Reinforcement Learning

Eligibility Traces

U Kang
Seoul National University

In This Lecture

- Eligibility traces

Overview

- Eligibility traces are one of the basic mechanisms of RL
  - E.g., in the TD(λ) algorithm, the λ refers to the use of an eligibility trace
- Almost any 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 unify and generalize TD and MC methods
- When TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has MC methods at one end (λ=1) and one-step TD methods at the other (λ=0)
- In between are intermediate methods that are often better than either extreme method
- Eligibility traces also provide a way of implementing MC methods online and on continuing problems without episodes

Overview

- In fact, the n-step TD methods also unify TD and MC methods
- What eligibility traces offer beyond these is an elegant algorithmic mechanism with significant computational advantages
- The mechanism is a short-term memory vector, the eligibility trace $z_t \in R^d$, that parallels the long-term weight vector $w_t \in R^d$
- The rough idea is that when a component of w_t participates in producing an estimated value, then the corresponding component of z_t is bumped up and then begins to fade away
- Learning will then occur in that component of w_t if a nonzero TD error occurs before the trace falls back to zero
- The trace-decay parameter $\lambda \in [0,1]$ determines the rate at which the trace falls

Overview

- The primary computational advantage of eligibility traces over n-step methods is that only a single trace vector is required rather than the last n feature vectors
- Learning also occurs continually and uniformly in time rather than being delayed and then catching up at the end of the episode
- In addition learning can occur and affect behavior immediately after a state is encountered rather than being delayed n steps

Overview

- Eligibility traces illustrate that a learning algorithm can sometimes be implemented in a different way to obtain computational advantages
- Many algorithms are most naturally formulated and understood as an update of a state's value based on events that follow that state over multiple future time steps (e.g., MC and TD)
- Such formulations, based on looking forward from the updated state, are called *forward views*
- Forward views are always somewhat complex to implement because the update depends on later things that are not available at the time
- However, it is often possible to achieve nearly the same updates—and sometimes exactly the same updates—with an algorithm that uses the current TD error, looking backward to recently visited states using an eligibility trace
- These alternative ways of looking at and implementing learning algorithms are called *backward views*

Outline

- **The λ-return**
- TD(λ)
- Conclusion

The λ-return

- An n-step return is the sum of the first n rewards plus the estimated value of the state reached in n steps, each appropriately discounted

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

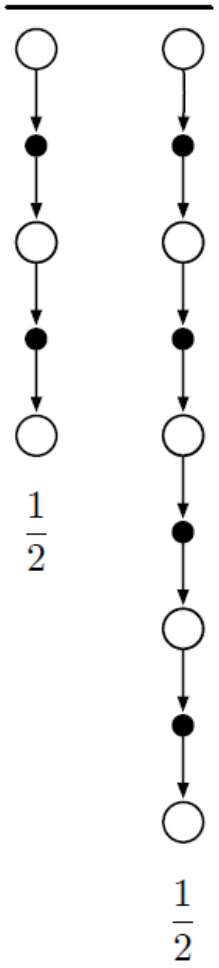
- $\hat{v}(s, w)$ is the approximate value of state s given weight vector w, and T is the time of episode termination, if any

The λ-return

- Note that a valid update can be done not just toward any n-step return, but toward any average of n-step returns for different ns
- For example, an update can be done toward a target that is half of a two-step return and half of a four-step return: $\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$
- Any set of n-step returns can be averaged in this way, as long as the weights on the component returns are positive and sum to 1
- The composite return possesses an error reduction property similar to that of individual n-step returns and thus can be used to construct updates with guaranteed convergence properties
- Averaging produces a substantial new range of algorithms
  - E.g., one could average one-step and infinite-step returns to obtain another way of interrelating TD and MC methods
- One could even average experience-based updates with DP updates to get a simple combination of experience-based and model-based methods

The λ-return

- An update that averages simpler component updates is called a *compound update*
- The backup diagram for a compound update consists of the backup diagrams for each of the component updates with a horizontal line above them and the weighting fractions below
- A compound update can only be done when the longest of its component updates is complete

