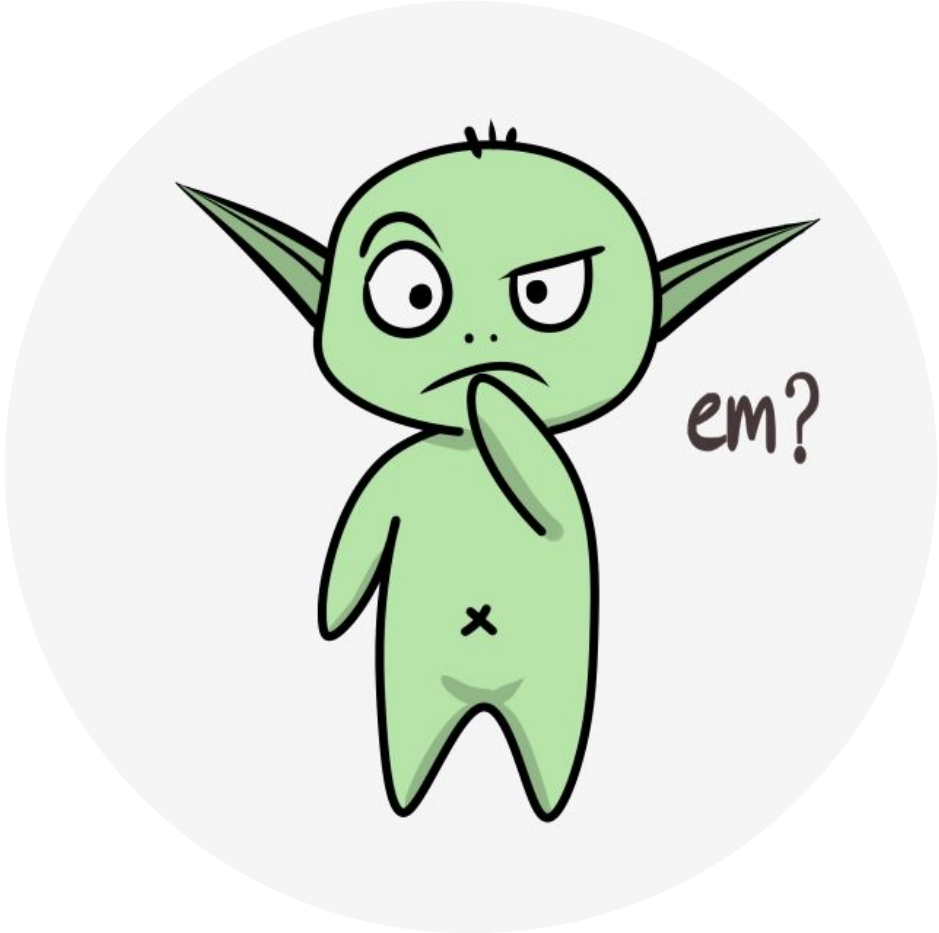
- What about the solution?
  - Dynamic programming
  - Monte Carlo methods ← Do we really need to wait until the end of an episode to learn something?

The core idea behind RL: Temporal-Difference Learning (Chapter 6)

An overview

- Main features of a reinforcement learning problem:
  - Trial-and-error learning
  - Exploration
  - Delayed credit assignment

- What about the solution?
  - Dynamic programming
  - Monte Carlo methods
  - TD learning ← We learn from experience like MC methods but we bootstrap like we do in DP.



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Chapter 6

Temporal-Difference Learning

Prediction

Temporal-difference learning – Why?

“If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning.”

TD Prediction

A simple every-visit Monte Carlo method is:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

What if we don't want to wait until we have a full return (end of episode)!

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \; \Big[Target - OldEstimate\Big]$$

TD Prediction

A simple every-visit Monte Carlo method is:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{G_t}_{\textbf{Target}} - V(S_t)\Big]$$

Temporal-Difference Learning:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\textbf{Target}} - V(S_t)\Big]$$

TD Prediction

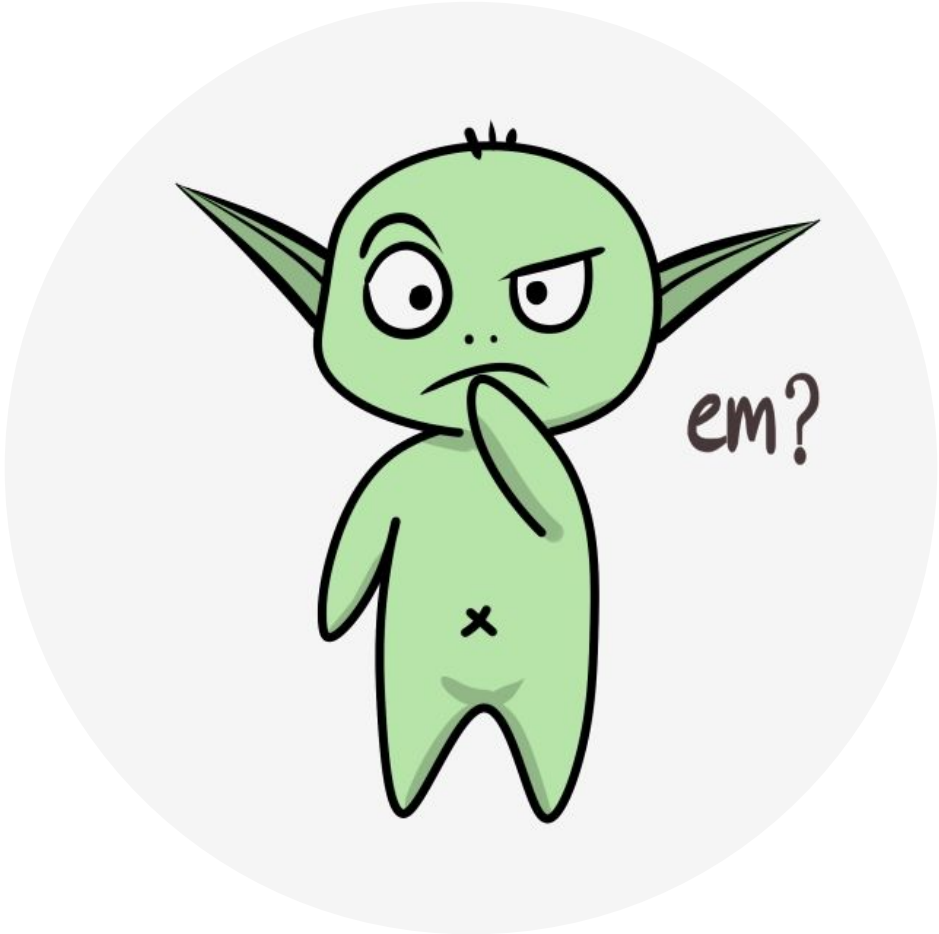
A simple every-visit Monte Carlo method is:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

Temporal-Difference Learning (specifically, **one-step TD**, or **TD(0)**):

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

These are estimates all the way down…



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

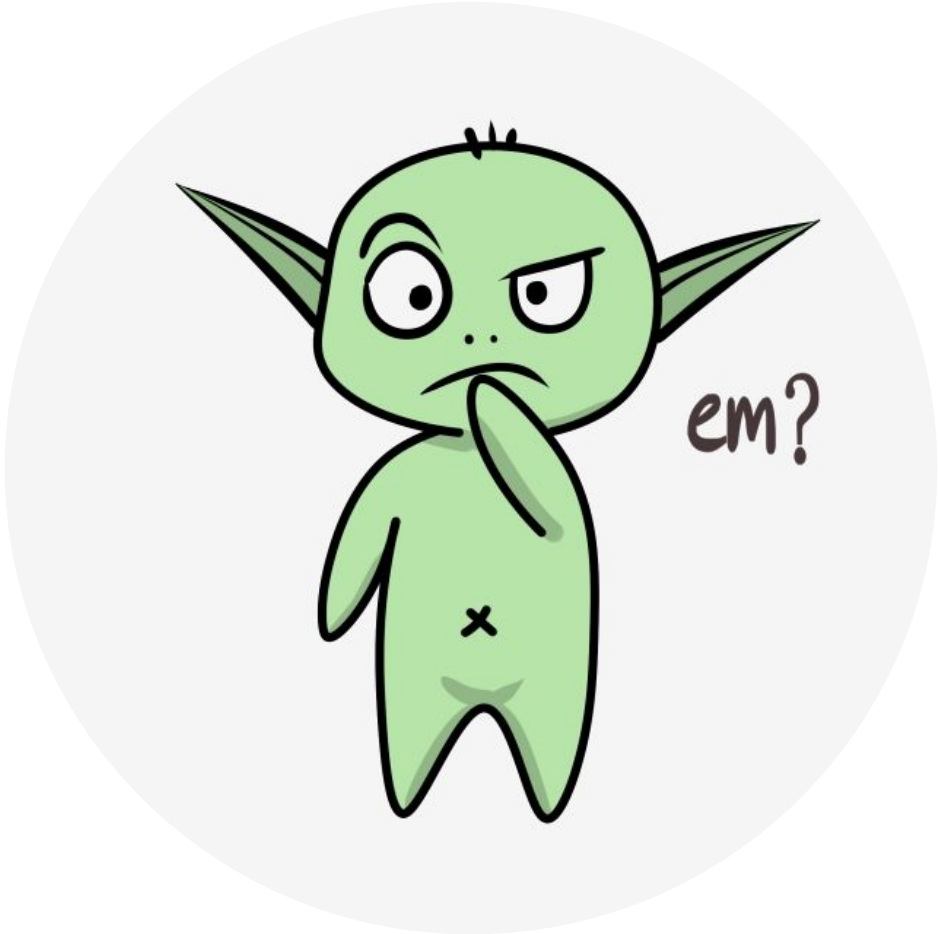
Tabular TD(0)

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

Sample update



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Temporal-Difference Error

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

Example – Driving Home

Example 6.1: Driving Home Each day as you drive home from work, you try to predict how long it will take to get home. When you leave your office, you note the time, the day of week, the weather, and anything else that might be relevant. Say on this Friday you are leaving at exactly 6 o'clock, and you estimate that it will take 30 minutes to get home. As you reach your car it is 6:05, and you notice it is starting to rain. Traffic is often slower in the rain, so you reestimate that it will take 35 minutes from then, or a total of 40 minutes. Fifteen minutes later you have completed the highway portion of your journey in good time. As you exit onto a secondary road you cut your estimate of total travel time to 35 minutes. Unfortunately, at this point you get stuck behind a slow truck, and the road is too narrow to pass. You end up having to follow the truck until you turn onto the side street where you live at 6:40. Three minutes later you are home.

Example – Driving Home

The sequence of states, times, and predictions is thus as follows:

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Example – Driving Home

The rewards in this example are the elapsed times on each leg of the journey.[1] We are not discounting ($\gamma = 1$), and thus the return for each state is the actual time to go from that state. The value of each state is the *expected* time to go. The second column of numbers gives the current estimated value for each state encountered.

