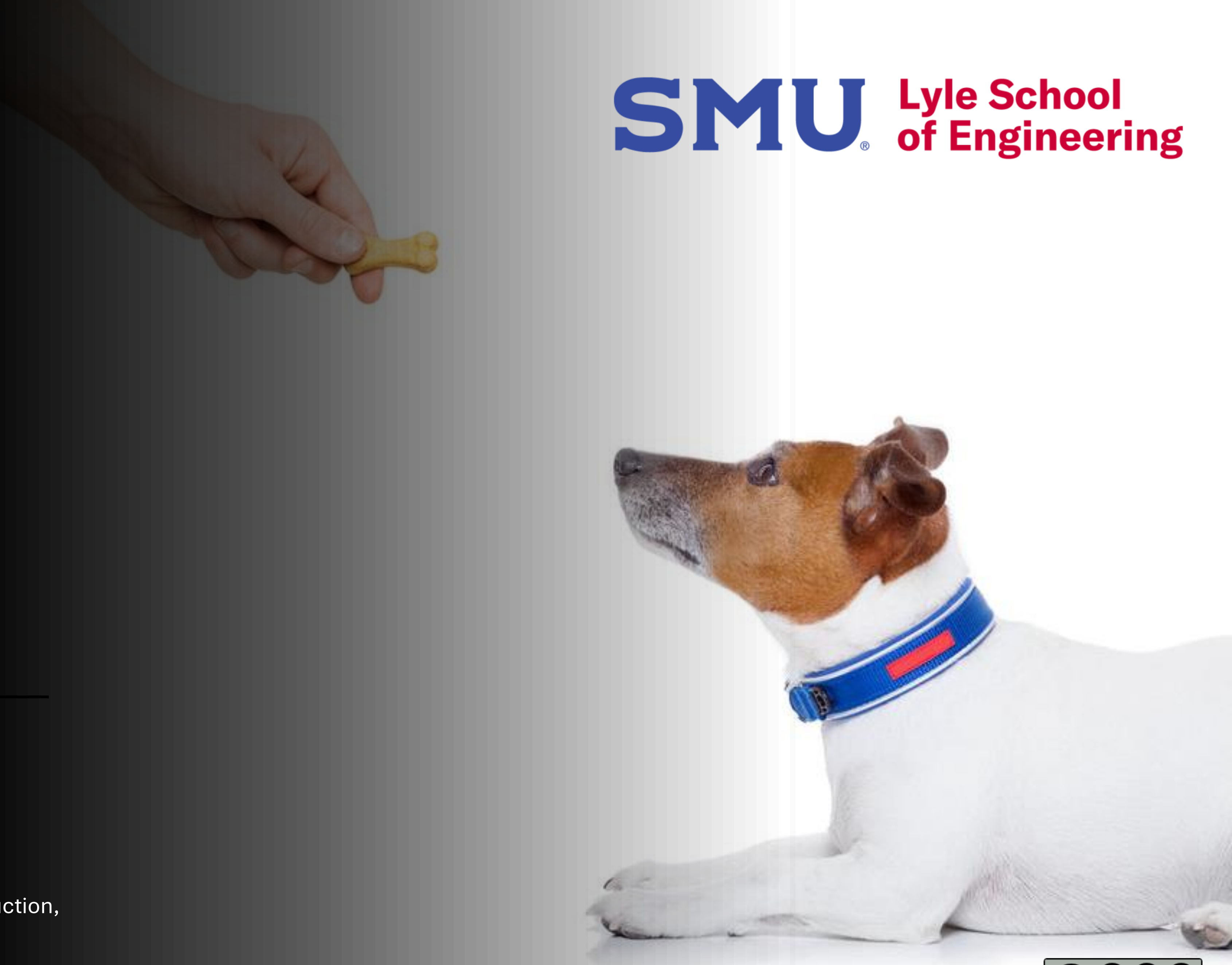
SMU Lyle School of Engineering

Reinforcement Learning

Eligibility Traces

Sutton/Barto* Chapter 12



Michael Hahsler, SMU

With figures from Sutton/Barto*

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

Topics of this Course

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

Summary of Notation

General

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

Value Function

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

MDP

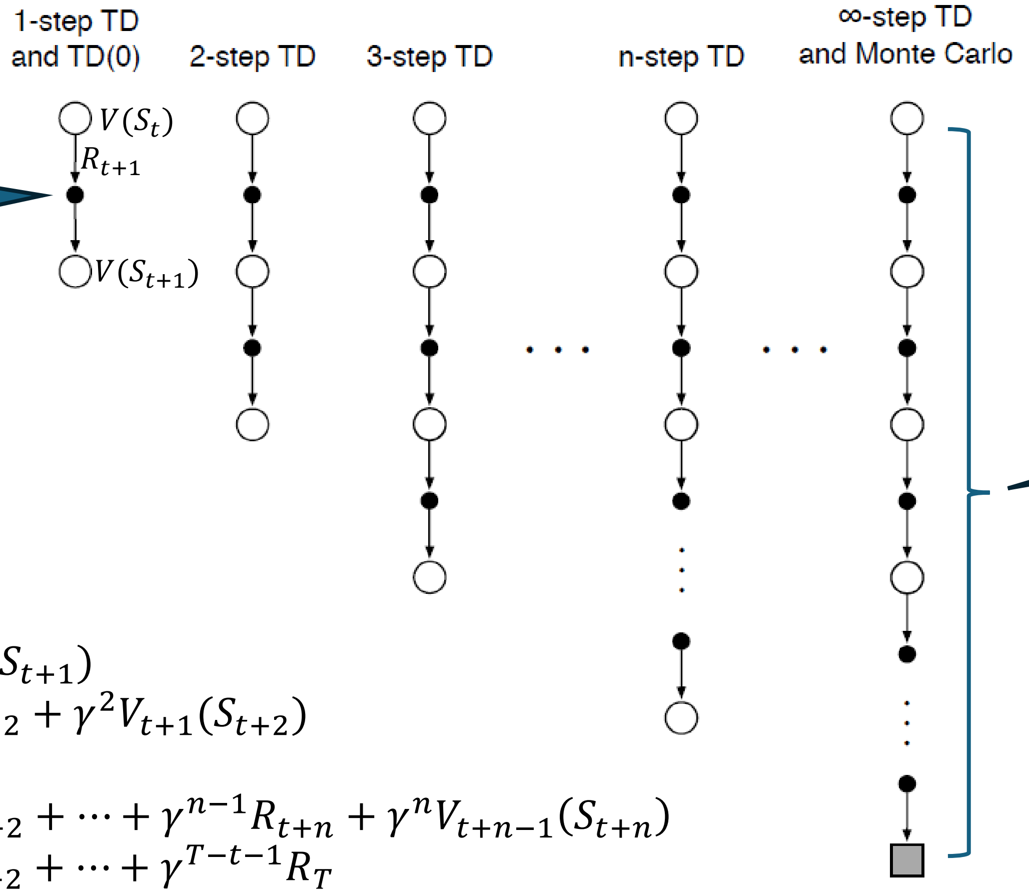
s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all (nonterminal) states, $\mathcal{S}^+$ are all states
$\mathcal{A}(s)$	set of all actions available in state s
γ	discount-rate parameter
t	discrete time step
T	final time step of an episode (a.k.a. horizon)
A_t	random variable for the action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

The λ-Return

Performing updates with compound return targets

Remember Chapter 7: n-step TD Prediction

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



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

Return

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

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

…

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

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

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

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

Question: The implementation drops the subscript for V!

