"Advice is a dangerous gift, even from the wise to the wise, and all courses may run ill."

J.R.R. Tolkien, The Fellowship of the Ring

CMPUT 655 Introduction to RL

Marlos C. Machado

Class 9/12

https://wallpaperaccess.com/full/803360.jpg

Plan

- A Brief Overview of Everything we discussed so far.
- Chapter 11: Off-policy Methods with Approximation.
- Chapter 12: Eligibility Traces.

I won’t talk about everything from these chapters.

Reminder

- Midterm
  - The midterm is next Friday. Nov 10th, in this very same room, from 15:30 to 16:50 (80 minutes)
  - The exam will be closed book and written in real-time. No cheat sheet.
- We will have “regular” classes next week, from 14:00 to 15:00.
  - I’ll give you a 30-minute break before the midterm. Everyone will need to leave the room.
- I’m still marking the project proposals. You should have them by next week.

Please, interrupt me at any time!

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

What have we discussed so far?

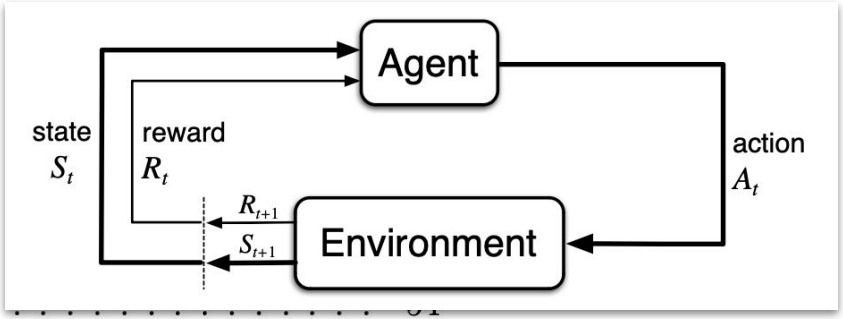
I Tabular Solution Methods

2 Multi-armed Bandits **25**

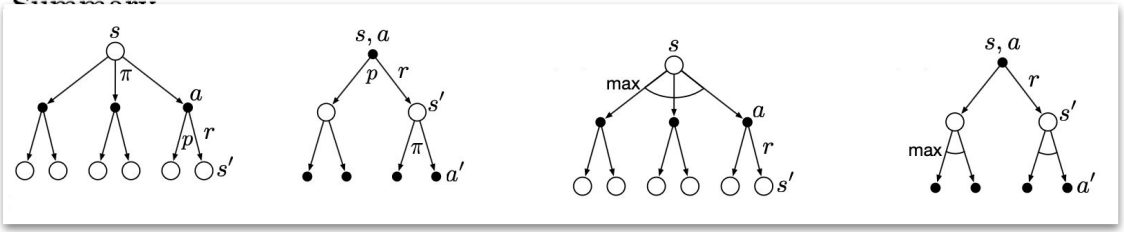
2.1 A k-armed Bandit Problem 25
2.2 Action-value Methods 27
2.3 The 10-armed Testbed 28
2.4 Incremental Implementation 30
2.5 Tracking a Nonstationary Problem 32
2.6 Optimistic Initial Values 34
2.7 Upper-Confidence-Bound Action Selection 35
~~2.8 Gradient Bandit Algorithms . . . . . 37~~
2.9 Associative Search (Contextual Bandits) 41
2.10 Summary 42

What have we discussed so far?

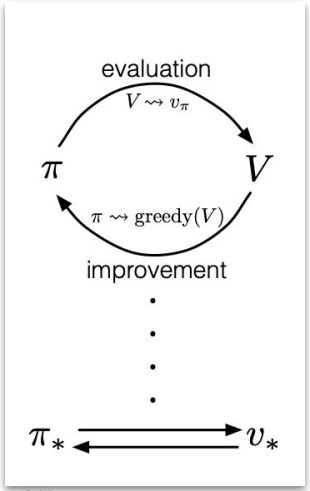
I Tabular Solution Methods



3 Finite Markov Decision Processes
3.1 The Agent–Environment Interface . . . . .
3.2 Goals and Rewards . . . . .
3.3 Returns and Episodes . . . . .
3.4 Unified Notation for Episodic and Continuing Tasks 57
3.5 Policies and Value Functions 58
3.6 Optimal Policies and Optimal Value Functions 62
3.7 Optimality and Approximation 67
3.8 Summary 68



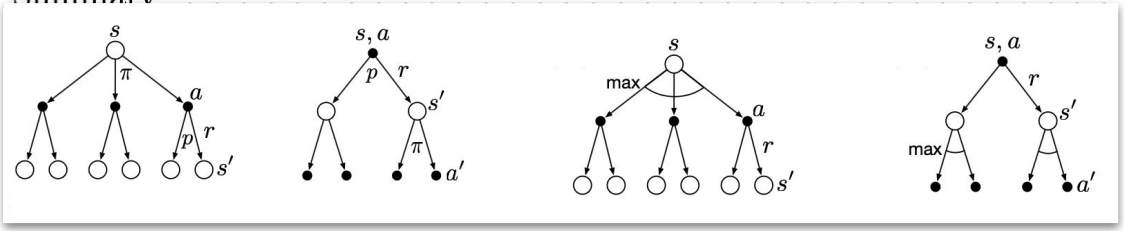
What have we discussed so far?



I Tabular Solution Methods

4 Dynamic Programming

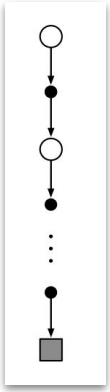
4.1 Policy Evaluation (Prediction) . . . .
4.2 Policy Improvement . . . .
4.3 Policy Iteration . . . .
4.4 Value Iteration . . . .
4.5 Asynchronous Dynamic Programming 85
4.6 Generalized Policy Iteration 86
4.7 Efficiency of Dynamic Programming 87
4.8 Summary 88



What have we discussed so far?

I Tabular Solution Methods

5 Monte Carlo Methods **91**
5.1 Monte Carlo Prediction 92
5.2 Monte Carlo Estimation of Action Values 96
5.3 Monte Carlo Control 97
5.4 Monte Carlo Control without Exploring Starts 100
5.5 Off-policy Prediction via Importance Sampling 103
5.6 Incremental Implementation 109
5.7 Off-policy Monte Carlo Control 110
~~5.8 *Discounting-aware Importance Sampling . . . . . 112~~
~~5.9 *Per-decision Importance Sampling . . . . . 114~~
5.10 Summary 115

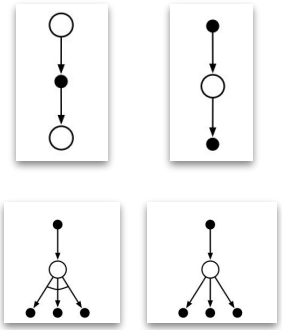


What have we discussed so far?

I Tabular Solution Methods

6 Temporal-Difference Learning **119**

6.1 TD Prediction 119
6.2 Advantages of TD Prediction Methods 124
6.3 Optimality of TD(0) 126
6.4 Sarsa: On-policy TD Control 129
6.5 Q-learning: Off-policy TD Control 131
6.6 Expected Sarsa 133
6.7 Maximization Bias and Double Learning 134
6.8 Games, Afterstates, and Other Special Cases 136
6.9 Summary 138

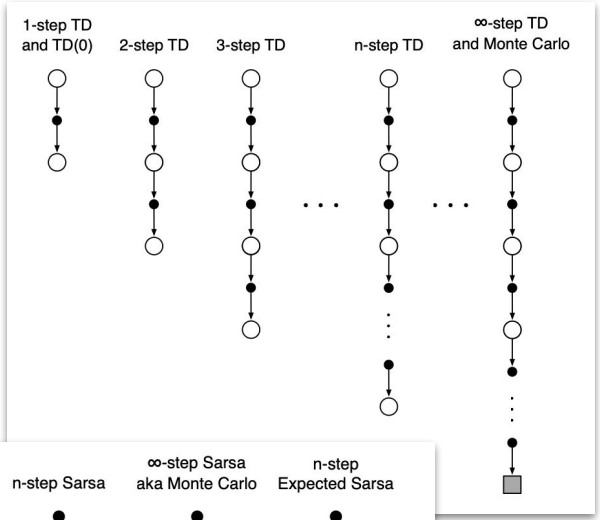


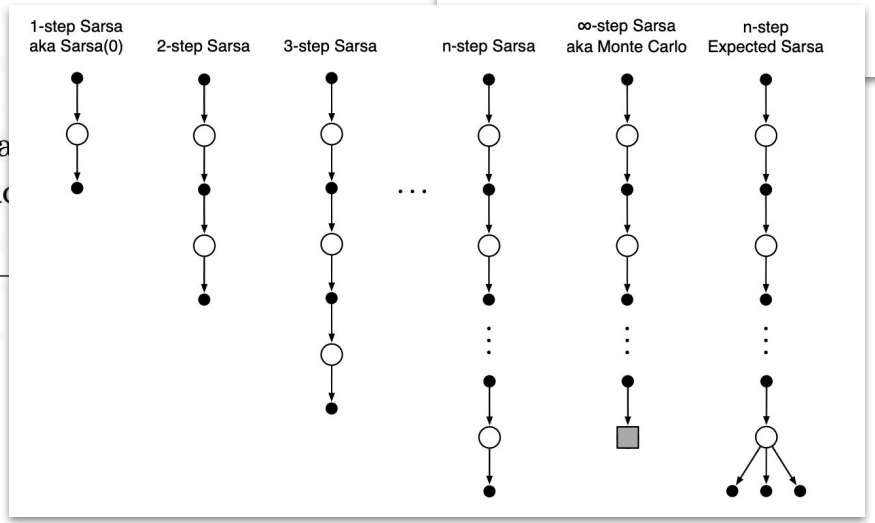
What have we discussed so far?

I Tabular Solution Methods

7 *n*-step Bootstrapping

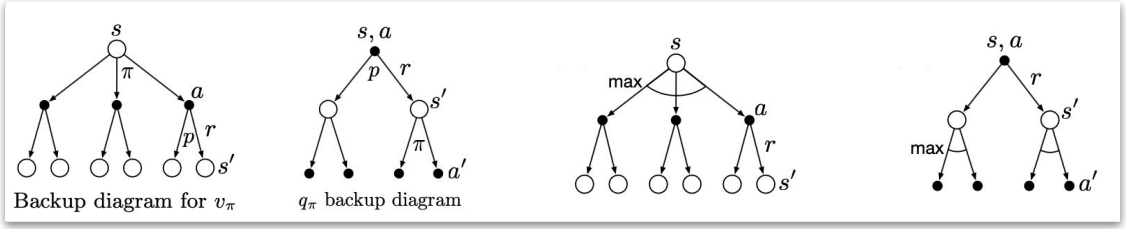
7.1 *n*-step TD Prediction . . . . . .
7.2 *n*-step Sarsa . . . . . .
7.3 *n*-step Off-policy Learning . . . . . .
7.4 *Per-decision Methods with Control Va
7.5 Off-policy Learning Without Importanc
The *n*-step Tree Backup Algorithm . .
~~7.6 *A Unifying Algorithm: *n*-step $Q(\sigma)$~~ .
7.7 Summary . . . . . .