The sequence of states, times, and predictions is thus as follows:

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Example – Driving Home

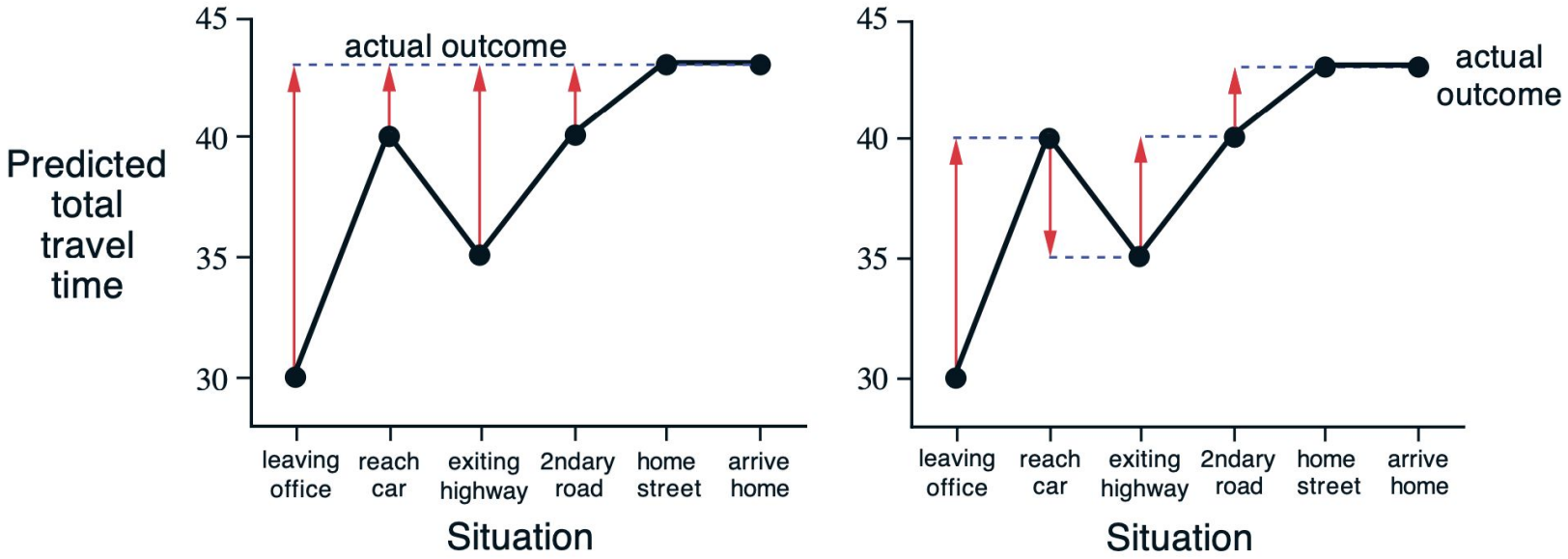
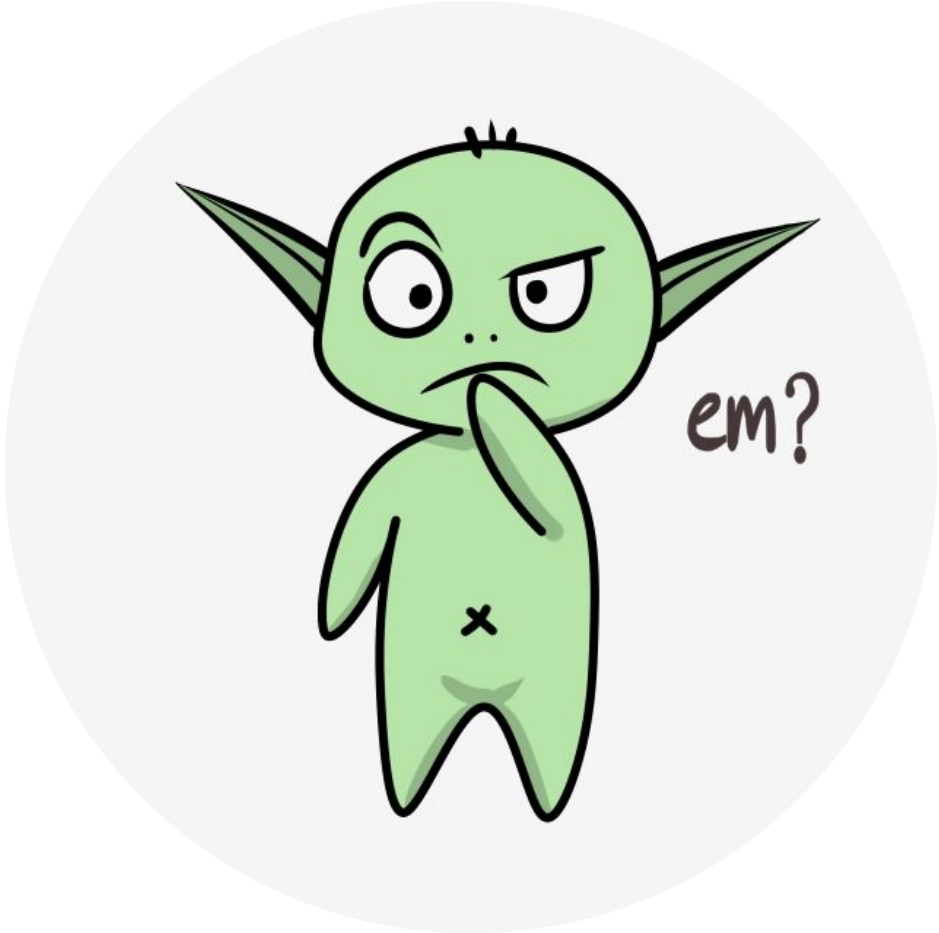


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Optimality of TD(0)

- Under batch training, constant-α MC converges to values, V(s), that are sample averages of the actual returns experienced after visiting each state s. These are optimal estimates in the sense that they minimize the mean square error from the actual returns in the training set.

- Bath TD(0) gives us the answer that it is based on first modeling the Markov process and then computing the correct estimates given the model (the *certainty-equivalence estimate*).

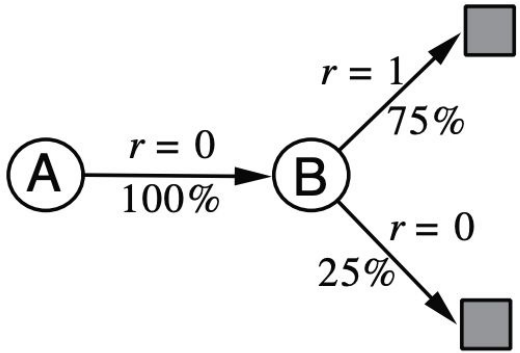
Example

Example 6.4: You are the Predictor Place yourself now in the role of the predictor of returns for an unknown Markov reward process. Suppose you observe the following eight episodes:

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

V(A) = ?

V(B) = ?



Example

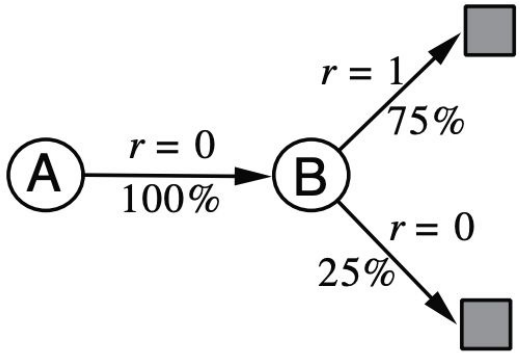
Example 6.4: You are the Predictor Place yourself now in the role of the predictor of returns for an unknown Markov reward process. Suppose you observe the following eight episodes:

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

TD MC

V(A) = ? ¾ or 0?

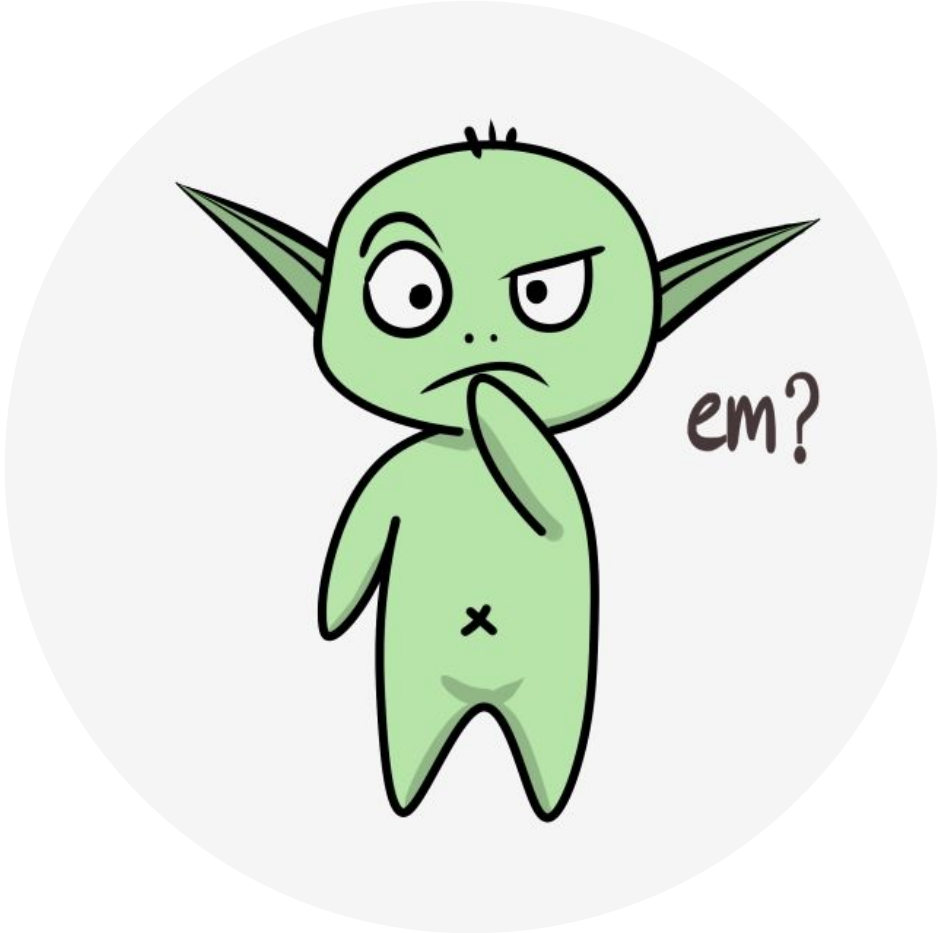
V(B) = ¾



TD vs Monte Carlo

"Batch Monte Carlo methods always find the estimates that minimize mean square error on the training set, whereas batch TD(0) always finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process."

In general, the *maximum-likelihood estimate* of a parameter is the parameter value whose probability of generating the data is greatest.