Notation

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

Valid n-step Update Targets

- n-step return from Chapter 7 with approximation:

Episode termination

$$G_{t:t+n} = \underbrace{R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}}_{n \text{ next rewards}} + \underbrace{\gamma^{n} \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1})}_{\text{approx. value}}, \qquad 0 \leq t \leq T - n$$

- Any n-step return $G_{t:t+n}$ is a valid update target for tabular learning and approx. SGD since it has the error reduction property.

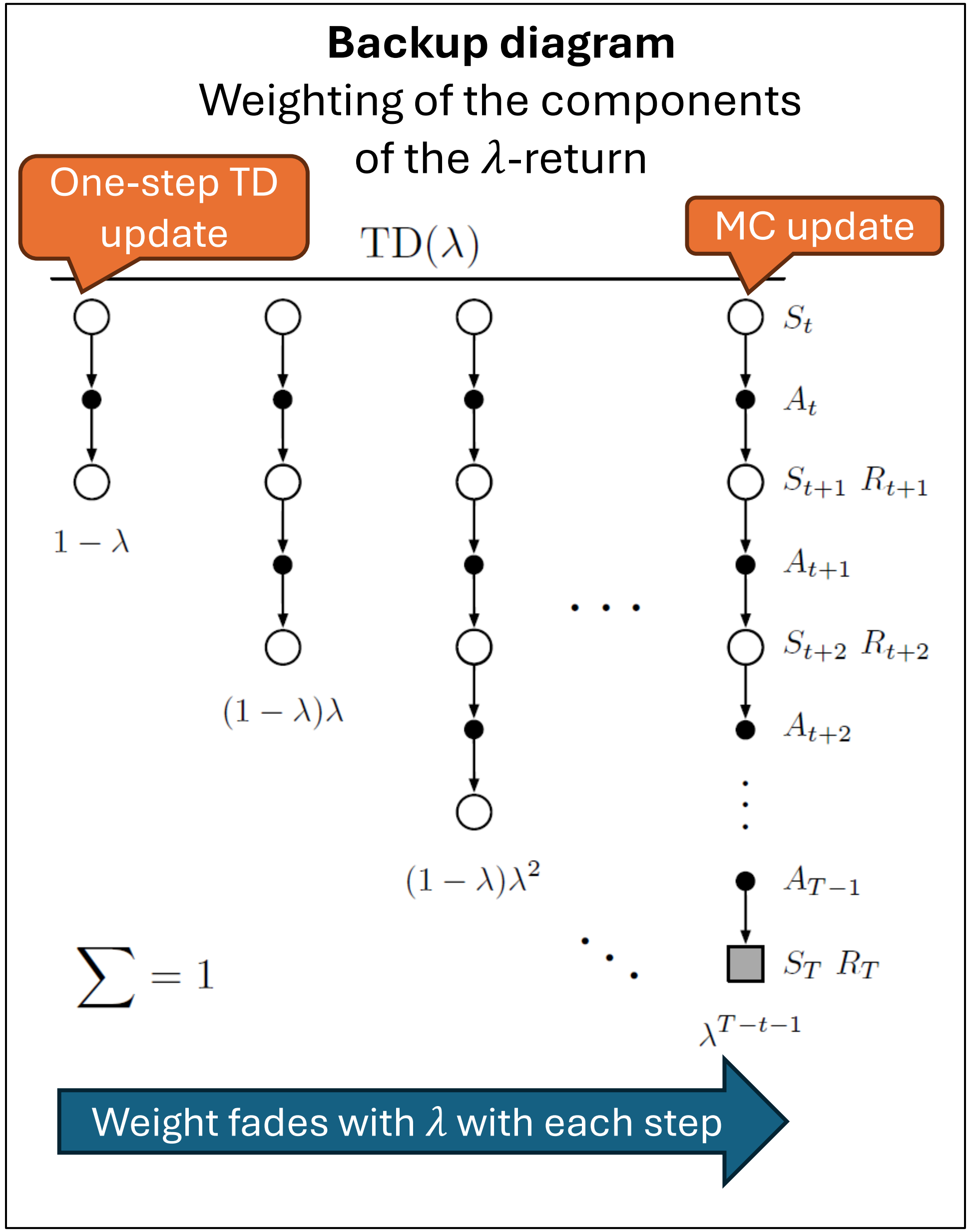
- **Observation:** any average of different n-step return is also valid. E.g., average 2-step and 4-step returns: $\frac{1}{2} G_{t:t+2} + \frac{1}{2} G_{t:t+4}$ Updating with an average of n-step returns is called a compound update.

TD(λ) and the λ-Return

- TD(λ) uses a compound return called the ***λ*-return** as the update target:

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

- Special cases:
  - **TD(0)** with $\lambda = 0$ uses only first component and is a one-step TD update.
  - **TD(1)** with $\lambda = 1$ uses only the last component and is a MC update.