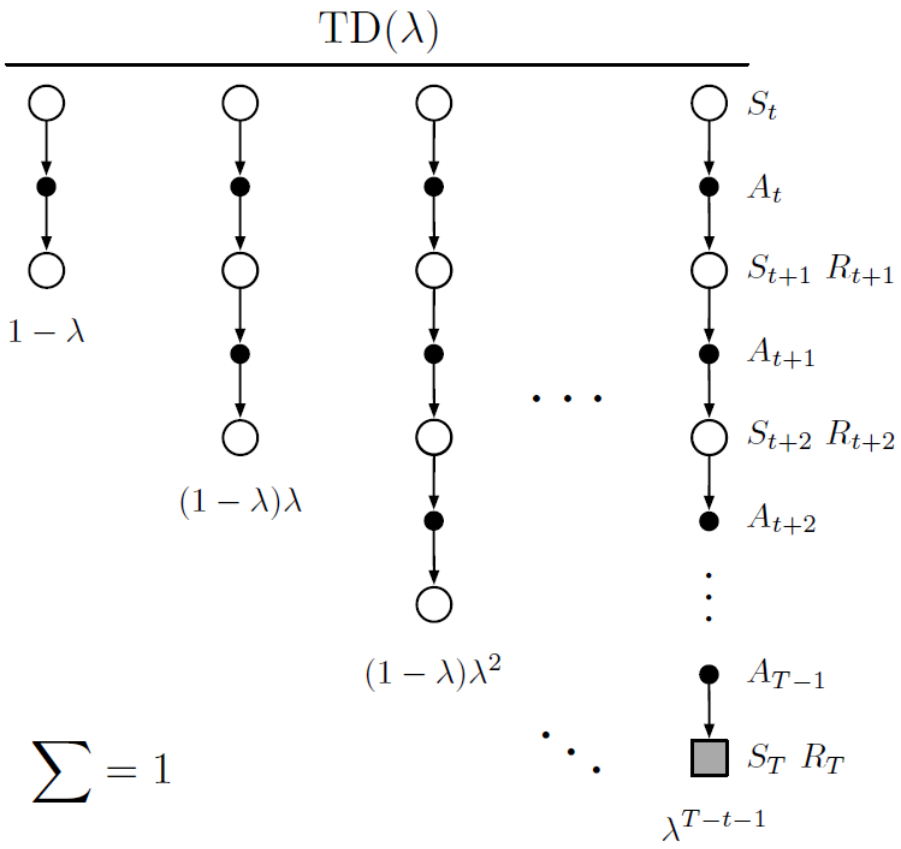


Sutton and Barto, Reinforcement Learning, 2018

The λ-return

- The λ-return algorithm can be understood as a way of averaging n-step updates
- This average contains all the n-step updates, each weighted proportionally to λ^{n-1} (where $\lambda \in [0,1]$), and is normalized by a factor of $1-\lambda$ 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

