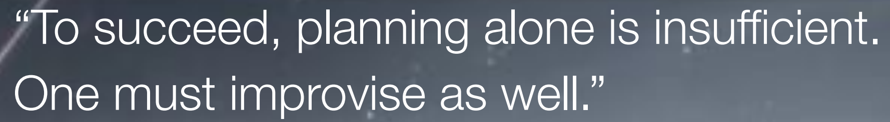


"To succeed, planning alone is insufficient.
One must improvise as well."

Isaac Asimov, *Foundation*

CMPUT 655
Introduction to RL

Marlos C. Machado

Class 6/12

Plan

- Chapter 7: n-Step Bootstrapping
- General Value Functions
- Chapter 8: Planning and Learning with Tabular Methods

Reminder I

You **should be enrolled in the private session** we created in Coursera for CMPUT 655.

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

- On the course workload.
- The project proposal is due next week. That's worth 15% of your marks.
- Next week there are <u>three</u> weeks of Coursera for you to go over:
    - Week 2 of Prediction and Control with FA: On-Policy Prediction with Approximation
    - Week 3 of Prediction and Control with FA: Constructing Features for Prediction
    - Week 4 of Prediction and Control with FA: Control with Approximation

Please, interrupt me at any time!

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

Last Class: TD Learning and Monte-Carlo Methods

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)**):

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

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

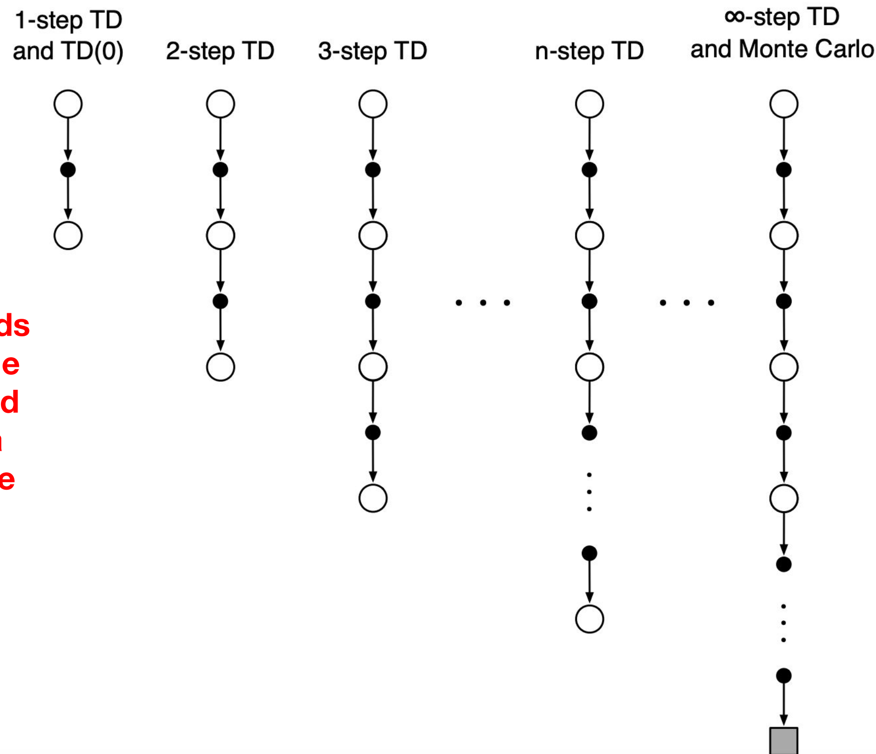
- MC methods perform an update for each state based on the entire sequence of observed rewards (until the end of the episode).
- One-step TD methods have an update that is based on just the one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards.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

Chapter 7

n-step Bootstrapping

Prediction

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

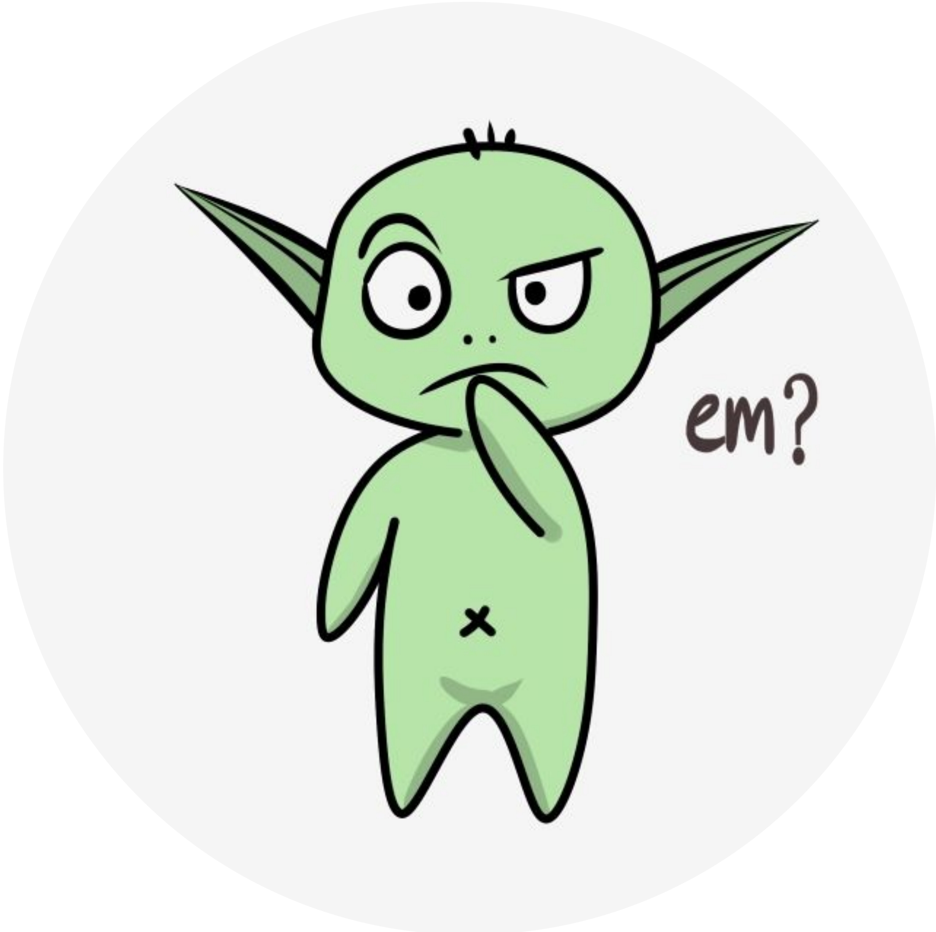
- Complete return: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
- One-step return: $G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

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 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

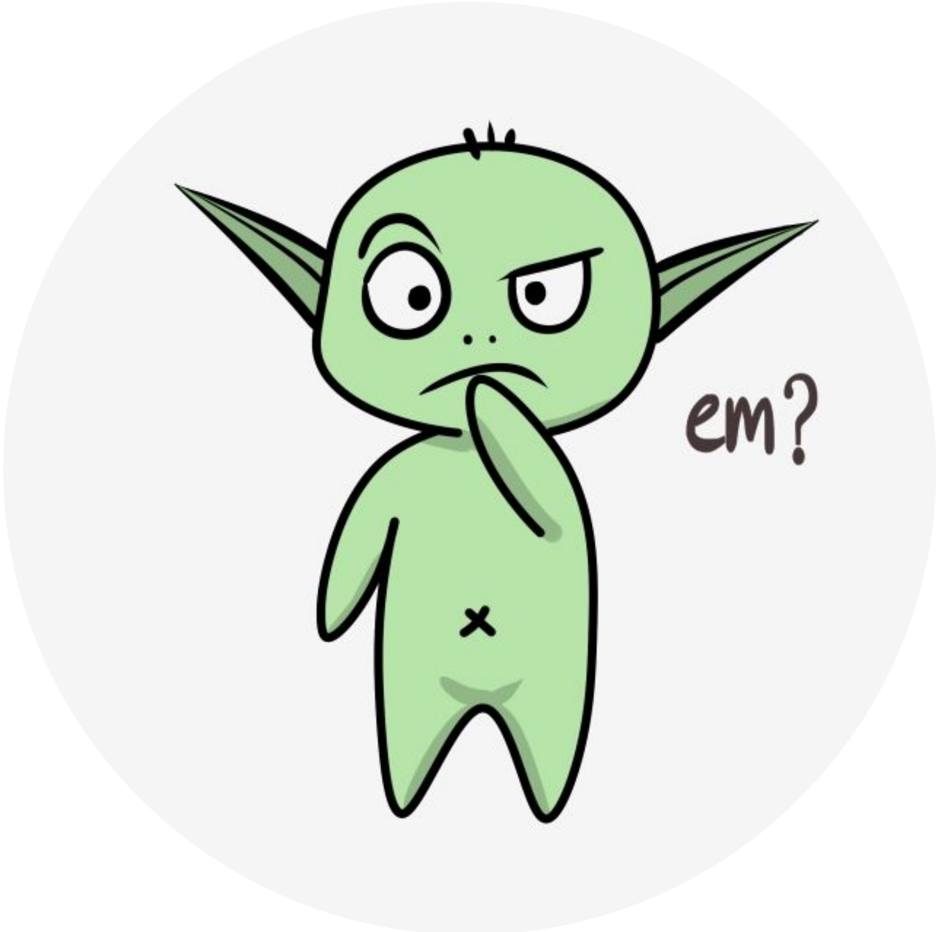
$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \leq t < T$$

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 $n+1$

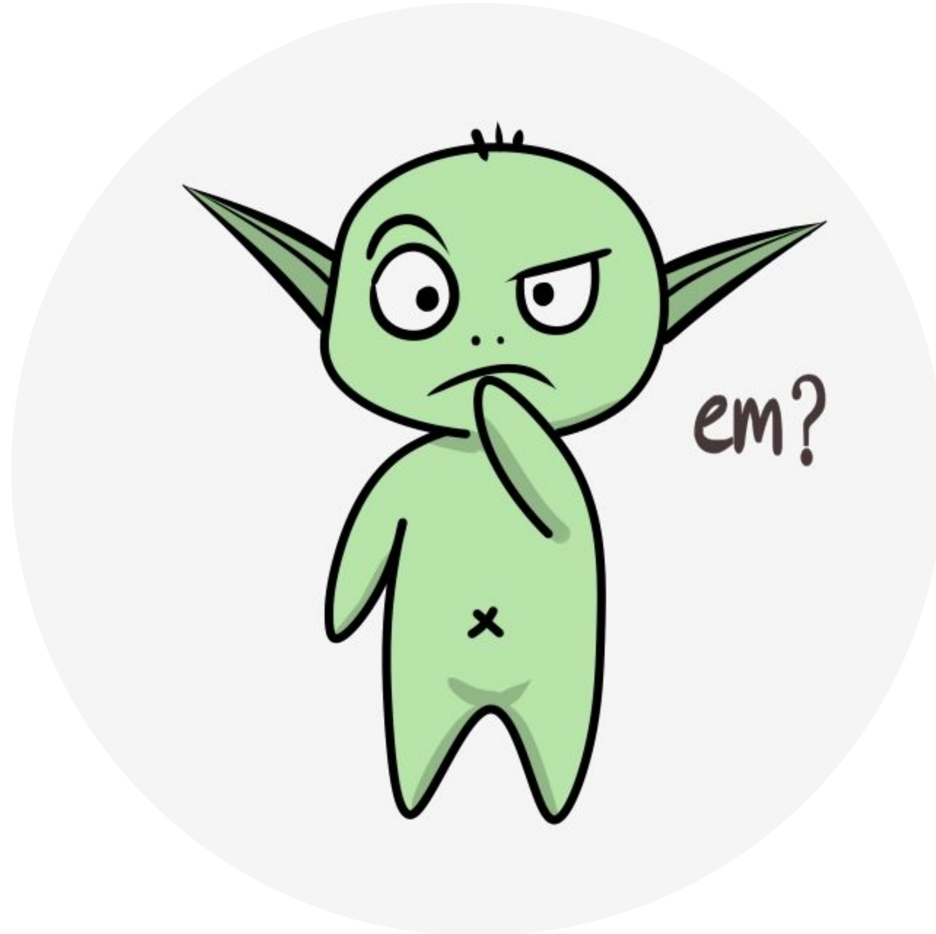
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \dots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (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. □