Backup diagram

Weighting of the components of the λ-return

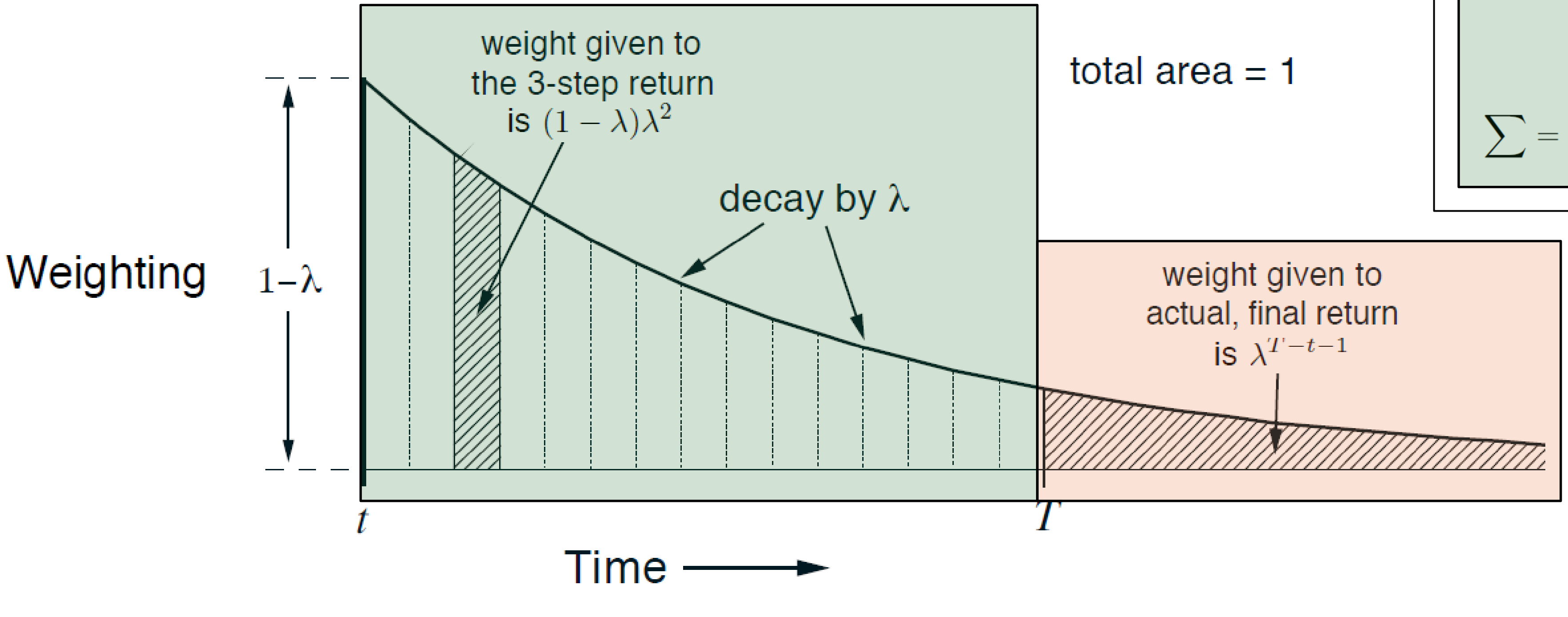
One-step TD update

MC update

Weight fades with λ with each step

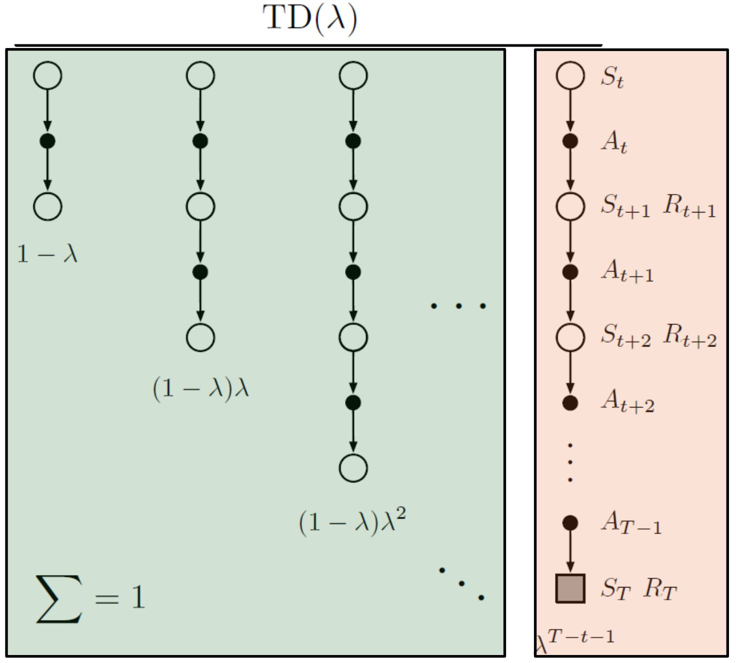
Exponential Weighting Scheme

$$G_t^{\lambda} = (1-\lambda)\underbrace{\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n}}_{n\text{-step return}} + \lambda^{T-t-1}\underbrace{G_t}_{\text{MC return}}$$



Backup diagram

Weighting of the components of the λ-return



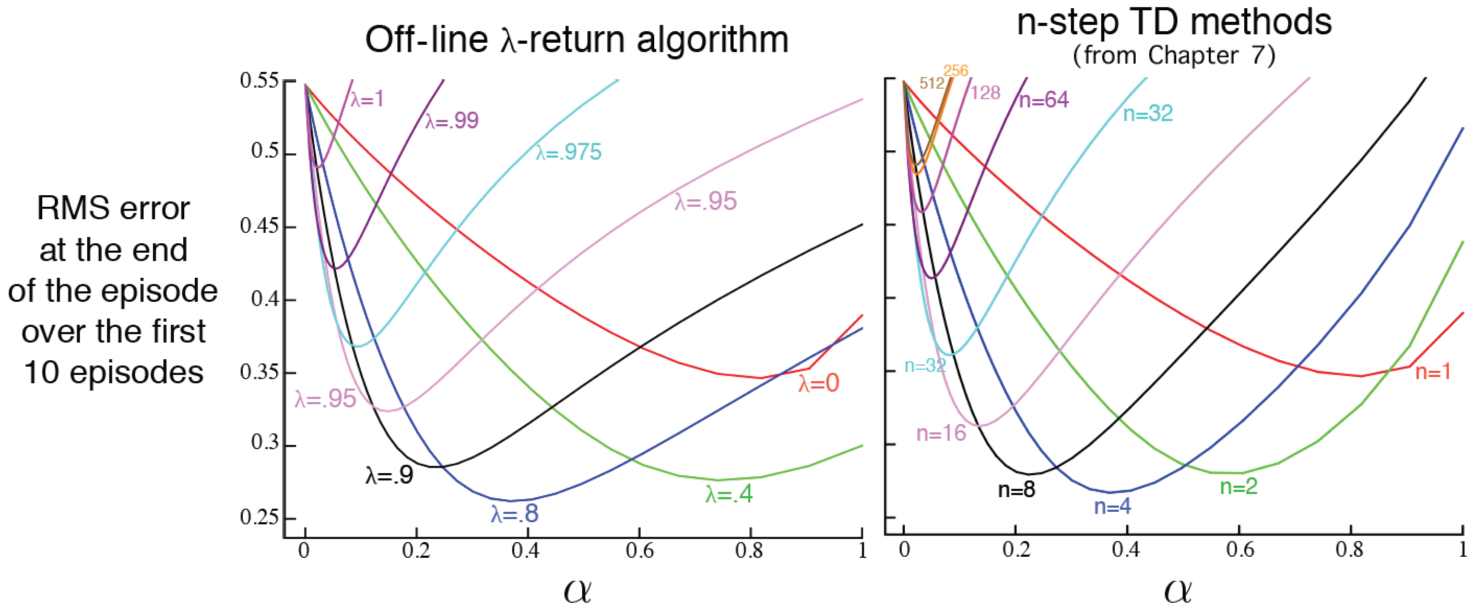
The Off-Line λ-Return Algorithm

- A simplistic algorithm is to use the λ-return as the update target for SGD (semi-gradient descent) with approximation.
- Update: Use $U_t = G_t^\lambda$ as the target