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

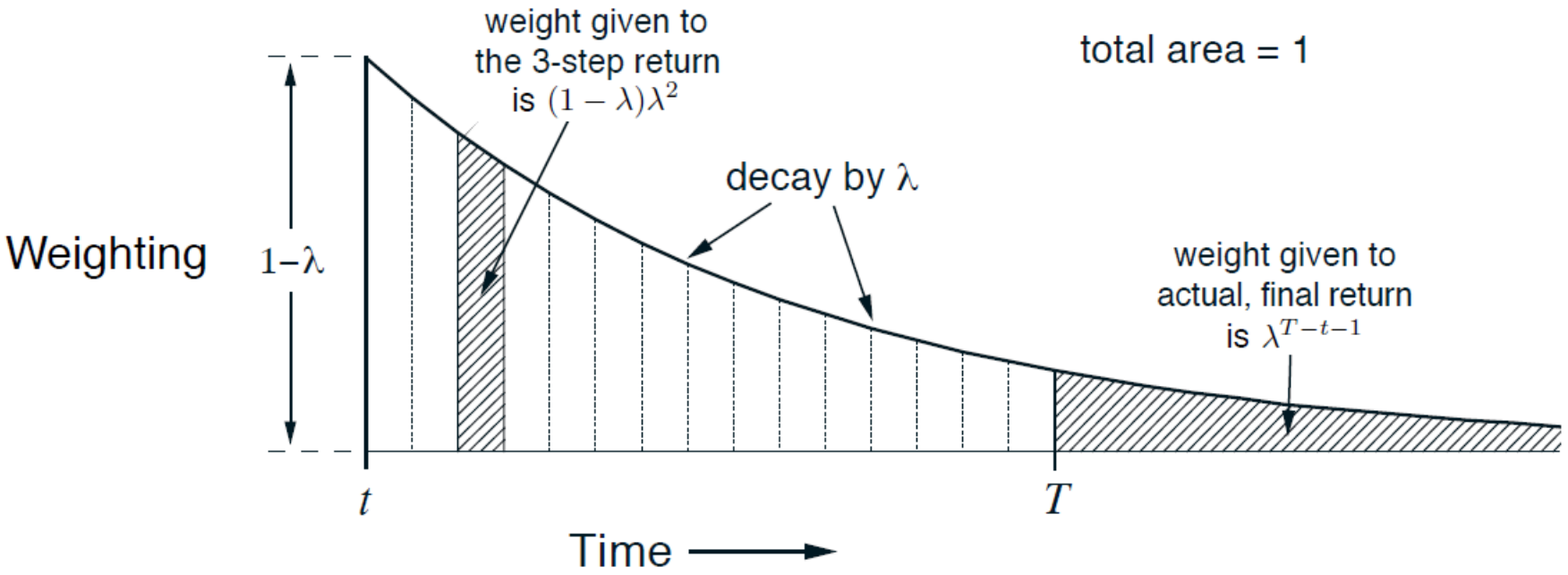


Sutton and Barto, Reinforcement Learning, 2018

The λ-return

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

Sutton and Barto, Reinforcement Learning, 2018



- If we want, we can separate these post-termination terms from the main sum, yielding

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

The λ-return

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

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

- Thus, for $\lambda = 1$, updating according to the λ-return is a MC algorithm
- On the other hand, if λ = 0, then the λ-return reduces to $G_{t:t+1}$, the one-step return. Thus, for λ = 0, updating according to the λ-return is a one-step TD method

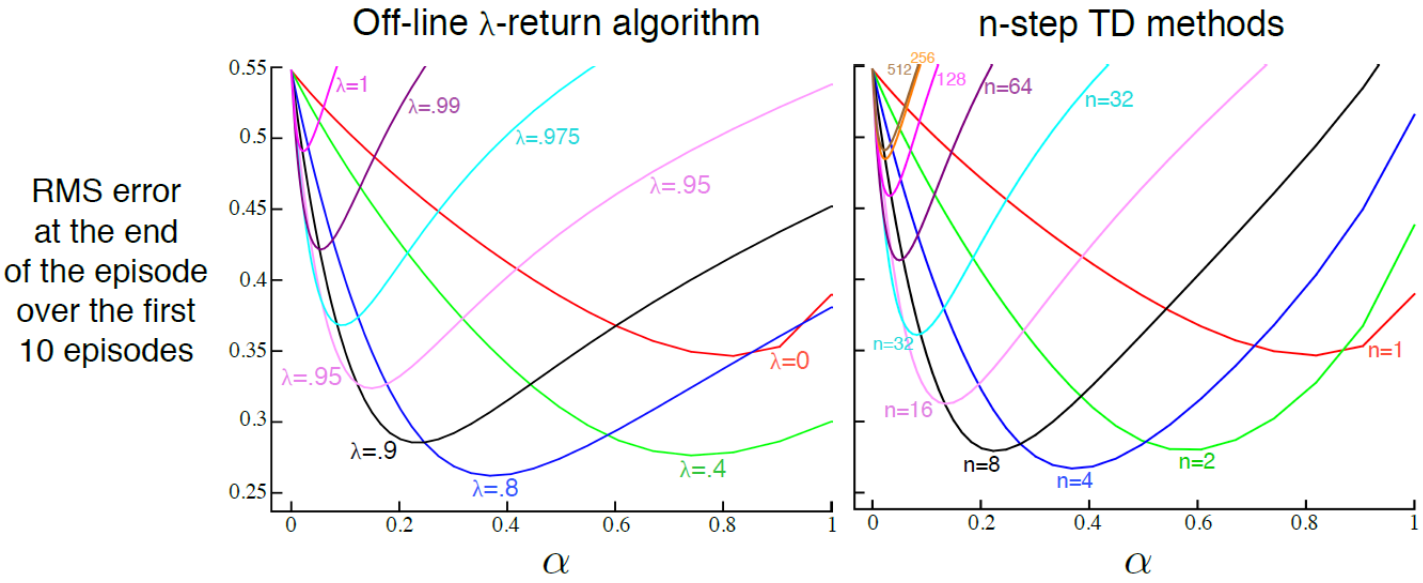
The λ-return

- Offline λ-return algorithm
  - It is based on the λ-return
  - As an offline algorithm, it makes no changes to the weight vector during the episode
  - Then, at the end of the episode, a whole sequence of offline updates are made according to our usual semi-gradient rule, using the λ-return as the target

$$w_{t+1} \doteq w_t + \alpha\left[G_t^\lambda - \hat{v}(S_t, w_t)\right]\nabla\hat{v}(S_t, w_t), \qquad t = 0, \ldots, T-1$$