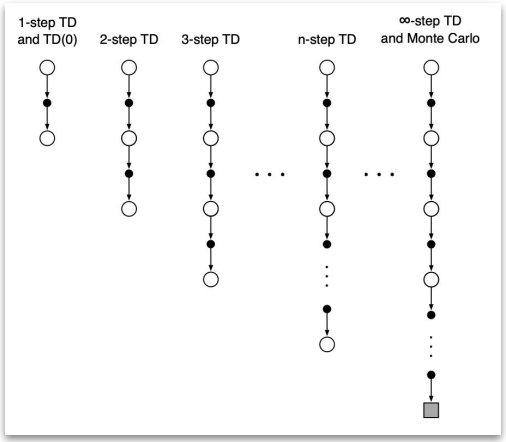


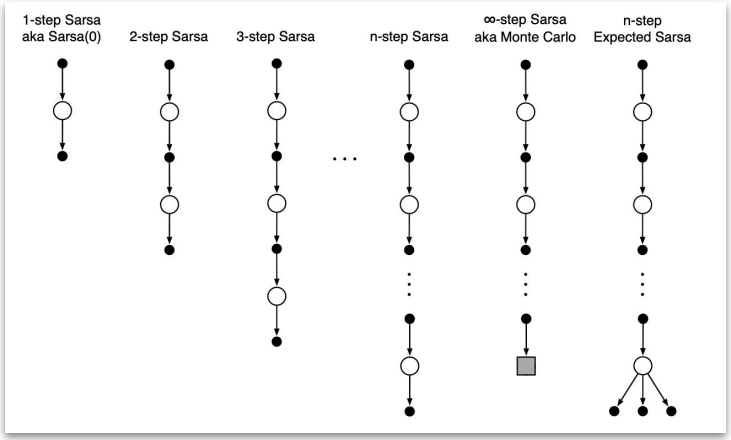


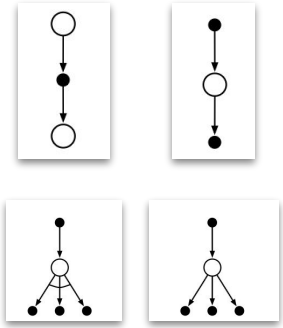
What have we discussed so far?

I Tabular Solution Methods







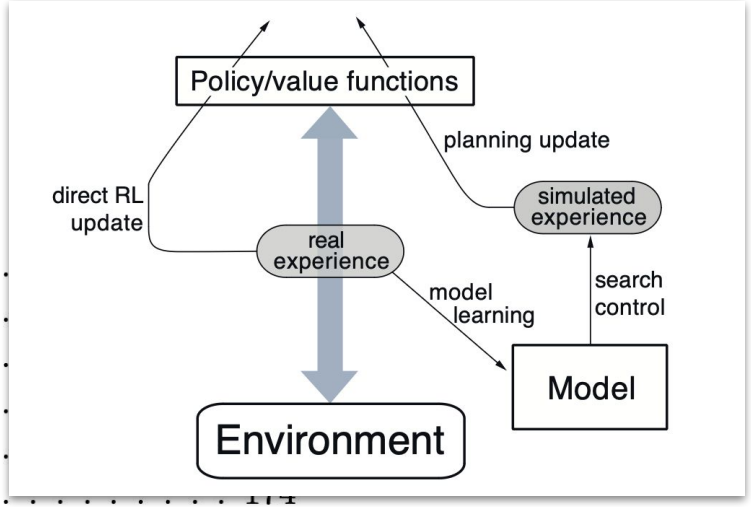


What have we discussed so far?

I Tabular Solution Methods

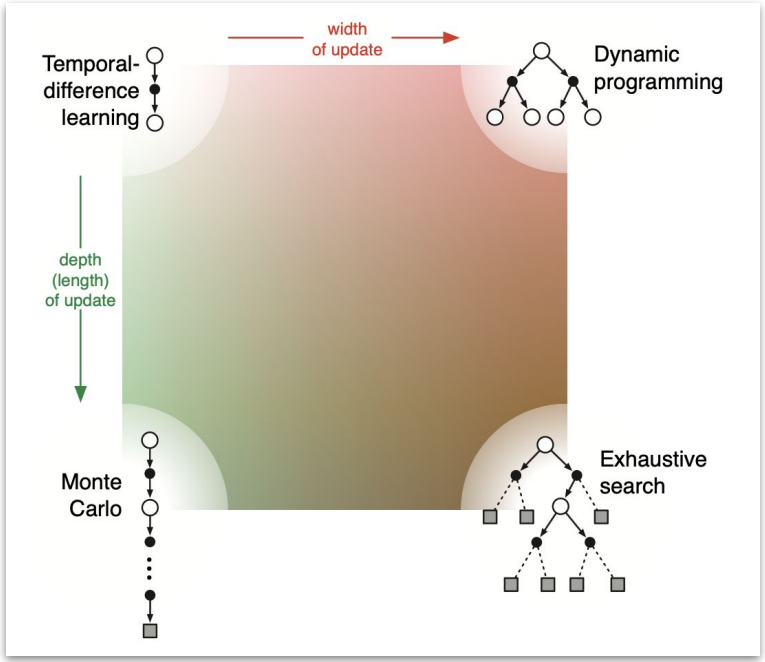
8 Planning and Learning with Tabular Methods

8.1 Models and Planning . . .
8.2 Dyna: Integrated Planning, Acting, and Learning . . .
8.3 When the Model Is Wrong . . .
8.4 Prioritized Sweeping . . .
8.5 Expected vs. Sample Updates . . .
8.6 Trajectory Sampling . . .
8.7 Real-time Dynamic Programming . . . 177
8.8 Planning at Decision Time . . . 180
8.9 Heuristic Search . . . 181
8.10 Rollout Algorithms . . . 183
8.11 Monte Carlo Tree Search . . . 185



What have we discussed so far?

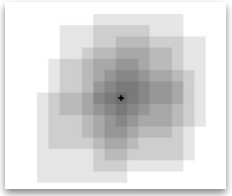
I Tabular Solution Methods

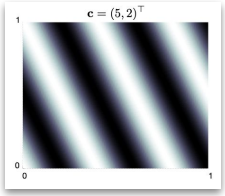


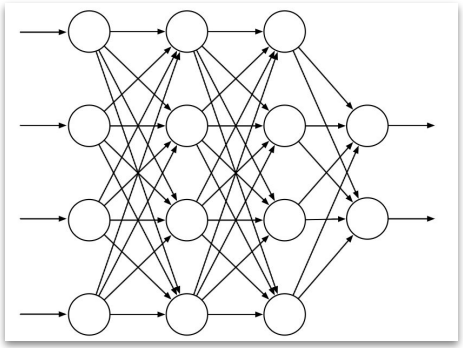
What have we discussed so far?

II Approximate Solution Methods

9 On-policy Prediction with Approximation **197**
9.1 Value-function Approximation 198
9.2 The Prediction Objective ($\overline{\text{VE}}$) 199
9.3 Stochastic-gradient and Semi-gradient Methods 200
9.4 Linear Methods 204
9.5 Feature Construction for Linear Methods 210
9.5.1 Polynomials 210
9.5.2 Fourier Basis 211
9.5.3 Coarse Coding 215
9.5.4 Tile Coding 217
9.5.5 Radial Basis Functions 221
9.6 Selecting Step-Size Parameters Manually 222
9.7 Nonlinear Function Approximation: Artificial Neural Networks 223
9.8 Least-Squares TD 228
9.9 Memory-based Function Approximation 230
9.10 Kernel-based Function Approximation 232
~~9.11 Looking Deeper at On-policy Learning: Interest and Emphasis . . . . . 234~~
9.12 Summary 236







What have we discussed so far?

II Approximate Solution Methods

10 On-policy Control with Approximation **243**

10.1 Episodic Semi-gradient Control 243

10.2 Semi-gradient n-step Sarsa 247

~~10.3 Average Reward: A New Problem Setting for Continuing Tasks . . . . . 249~~

~~10.4 Deprecating the Discounted Setting . . . . . 253~~

~~10.5 Differential Semi-gradient n-step Sarsa . . . . . 255~~

10.6 Summary 256

What have we discussed so far?

II Approximate Solution Methods

11 *Off-policy Methods with Approximation 257

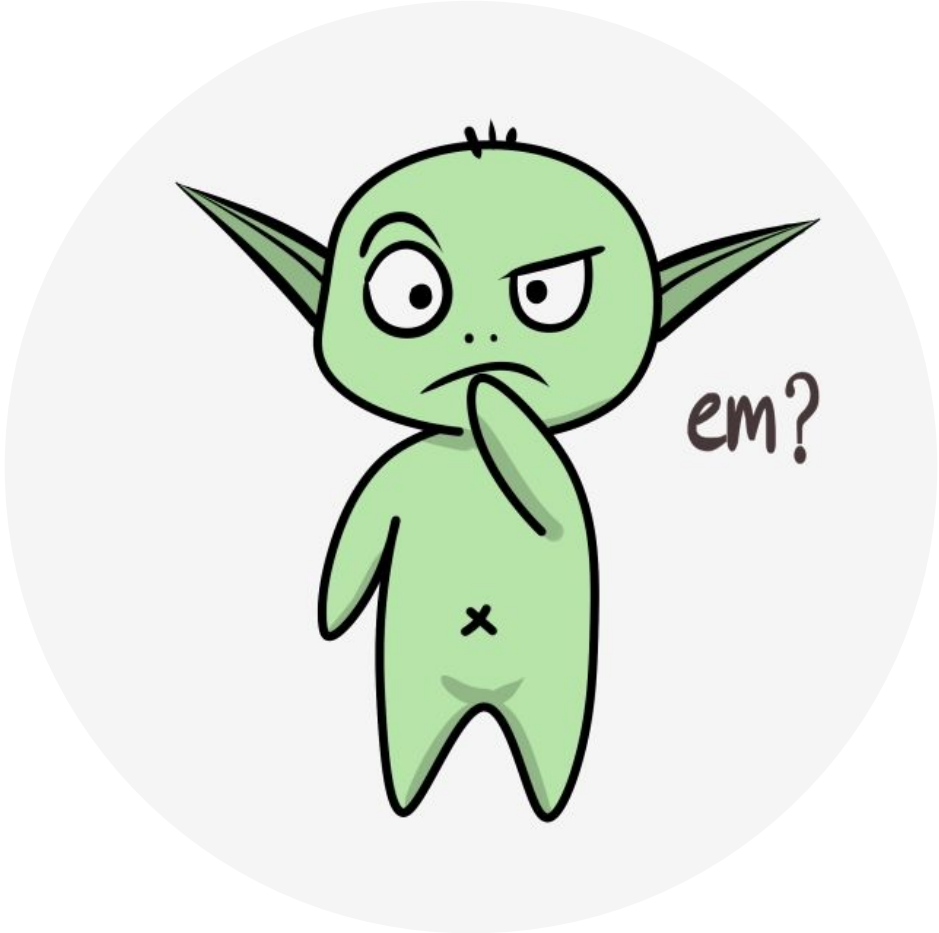
11.1 Semi-gradient Methods . . . 258
11.2 Examples of Off-policy Divergence . . . 260
11.3 The Deadly Triad . . . 264
~~11.4 Linear Value-function Geometry . . . 266~~
~~11.5 Gradient Descent in the Bellman Error . . . 269~~
~~11.6 The Bellman Error is Not Learnable . . . 274~~
~~11.7 Gradient-TD Methods . . . 278~~
~~11.8 Emphatic-TD Methods . . . 281~~
~~11.9 Reducing Variance . . . 283~~
11.10 Summary . . . 284

What have we discussed so far?