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

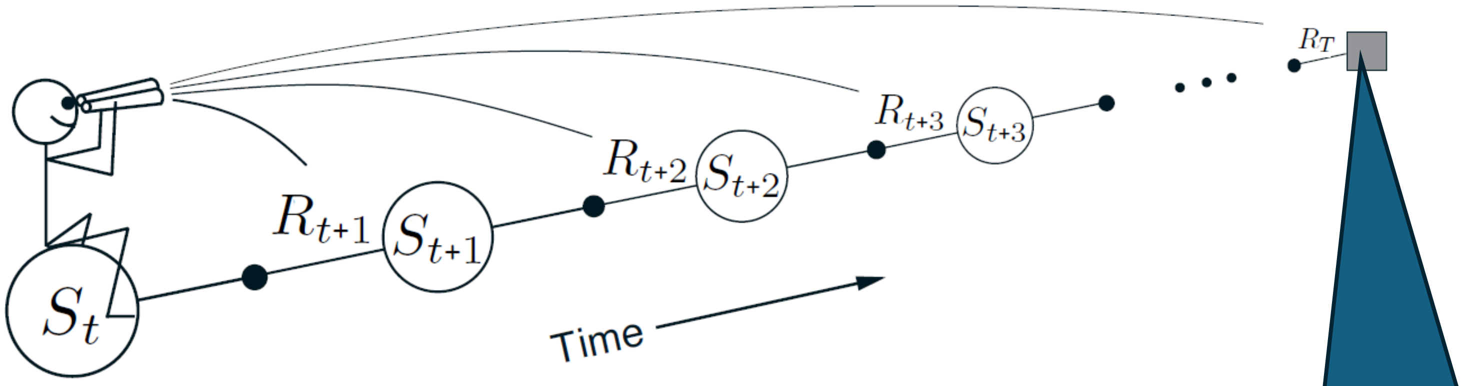
Example: 19-state random walk value estimation task.

λ-return algorithm mimics n-step methods.



The Forward View Issue with the Off-Line λ-Return Algorithm

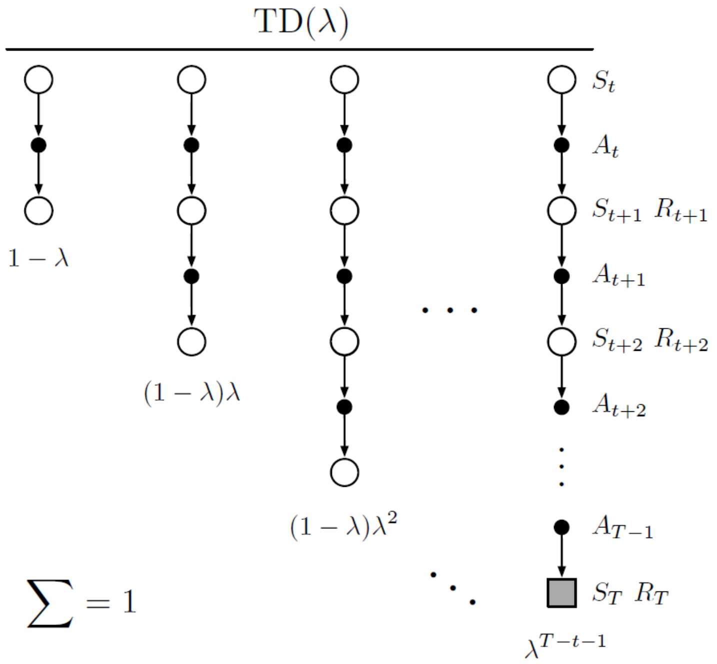
- Uses the **"forward view."**
- Updating S_t requires the calculation of G_t^λ, which requires all future rewards
- The **update is delayed** till the whole episode is observed (like for MC).



Agent can only update the value for S_t when it is here!

Backup diagram

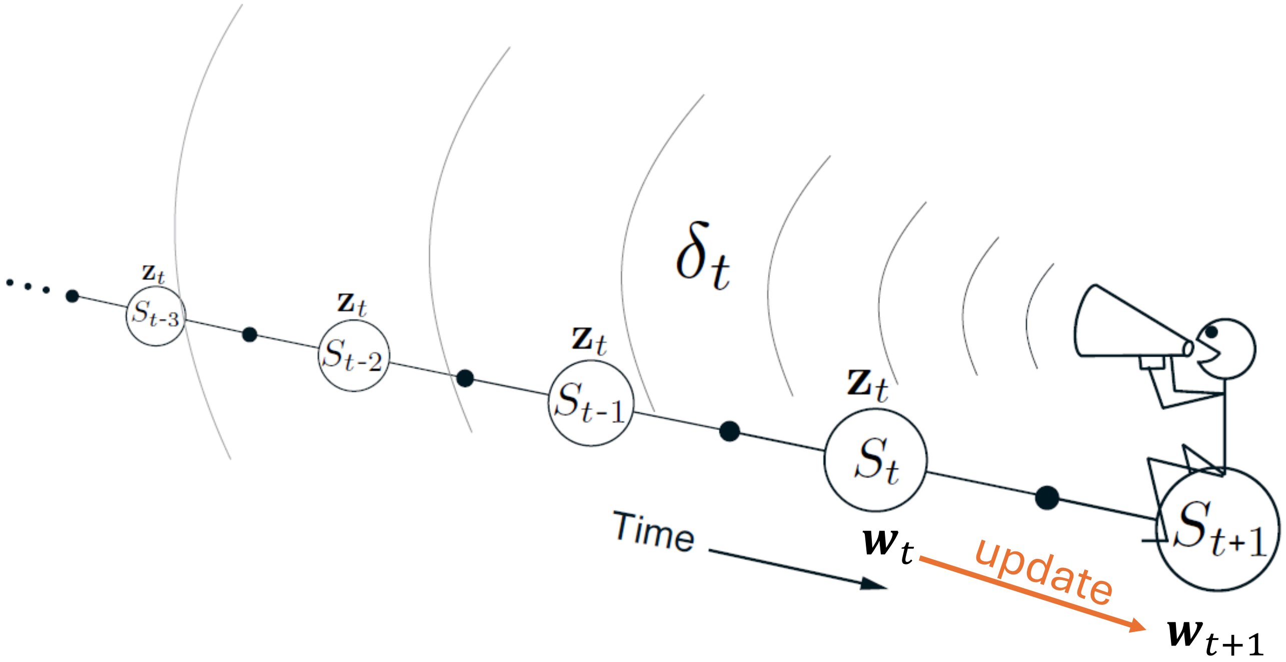
Weighting of the components of the λ-return