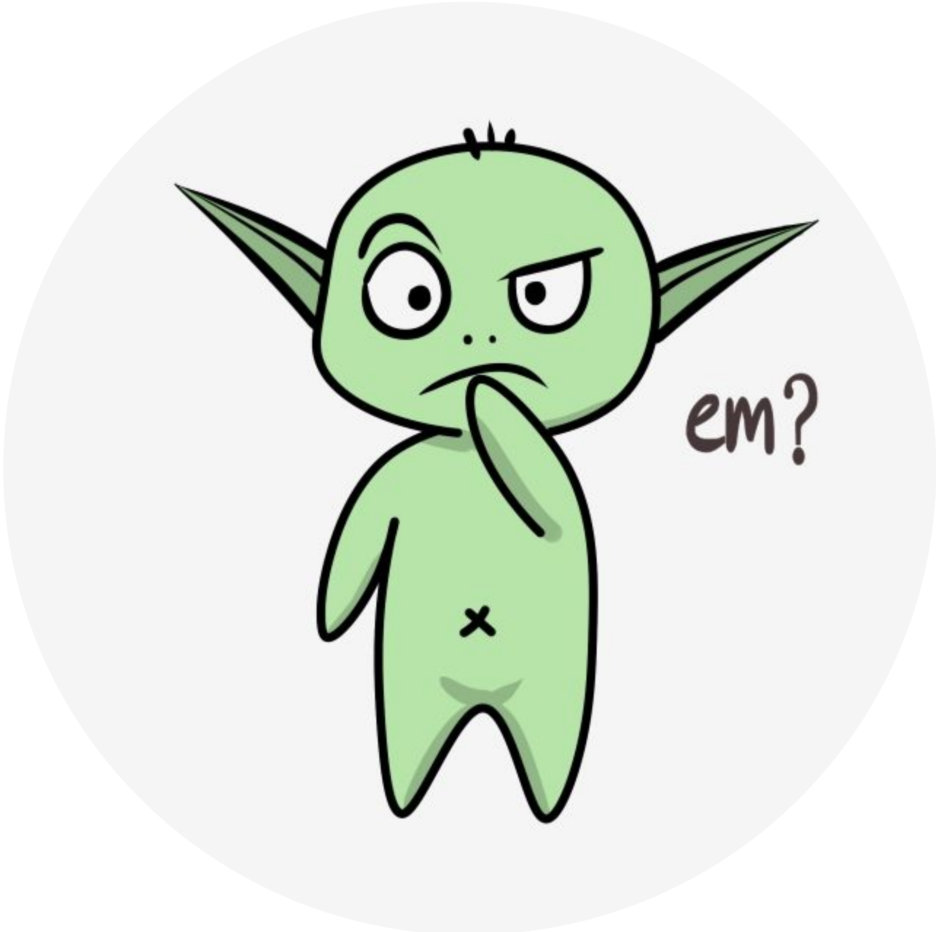
Answer: We adopt the convention that $G_{t:t} \doteq V(S_t)$. The recursive relationship among successive returns,

$$G_{t:t+n} = R_{t+1} + \gamma G_{t+1:t+n},$$

then holds even for $n = 1$. If $t + n > T$, then all of our n-step returns (Gs) are in fact full returns, and the earlier result applies. Thus, here we can assume $t + n \leq T$. Then we have

$$\begin{aligned}
G_{t:t+n} - V(S_t) &= R_{t+1} + \gamma G_{t+1:t+n} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \qquad \text{(from (3.9))} \\
&= \delta_t + \gamma\big(G_{t+1:t+n} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2:t+n} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{n-1}\delta_{t+n-1} + \gamma^n\big(G_{t+n:t+n} - V(S_{t+n})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{n-1}\delta_{t+n-1} + \gamma^n\big(V(S_{t+n}) - V(S_{t+n})\big) \\
&= \sum_{k=t}^{t+n-1} \gamma^{k-t}\delta_k,
\end{aligned}$$

Q.E.D. $\square$



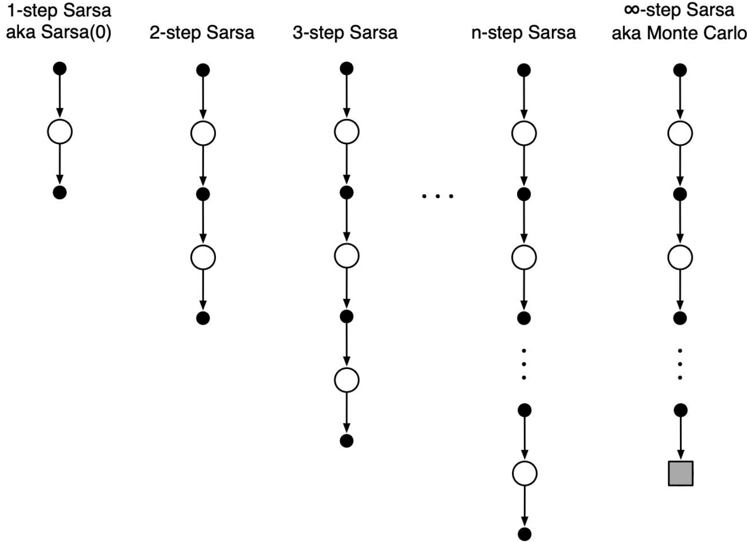
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Control

n-step Sarsa

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

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



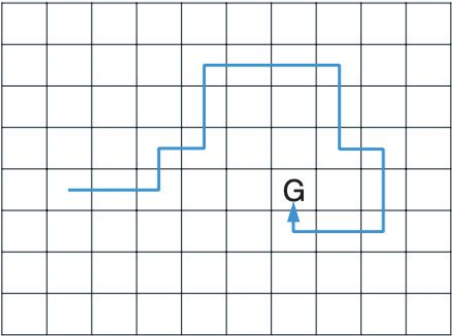
n-step Sarsa

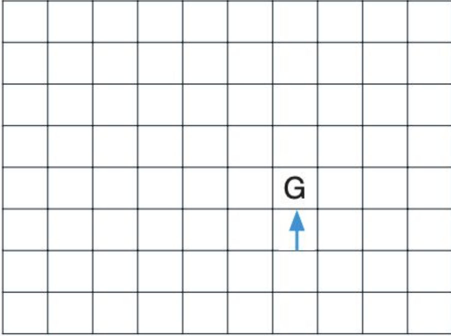
n-step Sarsa for estimating $Q \approx q_*$ or q_π

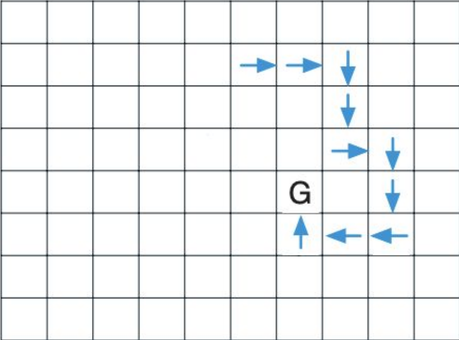
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and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

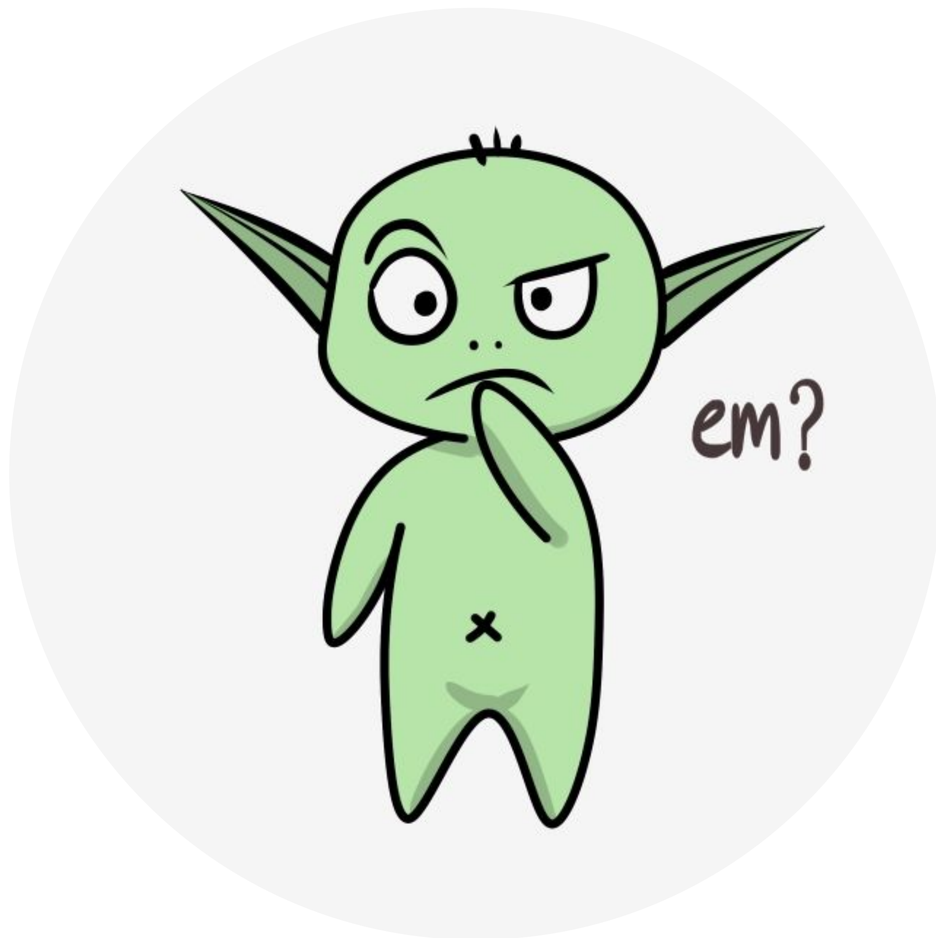
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | 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









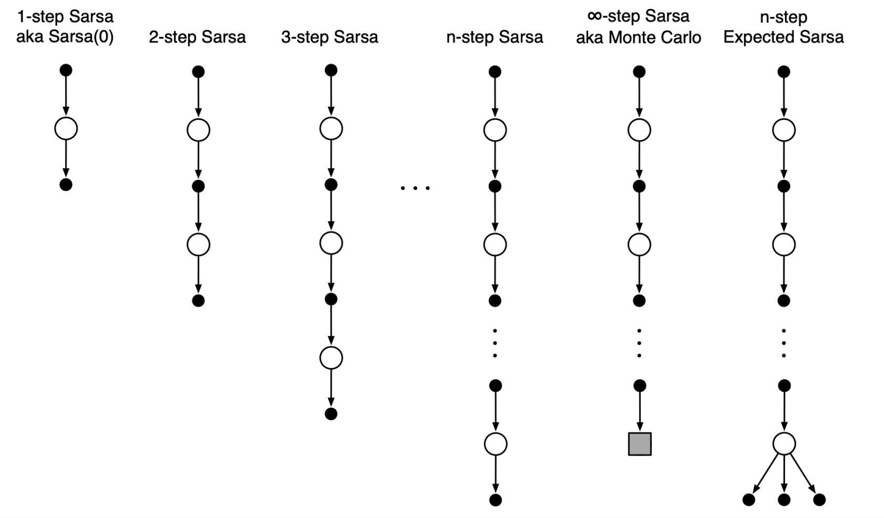
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

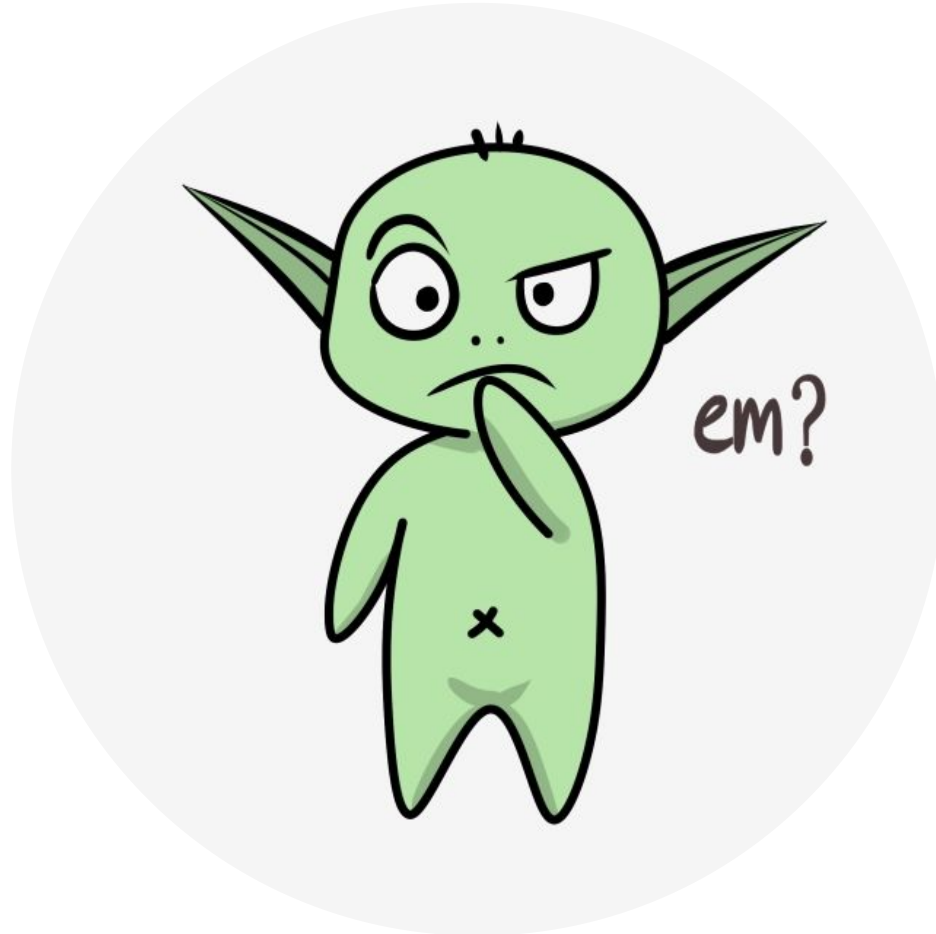
n-step Expected-Sarsa

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

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).