The λ-return

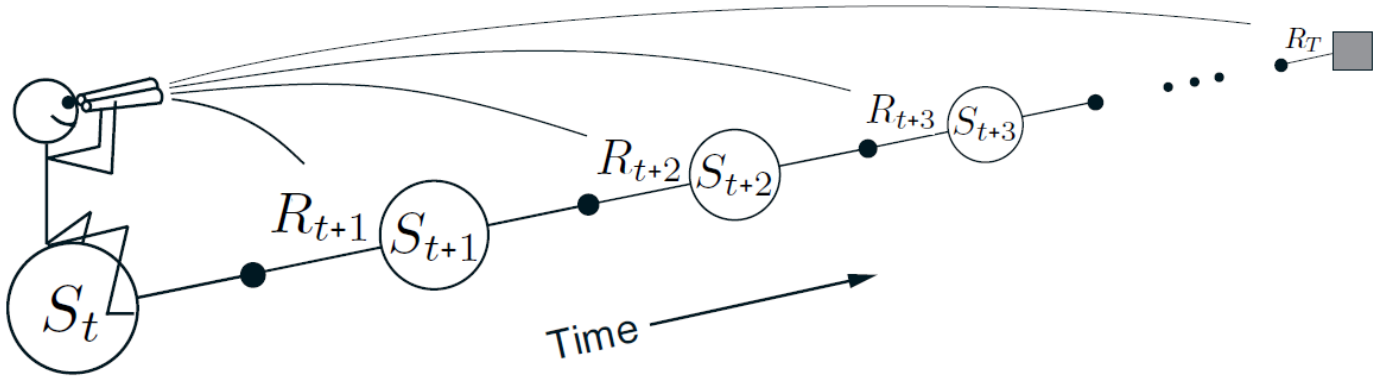
- The λ-return gives us an alternative way of moving smoothly between MC and one-step TD methods that can be compared with the n-step bootstrapping
- The figure shows the comparison on a 19-state random walk task
- Note that overall performance of the offline λ-return algorithms is comparable to that of the n-step algorithms; in both cases we get best performance with an intermediate value of the bootstrapping parameter



Sutton and Barto, Reinforcement Learning, 2018

The λ-return

- The approach that we have been taking so far is what we call *theoretical or forward view* of a learning algorithm
- For each state visited, we look forward in time to all the future rewards and decide how best to combine them
- We might imagine ourselves riding the stream of states, looking forward from each state to determine its update
- After looking forward from and updating one state, we move on to the next and never have to work with the preceding state again
- Future states, on the other hand, are viewed and processed repeatedly, once from each vantage point preceding them



Sutton and Barto, Reinforcement Learning, 2018

Outline

- ☑ The λ-return
- ☐ **TD(λ)**
- ☐ Conclusion

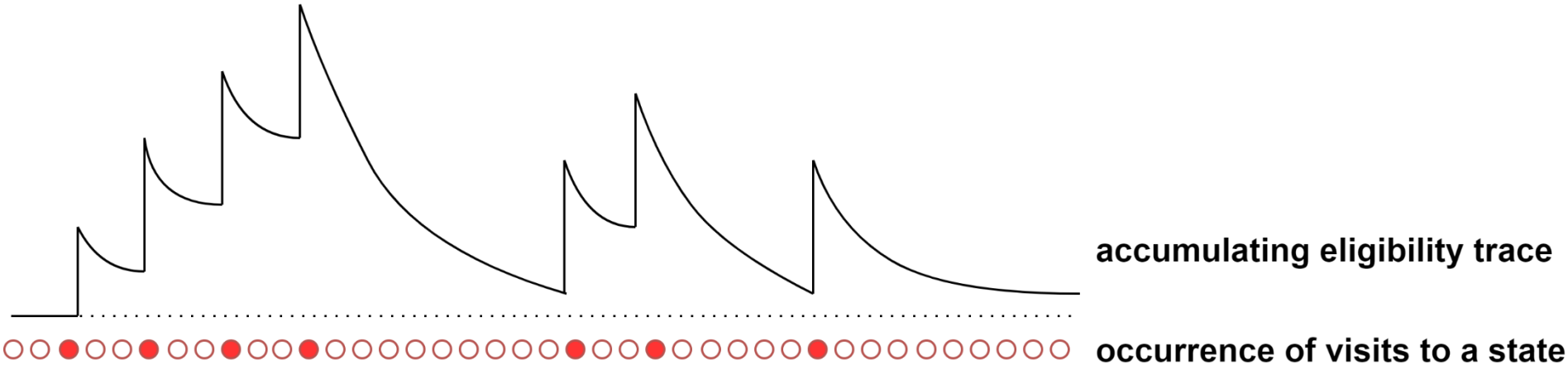
TD(λ)