The Backward View

Using eligibility traces

The Backward View



If we can update our current state value by looking back at the updates of previous states, then we can:

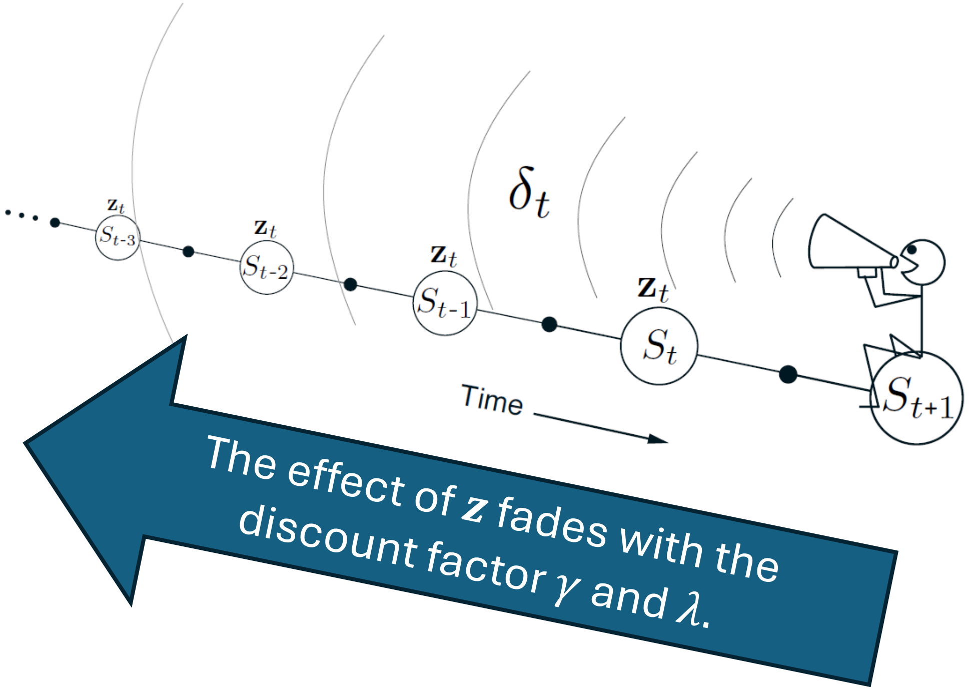
- Approximate the behavior of the forward-looking Off-Line λ-Return Algorithm.
- Update $\boldsymbol{w}$ at every time step.
- Computations are equally spread over time (not just the end of the episode).
- Works for continuing problems with no episode structure.

Eligibility Traces

- The approximation weights $\boldsymbol{w}_t \in \mathbb{R}^d$ represent what we have learned about the value function so far = long-term memory

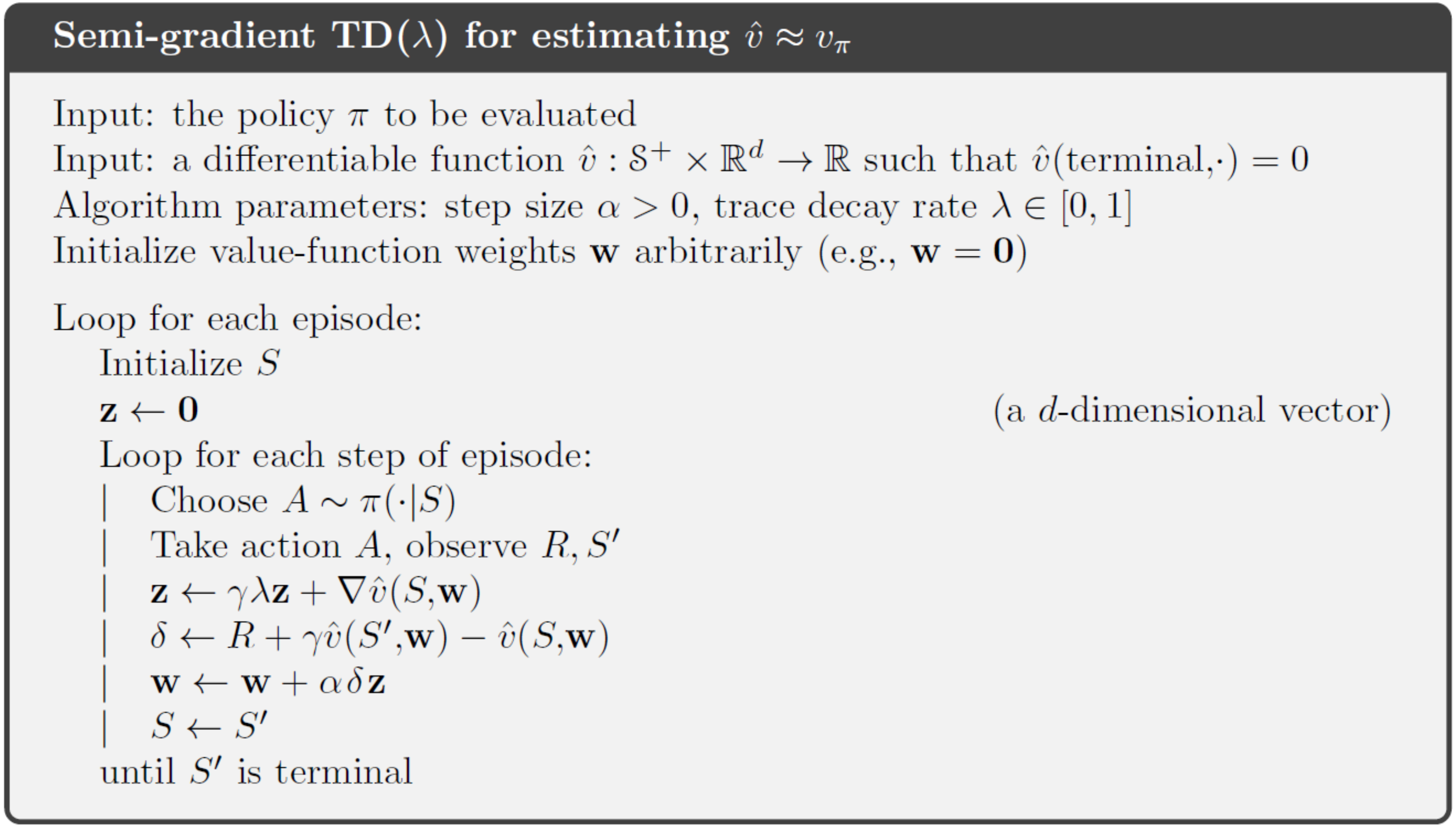
- An **eligibility trace** is a vector $\boldsymbol{z}_t \in \mathbb{R}^d$
- Update: $\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \nabla\hat{v}(S_t, \boldsymbol{w}_t)$

- Each component in $\boldsymbol{z}_t$ keeps track of which component in $\boldsymbol{w}_t$ contributed to the recent state valuation.
- In other words, $\boldsymbol{z}_t$ shows the eligibility of each component of $\boldsymbol{w}_t$ to undergo learning changes when a reinforcement event happens.