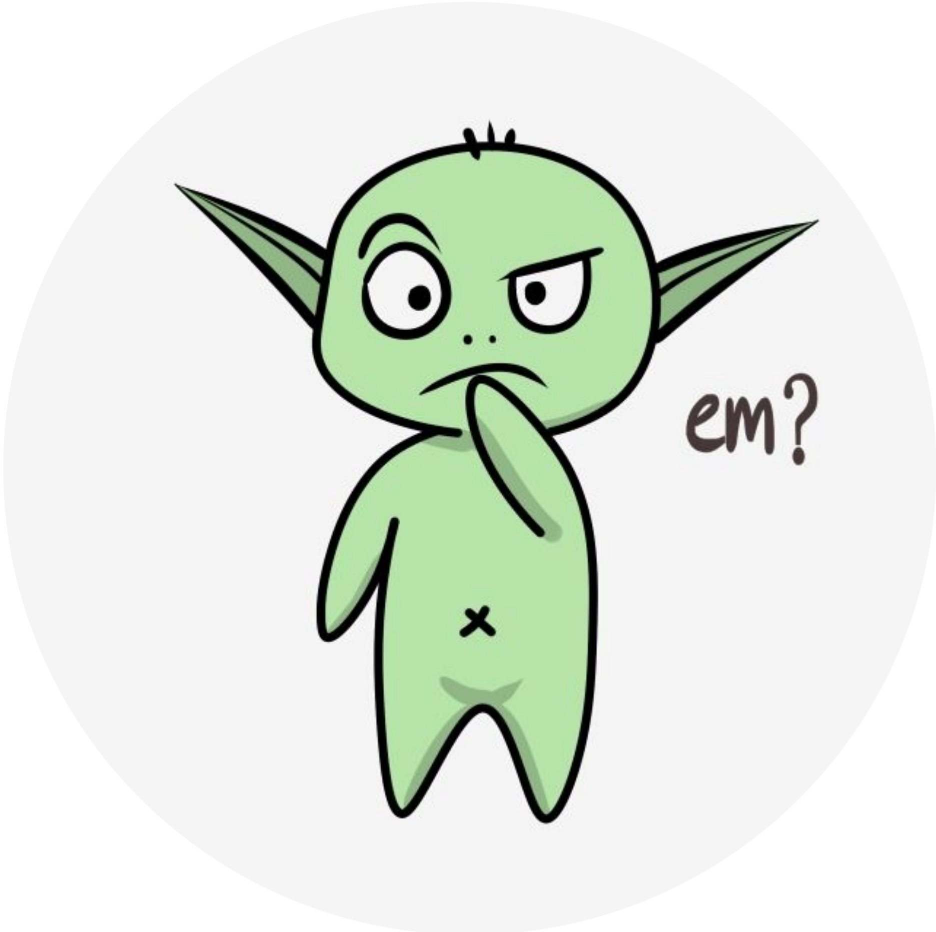
$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

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 $n+1$

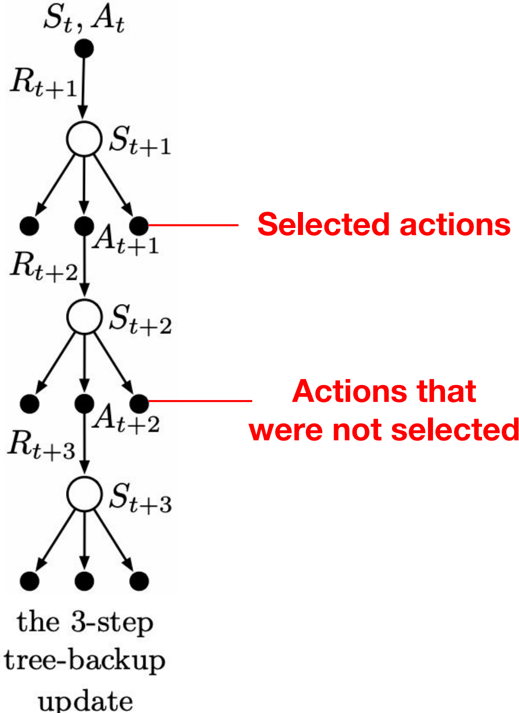
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | 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

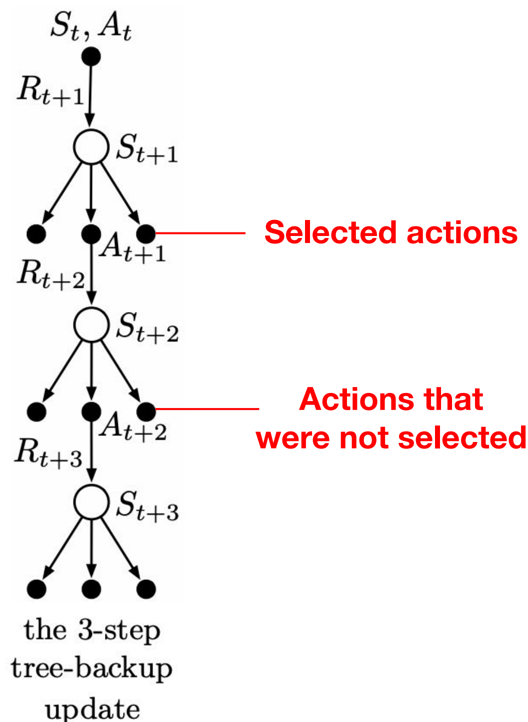
n-step Tree Backup

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



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.

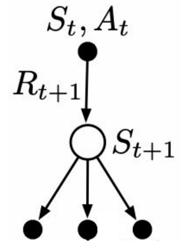


the 3-step tree-backup update

n-step Tree Backup

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

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



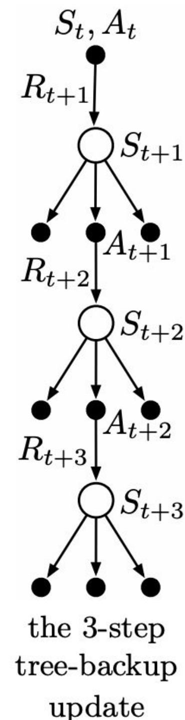
n-step Tree Backup

- 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

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



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:

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

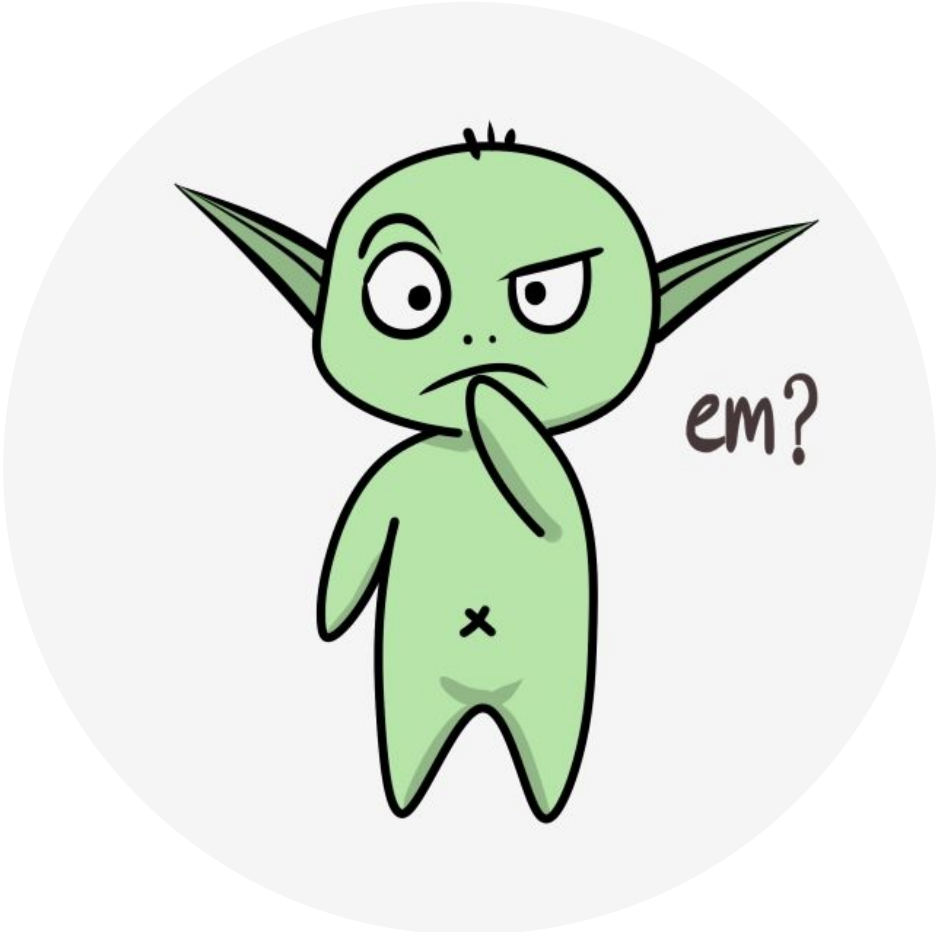
At each step along a trajectory, there are several possible choices of action according to the target policy. The one-step target combines the value estimates for these actions according to their probabilities of being taken under the target policy.

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

Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction

Richard S. Sutton, Joseph Modayil, Michael Delp
Thomas Degris, Patrick M. Pilarski, Adam White
Reinforcement Learning and Artificial Intelligence Laboratory
Department of Computing Science, University of Alberta, Canada

Doina Precup
School of Computer Science, McGill University, Montreal, Canada

General Value Functions (GVFs)

How should one learn, represent, and use knowledge of the world in a general sense?

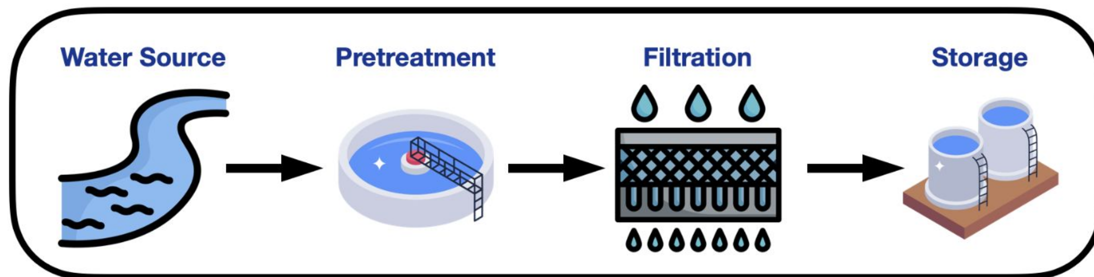
- GVFs are based on the idea that knowledge should be represented as a large number of approximate value functions.
- Value functions have an explicit semantics and a clear notion of truth grounded in sensorimotor interaction. We can define accuracy of knowledge.
- The idea of GVFs is that the value-function approach to grounding semantics can be extended beyond reward to a theory of all world knowledge.
- The theory of value functions provides a semantics for predictive knowledge.

General Value Functions (GVFs)

$$v_{\pi,\beta}^{\gamma,c,z}(s) \doteq \mathbb{E}_{\pi,\beta}\left[\sum_{j=1}^{K} \gamma^{j-1} c(S_j) + \gamma^{K-1} z(S_K) \middle| S_0 = s\right]$$

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

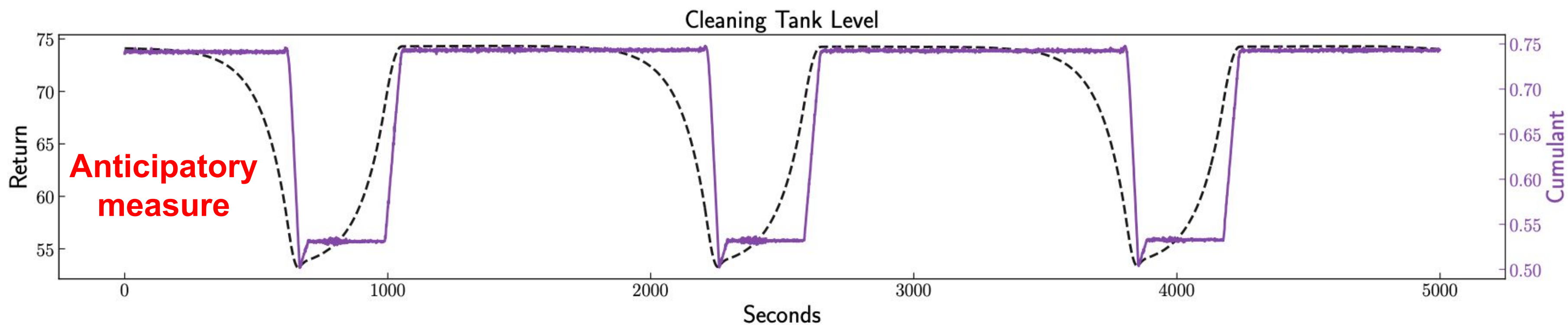
GVFs – Examples



M. K. Janjua, H. Shah, M. White, E. Miahi, M. C. Machado, and A. White.
GVFs in the Real World: Making Predictions Online for Water Treatment.
Machine Learning (2023).