- TD(λ) is one of the oldest and most widely used algorithms in RL
- It was the first algorithm for which a formal relationship was shown between a more theoretical forward view and a more computationally-congenial backward view using eligibility traces
- We will show empirically that it approximates the offline λ-return algorithm
- TD(λ) improves over the offline λ-return algorithm in three ways
  - 1) It updates the weight vector on every step of an episode rather than only at the end, and thus its estimates may be better sooner
  - 2) Its computations are equally distributed in time rather than all at the end of the episode
  - 3) It can be applied to continuing problems rather than just to episodic problems
- We present the semi-gradient version of TD(λ) with FA

ASIDE: Eligibility Trace for Tabular Setting

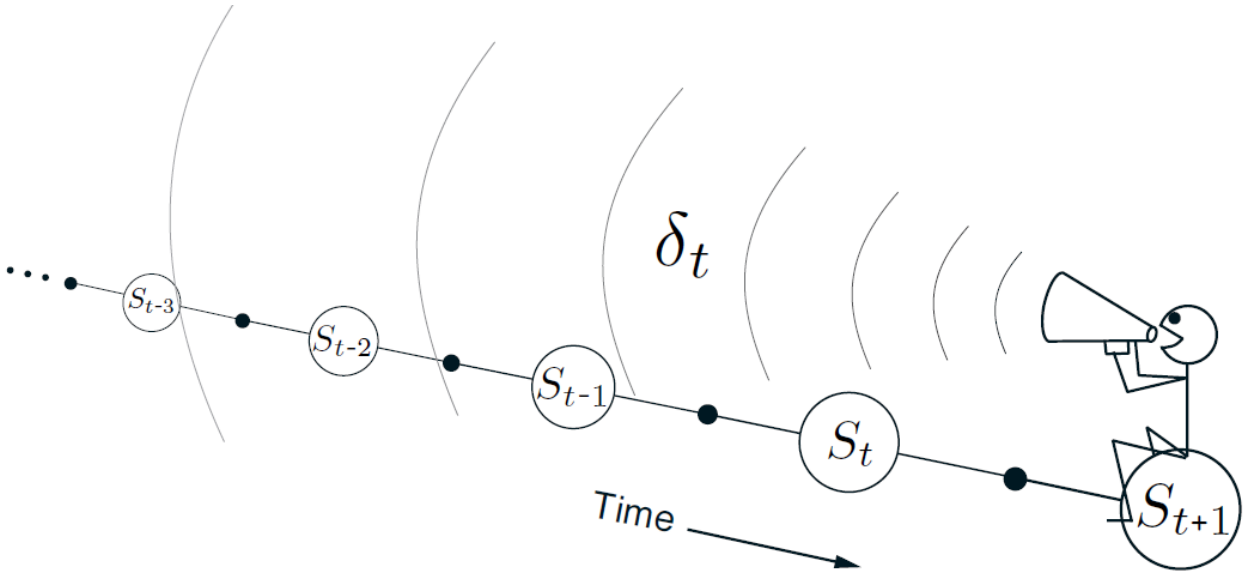
- Which state contributed to a reward?

- $E_0(s) = 0$
- $E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s)$



ASIDE: Eligibility Trace for Tabular Setting

- Keep an eligibility trace for every state s
- Update value V(s) for every state s, in proportion to TD-error δ_t and eligibility trace $E_t(s)$
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - $V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$



Sutton and Barto, Reinforcement Learning, 2018

ASIDE: Eligibility Trace for Tabular Setting

- Eligibility trace
  - $E_0(s) = 0$
  - $E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s)$
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - $V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$

- When $\lambda = 0$, only the current state is updated
  - $E_t(s) = 1(S_t = s)$
  - $V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$

- This is exactly equivalent to TD(0) update
  - $V(S_t) \leftarrow V(S_t) + \alpha\delta_t$

ASIDE: Eligibility Trace for Tabular Setting

- Eligibility trace
  - $E_0(s) = 0$
  - $E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s)$
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - $V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$

- When $\lambda = 1$, TD(λ) behaves similarly to MC method
- Consider episodic environments
- Over the course of an episode, total update for TD(1) is the same as total update for MC
  - $\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t - V(S_t)\right) 1(S_t = s)$