The effect of $\boldsymbol{z}$ fades with the discount factor γ and λ.

Semi-gradient TD(λ) for Estimation



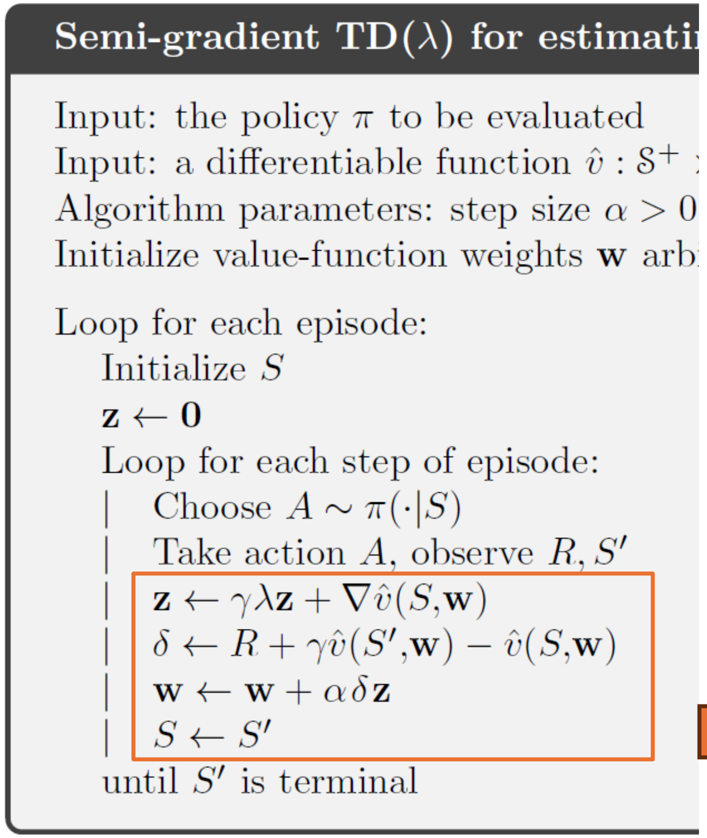
Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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
```

True Online TD(λ) for Estimation

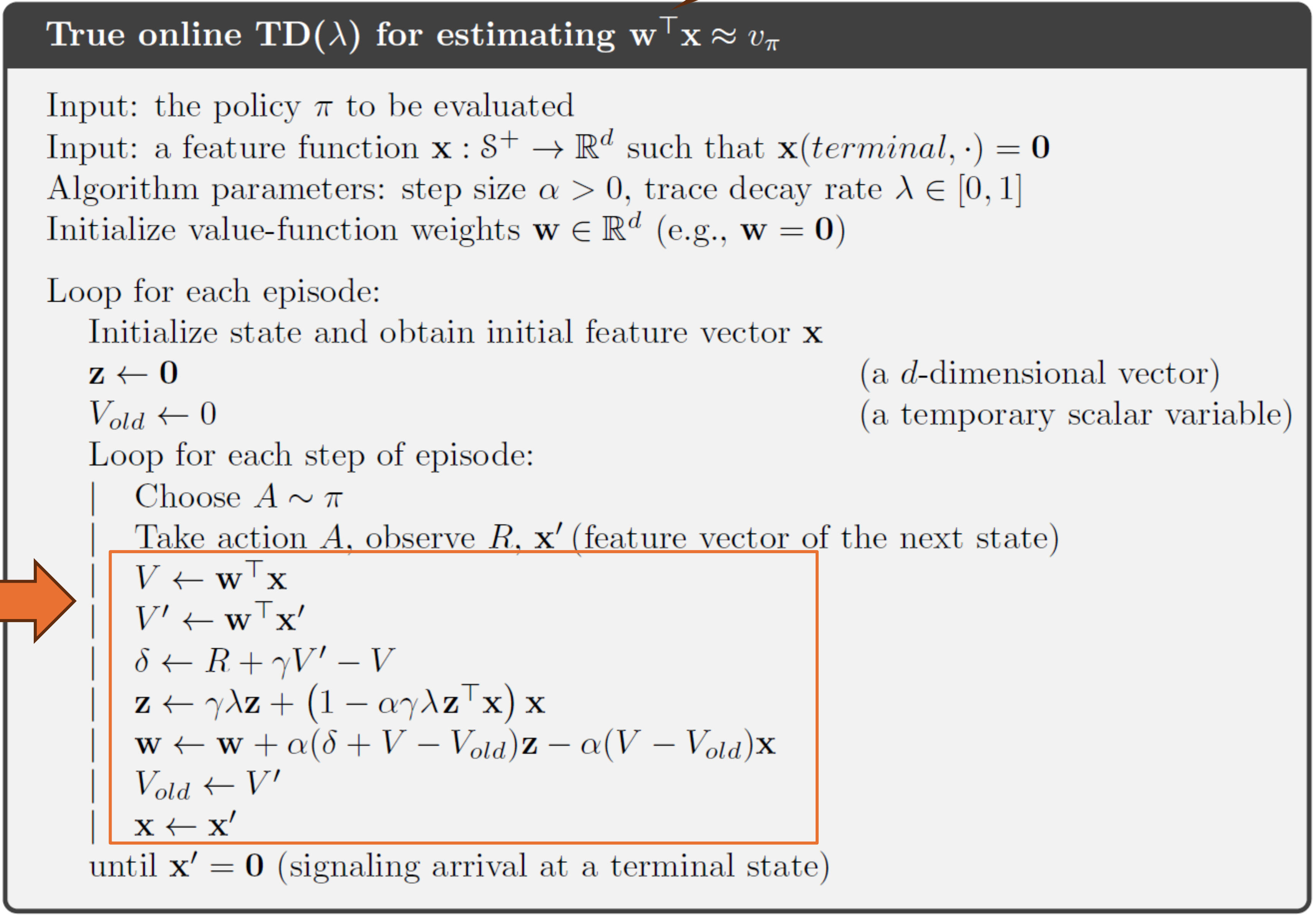
States are represented by feature vectors $\boldsymbol{x}$
Replace $\hat{v}(S, \boldsymbol{w})$ with $\boldsymbol{w}^\top \boldsymbol{x}$



Semi-gradient TD(λ) for estimati

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+$ :
Algorithm parameters: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w}$ arb

Loop for each episode:
    Initialize $S$
    $\mathbf{z} \leftarrow \mathbf{0}$
    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
```