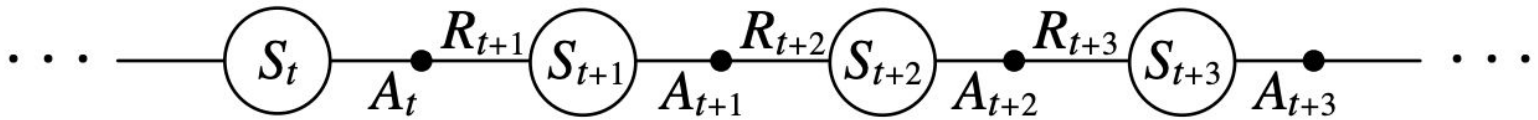


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Control

Sarsa: On-policy Control

- We again use generalized policy iteration (GPI), but now using TD for evaluation.
- We need to learn an action-value function instead of a state-value function. We can do this!



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

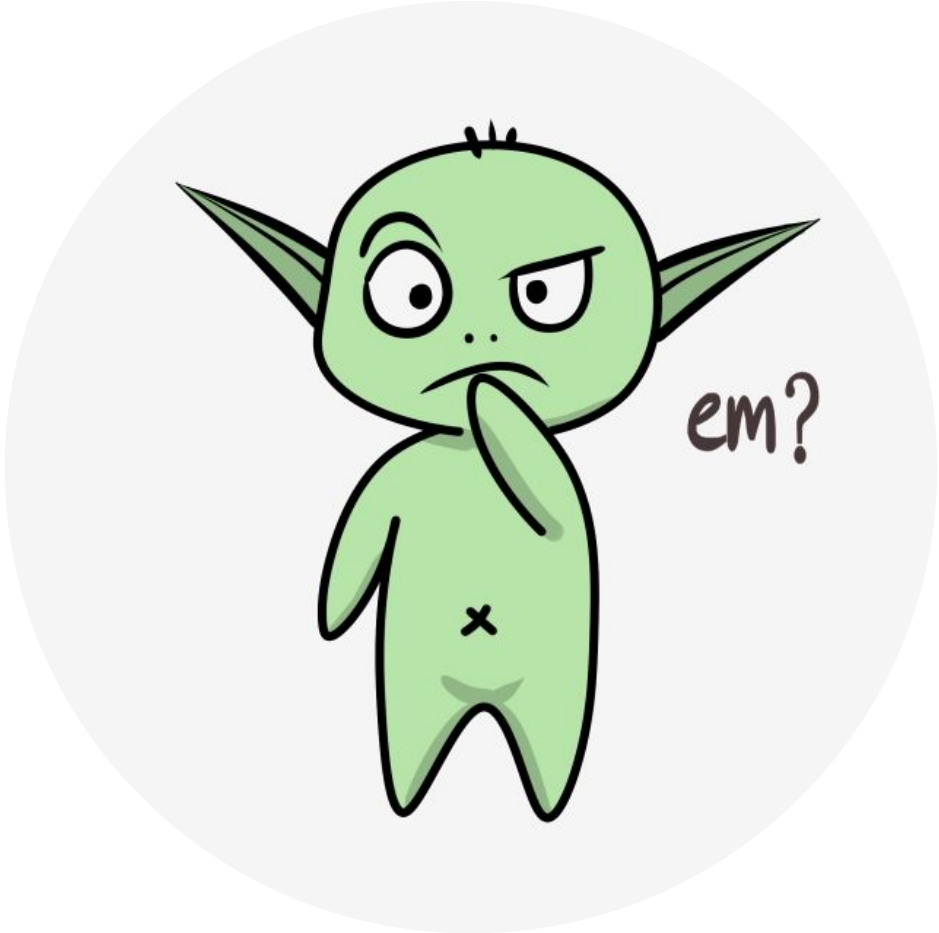
Sarsa: On-policy Control

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

We need to explore!



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Q-Learning: Off-Policy Control

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

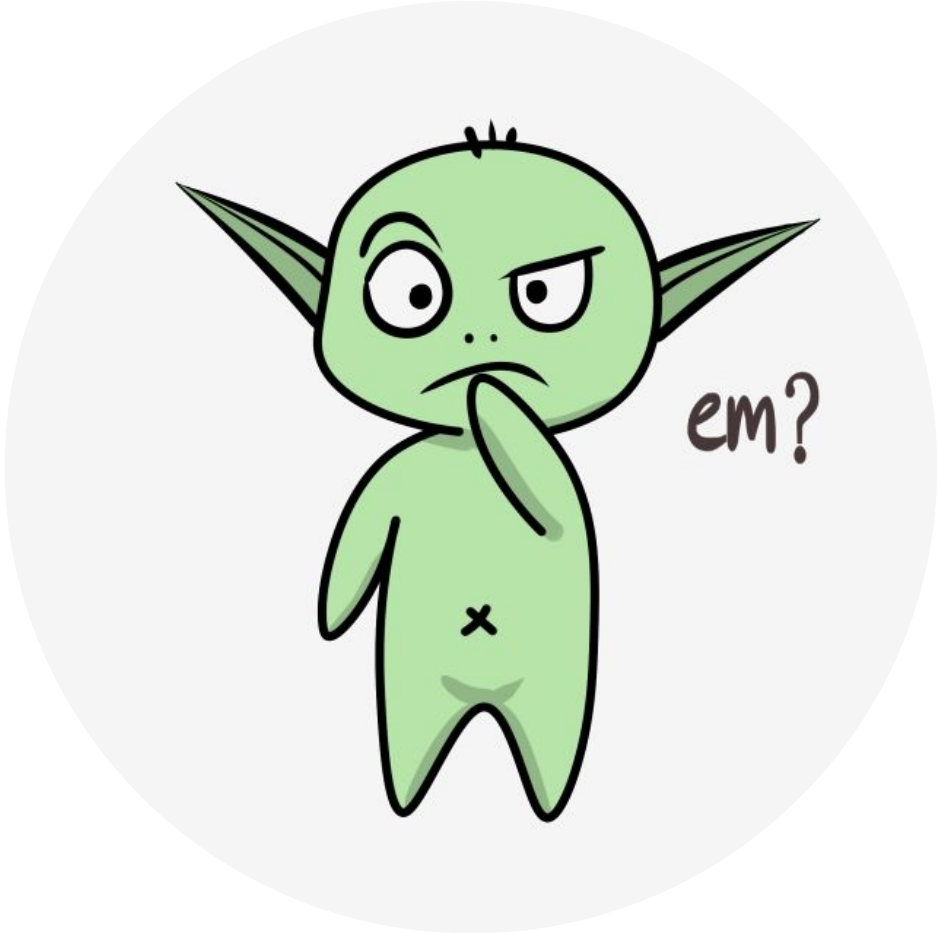
- Q directly approximates q_*, regardless of the policy being followed.
- Notice we do not need importance sampling. We are updating a state–action pair. We do not have to care how likely we were to select the action; now that we have selected it we want to learn fully from what happens.

Q-Learning: Off-Policy Control

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

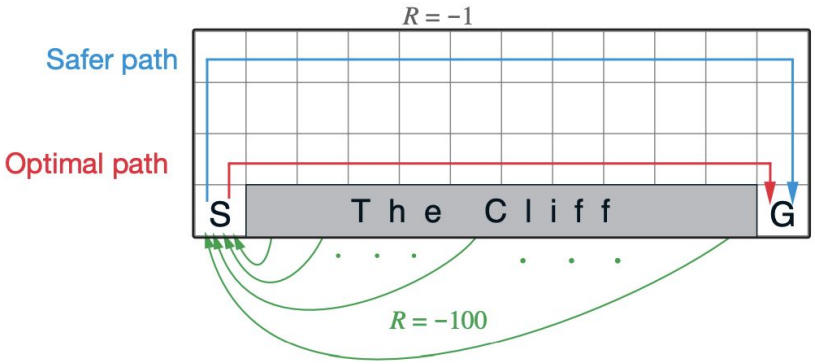
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

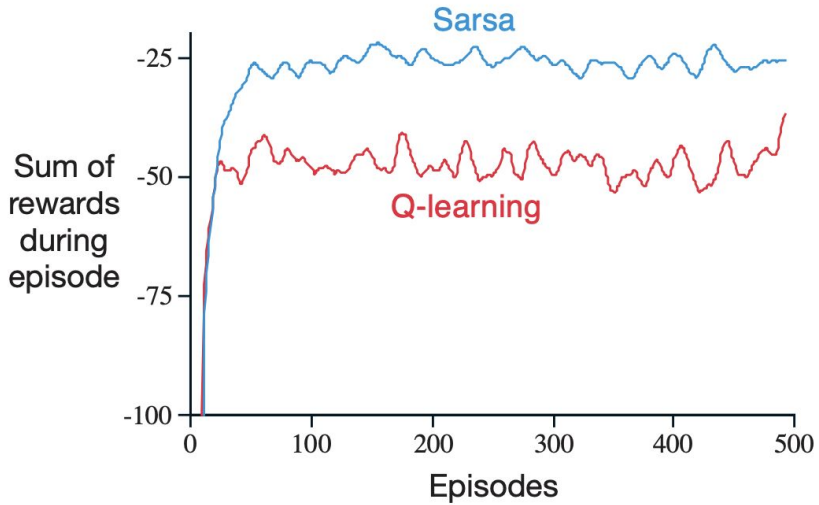
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Example – Q-Learning vs Sarsa



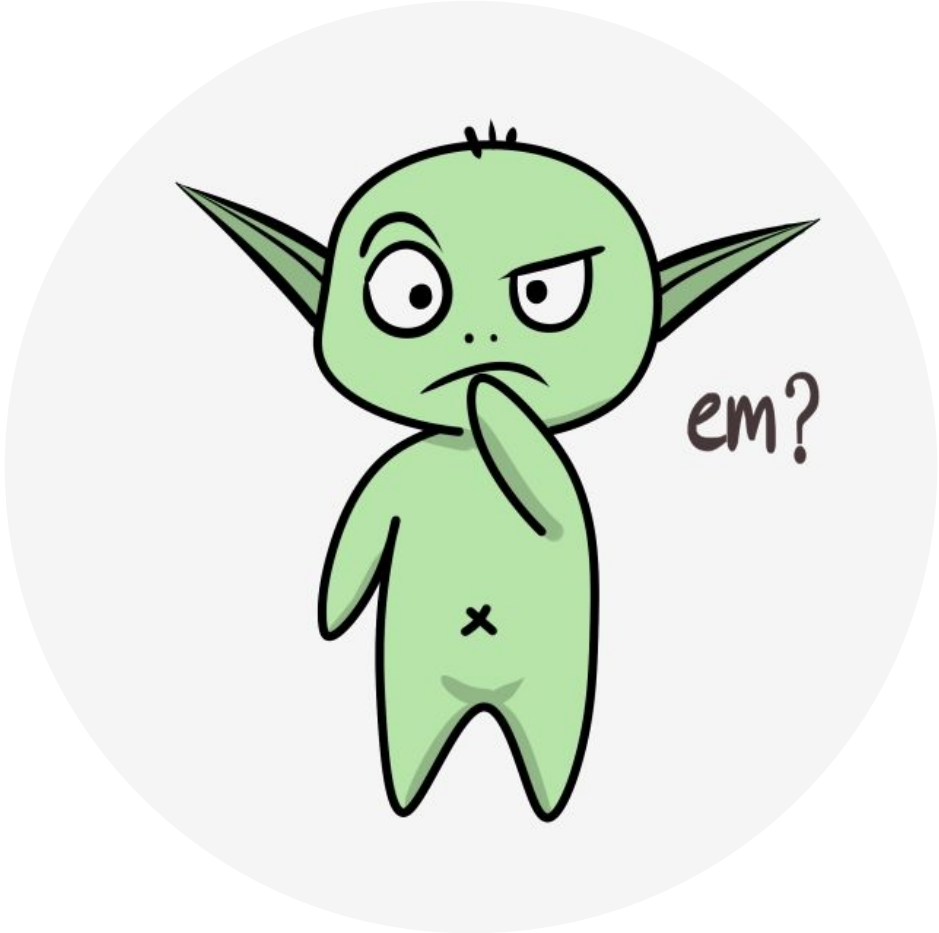


Discussion

Exercise 6.12. Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates?