ASIDE: Eligibility Trace for Tabular Setting

- Theorem: $\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \big(G_t - V(S_t)\big) 1(S_t = s)$
- Proof
  - Consider an episode where s is visited once at time-step k
  - When $\lambda = 1$, TD(λ) eligibility trace is given by

$$E_t(s) = \gamma E_{t-1}(s) + \mathbf{1}(S_t = s)$$
$$= \begin{cases} 0 & if\ t < k \\ \gamma^{t-k} & if\ t \geq k \end{cases}$$

  - By the end of episode, TD(1) accumulates the total error

$$\delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \cdots + \gamma^{T-1-k} \delta_{T-1}$$

ASIDE: Eligibility Trace for Tabular Setting

- Theorem: $\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t - V(S_t) \right) 1(S_t = s)$
- Proof (cont.)

$$
\begin{aligned}
&\delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \cdots + \gamma^{T-1-k} \delta_{T-1} \\
&= R_{k+1} + \gamma V(S_{k+1}) - V(S_k) \\
&+ \gamma R_{k+2} + \gamma^2 V(S_{k+2}) - \gamma V(S_{k+1}) \\
&+ \gamma^2 R_{k+3} + \gamma^3 V(S_{k+3}) - \gamma^2 V(S_{k+2}) \\
&\quad \vdots \\
&+ \gamma^{T-1-k} R_T + \gamma^{T-k} V(S_T) - \gamma^{T-1-k} V(S_{T-1}) \\
&= R_{k+1} + \gamma R_{k+2} + \gamma^2 R_{k+3} + \cdots + \gamma^{T-1-k} R_T - V(S_k) \\
&= G_k - V(S_k)
\end{aligned}
$$

TD(λ)

- With FA, the eligibility trace is a vector $z_t \in R^d$ with the same number of components as the weight vector w_t
- Whereas the weight vector is a long-term memory, accumulating over the lifetime of the system, the eligibility trace is a short-term memory, typically lasting less time than the length of an episode
- Eligibility traces assist in the learning process; their only consequence is that they affect the weight vector, and then the weight vector determines the estimated value
- In TD(λ), the eligibility trace vector is initialized to 0 at the beginning of the episode, is incremented on each time step by the value gradient, and then fades away by $\gamma\lambda$:

$$z_{-1} \doteq \mathbf{0},$$
$$z_t \doteq \gamma\lambda z_{t-1} + \nabla\hat{v}(S_t, w_t), \qquad 0 \le t \le T$$

- where γ is the discount rate and λ is the trace-decay parameter

TD(λ)

$$z_{-1} \doteq \mathbf{0},$$
$$z_t \doteq \gamma\lambda z_{t-1} + \nabla\hat{v}(S_t, w_t), \qquad 0 \leq t \leq T$$

- The eligibility trace keeps track of which components of the weight vector have contributed, positively or negatively, to recent state valuations, where "recent" is defined in terms of $\gamma\lambda$
  - In linear FA, $\nabla\hat{v}(S_t, w_t)$ is just the feature vector x_t in which case the eligibility trace vector is just a sum of past, fading, input vectors
- The trace is said to indicate the eligibility of each component of the weight vector for undergoing learning changes should a reinforcing event occur
- The reinforcing events we are concerned with are the moment-by-moment one-step TD errors

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

- In TD(λ), the weight vector is updated on each step proportional to the scalar TD error and the vector eligibility trace:

$$w_{t+1} = w_t + \alpha\delta_t z_t$$

TD(λ)

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

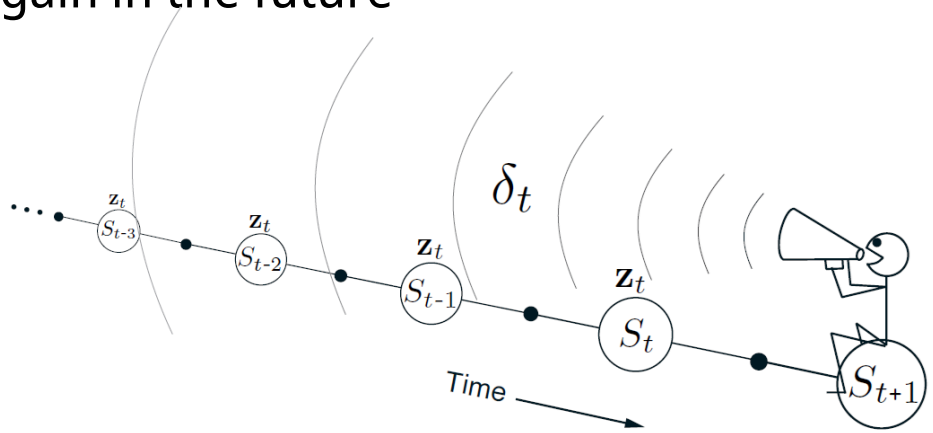
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize value-function weights $\mathbf{w}$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