II Approximate Solution Methods

12 Eligibility Traces **287**

12.1 The λ-return . . . 288
12.2 TD(λ) . . . 292
~~12.3 n-step Truncated λ-return Methods . . . 295~~
~~12.4 Redoing Updates: Online λ-return Algorithm . . . 297~~
~~12.5 True Online TD(λ) . . . 299~~
~~12.6 *Dutch Traces in Monte Carlo Learning . . . 301~~
12.7 Sarsa(λ) . . . 303
~~12.8 Variable λ and γ . . . 307~~
~~12.9 Off-policy Traces with Control Variates . . . 309~~
12.10 Watkins's Q(λ) to Tree-Backup(λ) . . . 312
~~12.11 Stable Off-policy Methods with Traces . . . 314~~
~~12.12 Implementation Issues . . . 316~~
12.13 Conclusions . . . 317



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

Chapter 11

*Off-policy Methods with Approximation

Off-Policy Methods with Approximation – Why?

Off-policy learning is extremely important, but we need to be able to approximate our estimates when dealing with large (or infinite) state spaces ¯_(ツ)_/¯

Off-Policy Methods with Approximation

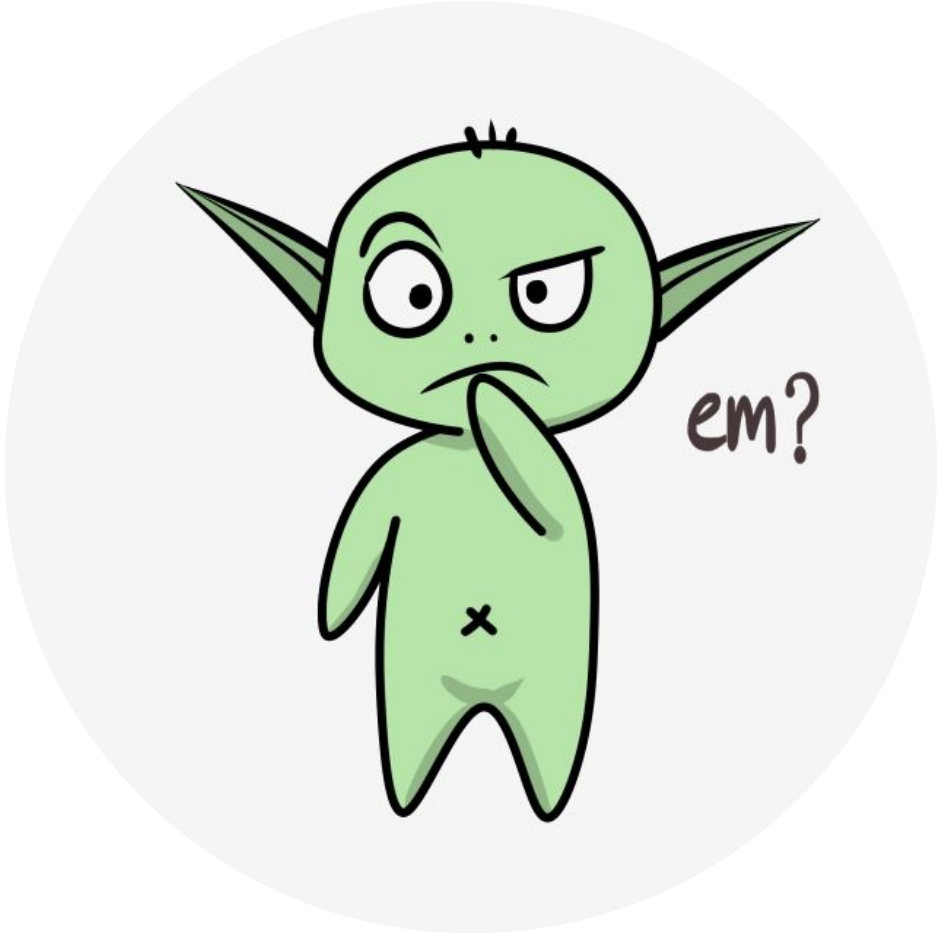
- “The extension to function approximation turns out to be significantly and harder for off-policy learning than it is for on-policy learning.”
    - “The tabular off-policy methods developed in Chapter 6 and 7 readily extend to semi-gradient algorithms, but these algorithms do not converge as robustly as they do under on-policy training.”

Off-Policy Methods with Approximation

- The challenge of off-policy learning can be divided into two parts:
  - The target update (not to be confused with the target policy).
  - The distribution of the updates.

Off-Policy Methods with Approximation

- The challenge of off-policy learning can be divided into two parts:
  - The target update (not to be confused with the target policy).
    - Importance sampling.
  - The distribution of the updates.
    - Importance sampling.
    - True gradient methods that do not rely on any special distribution for stability.



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

Semi-gradient Methods

- Addressing the target of the update (but not the distribution of the updates).
  - “These methods may diverge in some cases, and in that sense they are not sound, but still they are often successfully used.”

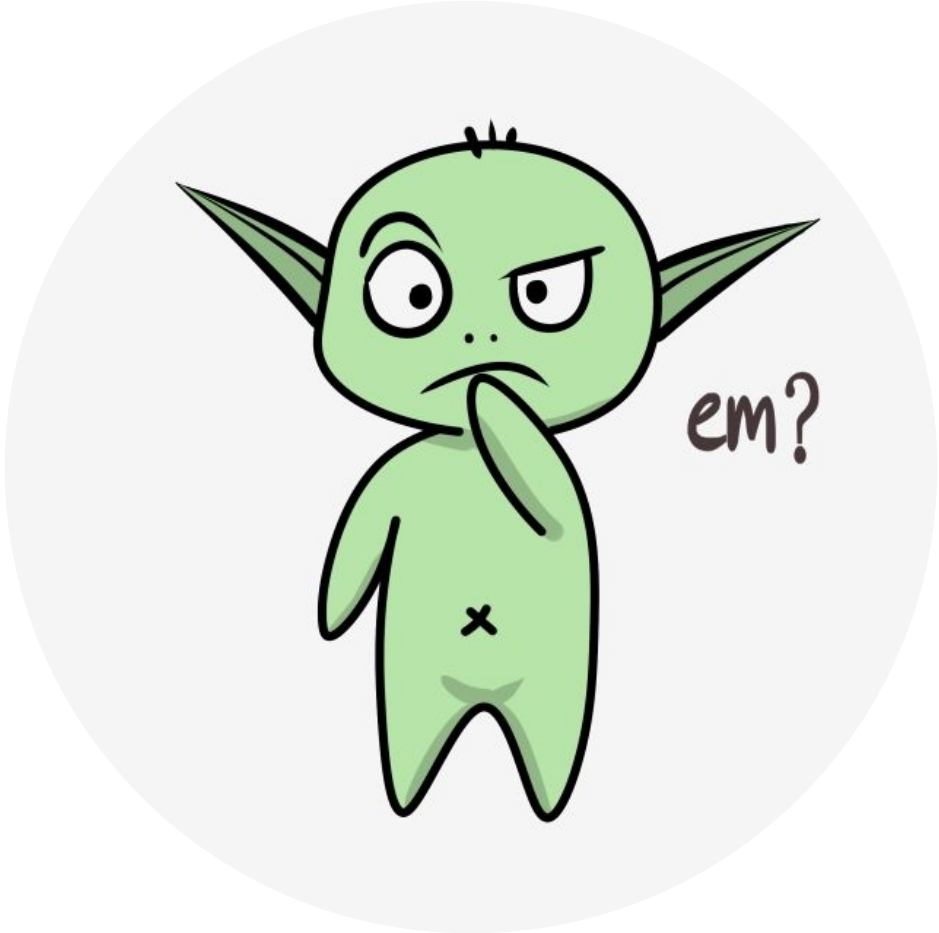
Semi-gradient Methods

- Addressing the target of the update (but not the distribution of the updates).
  - “These methods may diverge in some cases, and in that sense they are not sound, but still they are often successfully used.”
- Off-policy TD:

$$\rho_t \doteq \rho_{t:t} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t,\mathbf{w}_t)$$

$$\delta_t \doteq R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t) - \hat{v}(S_t,\mathbf{w}_t)$$



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

Expected Sarsa

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \text{ with}$$

$$\delta_t \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

"Note that this algorithm does not use importance sampling. In the tabular case it is clear that this is appropriate because the only sample action is A_t, and in learning its values we do not have to consider any other actions. With function approximation it is less clear because we might want to weight different state-action pairs differently once they all contributed to the same overall approximation."

Expected Sarsa

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \text{ with}$$

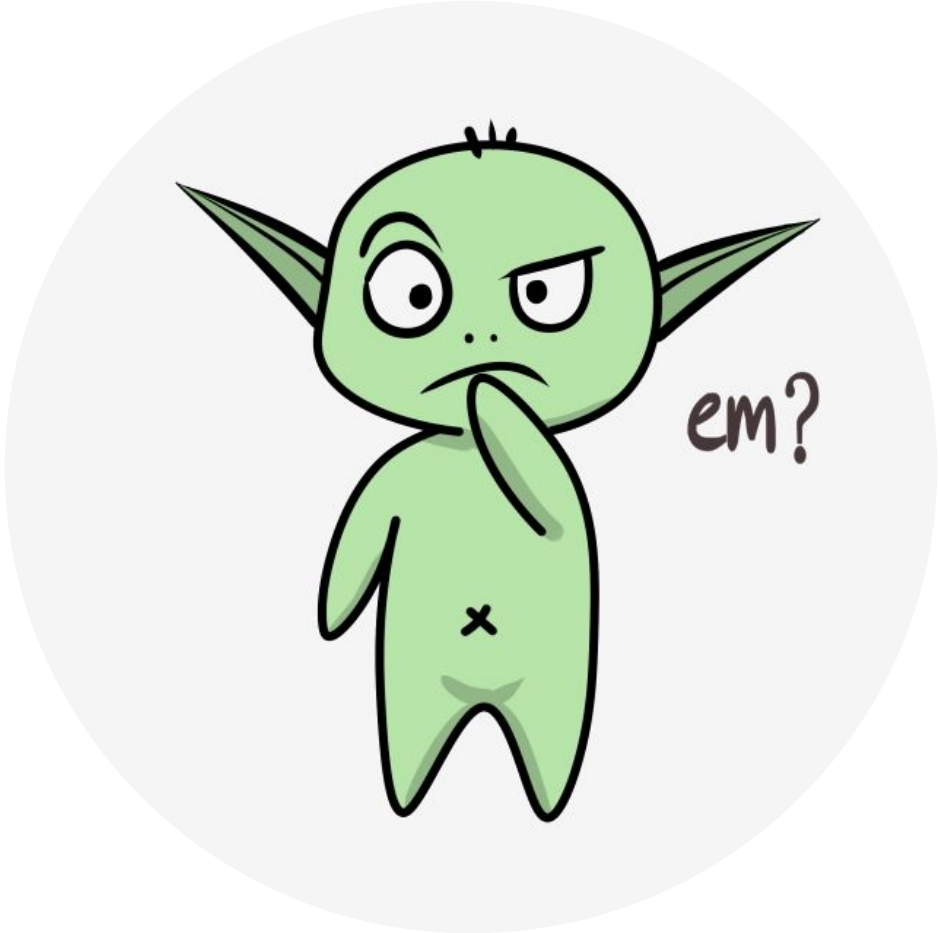
$$\delta_t \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

"Note that this algorithm does not use importance sampling. In the tabular case it is clear that this is appropriate because the only sample action is A_t, and in learning its values we do not have to consider any other actions. With function approximation it is less clear because we might want to weight different state-action pairs differently once they all contributed to the same overall approximation."