True online TD(λ) for estimating $\mathbf{w}^\top\mathbf{x} \approx v_\pi$

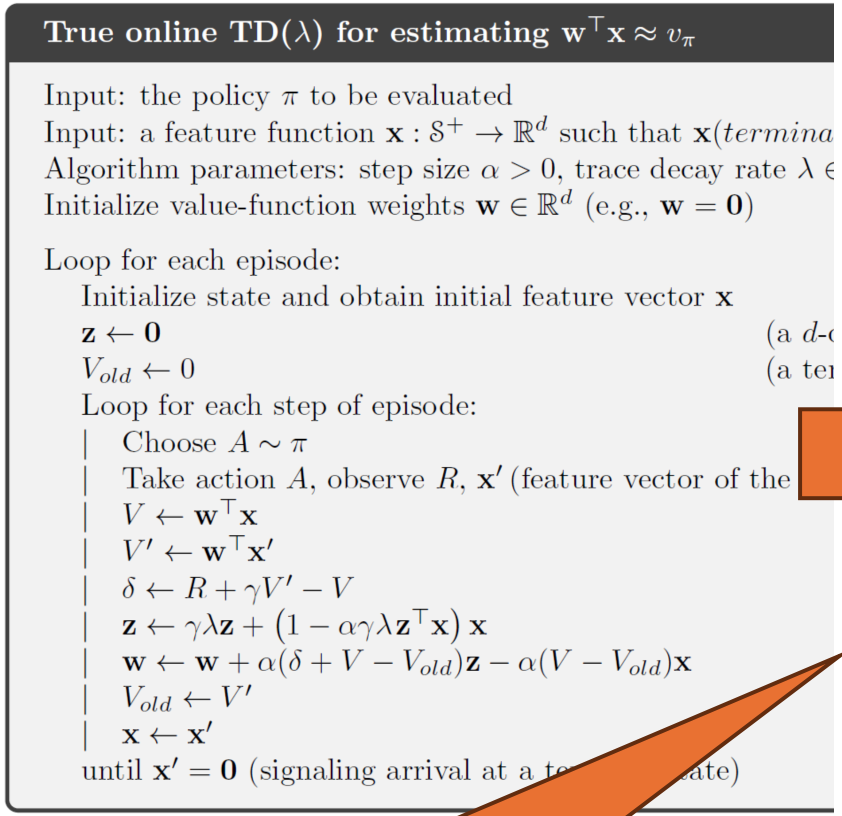
```
Input: the policy $\pi$ to be evaluated
Input: a feature function $\mathbf{x} : \mathcal{S}^+ \rightarrow \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize value-function weights $\mathbf{w}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

Loop for each episode:
    Initialize state and obtain initial feature vector $\mathbf{x}$
    $\mathbf{z} \leftarrow \mathbf{0}$                                   (a $d$-dimensional vector)
    $V_{old} \leftarrow 0$                                  (a temporary scalar variable)
    Loop for each step of episode:
    |   Choose $A \sim \pi$
    |   Take action $A$, observe $R$, $\mathbf{x}'$ (feature vector of the next state)
    |   $V \leftarrow \mathbf{w}^\top\mathbf{x}$
    |   $V' \leftarrow \mathbf{w}^\top\mathbf{x}'$
    |   $\delta \leftarrow R + \gamma V' - V$
    |   $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
    |   $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{old})\mathbf{z} - \alpha(V - V_{old})\mathbf{x}$
    |   $V_{old} \leftarrow V'$
    |   $\mathbf{x} \leftarrow \mathbf{x}'$
    until $\mathbf{x}' = \mathbf{0}$ (signaling arrival at a terminal state)
```

Control with Eligibility Traces

Sarsa(λ)