GVFs – Examples

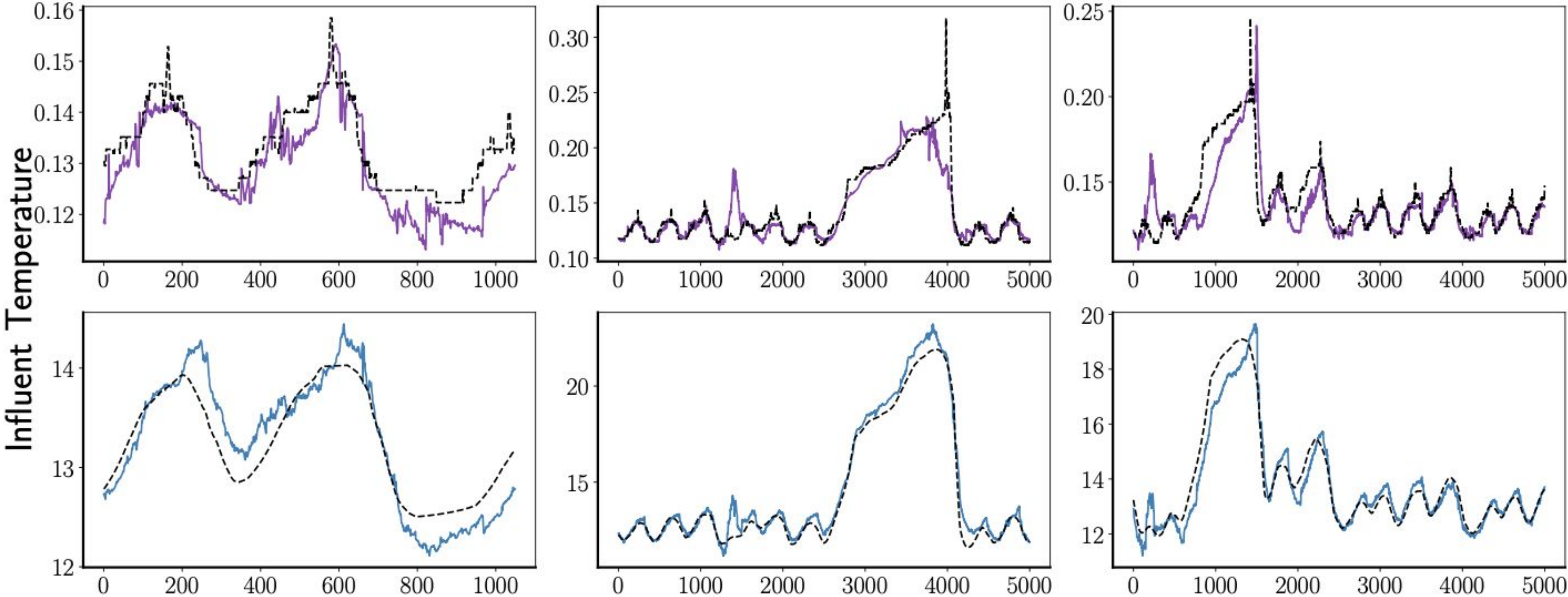
Idealized prediction:



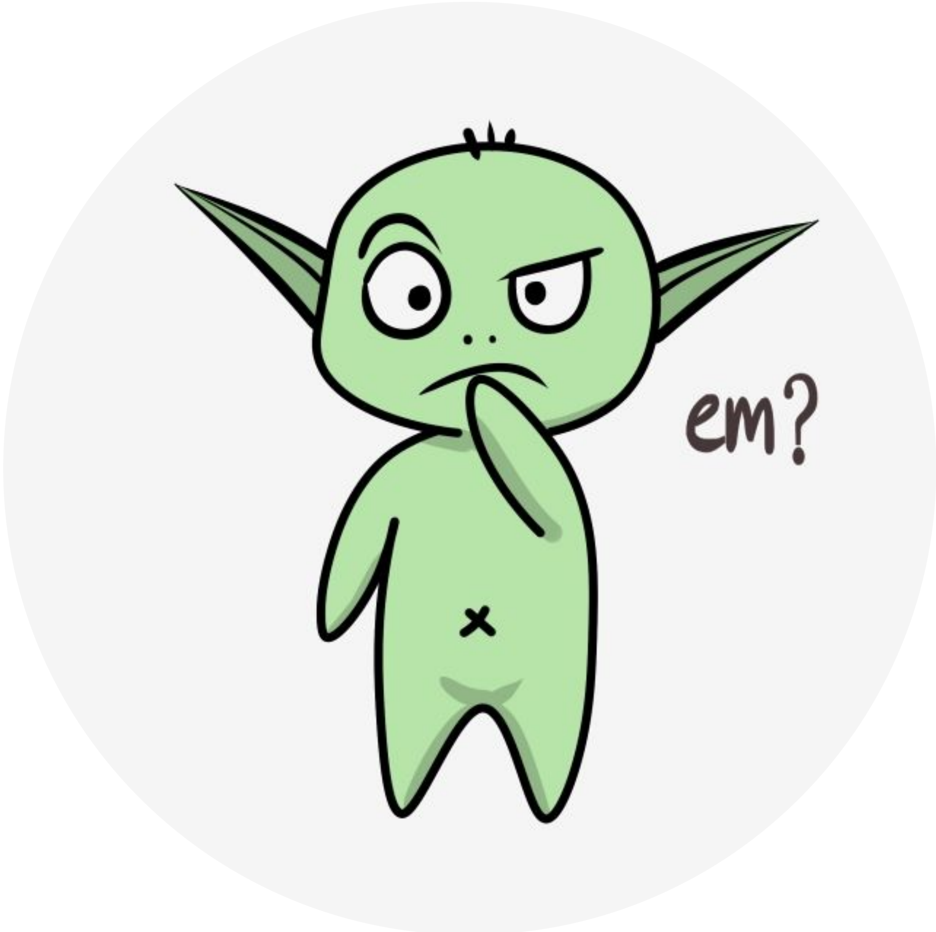
[Janjua et al., 2023]

GVFs – Examples

Actual prediction:



[Janjua et al., 2023]



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

Chapter 8

Planning and Learning with Tabular Methods

Motivation

- Can we be more sample efficient?
- Do we really need to learn only from the transitions we observe?
- Can we learn how the world works and then imagine what we would do in different situations?

Model-based Reinforcement Learning

Models and Planning

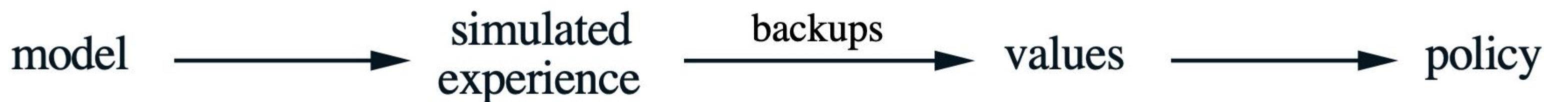
- Model: Anything that an agent can use to predict how the environment will respond to its actions. Given a state and an action, a model produces a prediction of the resultant next state and next reward.
- There are different types of models, such as *sample models*, *expectation models*, and *distribution models*.
- Models can be used to mimic or *simulate* experience.
- Planning is any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment:

Planning

- State-space planning: a search through the state space for an optimal policy or an optimal path to a goal. Actions cause transitions from state to state, and value functions are computed over states. ← **Our focus!**
- Plan-space planning: search through the space of plans. Operators transform one plan into another, and value functions, if any, are defined over the space of plans. Plan-space planning includes evolutionary methods and "partial-order planning".

A Common Structure for State-Space Planning

1. All state-space planning methods involve computing value functions as a key intermediate step toward improving the policy,
2. they compute value functions by updates or backup operations applied to simulated experience.



Arguably, other state-space planning methods differ only in the kinds of updates they do, the order in which they do them, and in how long the backed-up information is retained.

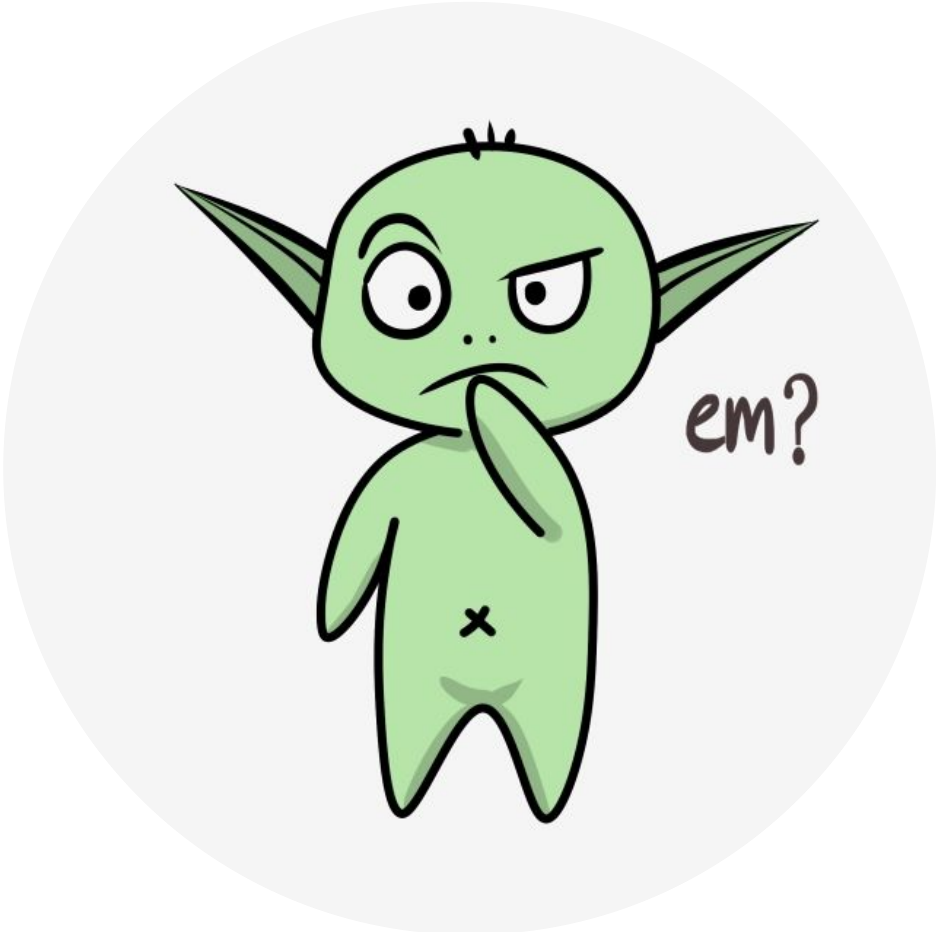
Planning vs. Learning

- The heart of both learning and planning methods is the estimation of value functions by backing-up update operations.
  - Planning uses simulated experience generated by a model.
  - Learning methods use real experience generated by the environment.
- Learning methods require only experience as input, and in many cases they can be applied to simulated experience just as well as to real experience.

Random-sample one-step tabular Q-planning

Loop forever:

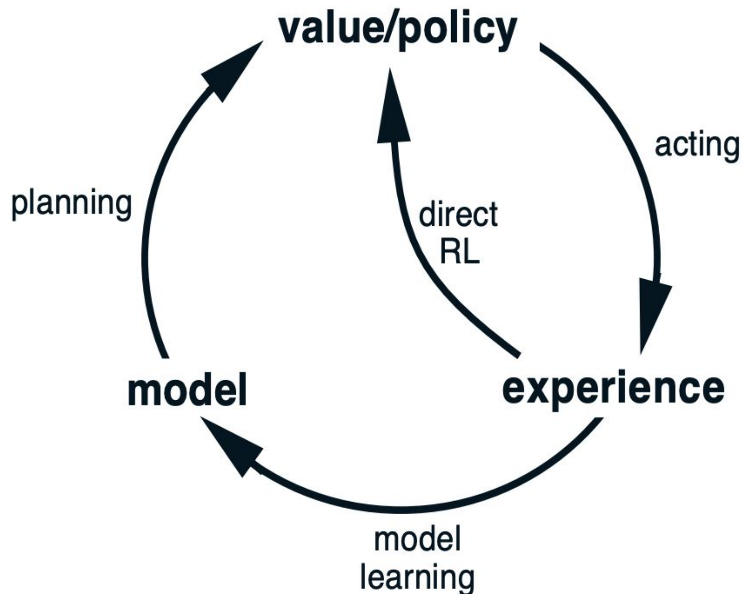
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$



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

Dyna: Integrated Planning, Acting, and Learning

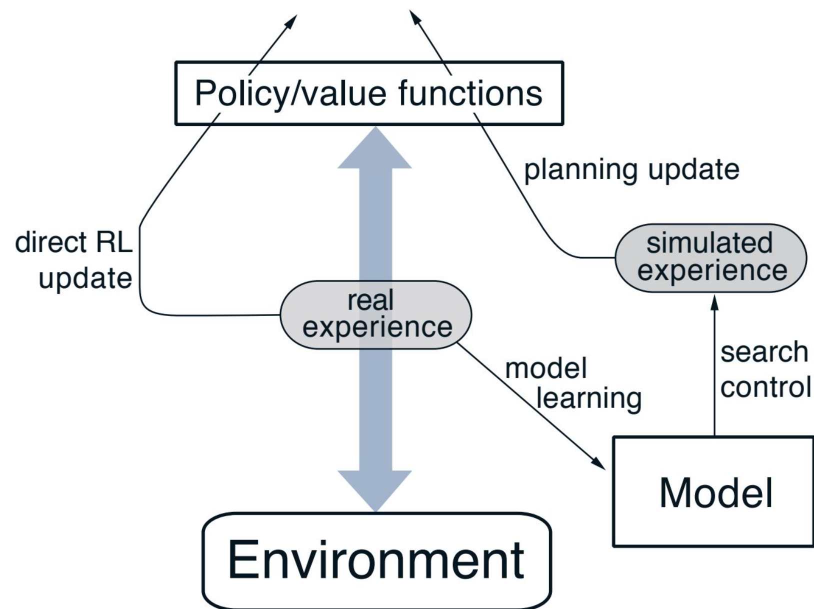
- Online planning, in small, incremental steps.