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

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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Expected Sarsa

What if instead of the maximum over next state-action pairs we used the expected value, taking into account how likely each action is under the current policy?

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\
&= Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big],
\end{aligned}
$$

Expected Sarsa

What if instead of the maximum over next state-action pairs we used the expected value, taking into account how likely each action is under the current policy?

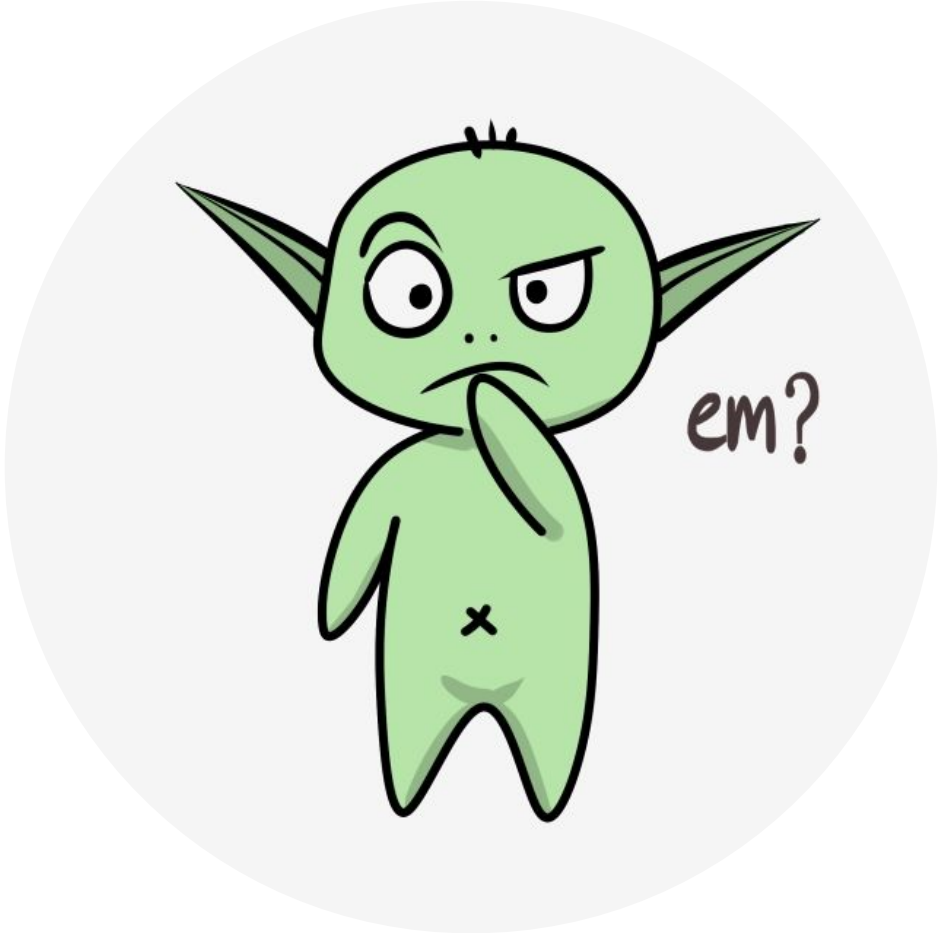
$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\
&= Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big],
\end{aligned}
$$

Expected Sarsa is more computationally expensive than Sarsa but, in return, it eliminates the variance due to the random selection of A_{t+1}.

Is Expected Sarsa on-policy or off-policy?

Is Expected Sarsa on-policy or off-policy?

Expected can use a policy different from the target policy π to generate behavior (thus, it can be off-policy; although one can use it on-policy as well).



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

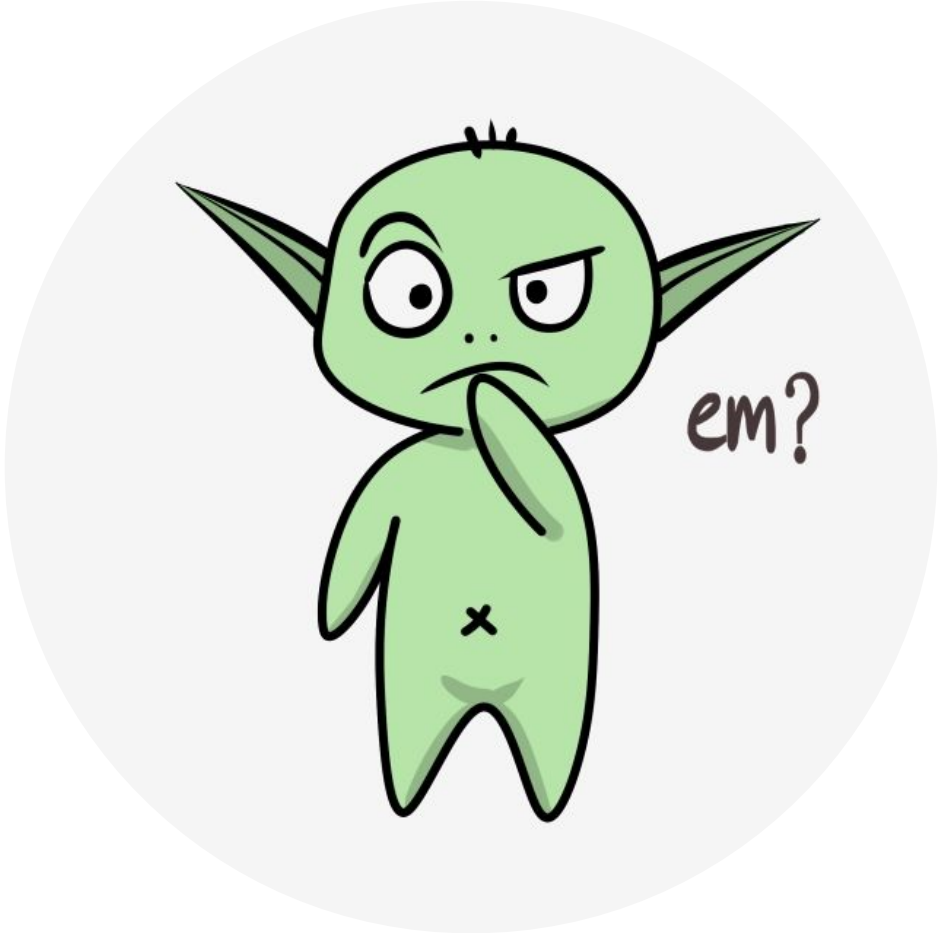
Maximization Bias

- The control algorithms we discussed so far use a maximization to get their target policies (either a max/greedy policy or an ε-greedy policy).
- *Maximization bias:* A maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias.

Double Learning

- The issue is that we use the same samples to determine the maximizing action and to estimate its value.
- In Bandits:
  - Split the data, learn $Q_1(a)$ and $Q_2(a)$ to estimate $q(a)$.
  - Choose actions according to one estimate and get estimate from the other:
    $A^* = \text{argmax}_a\ Q_1(a)$ $\quad\quad$ $Q_2(A^*) = Q_2(\text{argmax}_a\ Q_1(a))$
  - This leads to unbiased estimates, that is: $\mathbb{E}[Q_2(A^*)] = q(A^*)$

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2\left(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\right) - Q_1(S_t, A_t) \right]$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

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 $Q(\textit{terminal}, \cdot) = 0$

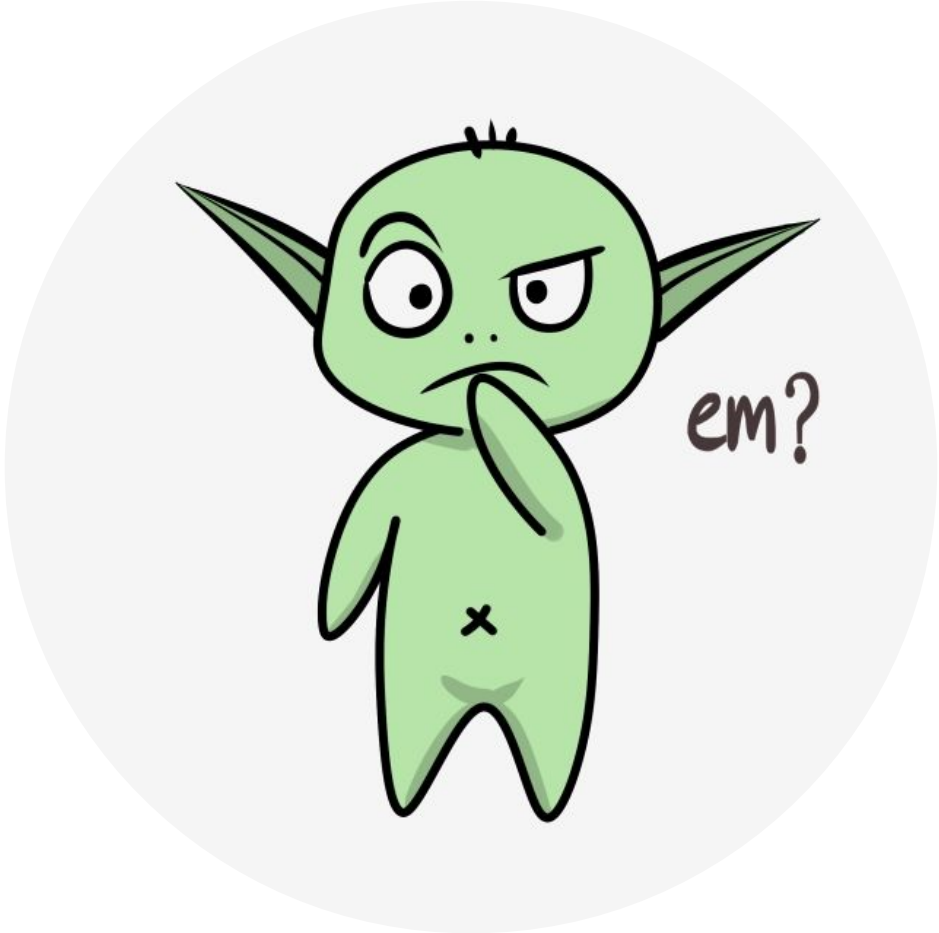
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