Sutton and Barto, Reinforcement Learning, 2018

TD(λ)

- TD(λ) is oriented backward in time
- At each moment we look at the current TD error and assign it backward to each prior state according to how much that state contributed to the current eligibility trace at that time
- We might imagine ourselves riding along the stream of states, computing TD errors, and shouting them back to the previously visited states
- Where the TD error and traces come together, we get the update given by $w_{t+1} = w_t + \alpha\delta_t z_t$, changing the values of those past states for when they occur again in the future



Sutton and Barto, Reinforcement Learning, 2018

TD(λ)

- To better understand the backward view of TD(λ), consider what happens at various values of λ

$$z_{-1} \doteq \mathbf{0},$$
$$z_t \doteq \gamma\lambda z_{t-1} + \nabla\hat{v}(S_t, w_t), \qquad 0 \leq t \leq T$$

- If λ = 0, the trace at t is exactly the value gradient corresponding to S_t
- Thus the TD(λ) update $w_{t+1} = w_t + \alpha\delta_t z_t$ reduces to the one-step semi-gradient TD update (and, in the tabular case, to the simple TD rule)
- This is why that algorithm was called TD(0)
- TD(0) is the case in which only the one state preceding the current one is changed by the TD error
- For larger values of λ, but still $\lambda < 1$, more of the preceding states are changed, but each more temporally distant state is changed less because the corresponding eligibility trace is smaller
- We say that the earlier states are given less *credit* for the TD error

TD(λ)

- If λ = 1, then the credit given to earlier states falls only by γ per step
- This turns out to be just the right thing to do to achieve MC behavior
- For example, remember that the TD error δ_t includes an undiscounted term of R_{t+1}
- In passing this back k steps it needs to be discounted, like any reward in a return, by γ^k, which is just what the falling eligibility trace achieves

- If λ = 1 and γ = 1, then the eligibility traces do not decay at all with time
- In this case the method behaves like an MC method for an undiscounted, episodic task
- If λ = 1, the algorithm is also known as TD(1)

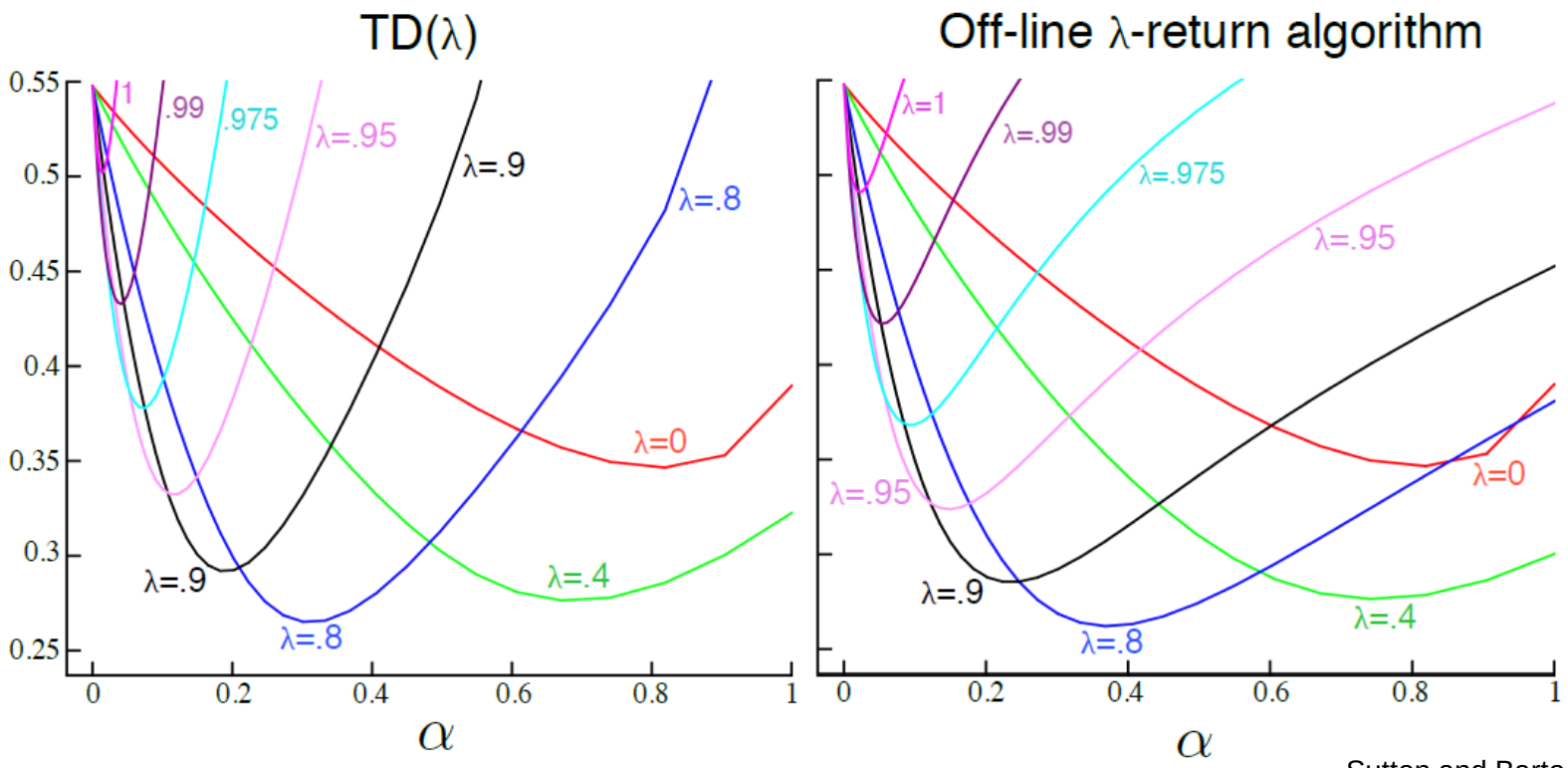
TD(λ)