- n-step Expected Sarsa

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1})$$

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha \rho_{t+1} \cdots \rho_{t+n} \left[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})$$



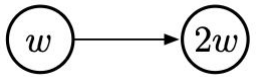
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

The Mismatch between Distribution of Updates

- In off-policy learning, the distribution of updates does not match the on-policy distribution.

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t,\mathbf{w}_t)$$
$$\delta_t \doteq R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t) - \hat{v}(S_t,\mathbf{w}_t)$$

Example:



$\mathbf{x}' = [0, \ldots, 0, 2]^\top$

$\mathbf{x} = [0, \ldots, 0, 1]^\top$

$\alpha = 0.1$
$\gamma = 0.9$
$\rho = 1.0$

First time:

$\mathbf{w}_t = [0, \ldots, 0, 10]^\top$
$\delta_t = 0 + 0.9 \times \mathbf{x}'^\top\mathbf{w}_t - \mathbf{x}^\top\mathbf{w}_t$
$= 0 + 0.9 \times 20 - 10 = 8$

$\mathbf{w}_{t+1} = \mathbf{w}_t + 0.1 \times 8 \times \mathbf{x}$
$= [0, \ldots, 0, 10.8]$

Second time:

$\mathbf{w}_{t+1} = [0, \ldots, 0, 10.8]^\top$
$\delta_{t+1} = 0 + 0.9 \times \mathbf{x}'^\top\mathbf{w}_t - \mathbf{x}^\top\mathbf{w}_t$
$= 0 + 0.9 \times 21.6 - 10.8 = 8.64$

$\mathbf{w}_{t+2} = \mathbf{w}_{t+1} + 0.1 \times 8.64 \times \mathbf{x}$
$= [0, \ldots, 0, 11.7]$

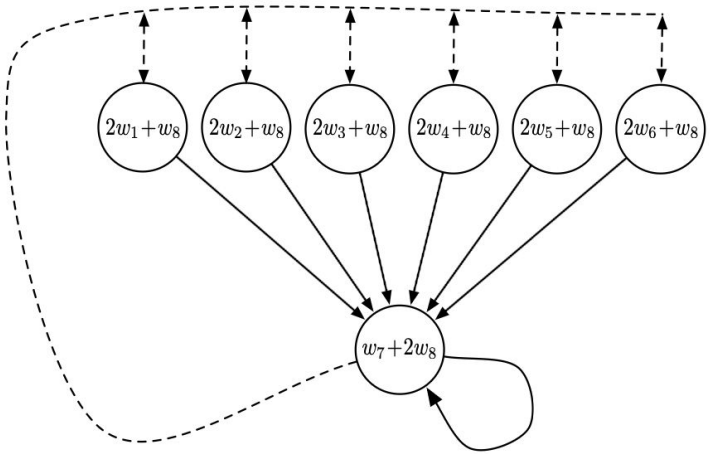
$$w_{t+1} = w_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t,w_t) = w_t + \alpha \cdot 1 \cdot (2\gamma - 1)w_t \cdot 1 = \big(1 + \alpha(2\gamma - 1)\big)w_t.$$

The Mismatch between Distribution of Updates

- Maybe we need overparameterization, or linearly independent features.

The Mismatch between Distribution of Updates

- Maybe we need overparameterization, or linearly independent features.
- Nope. Baird's counter-example:

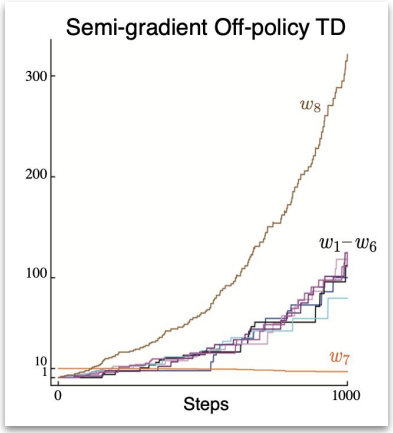


$\pi(\text{solid}|\cdot) = 1$

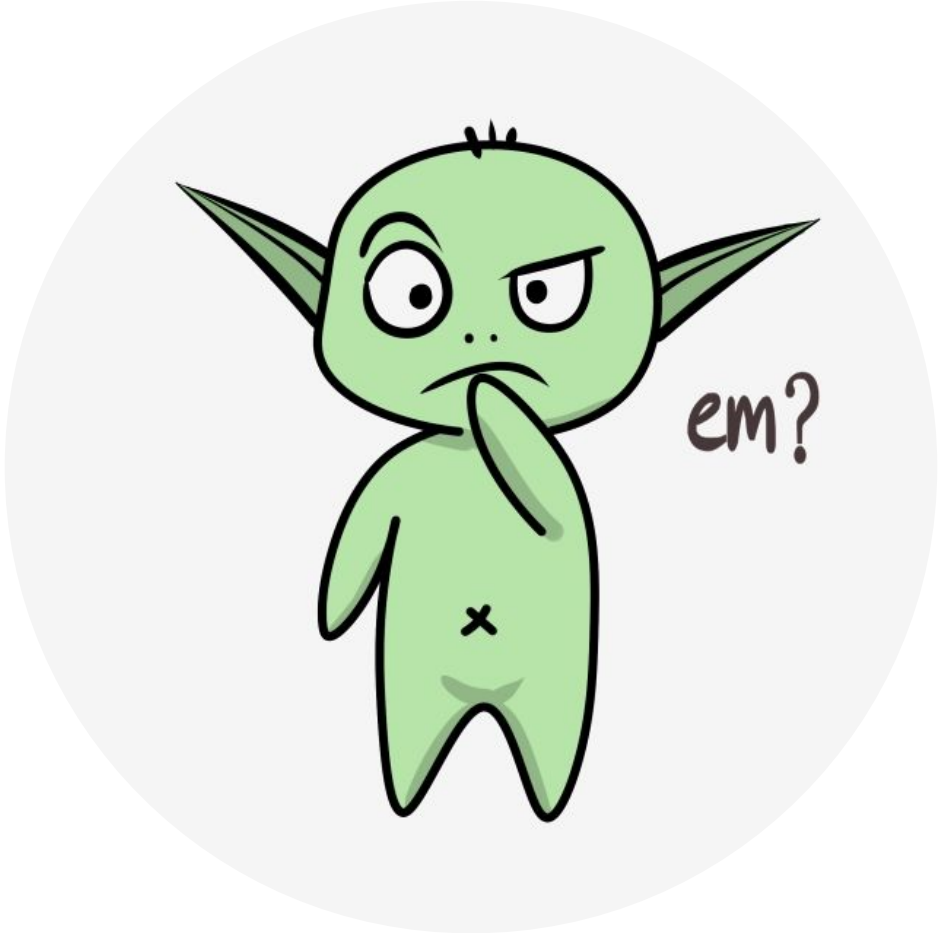
$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$



There are also counterexamples similar to Baird's showing divergence for Q-learning.



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

The Deadly Triad

1. **Function approximation**
2. **Bootstrapping**
3. **Off-policy training**

Should we give up on something?

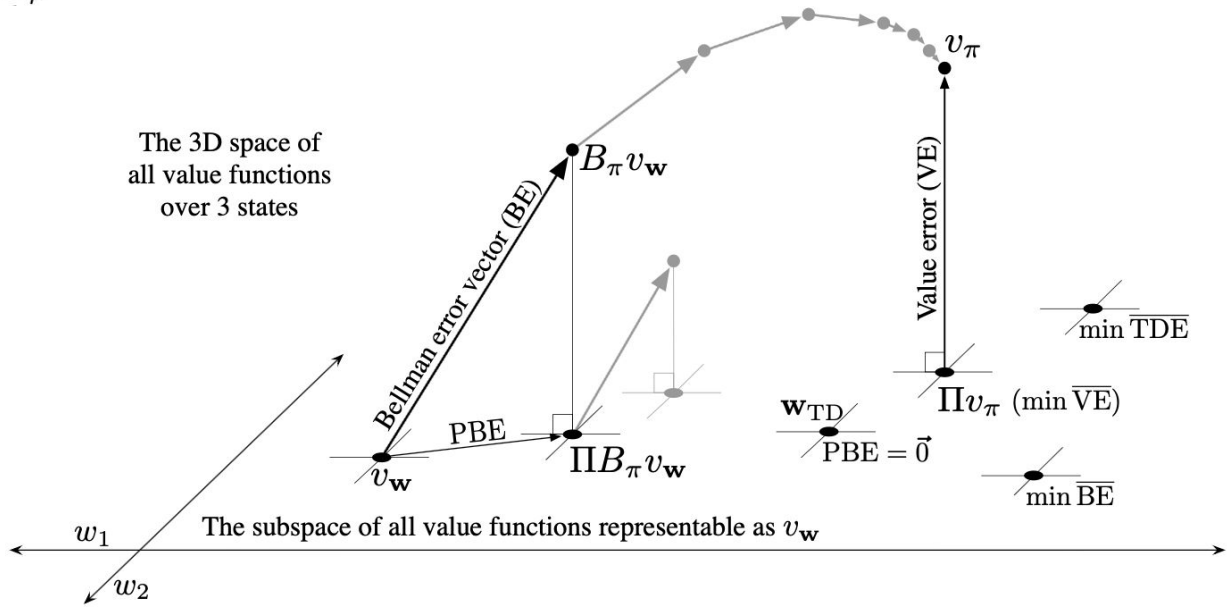
Notice, not control, nor GPI. Prediction!