Sarsa(λ) Control

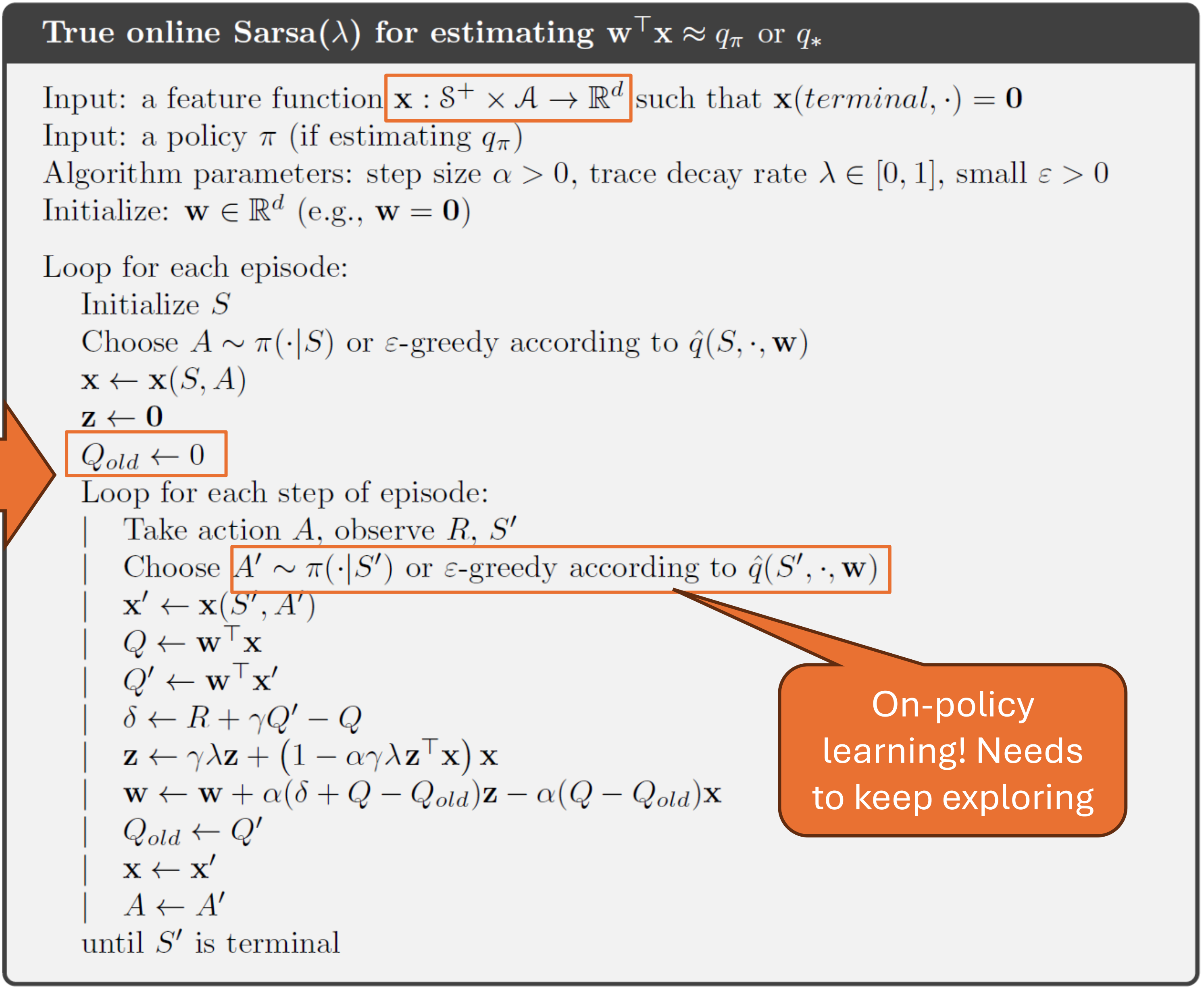


True online TD(λ) for estimating $\mathbf{w}^\top\mathbf{x} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a feature function $\mathbf{x} : \mathcal{S}^+ \rightarrow \mathbb{R}^d$ such that $\mathbf{x}(termina$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize state and obtain initial feature vector $\mathbf{x}$
    $\mathbf{z} \leftarrow \mathbf{0}$                                    (a $d$-
    $V_{old} \leftarrow 0$                                    (a ter
    Loop for each step of episode:
    |   Choose $A \sim \pi$
    |   Take action $A$, observe $R$, $\mathbf{x}'$ (feature vector of the
    |   $V \leftarrow \mathbf{w}^\top\mathbf{x}$
    |   $V' \leftarrow \mathbf{w}^\top\mathbf{x}'$
    |   $\delta \leftarrow R + \gamma V' - V$
    |   $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
    |   $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{old})\mathbf{z} - \alpha(V - V_{old})\mathbf{x}$
    |   $V_{old} \leftarrow V'$
    |   $\mathbf{x} \leftarrow \mathbf{x}'$
    until $\mathbf{x}' = \mathbf{0}$ (signaling arrival at a te          ate)
```

- Estimate action values $\hat{q}(s, a, \boldsymbol{w})$. V becomes Q.
- $\boldsymbol{x}$ is now a state+action feature



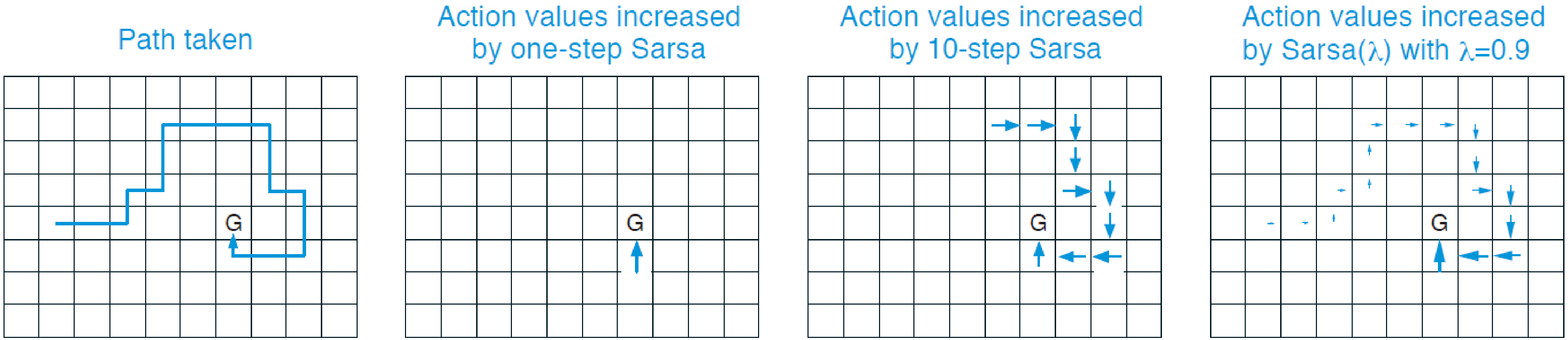
True online Sarsa(λ) for estimating $\mathbf{w}^\top\mathbf{x} \approx q_\pi$ or q_*

```
Input: a feature function $\boxed{\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \rightarrow \mathbb{R}^d}$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy $\pi$ (if estimating $q_\pi$)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$, small $\varepsilon > 0$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize $S$
    Choose $A \sim \pi(\cdot|S)$ or $\varepsilon$-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
    $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
    $\mathbf{z} \leftarrow \mathbf{0}$
    $\boxed{Q_{old} \leftarrow 0}$
    Loop for each step of episode:
    |   Take action $A$, observe $R$, $S'$
    |   Choose $\boxed{A' \sim \pi(\cdot|S') \text{ or } \varepsilon\text{-greedy according to } \hat{q}(S', \cdot, \mathbf{w})}$
    |   $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
    |   $Q \leftarrow \mathbf{w}^\top\mathbf{x}$
    |   $Q' \leftarrow \mathbf{w}^\top\mathbf{x}'$
    |   $\delta \leftarrow R + \gamma Q' - Q$
    |   $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
    |   $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
    |   $Q_{old} \leftarrow Q'$
    |   $\mathbf{x} \leftarrow \mathbf{x}'$
    |   $A \leftarrow A'$
    until $S'$ is terminal
```

On-policy learning! Needs to keep exploring

Example: Traces in a Gridworld


Path taken
Action values increased by one-step Sarsa
Action values increased by 10-step Sarsa
Action values increased by Sarsa(λ) with λ=0.9
G
G
G
G

- The first panel shows the path taken by an agent in a single episode.
- The initial estimated values were zero, and all rewards were zero except for a positive reward at the goal location marked by G.
- The arrows in the other panels show, for various algorithms, which action-values would be increased, and by how much, upon reaching the goal.
- Sarsa(λ) updates more values and also updates values closer to the goal more. This leads to **faster and more robust learning.**

Implementation Issues

- **Off-policy**: Use importance sampling (see Chapter 7).
  Note: Semi-gradient methods do not guarantee stability and may need extensions to improve stability.

- Most values of $\boldsymbol{z}_t$ are typically close to 0. Many algorithms set very small values to zero and save computation.

- **How to chose $\boldsymbol{\lambda}$**: There is no clear optimal value. It depends on
  - Stochasticiy of the environment
  - How sparse rewards are
  - The learning rate α
- Practitioners often start with lambda equals 0.9 and then lower it if behavior is unstable.

Conclusion

- Let’s us incrementally move between MC and one-step TD.
- Eligibility traces are very general, often learn faster, and are computationally efficient.
- Since eligibility traces can move close to MC behavior, they should be used if
  - the problem is suspected to be partially non-Markov or
  - the rewards are very delayed.
- Eligibility traces can also be used to extend Q-learning. This leads to algorithms like Watkins’s $Q(\lambda)$ and Tree-Backup(λ)