- TD(1) is a way of implementing MC algorithms that is more general and that significantly increases their range of applicability
- Whereas the earlier MC methods were limited to episodic tasks, TD(1) can be applied to discounted continuing tasks as well
- Moreover, TD(1) can be performed incrementally and online
- One disadvantage of MC methods is that they learn nothing from an episode until it is over; e.g., if an MC control method takes an action that produces a very poor reward but does not end the episode, then the agent's tendency to repeat the action will be undiminished during the episode
- Online TD(1), on the other hand, learns in an n-step TD way from the incomplete ongoing episode, where the n steps are all the way up to the current step
- If something unusually good or bad happens during an episode, control methods based on TD(1) can learn immediately

TD(λ)

- The figure shows the performance of TD(λ) on the 19-state random walk
- The two algorithms perform virtually identically

RMS error at the end of the episode over the first 10 episodes



U Kang

Sutton and Barto, Reinforcement Learning, 2018

Outline

- ☑ The λ-return
- ☑ TD(λ)
- ☐ **Conclusion**

Conclusion

- Eligibility traces in conjunction with TD errors provide an efficient, incremental way of shifting and choosing between MC and TD methods
- The n-step methods also enabled this, but eligibility trace methods are more general, often faster to learn, and offer different computational complexity tradeoff
- Eligibility traces enable more efficient and incremental backward-view algorithms compared to intuitive forward-view methods

Conclusion

- MC methods may have advantages in non-Markov tasks because they do not bootstrap
  - I.e., MC methods do not update their value estimates on the basis of those of successor states
- Because eligibility traces make TD methods more like MC methods, they also can have advantages in these cases
- If one wants to use TD methods because of their other advantages, but the task is at least partially non-Markov, then the use of an eligibility trace method is desired
- Eligibility traces are the first line of defense against both long-delayed rewards and non-Markov tasks

Conclusion

- By adjusting λ, we can place eligibility trace methods anywhere along a continuum from MC to one-step TD methods
- An intermediate mixture appears to be the best choice; eligibility traces should be used to bring us toward MC methods, but not all the way there

Conclusion

- Methods using eligibility traces require more computation than one-step methods, but in return they offer significantly faster learning, particularly when rewards are delayed by many steps
- Thus it often makes sense to use eligibility traces in online applications where data are scarce and cannot be repeatedly processed
- On the other hand, in offline applications in which data can be generated cheaply, perhaps from an inexpensive simulation, then it often does not pay to use eligibility traces
- In these cases the objective is not to get more out of a limited amount of data, but simply to process as much data as possible as quickly as possible
- In these cases the speedup per datum due to eligibility traces is typically not worth their computational cost

Questions?