- Indirect methods often make fuller use of a limited amount of experience and thus achieve a better policy with fewer environmental interactions.
- Direct methods are much simpler and are not affected by biases in the design of the model.

Dyna-Q

- Dyna-Q includes all of the processes shown in the diagram—planning, acting, model-learning, and direct RL—all occurring continually (and *simultaneously*).
- Planning method: random-sample one-step tabular Q-planning.
- Direct RL method: one-step tabular Q-learning.
- Model-learning method: table-based and assumes the environment is deterministic.

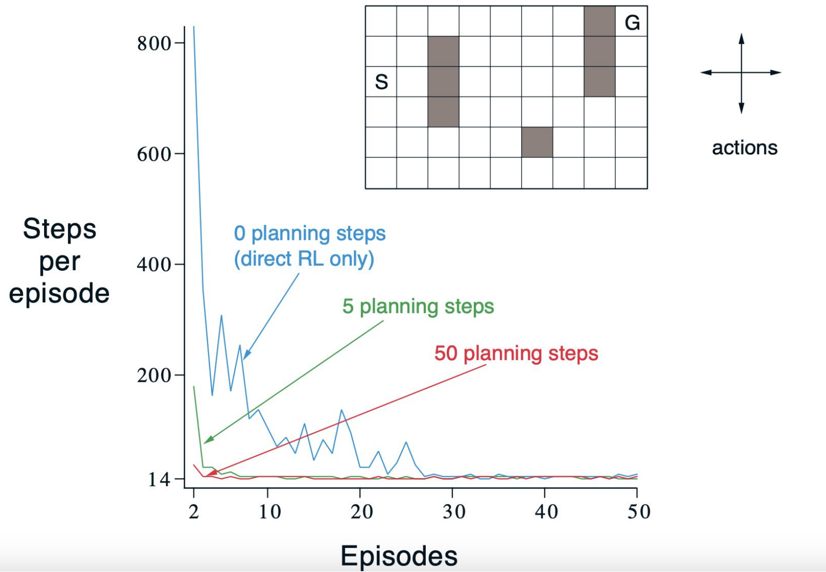


Dyna-Q

Tabular Dyna-Q

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

Example 8.1 – Dyna Maze



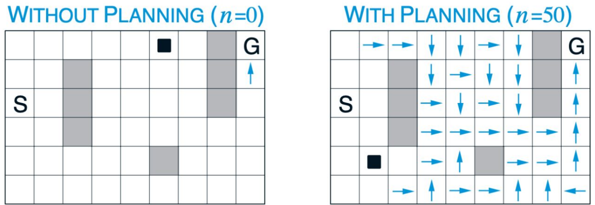
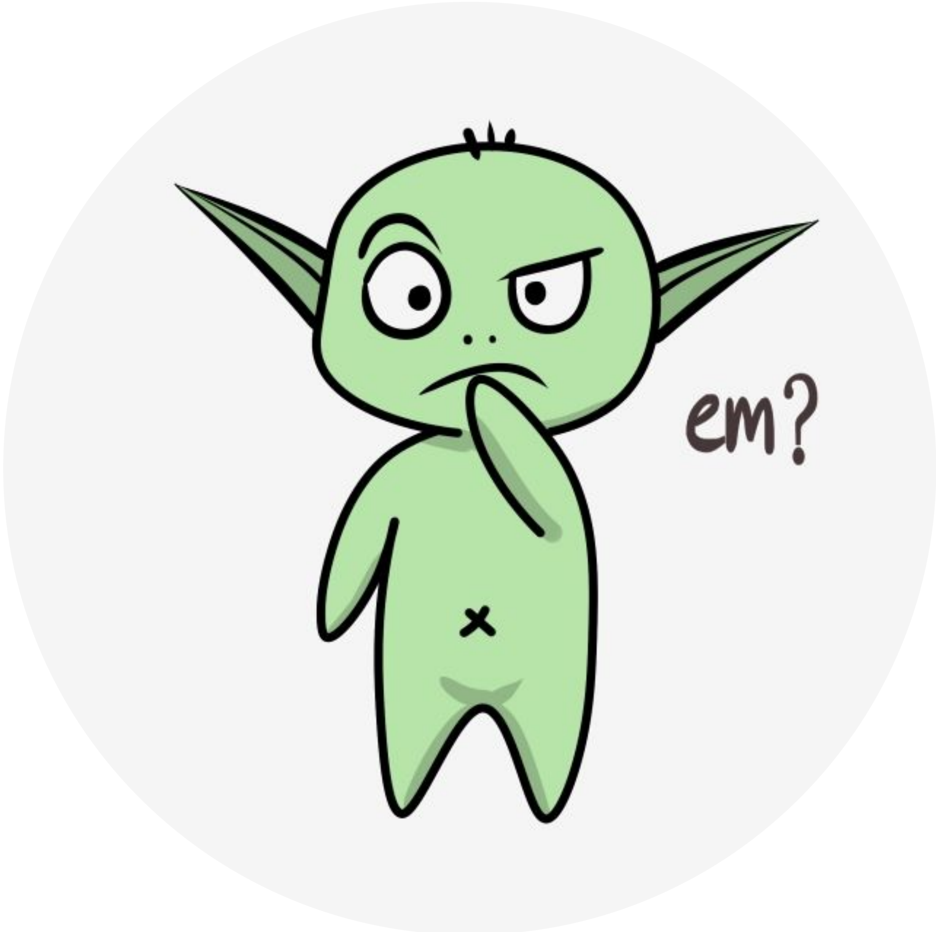


Figure 8.3: Policies found by planning and nonplanning Dyna-Q agents halfway through the second episode. The arrows indicate the greedy action in each state; if no arrow is shown for a state, then all of its action values were equal. The black square indicates the location of the agent. ■

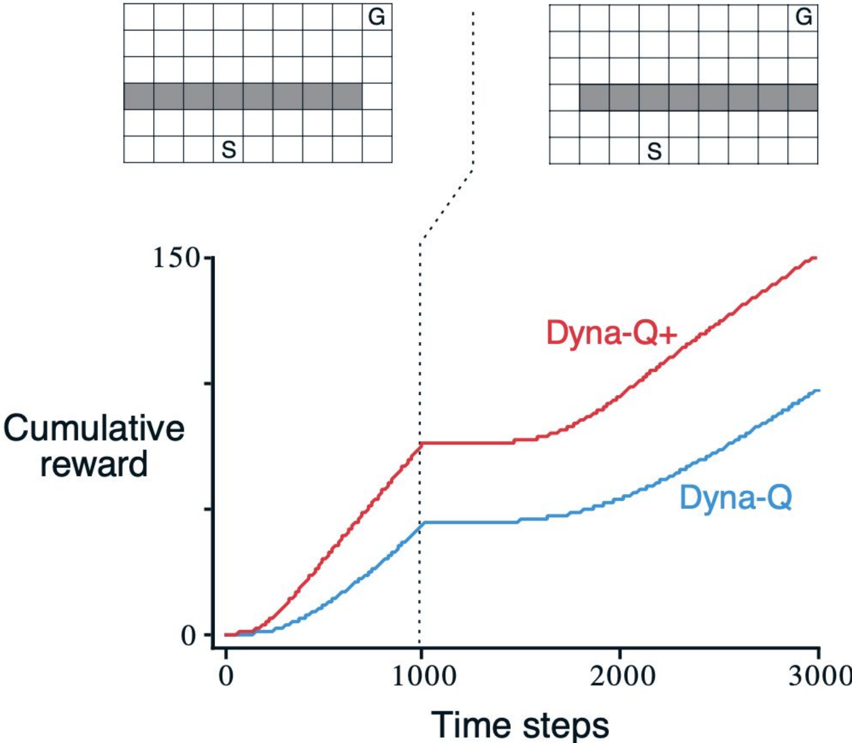


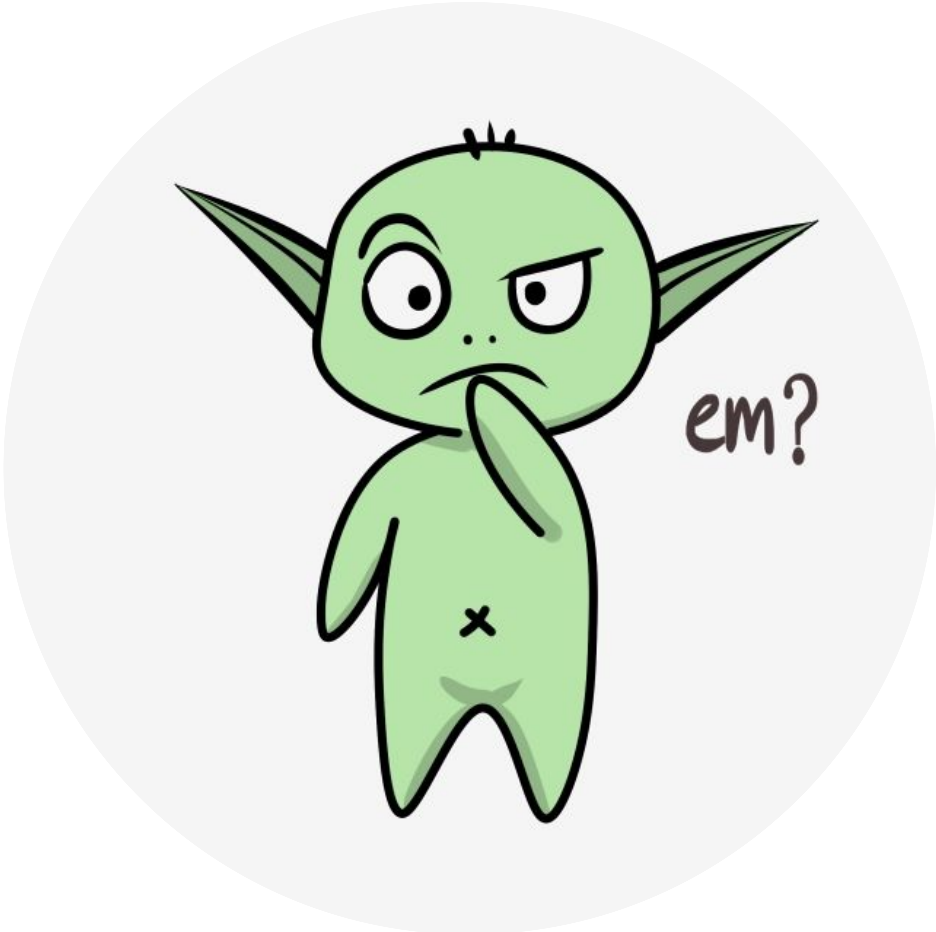
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

When the Model Is Wrong

- A model can be wrong for all sorts of reasons (e.g., stochastic environment, function approximation, non-stationarity in the environment).
- An incorrect model often leads to suboptimal policies.
- One needs to constantly explore to refine the learned model.
  - Exploration: take actions that improve the model.
  - Exploitation: behaving in the optimal way given the current model.
- Dyna-Q+: Provides "bonus rewards" for long-untried actions.
  Specifically, consider the reward $r + \kappa\sqrt{\tau}$, where τ is the number of time steps since that transition was tried for the last time.

Dyna-Q+ Sometimes Works ¯_(ツ)_/¯





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

Prioritized Sweeping

- How should we select transitions to simulate for the planning update?
- Working backward from goal states seem like a good idea, but it is dependent on the idea of a “goal state”.
    - More generally, we want to work back from any state whose value has changed.
- *Prioritized sweeping* is the idea of prioritizing updates according to their urgency.
    - When the effect of a change is greater than a threshold, the state in which that change happened is added to a priority queue.