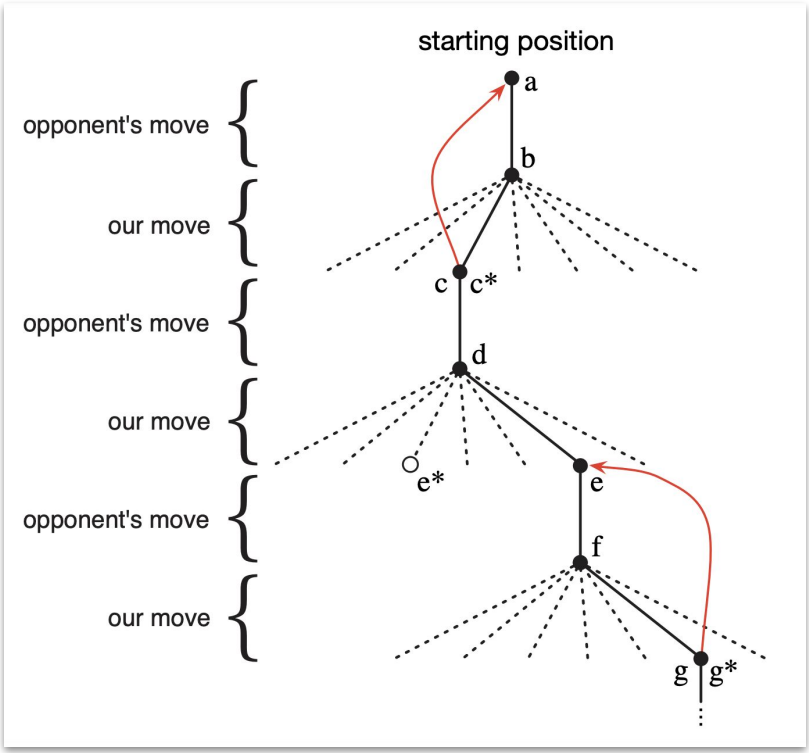


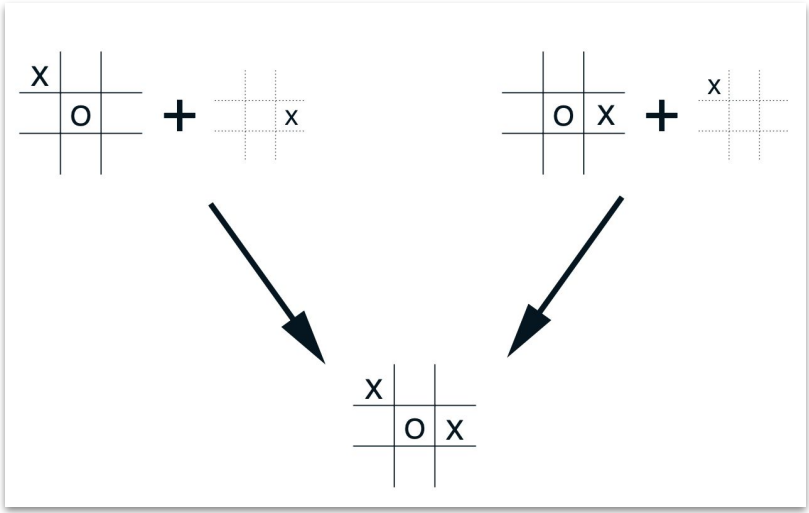
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

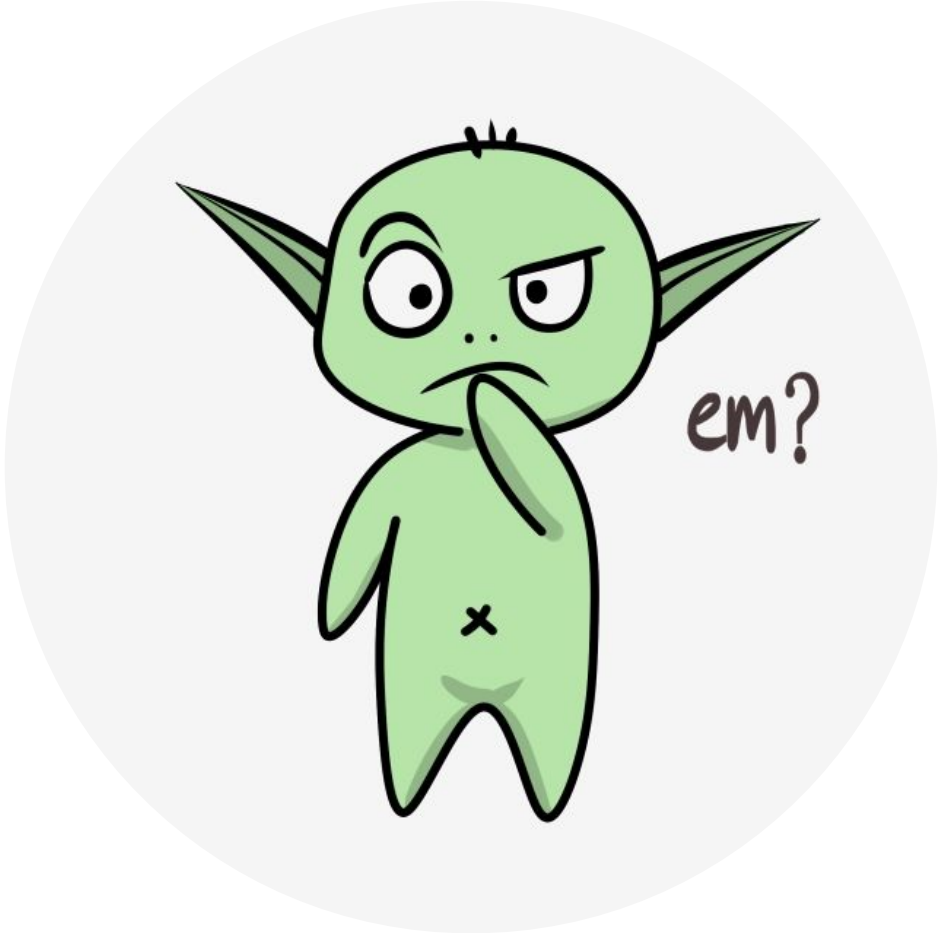
Afterstates

- One could evaluate states *after* the agent has taken an action (instead of states in which the agent has the option to select an action).
- This is particularly useful when we have knowledge of an initial part of the environment's dynamics but not necessarily of the full dynamics (e.g., how an opponent will reply in a game).
- This can be much more efficient!

Example – Tic-Tac-Toe







https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Chapter 7

n-step Bootstrapping

Prediction

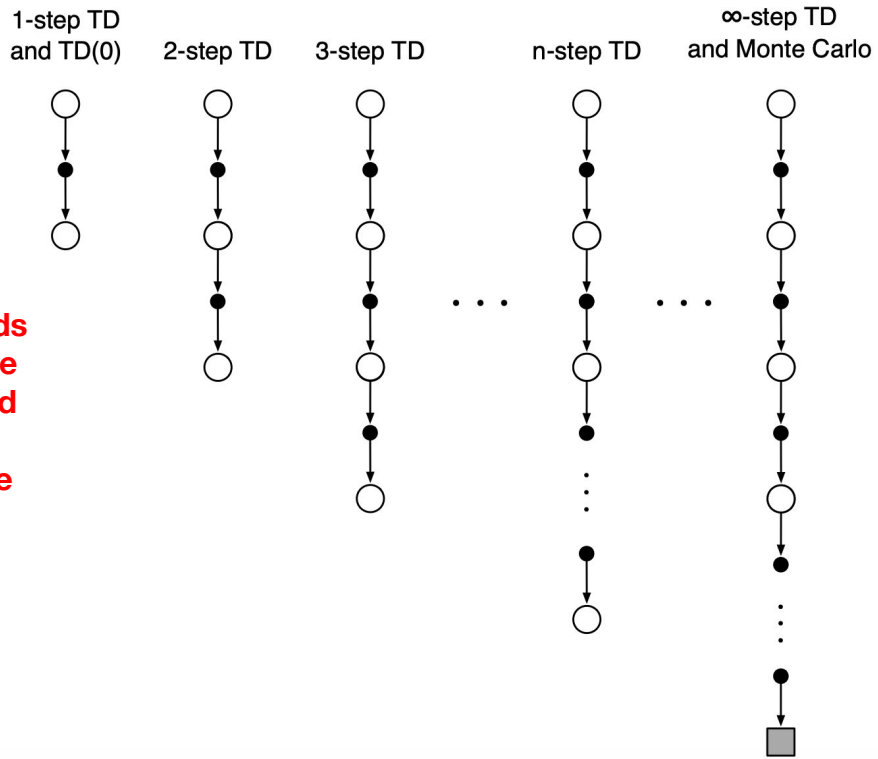
n-step Bootstrapping

- Can we unify Monte Carlo and TD methods to get the best of two worlds?
- n-step methods span a spectrum with MC methods at one end and one-step TD methods at the other. The best methods are often intermediate between the two extremes.
- n-step methods enable bootstrapping to occur over multiple steps, freeing us from the tyranny of the single time step.

n-step TD Prediction

- MC methods perform an update for each state based on the entire sequence of observed rewards (until the end of the episode).
- One-step TD methods have an update that is based on just the one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards.
- Intermediate method: perform an update based on an intermediate number of rewards: more than one, but less than all of them until termination ¯_(ツ)_/¯

n-step TD Prediction



These are still TD methods because they still change an earlier estimate based on how it differs from a later estimate. These are n-step TD methods.