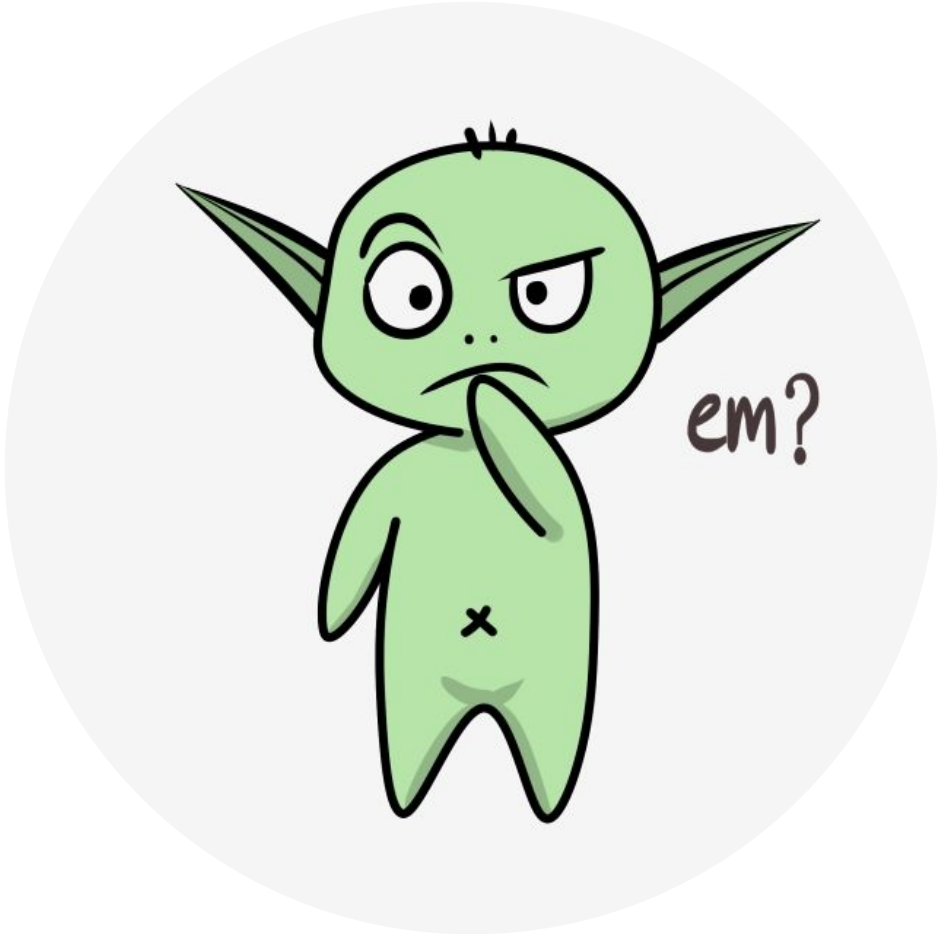
There's much more to be said, but not in this class

$$\overline{\text{VE}}(\mathbf{w}) = \|v_{\mathbf{w}} - v_\pi\|_\mu^2$$

$$\overline{\text{BE}}(\mathbf{w}) = \|\bar{\delta}_{\mathbf{w}}\|_\mu^2$$

$$\overline{\text{PBE}}(\mathbf{w}) = \|\Pi\bar{\delta}_{\mathbf{w}}\|_\mu^2$$





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

Chapter 12

Eligibility Traces

Eligibility Traces – Why?

“Eligibility traces are one of the basic mechanisms of reinforcement learning. (...) Almost any temporal-difference (TD) method, such as Q-learning or Sarsa, can be combined with eligibility traces to obtain a more general method that may learn more efficiently.”

Eligibility traces

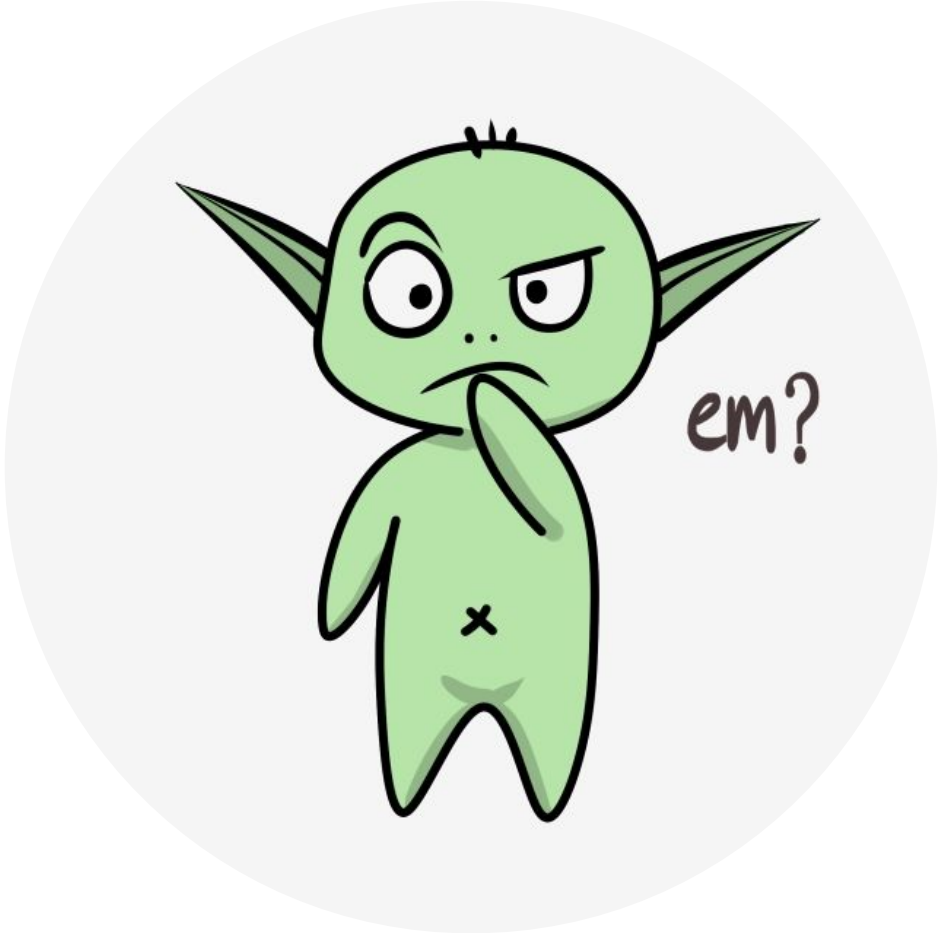
- They unify and generalize TD and Monte Carlo methods.
  - $\lambda = 0$ give us one-step TD methods.
  - $\lambda = 1$ gives us Monte Carlo methods, but it is a way of implementing them online and on continuing problems without episodes.

Eligibility traces

- They unify and generalize TD and Monte Carlo methods.
  - $\lambda = 0$ give us one-step TD methods.
  - $\lambda = 1$ gives us Monte Carlo methods, but it is a way of implementing them online and on continuing problems without episodes.
- What else allows us to unify TD and Monte Carlo methods?

Eligibility traces

- They unify and generalize TD and Monte Carlo methods.
  - $\lambda = 0$ give us one-step TD methods.
  - $\lambda = 1$ gives us Monte Carlo methods, but it is a way of implementing them online and on continuing problems without episodes.
- What else allows us to unify TD and Monte Carlo methods?
  - n-step returns!



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

Eligibility traces

- Eligibility trace is a short-term memory vector, $\mathbf{z}_t \in \mathbb{R}^d$, that parallels the long-term weight vector $\mathbf{w}_t \in \mathbb{R}^d$.
  - When a component of $\mathbf{w}_t$ participates in producing an estimated value, then the corresponding component of $\mathbf{z}_t$ is bumped up and then begins to fade away.
  - The trace decay parameter $\lambda \in [0, 1]$ determines the rate at which the trace falls.
- The computational advantage of eligibility traces over *n*-step methods is that only a single trace vector is required rather than a store of the last *n* feature vectors.
- The results with eligibility traces are much stronger with linear function approx.

The λ-return

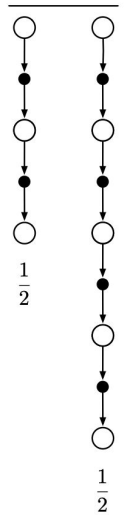
- Chapter 7, the *n*-step return is the sum of the first *n* rewards plus the estimated value of the state reached in *n* steps:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad 0 \leq t \leq T - n$$

- What if we averaged *n*-step returns for different *n*s? For example:

$$\tfrac{1}{2} G_{t:t+2} + \tfrac{1}{2} G_{t:t+4}$$

Any set of n-step returns can be avg. in this way, even an infinite set, as as the weights on the component returns are positive and sum to 1.



The λ-return

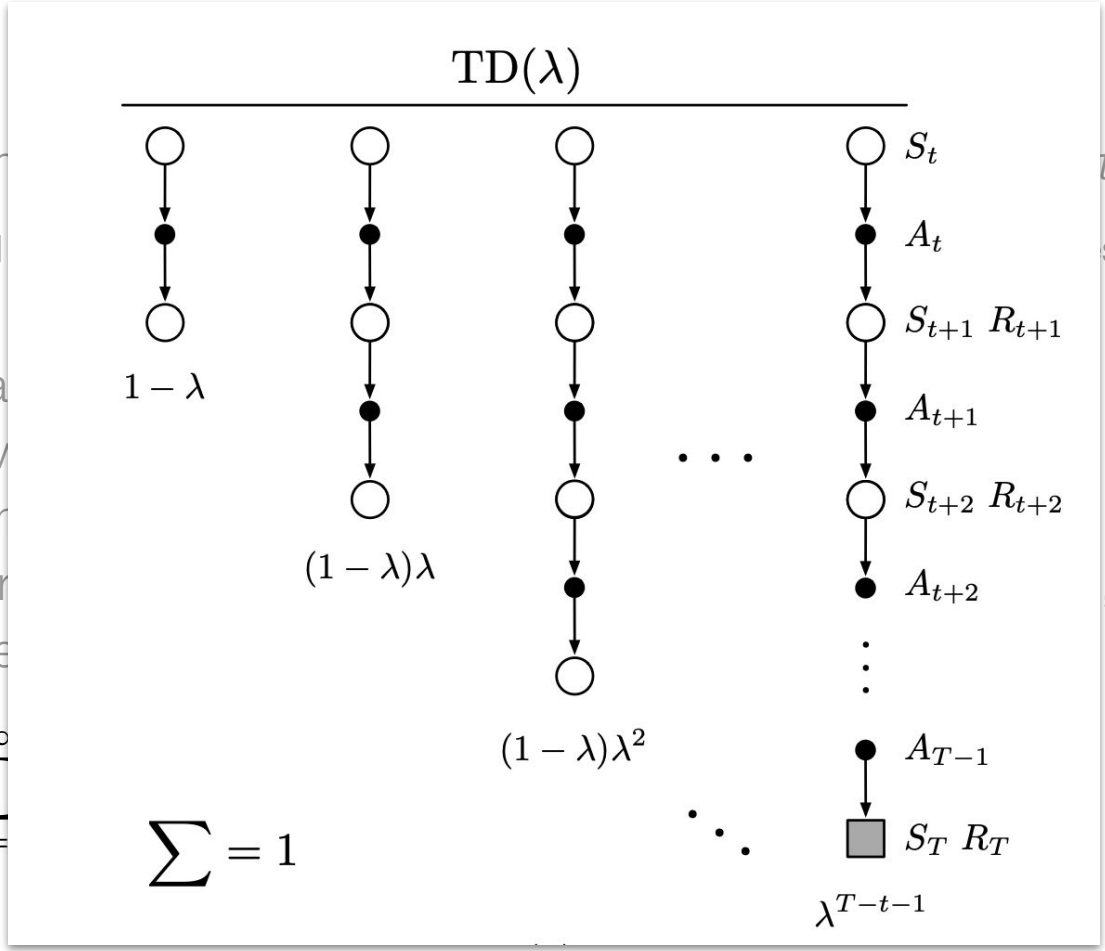
- An update that avg. simpler component updates is called a *compound update*.
  - Obviously, a compound update can only be done when the longest of its component updates is complete.
- "The TD(λ) algorithm can be understood as one particular way of averaging *n*-step updates. This average contains all the n-step updates, each weighted proportionally to λ^{n-1}, and is normalized by a factor of 1-λ to ensure that the weights sum to 1. The resulting update is toward a return, called the λ-return, defined in its state-based form by"

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}.$$

The λ-return

- An update that avg. sim *te*.
  - Obviously, a compound s is complete.
- "The TD(λ) algorithm ca *n*-step updates. This av proportionally to λ^{n-1}, ar weights sum to 1. The r defined in its state-base ,