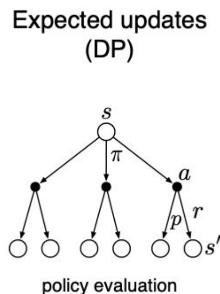
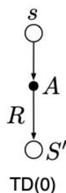
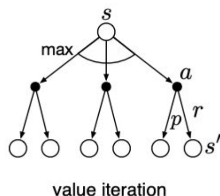
Prioritized Sweeping

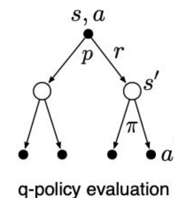
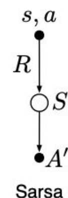
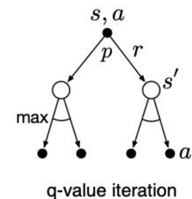
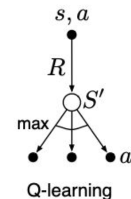
Prioritized sweeping for a deterministic environment

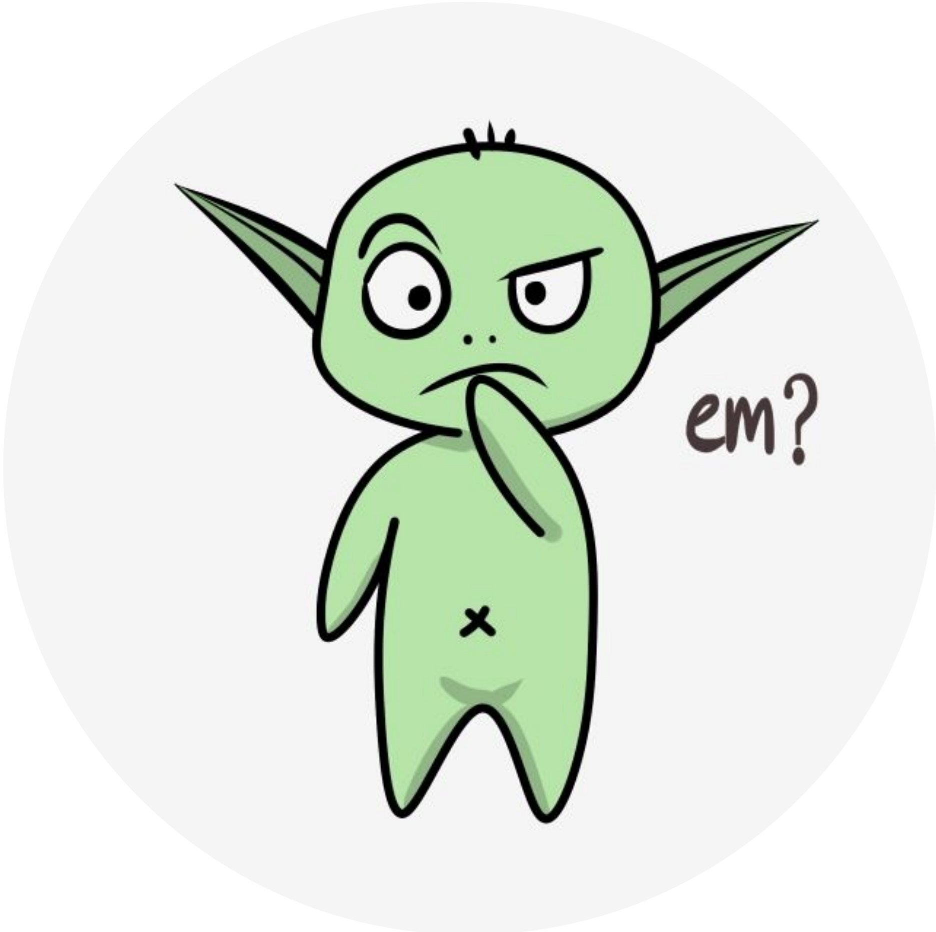
Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S, Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Model(S, A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S',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

Expected vs. Sample Updates

- There are three dimensions in the updates one can do:
  - Should we use state values or action values?
  - Should we estimate the value for the optimal policy or for an arbitrary given policy?
  - Should we use expected or sample updates?

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$v_\pi(s)$		
$v_*(s)$		

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$q_\pi(s, a)$		
$q_*(s, a)$		



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

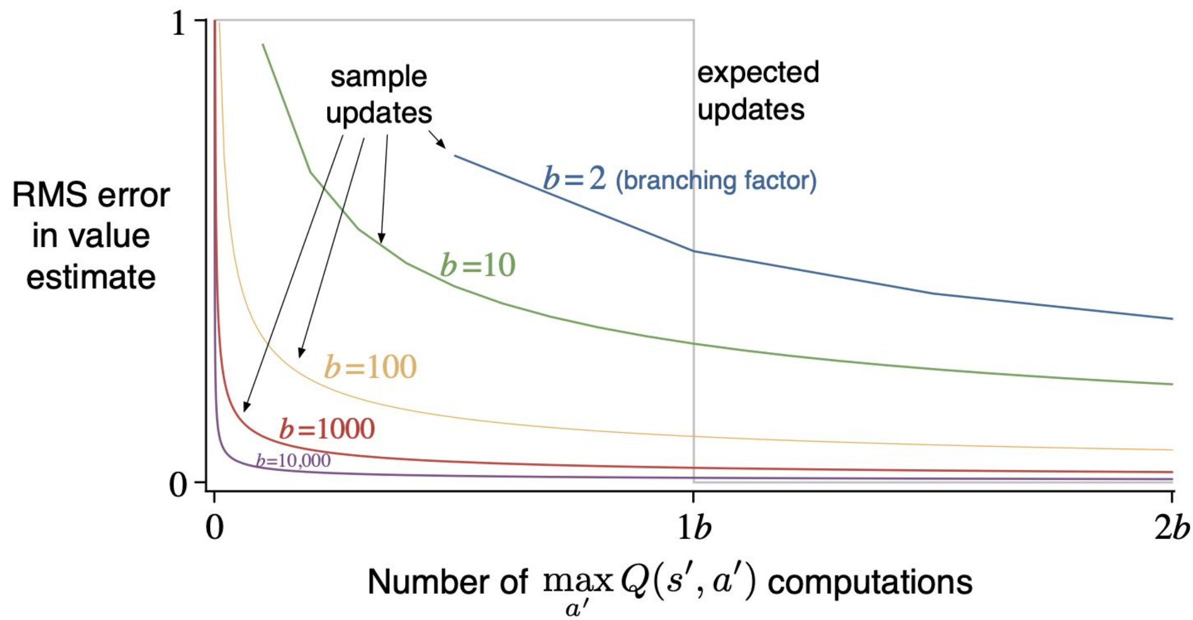
Expected vs. Sample Updates

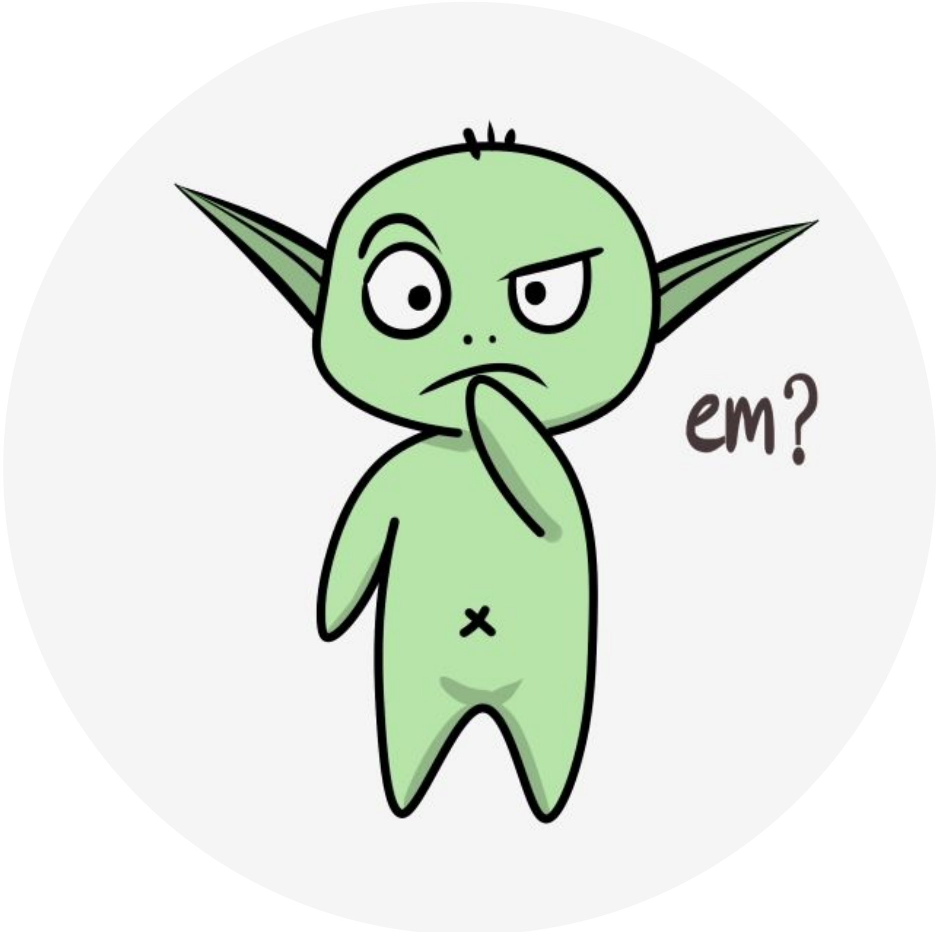
- If possible, are expected updates always preferable?
  - They yield a better estimate because they are uncorrupted by sampling error, but they also require more computation, and computation is often the limiting resource in planning.

$$Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r|s,a)\Big[r + \gamma \max_{a'} Q(s',a')\Big] \qquad Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R + \gamma \max_{a'} Q(S',a') - Q(s,a)\Big]$$

- Do we have enough time to do an expected update?
- Is it better to have a few sample updates at many state–action pairs or to have expected updates at a few pairs?

Often, the error falls dramatically with a fraction of b updates



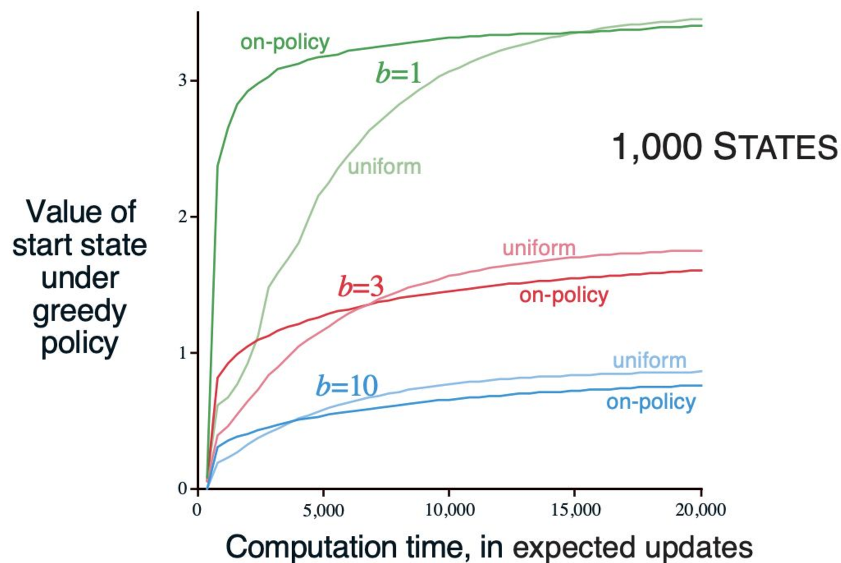


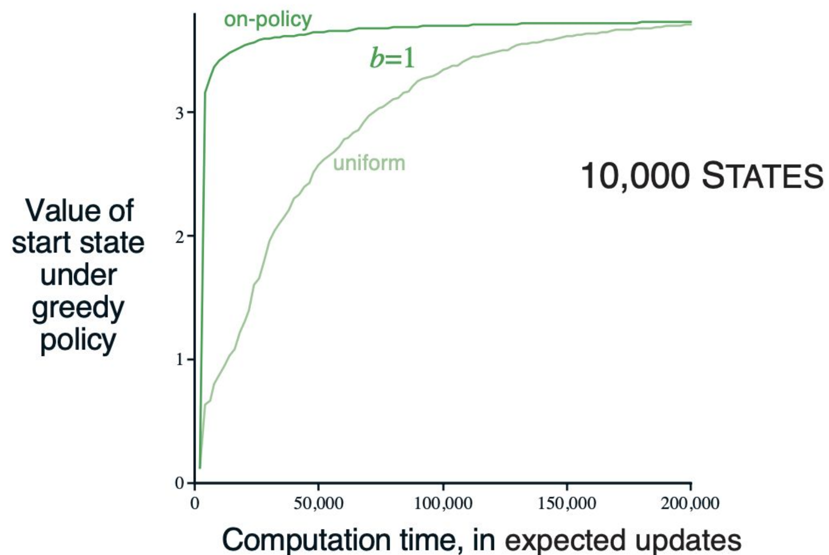
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Trajectory Sampling

- The classical approach, from dynamic programming, is to perform sweeps through the entire state space, updating each state once per sweep.
- However, in many tasks, most states are irrelevant under good policies.
- What if we sampled states from the state or state–action space according to some distribution?
- *Trajectory sampling* is the idea of sampling states from the on-policy distribution.