n-step TD Prediction

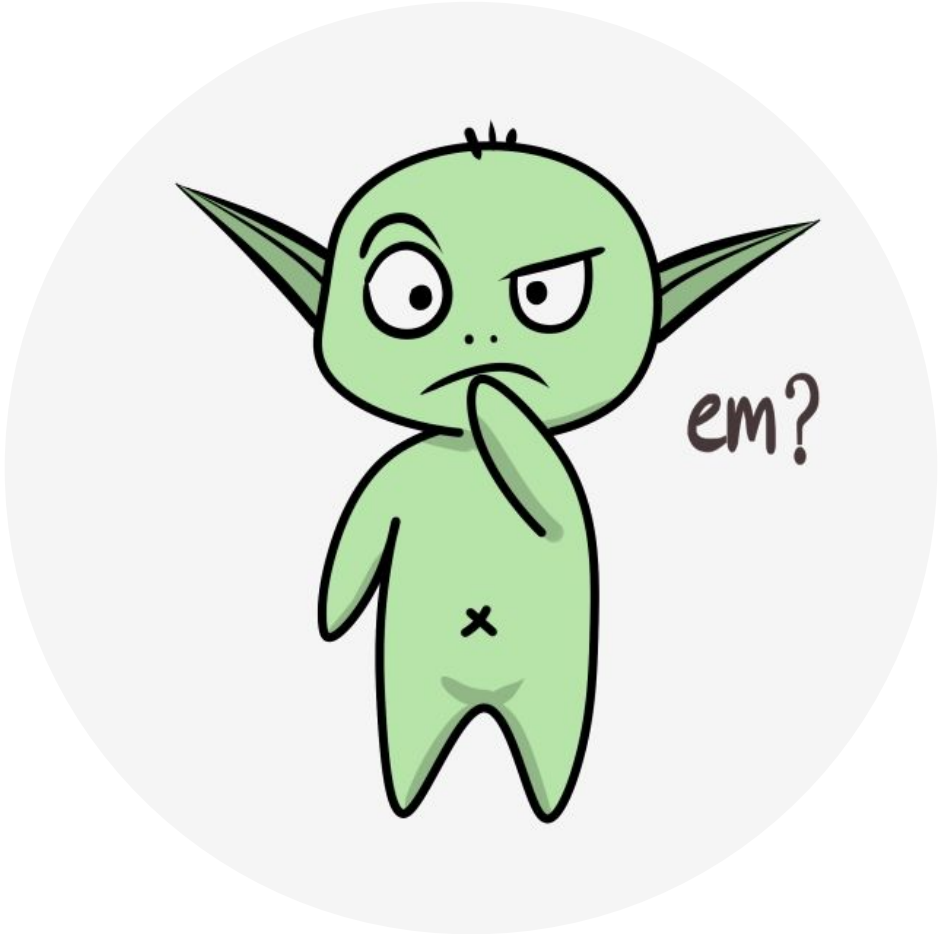
- Complete return: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

n-step TD Prediction

- Complete return: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
- One-step return: $G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

n-step TD Prediction

- Complete return: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
- One-step return: $G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
- Two-step return: $G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$
- *n*-step return: $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

n-step TD Learning Update Rule

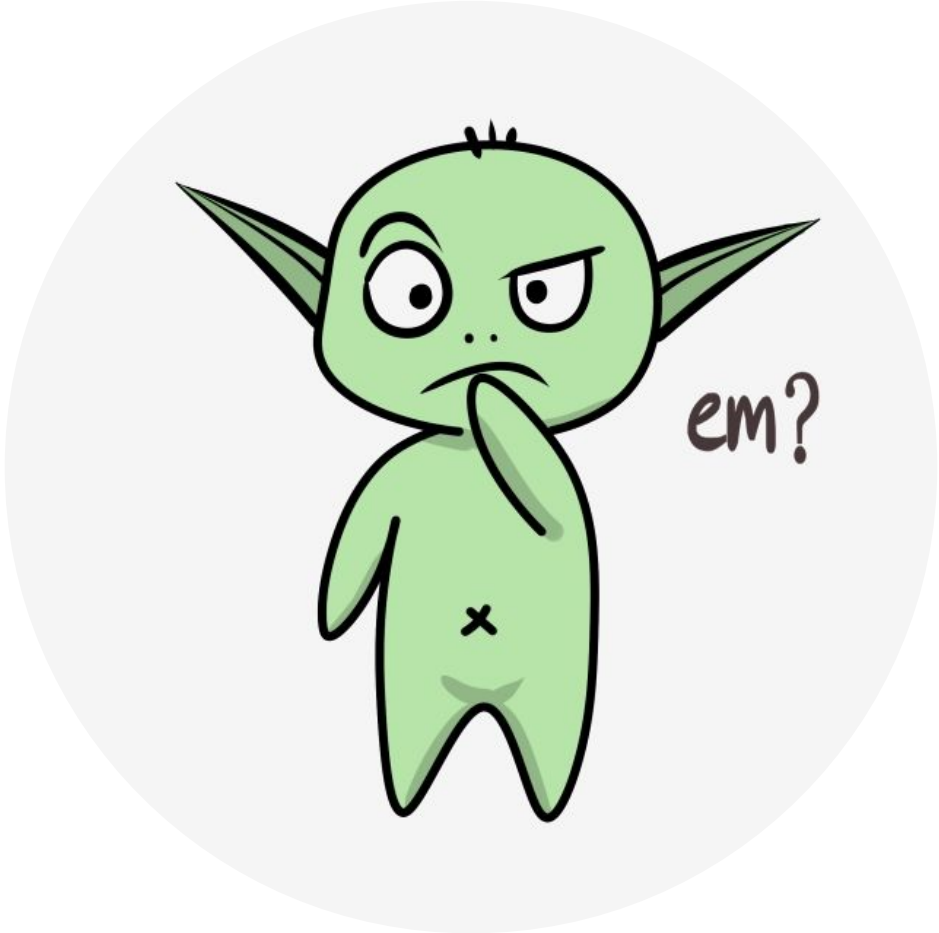
$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \leq t < T$$

n-step TD Learning Update Rule

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 $n+1$

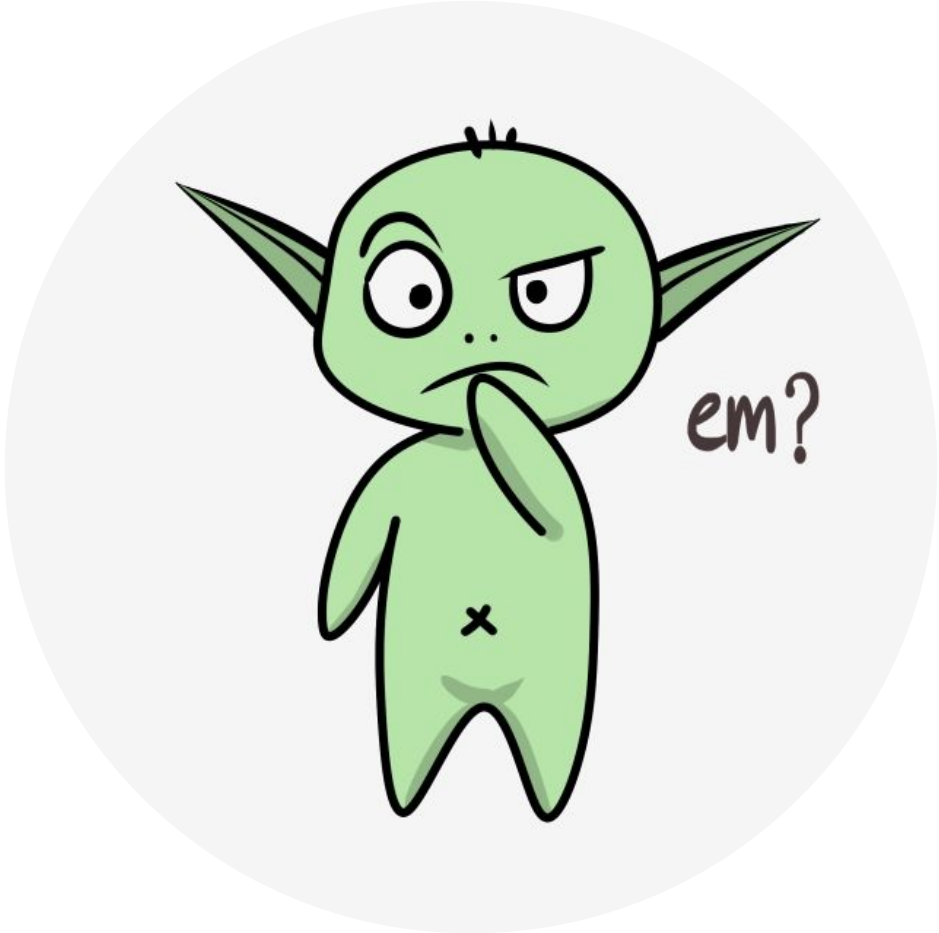
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Error reduction property of n-step returns

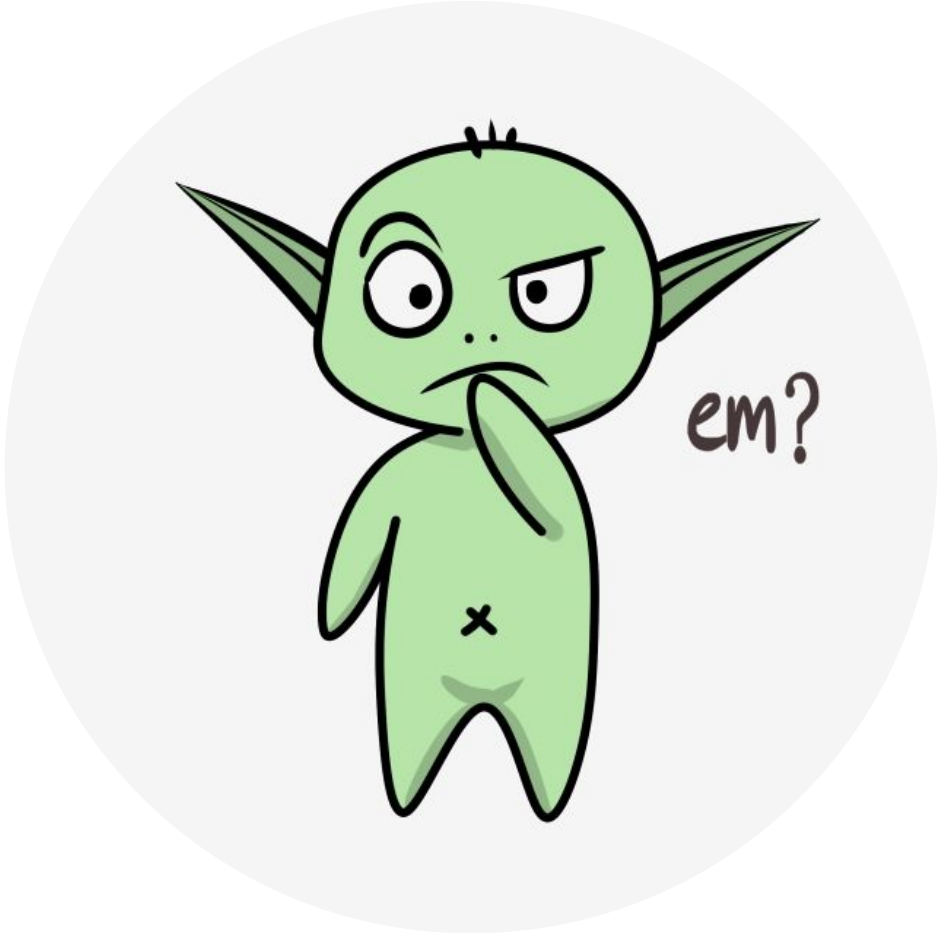
$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \leq \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Exercise – Textbook

Exercise 7.1 In Chapter 6 we noted that the Monte Carlo error can be written as the sum of TD errors (6.6) if the value estimates don't change from step to step. Show that the n-step error used in (7.2) can also be written as a sum of TD errors (again if the value estimates don't change) generalizing the earlier result. □