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=} \infty$$



$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}.$$

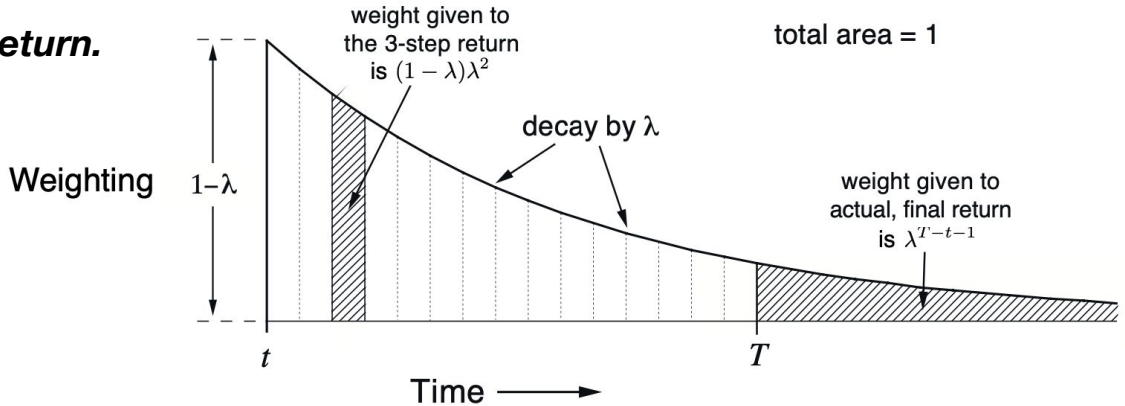
The λ-return

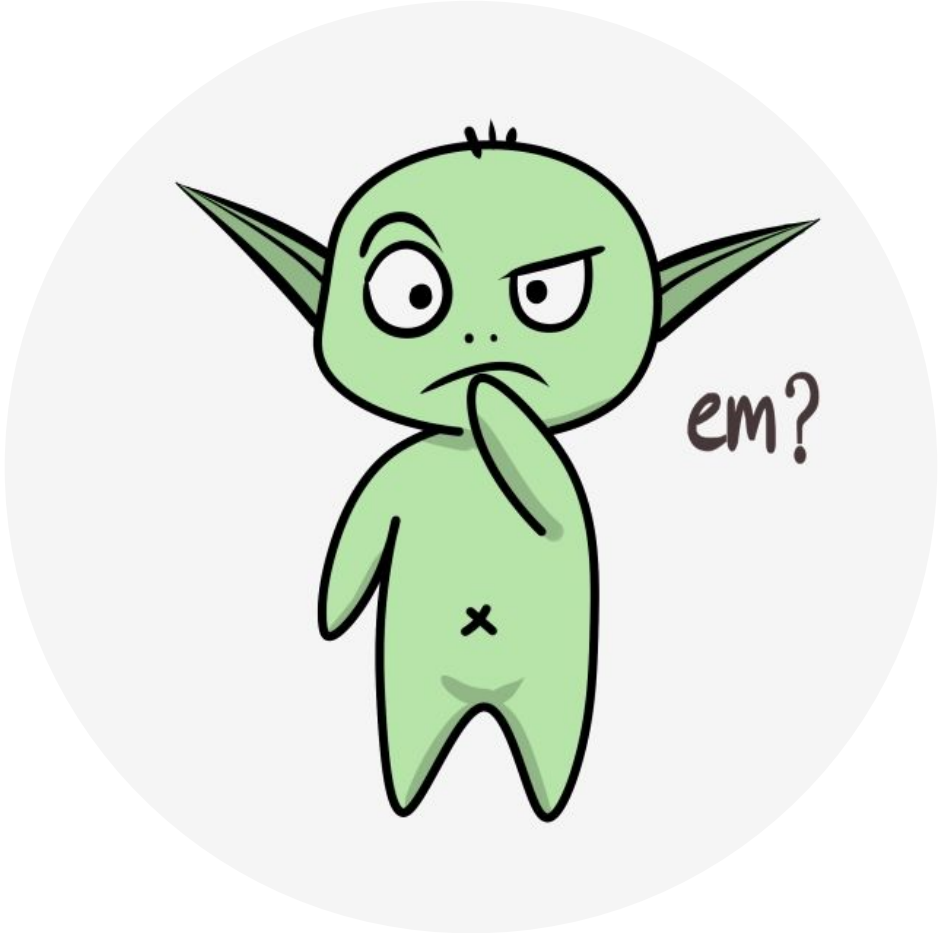
- The weight fades by λ with each additional step. After a terminal state has been reached, all subsequent *n*-step returns are equal to the conventional return, G_t.

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t$$

When λ = 1: The main sum goes to zero, and the remaining term reduces to the conventional return.

When λ = 0: The λ-return reduces to $G_{t:t+1}$, the one-step return.

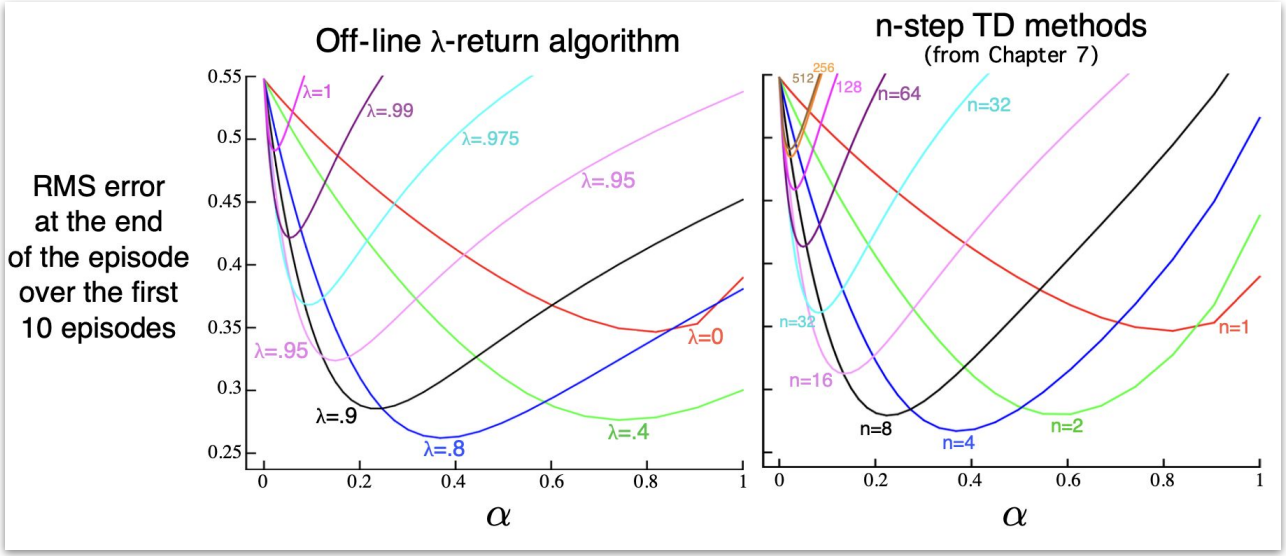




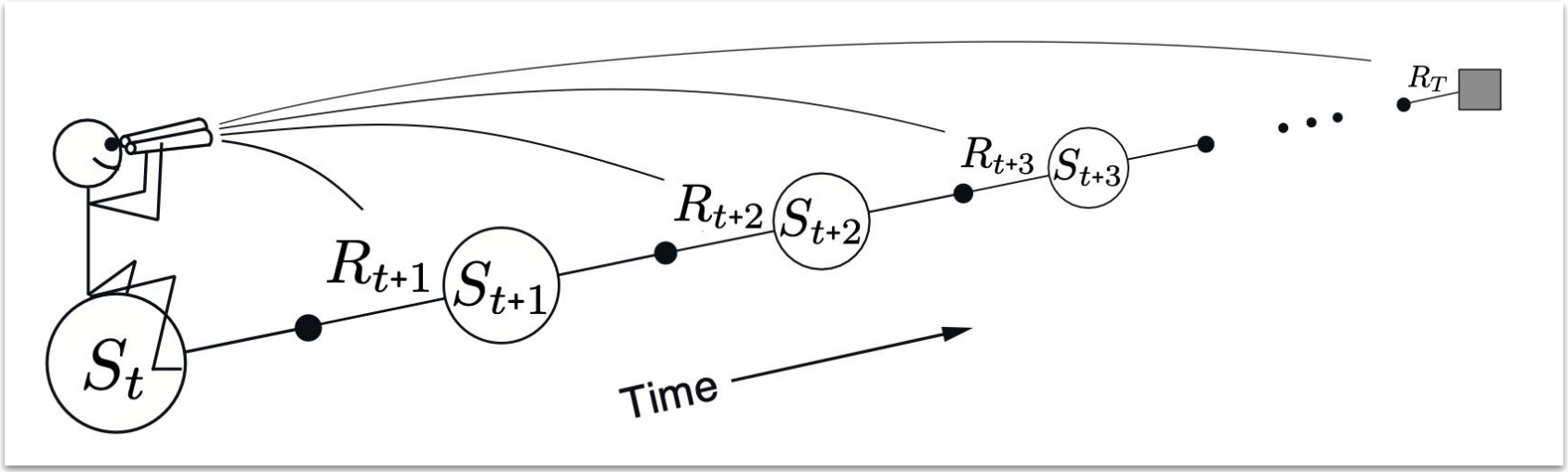
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

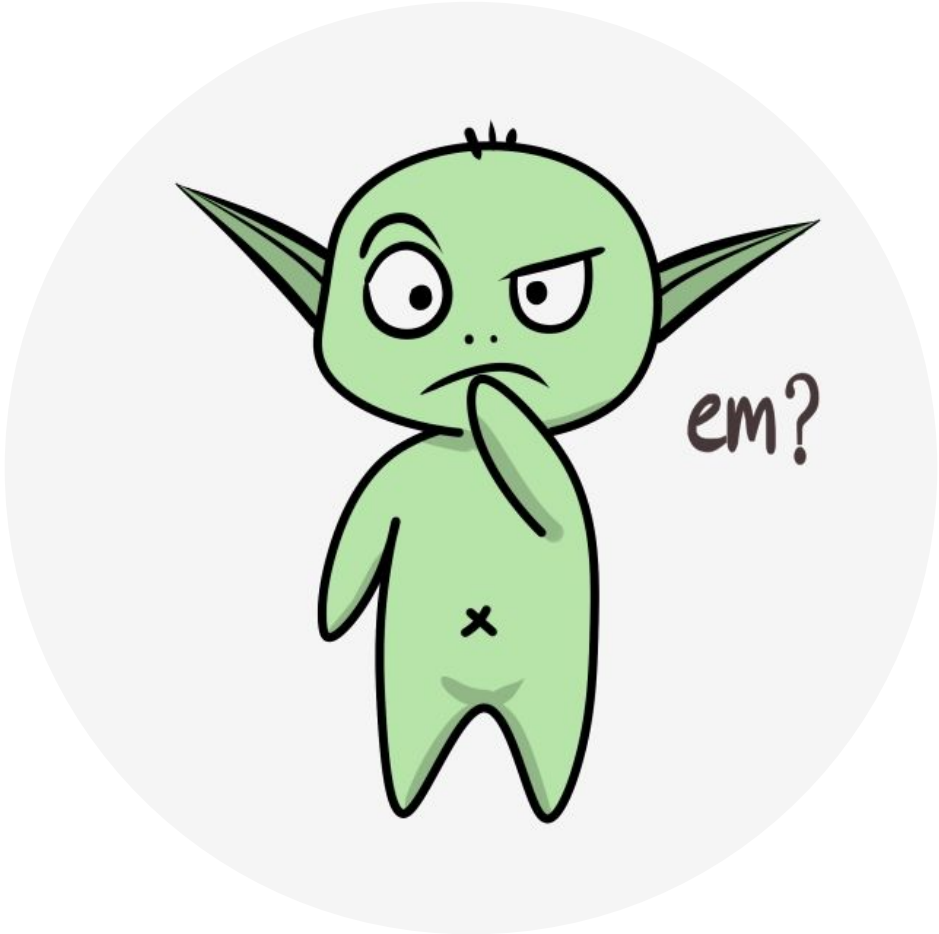
Off-line λ-return Algorithm

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t), \quad t = 0, \ldots, T-1$$