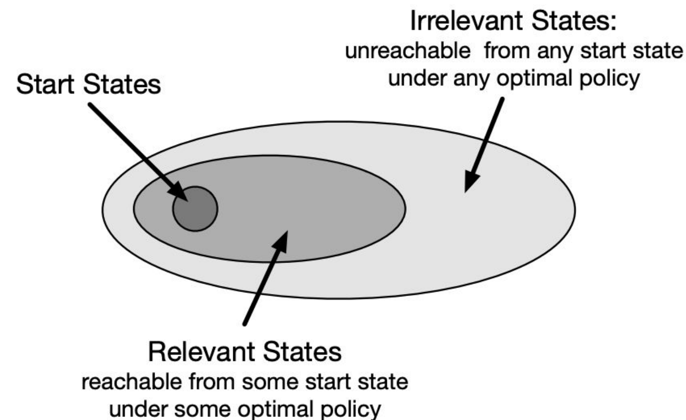
Inconclusive Results

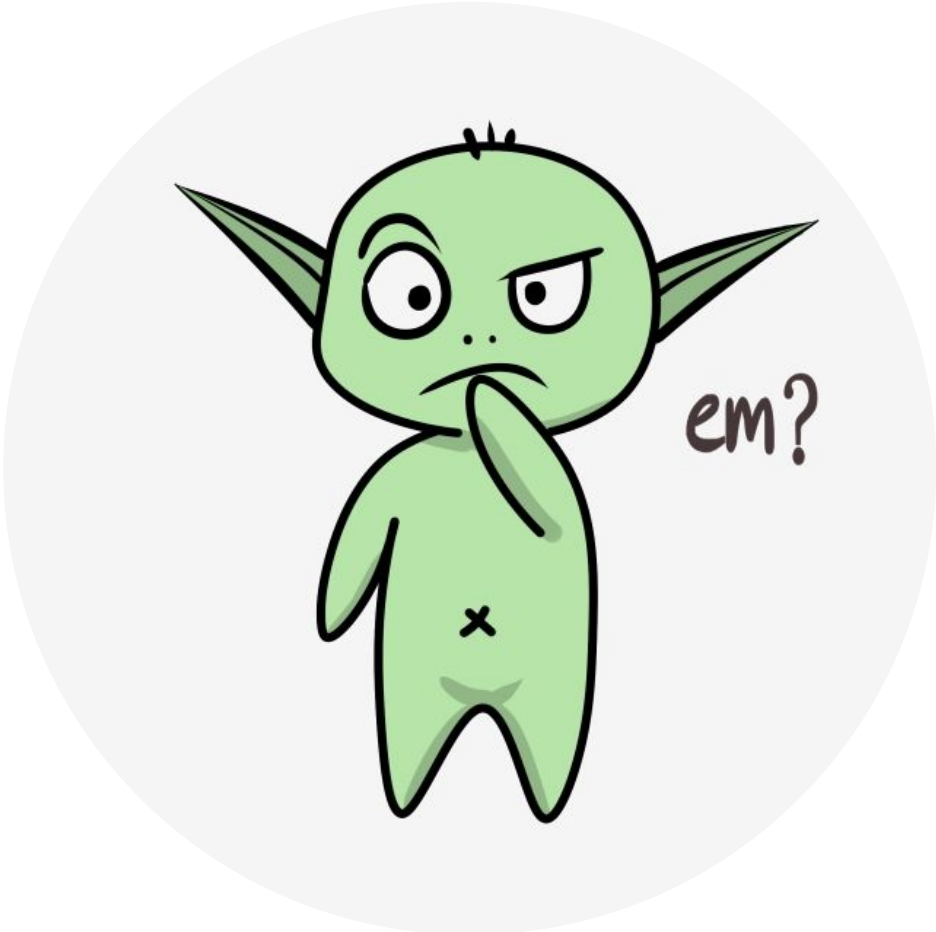




Real-time Dynamic Programming

- Real-time dynamic programming, or RTDP, is an on-policy trajectory-sampling version of the value-iteration algorithm of dynamic programming (DP).
- RTDP updates the values of states visited in actual or simulated trajectories by means of expected tabular value-iteration updates.
- RTDP is an example of an asynchronous DP algorithm. In RTDP, the update order is dictated by the order states are visited in real or simulated trajectories.
- It has some interesting convergence results in stochastic optimal path problems.





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

Decision-time Planning

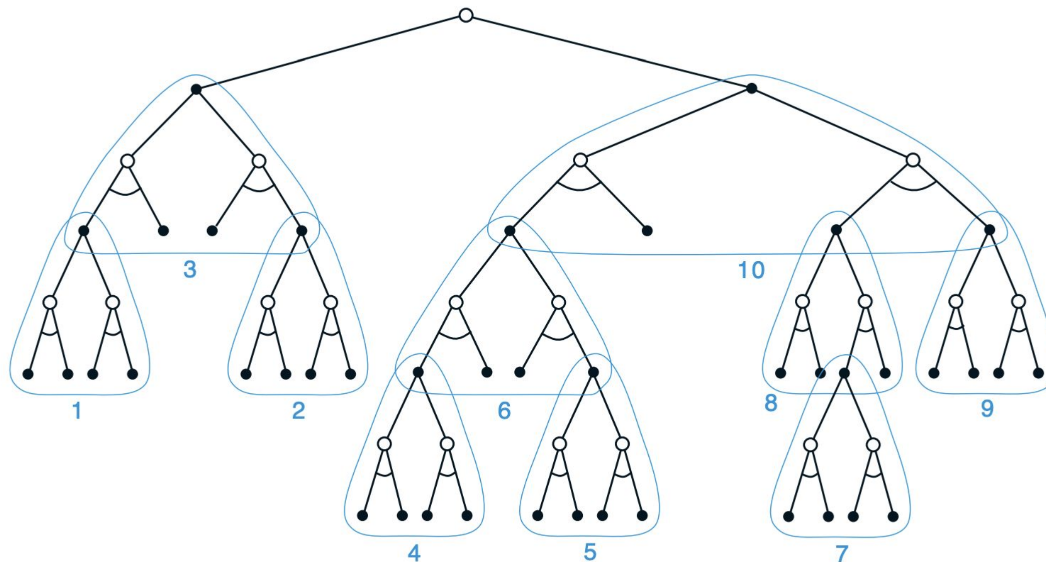
- We've been discussing *background planning:* using planning to gradually improve a policy or value function based on simulated experience obtained from a model.
  - Well before an action is selected for any current state S_t, planning has played a part in improving the table entries needed to select actions for many states, including S_t.
- *Decision-time planning* uses planning to begin and complete it after encountering each new state S_t, as a computation whose output is the selection of an action A_t; on the next step planning begins anew with S_{t+1} to produce A_{t+1}, and so on.
- We can still see decision-time planning as as proceeding from simulated experience to updates and values, and ultimately to a policy.
  - Now the values and policy are specific to the current state and the action choices available there.
- The response time really matters in this choice.

Heuristic Search

- The classical state-space planning methods in artificial intelligence are decision-time planning method collectively known as *heuristic search*.
- If one has a perfect model and an imperfect action-value function, then in fact deeper search will usually yield better policies.
- Much of its effectiveness is due to its search tree being focused on the states and actions that might immediately follow the current state.

Heuristic Search

Heuristic search can be implemented as a sequence of one-step updates (shown here outlined in blue) backing up values from the leaf nodes toward the root. The ordering shown below is for a selective depth-first search.



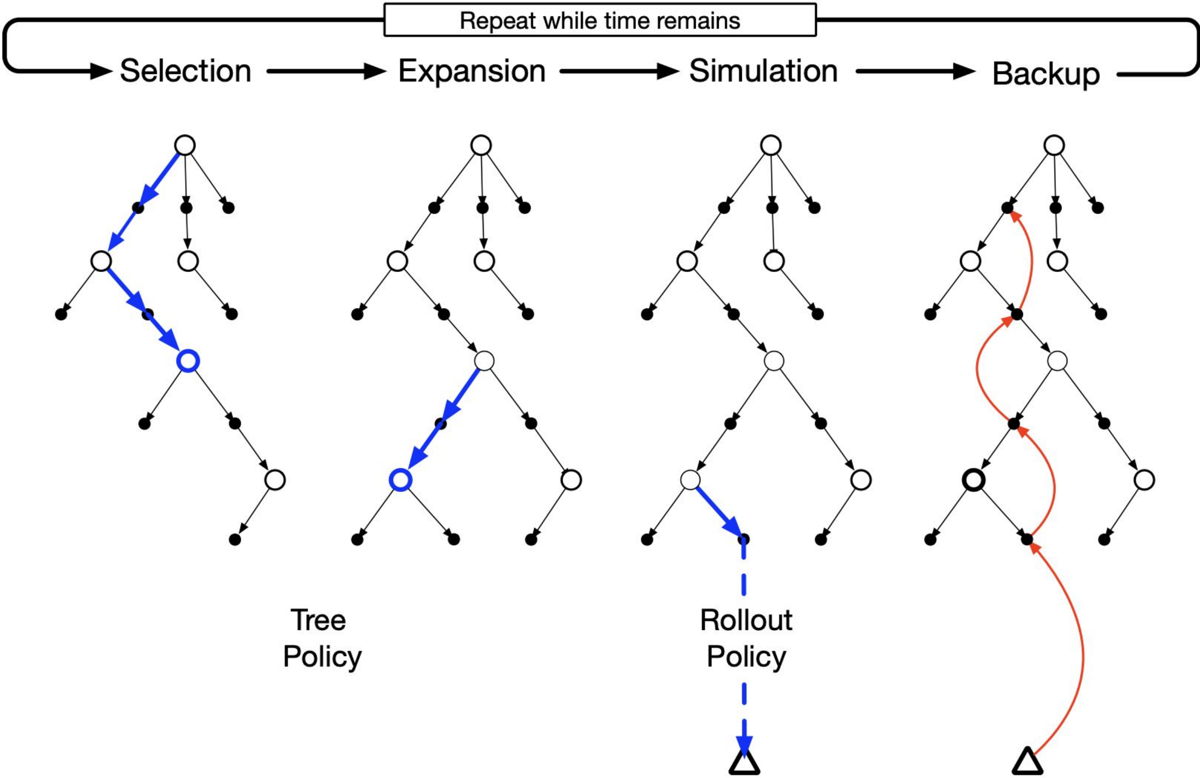
Rollout Algorithms

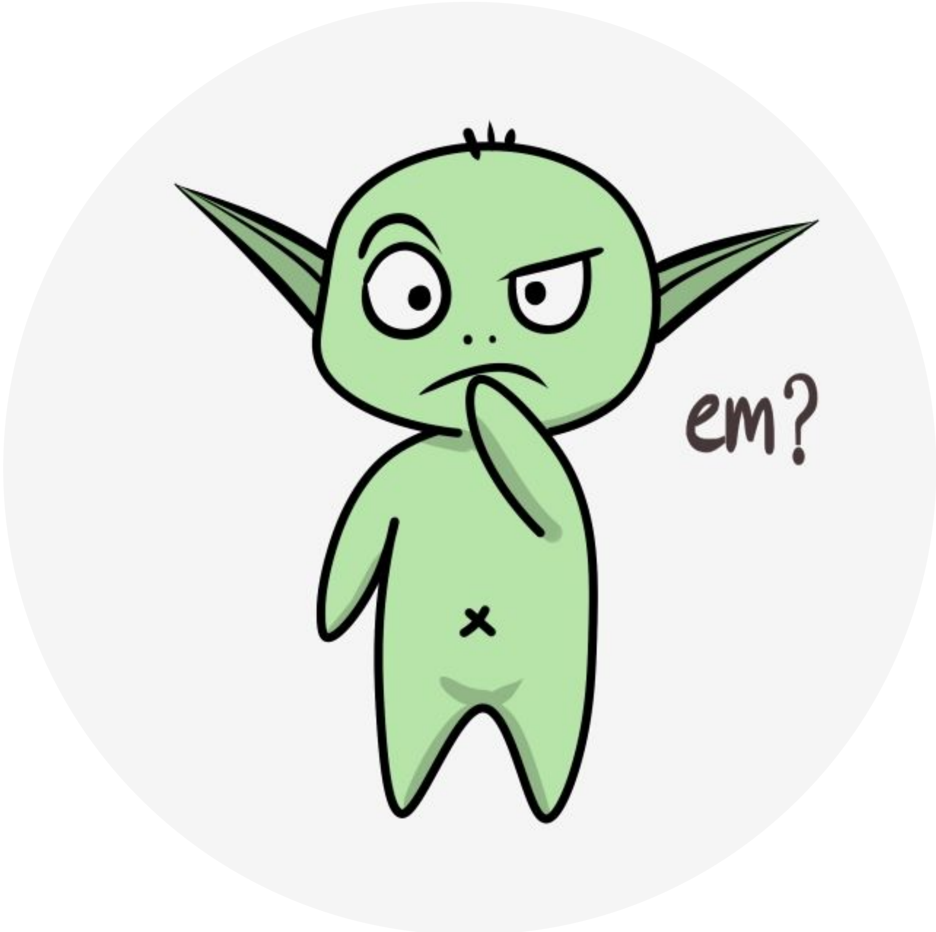
- Rollout algorithms are decision-time planning algorithms based on MC control applied to simulated trajectories that all begin at the current environment state.
    - They estimate action values for a given policy by averaging the returns of many simulated trajectories that start with each possible action and then follow the given policy.
- Unlike the Monte Carlo control algorithms previously described, the goal of a rollout algorithm is not to estimate a complete optimal action-value function, q_*, or a complete action-value function, q_π, for a given policy π.
    - They produce Monte Carlo estimates of action values only for each current state and for a given policy usually called the rollout policy.
- They are not learning algorithms *per se*, but they do leverage the RL toolkit.

Monte Carlo Tree Search (MCTS)

- MCTS is a great example of a rollout, decision-time planning algorithm.
  - But enhanced by the addition of a means for accumulating value estimates obtained from the MC simulations in order to successively direct simulations toward more highly-rewarding trajectories.
- The core idea of MCTS is to successively focus multiple simulations starting at the current state by extending the initial portions of trajectories that have received high evaluations from earlier simulations.
  - Monte Carlo value estimates are maintained only for the subset of state–action pairs that are most likely to be reached in a few steps, which form a tree rooted at the current state.

Monte Carlo Tree Search (MCTS)





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

Upper Confidence Bound 1 Applied to Trees (UCT)

Bandit based Monte-Carlo Planning

Levente Kocsis and Csaba Szepesvári

Computer and Automation Research Institute of the
Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary
kocsis@sztaki.hu

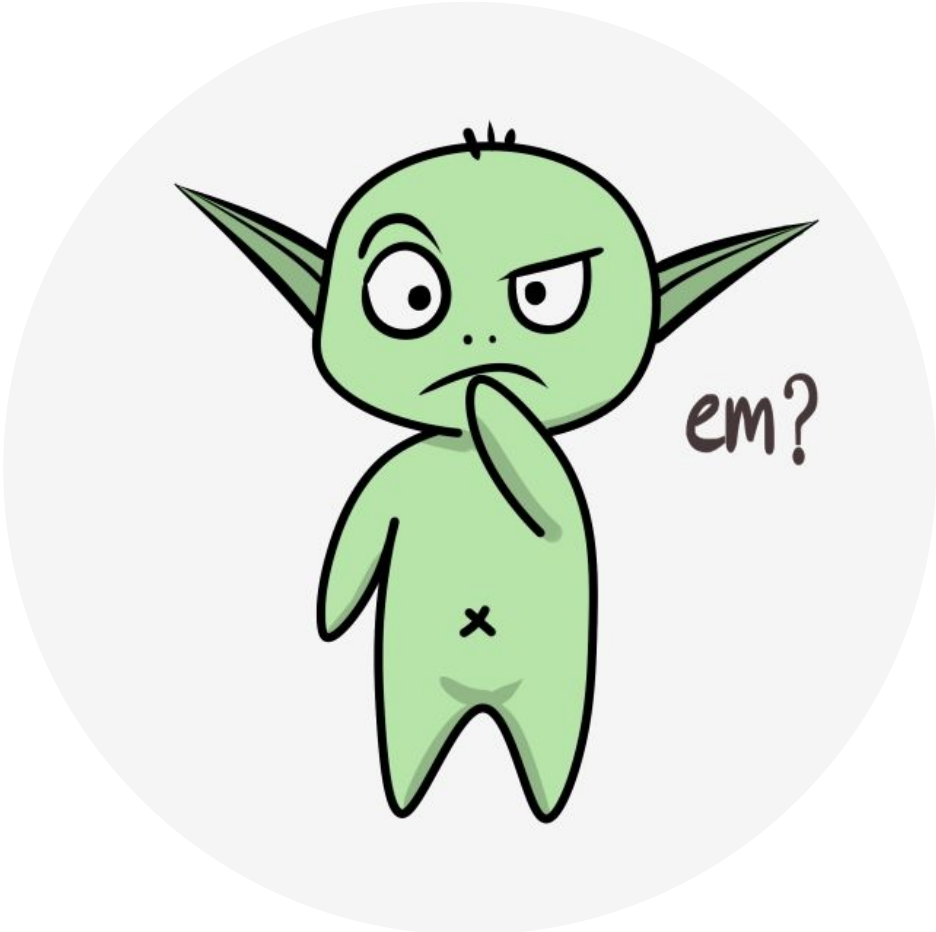
Monte Carlo Tree Search (MCTS)

Choose in each node of the game tree the move as the argmax of

$$\frac{w_i}{n_i} + \kappa\sqrt{\frac{\ln N_i}{n_i}}$$

- w_i: number of wins for the node considered after the i-th move.
- n_i: number of times the child node has been visited after the i-th move.
- N_i: number of times the parent node has been visited after the i-th move.
- κ: scalar parameter for trading-off exploration and exploitation.

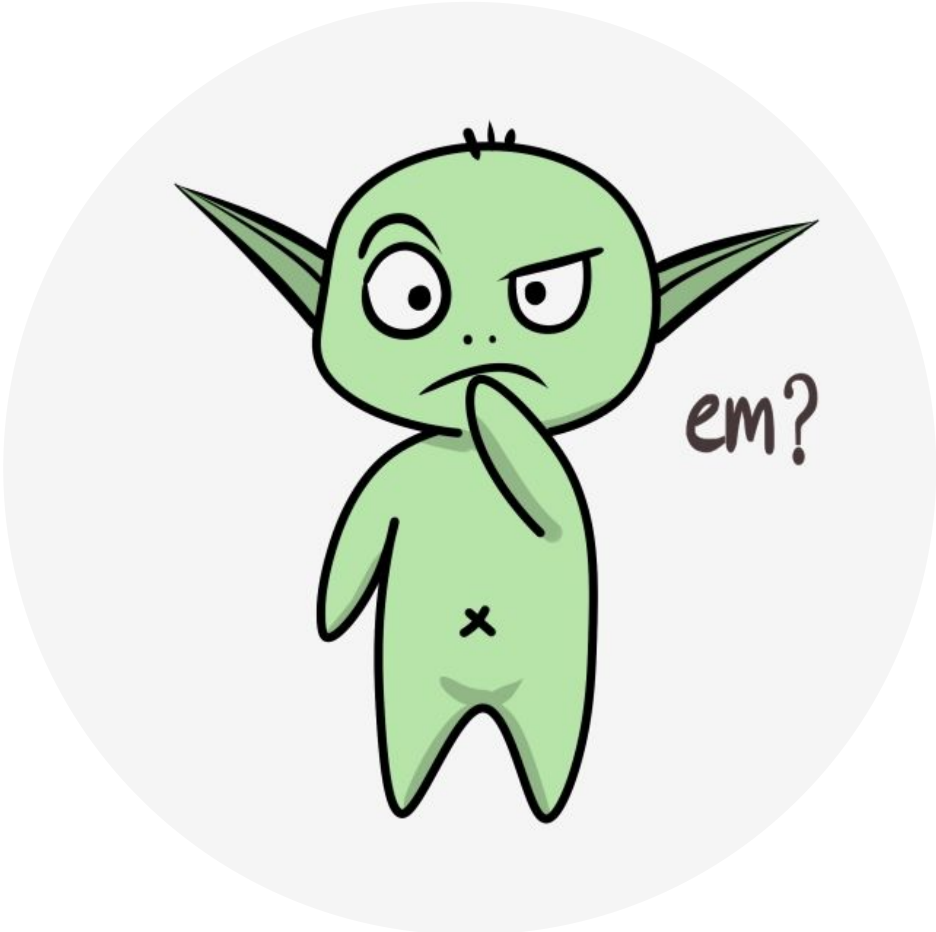
[Kocsis and Szepesvari, 2006]



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

MCTS incorporates several RL principles

- MCTS is a decision-time planning algorithm based on MC control applied to simulations that start from the root state (it is a kind of rollout algorithm).
    - It benefits from online, incremental, sample-based value estimation and policy improvement.
- It saves action-value estimates attached to the tree edges and updates them using reinforcement learning's sample updates.
    - It focuses the Monte Carlo trials on trajectories whose initial segments are common to high-return trajectories previously simulated.
- By incrementally expanding the tree, MCTS effectively grows a lookup table to store a partial action-value function, with memory allocated to the estimated values of state–action pairs visited in the initial segments of high-yielding sample trajectories
    - MCTS avoids the problem of globally approximating an action-value function while it retains the benefit of using past experience to guide exploration.

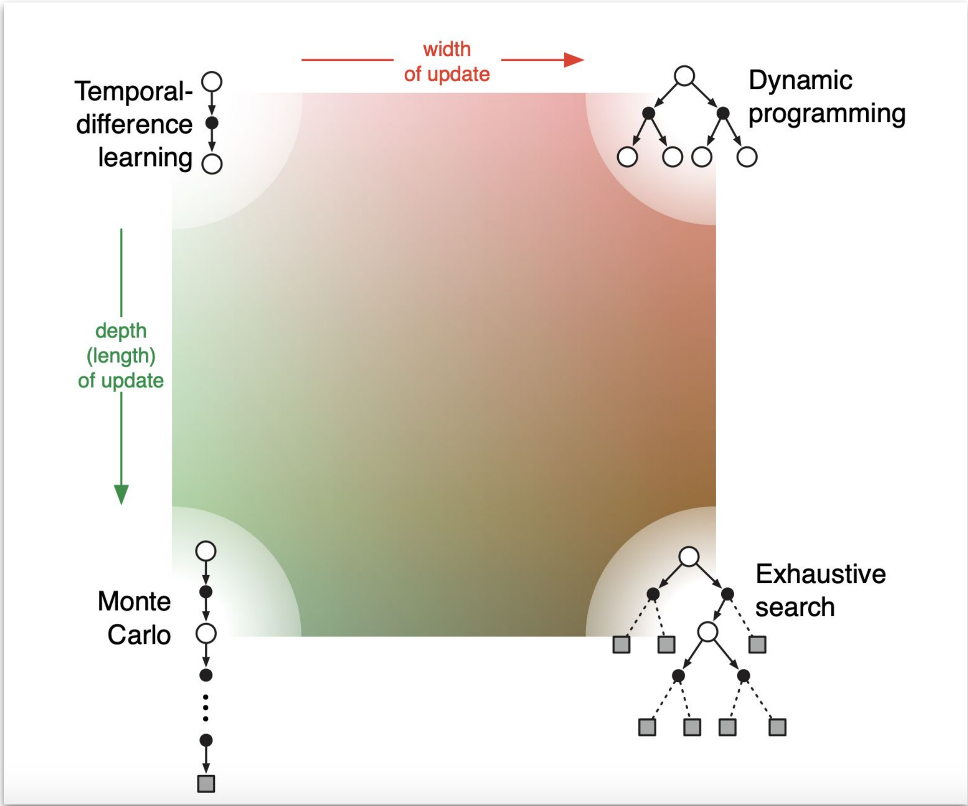


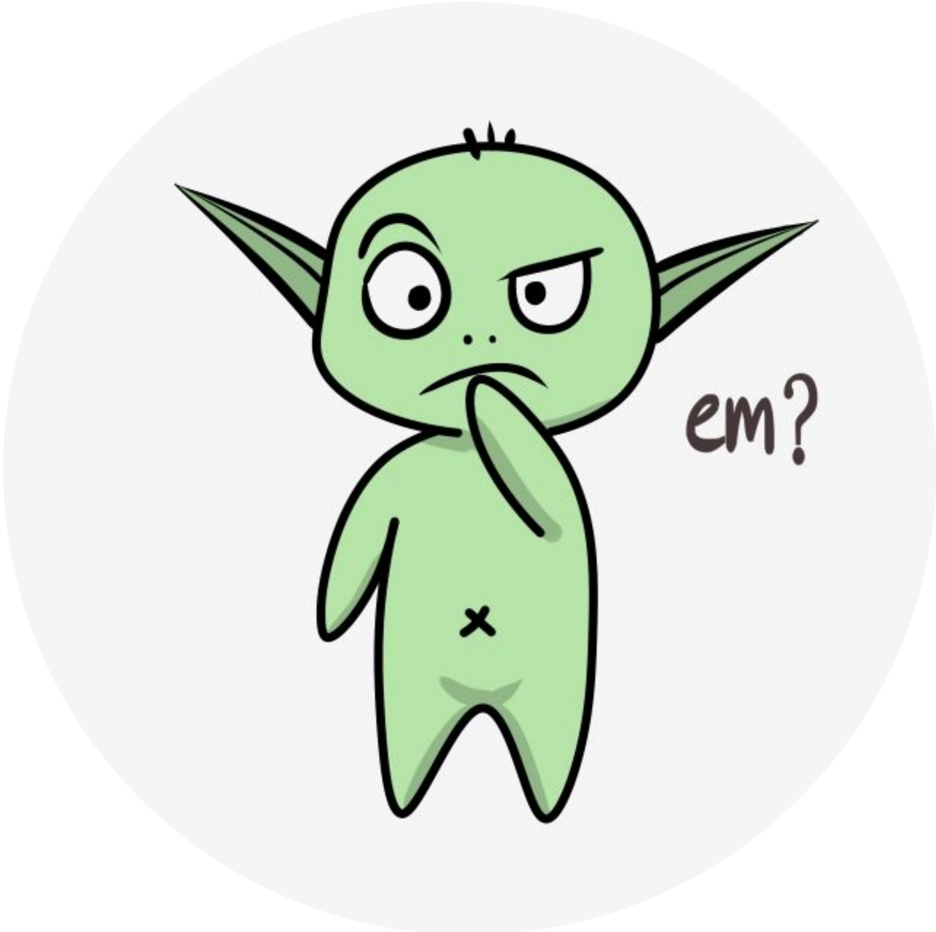
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Wrapping Up

- We have finished Part I of the textbook, Tabular Solution Methods.
- Reinforcement learning can be seen as being more than a collection of individual methods, but a coherent set of ideas cutting across methods.
    - They all seek to estimate value functions.
    - They all operate by backing up values along actual or possible state trajectories.
    - They all follow the general strategy of generalized policy iteration (GPI).

Wrapping Up





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