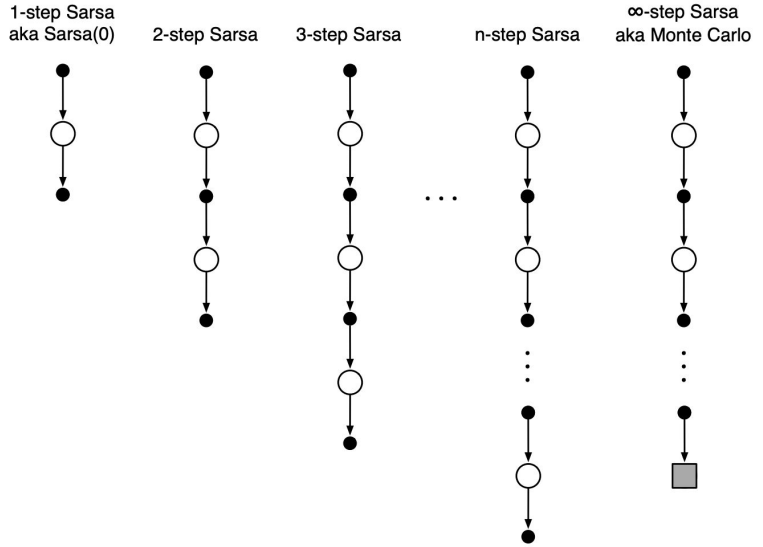
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Control

n-step Sarsa

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T - n$$

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



n-step Sarsa

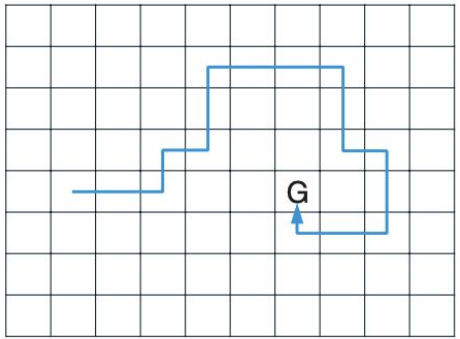
n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-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 $n+1$

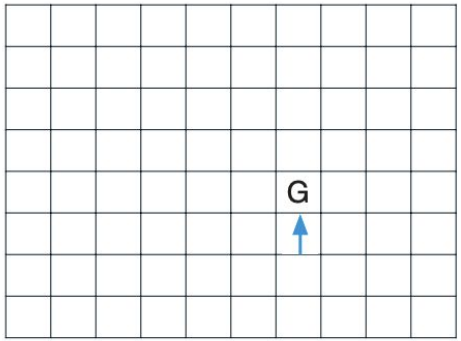
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ 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 π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

n-step Sarsa – Example

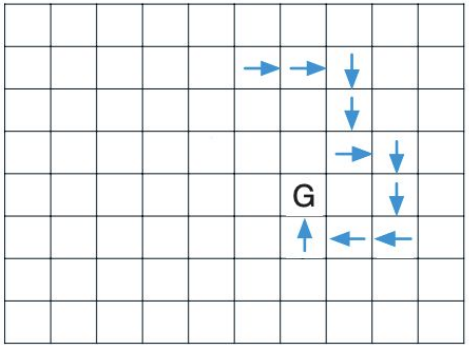
Path taken

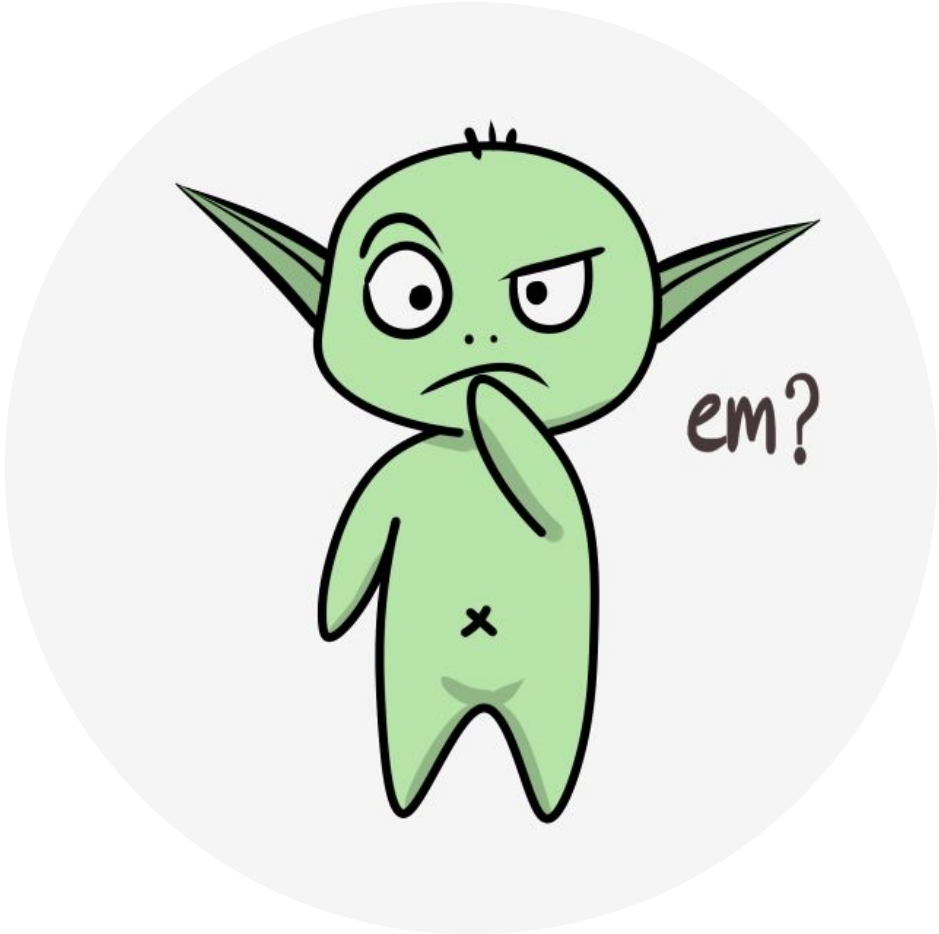


Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa





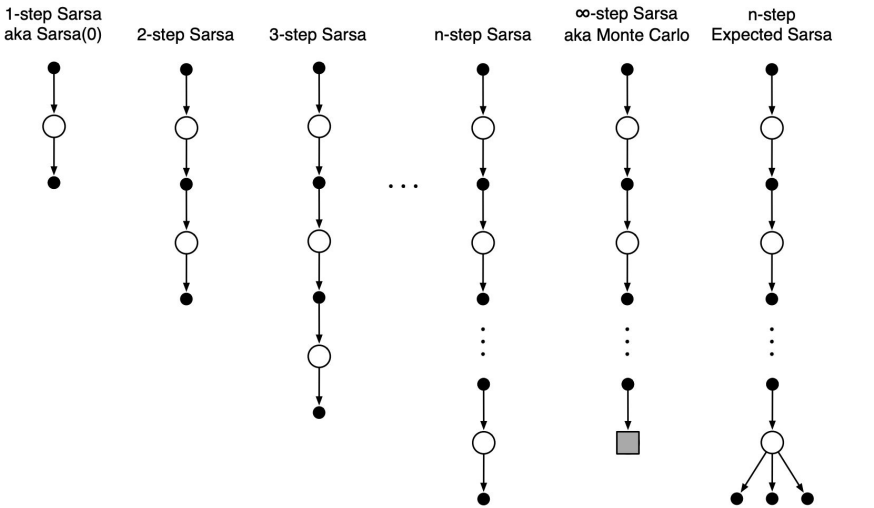
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

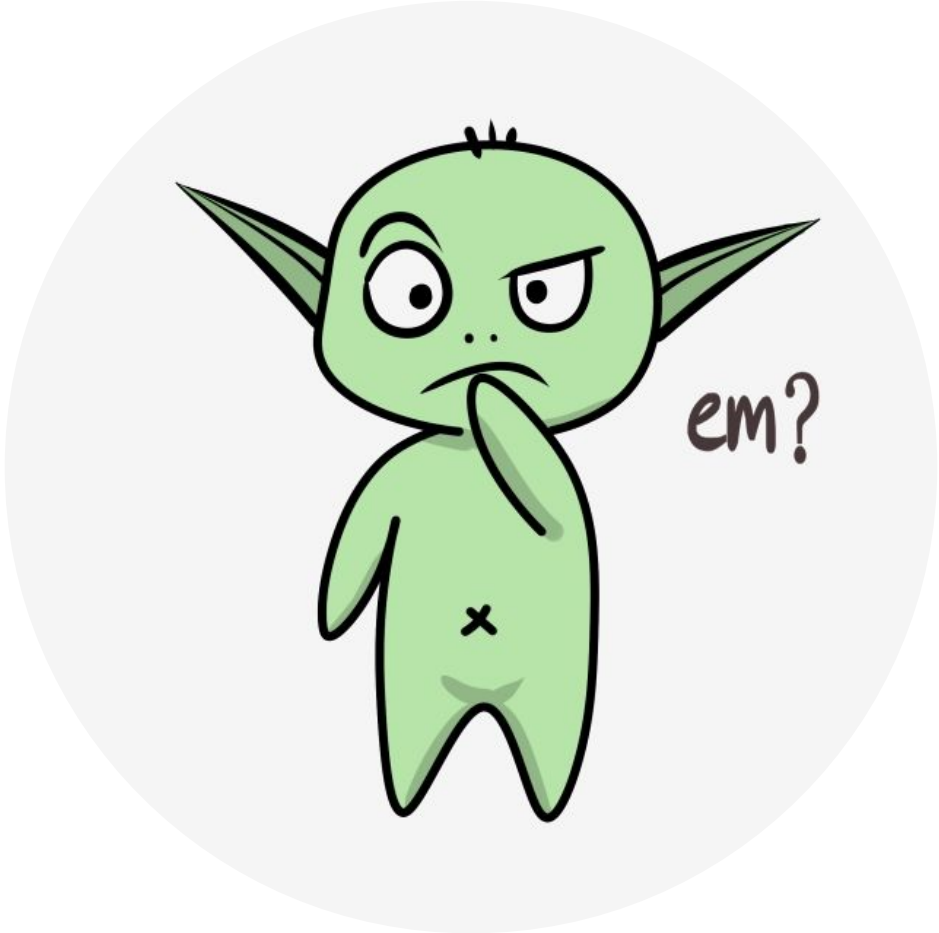
n-step Expected-Sarsa

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}), \qquad t+n < T$$

Expected approximate value of state s

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \qquad \text{for all } s \in \mathcal{S}.$$





https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

n-step Off-Policy Learning

- To use data from a behaviour policy, *b*, we need to consider the difference between the target policy, π, and *b* (*i.e.,* their relative probability of taking the actions that were taken).
- We need to compute the relative probability of the *n* actions.