The Forward View





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

TD(λ)

- “TD(λ) is one of the oldest and most widely used algorithms in RL.”
- TD(λ) is better than the off-line λ-return algorithm because:
  a. It updates the weight vector on every step of an episode rather than only at the end.
  b. Its computations are equally distributed in time rather than all at the end of the episode.
  c. It can be applied to continuing problems rather than just episodic problems.

$$\mathbf{z}_{-1} \doteq \mathbf{0},$$
$$\mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t,\mathbf{w}_t), \quad 0 \leq t \leq T$$

$$\delta_t \doteq R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t) - \hat{v}(S_t,\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t$$

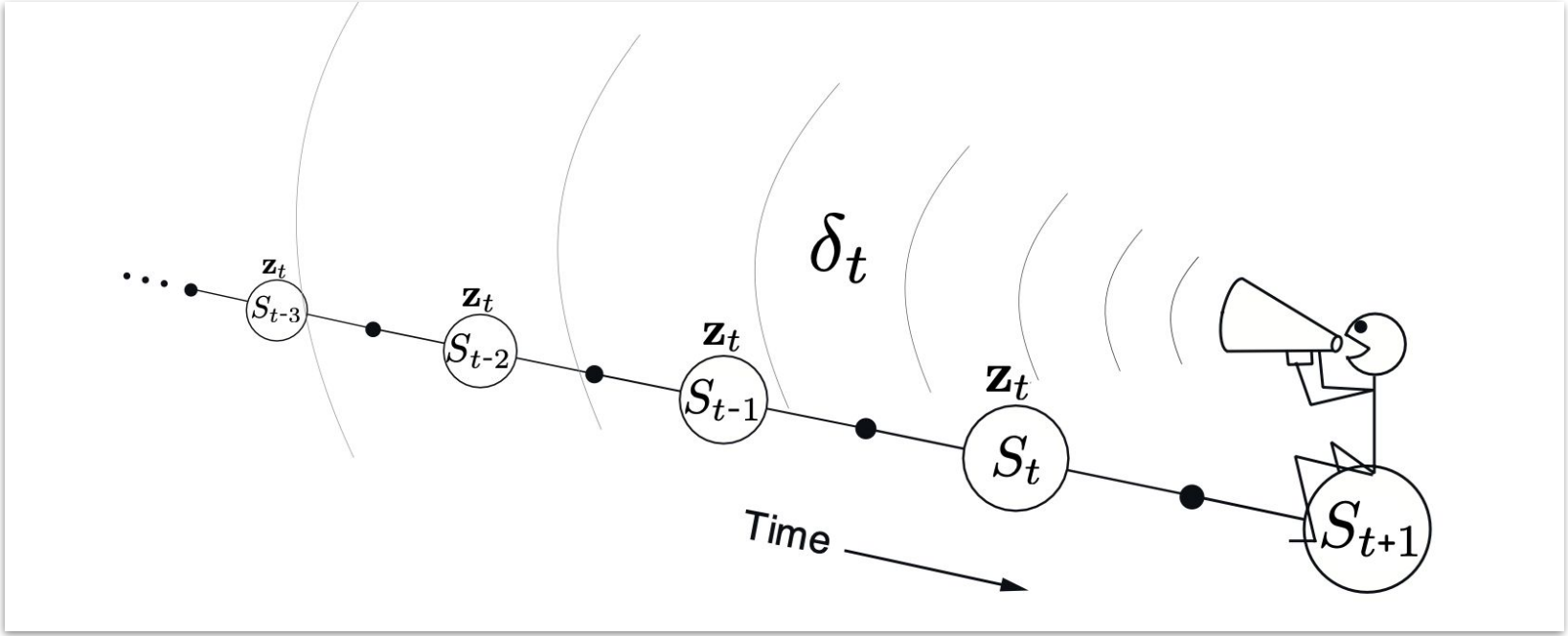
TD(λ)

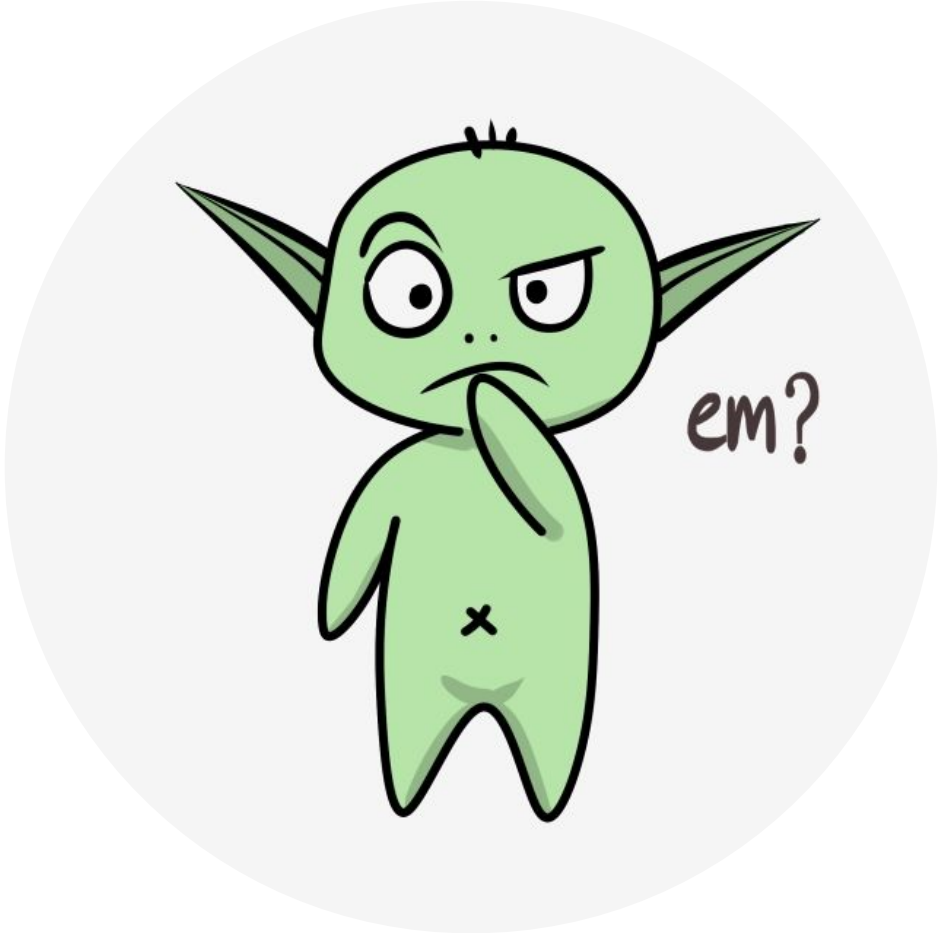
Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
  Loop for each step of episode:
  | Choose $A \sim \pi(\cdot|S)$
  | Take action A, observe R, S'
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S,\mathbf{w})$
  | $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
  | $S \leftarrow S'$
  until S' is terminal

The Backward View





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

Sarsa(λ)

- The action-value form of the off-line λ-return algorithm simply uses *q* instead of *v*:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[G_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t), \quad t = 0, \ldots, T-1$$

- Sarsa(λ) approximates this forward view:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t,$$

$$\delta_t \doteq R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

$$\mathbf{z}_{-1} \doteq \mathbf{0},$$
$$\mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t), \quad 0 \le t \le T.$$

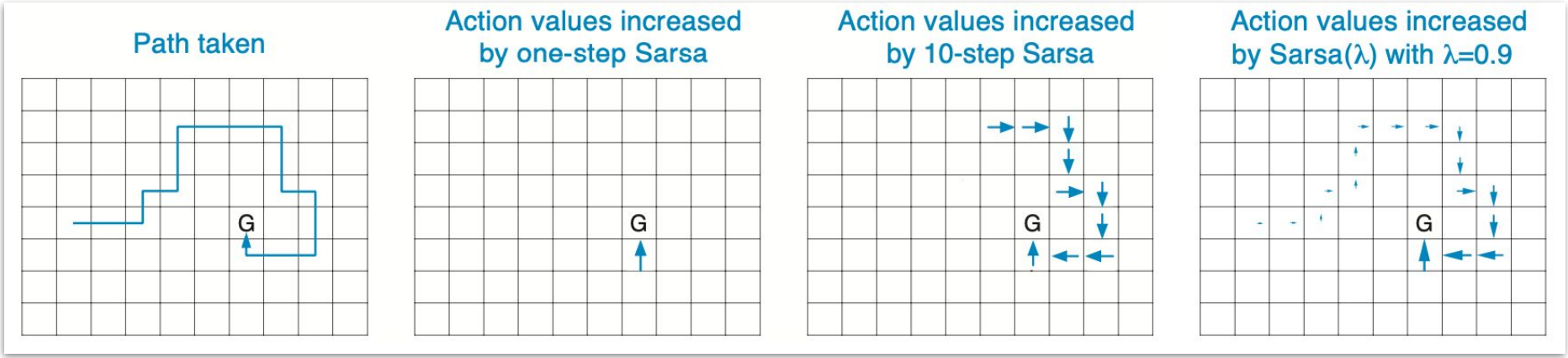
Sarsa(λ)

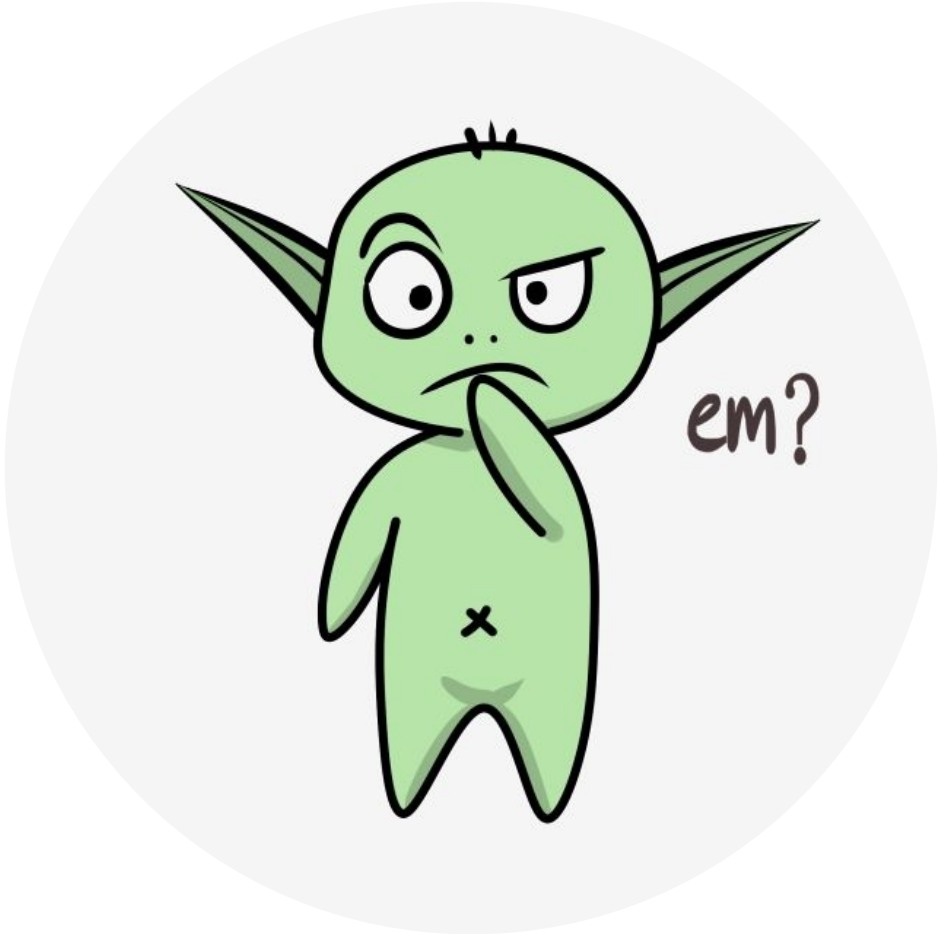
Sarsa(λ) with binary features and linear function approximation for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a function $\mathcal{F}(s,a)$ returning the set of (indices of) active features for s, a
Input: a policy π
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$, small $\varepsilon > 0$
Initialize: $\mathbf{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1, \ldots, z_d)^\top \in \mathbb{R}^d$

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{z} \leftarrow \mathbf{0}$
  Loop for each step of episode:
    Take action A, observe R, S'
    $\delta \leftarrow R$
    Loop for i in $\mathcal{F}(S, A)$:
      $\delta \leftarrow \delta - w_i$
      $z_i \leftarrow z_i + 1$ (accumulating traces)
      or $z_i \leftarrow 1$ (replacing traces)
    If S' is terminal then:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
      Go to next episode
    Choose $A' \sim \pi(\cdot|S')$ or ε-greedy according to $\hat{q}(S', \cdot, \mathbf{w})$
    Loop for i in $\mathcal{F}(S', A')$: $\delta \leftarrow \delta + \gamma w_i$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
    $S \leftarrow S'; A \leftarrow A'$

Traces in Gridworld

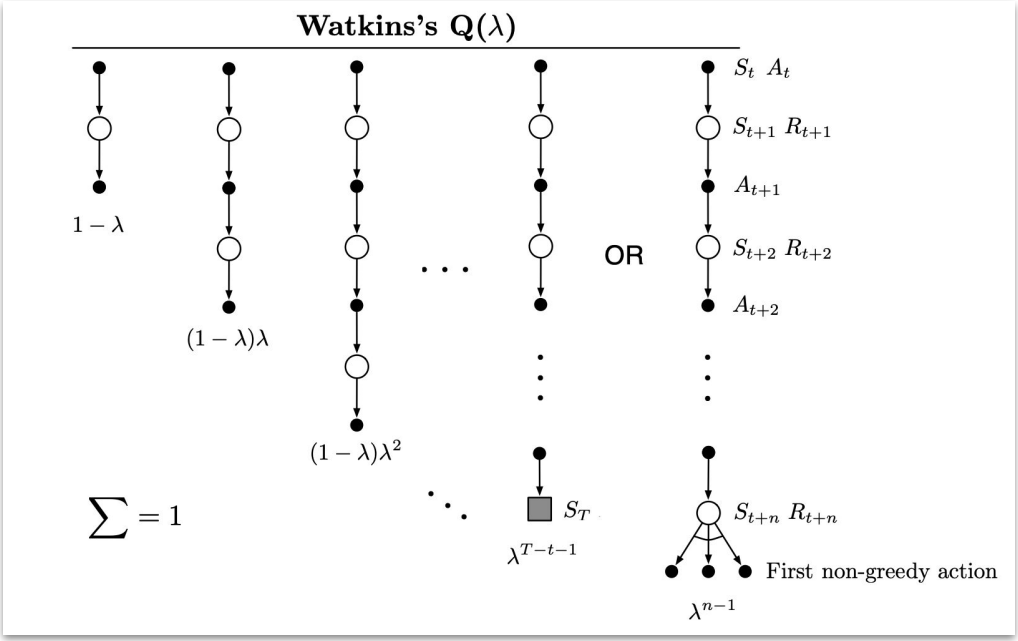


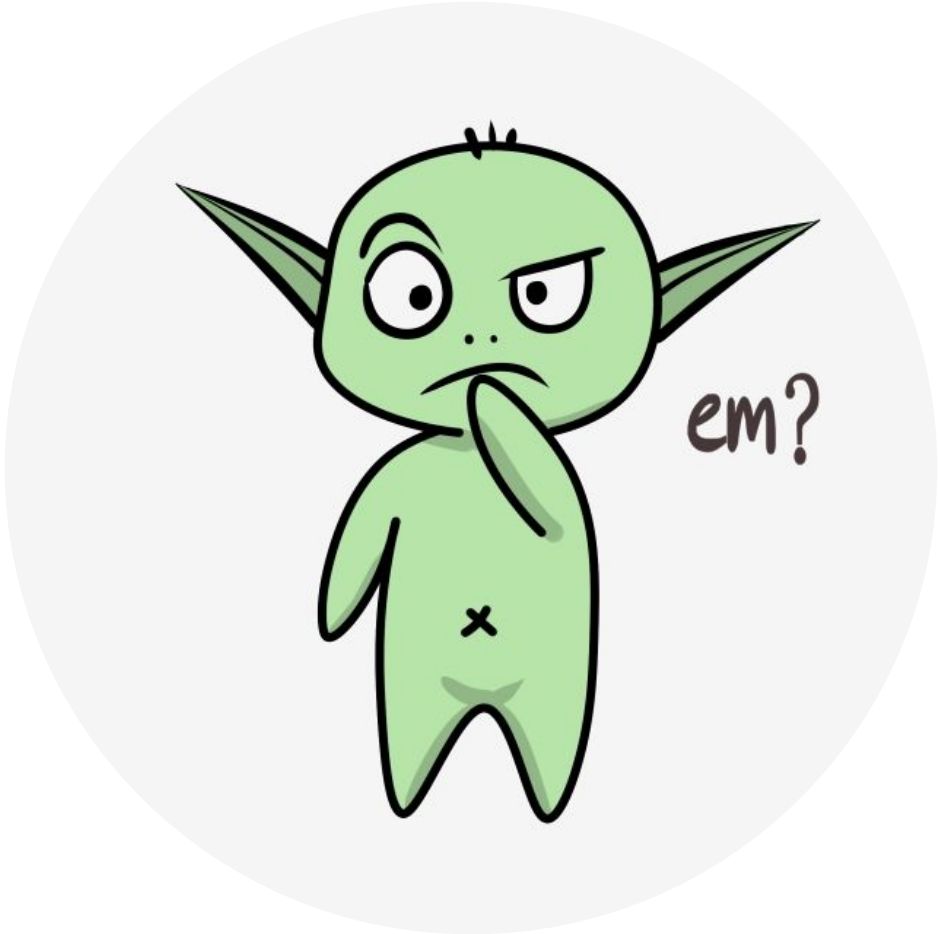


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Watkin's Q(λ)

- It decays the eligibility traces in the usual way as long as a greedy action was taken, but it then cuts the traces to zero after the first non-greedy action is taken.

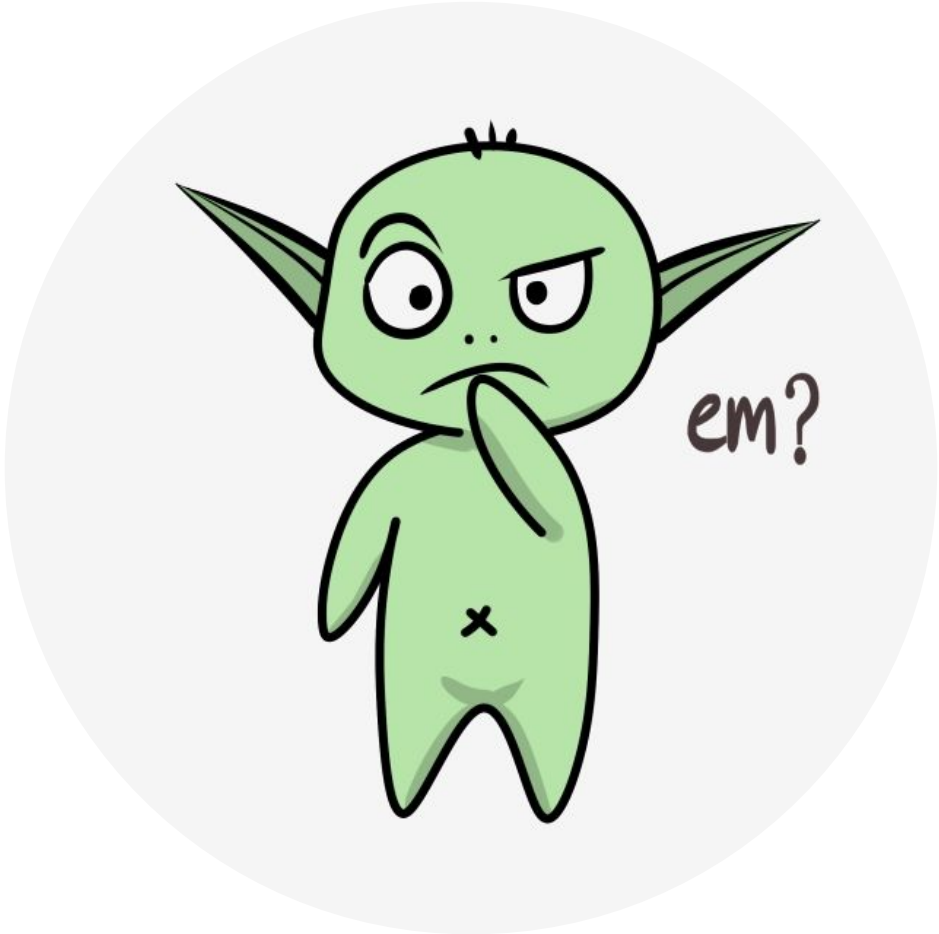




https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Implementation Issues

In practice, then, implementations on conventional computers may keep track of and update only the few traces that are significantly greater than zero. Using this trick, the computational expense of using traces in tabular methods is typically just a few times that of a one-step method. The exact multiple of course depends on λ and γ and on the expense of the other computations. Note that the tabular case is in some sense the worst case for the computational complexity of eligibility traces. When function approximation is used, the computational advantages of not using traces generally decrease. For example, if ANNs and backpropagation are used, then eligibility traces generally cause only a doubling of the required memory and computation per step. Truncated λ-return methods (Section 12.3) can be computationally efficient on conventional computers though they always require some additional memory.



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