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \qquad 0 \leq t < T$$

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

n-step Off-Policy Learning

- If an action would never be taken by π, we ignore the n-step return ($\rho = 0$).
- If by chance an action that π would take with much greater probability than b is taken, we need to over-weight that n-step return by a lot (very large ρ).
- Again, this can lead to really high variance (and/or really slow learning).

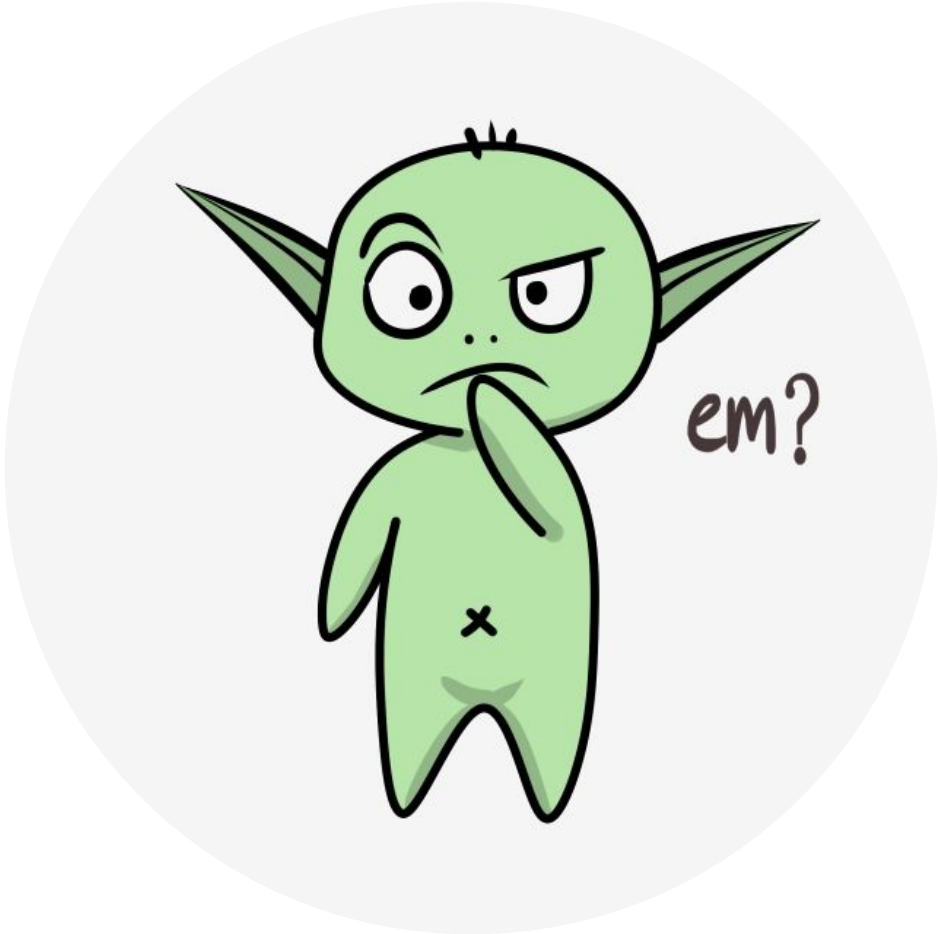
$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

n-step Off-Policy Learning

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

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, 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 π to be 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 $n+1$

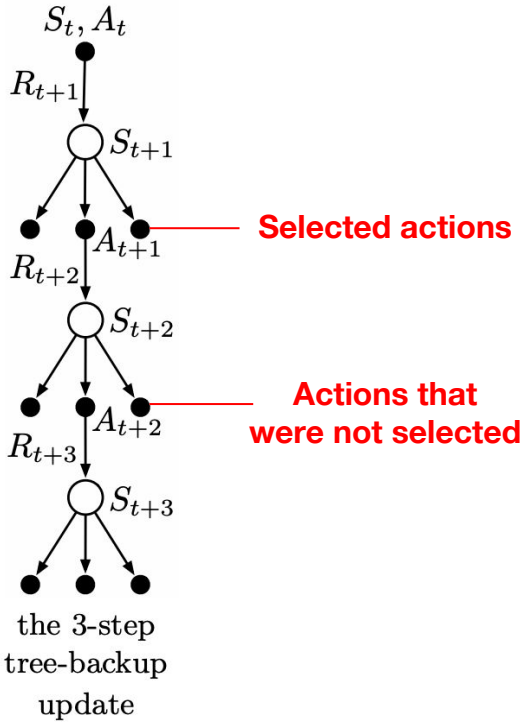
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  |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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

n-step Tree Backup

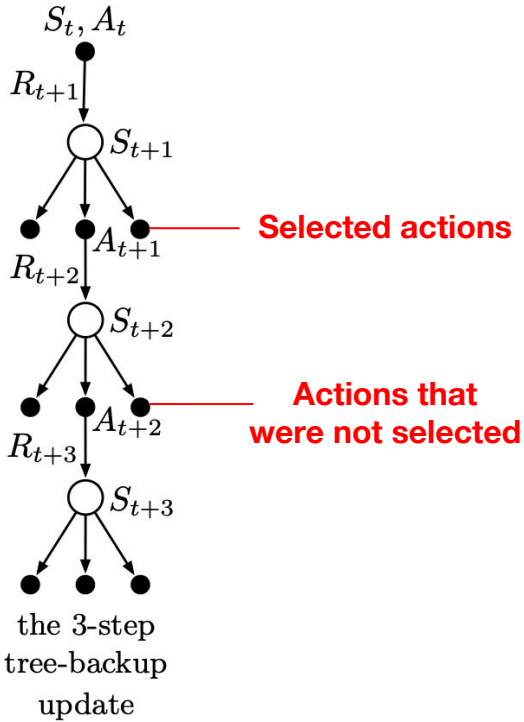
- An n–step off-policy learning algorithm without importance sampling.



the 3-step tree-backup update

n-step Tree Backup

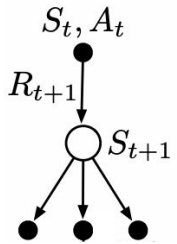
- An n–step off-policy learning algorithm without importance sampling.
- In Tree-Backup, we update the estimated value of the node at the top of the diagram toward a target combining the rewards along the way and the estimated values of the notes at the bottom **plus** the estimated values of the dangling action nodes hanging off the sides, at all levels.
- Each leaf node contributes to the target with a weight proportional to its probability of occurring under the target policy.



n-step Tree Backup

- One-step return (target) is the same as that of Expected Sarsa:

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a)$$



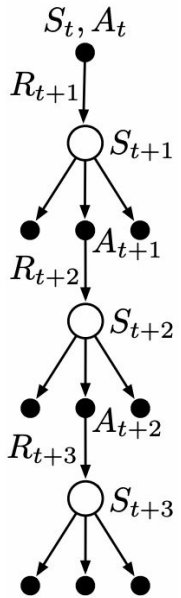
n-step Tree Backup

- One-step return (target) is the same as that of Expected Sarsa:

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a)$$

- The two-step tree-backup return is

$$\begin{aligned} G_{t:t+2} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ &\quad + \gamma \pi(A_{t+1}|S_{t+1}) \Big(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2}, \end{aligned}$$



the 3-step tree-backup update

n-step Tree Backup

- n-step return for Tree Backup:

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \; + \; \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}$$

n-step Tree Backup

- n-step return for Tree Backup:

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \; + \; \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}$$

- Update rule:

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

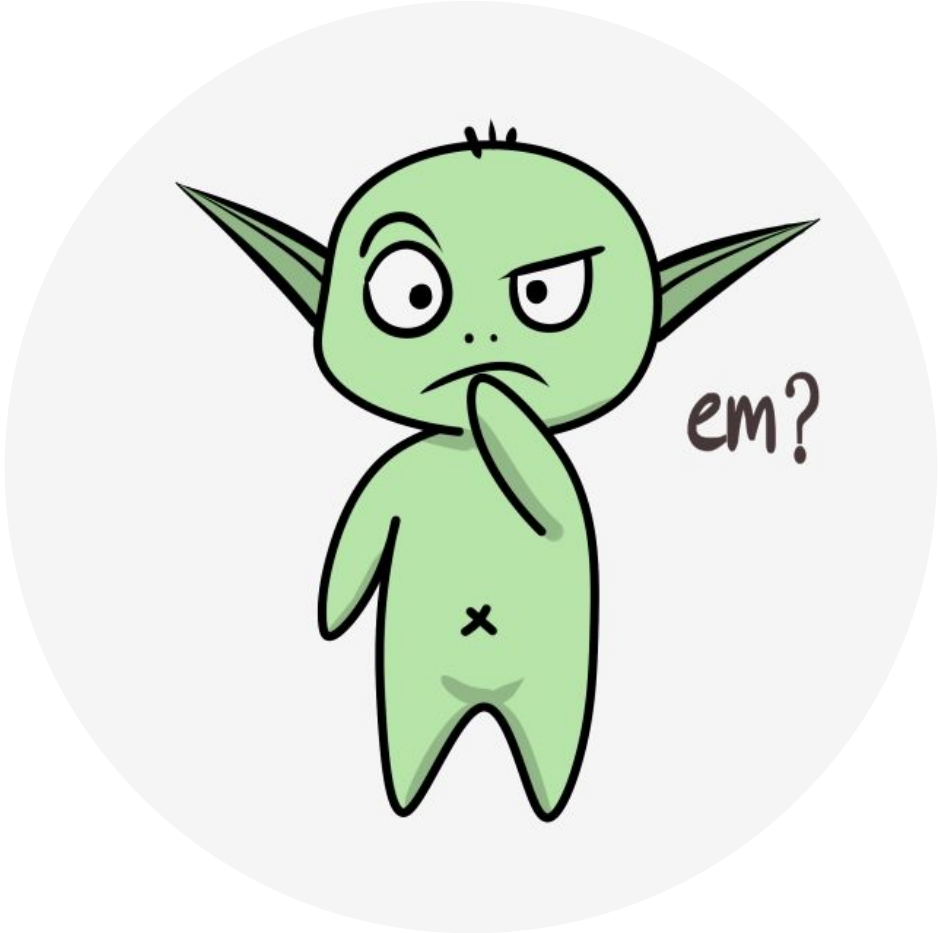
At each step along a trajectory, there are several possible choices of action according to the target policy. The one-step target combines the value estimates for these actions according to their probabilities of being taken under the target policy.

n-step Tree Backup

n-step Tree Backup for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be 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 can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$:
  | | Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
  | | If S_{t+1} is terminal:
  | | | $T \leftarrow t+1$
  | | else:
  | | | Choose an action A_{t+1} arbitrarily as a function of S_{t+1}; Store A_{t+1}
  | $\tau \leftarrow t+1-n$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | | If $t+1 \geq T$:
  | | | $G \leftarrow R_T$
  | | else
  | | | $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$
  | | Loop for $k = \min(t, T-1)$ down through $\tau + 1$:
  | | | $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
  | | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau) \right]$
  | | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht