ELTE | FACULTY OF INFORMATICS

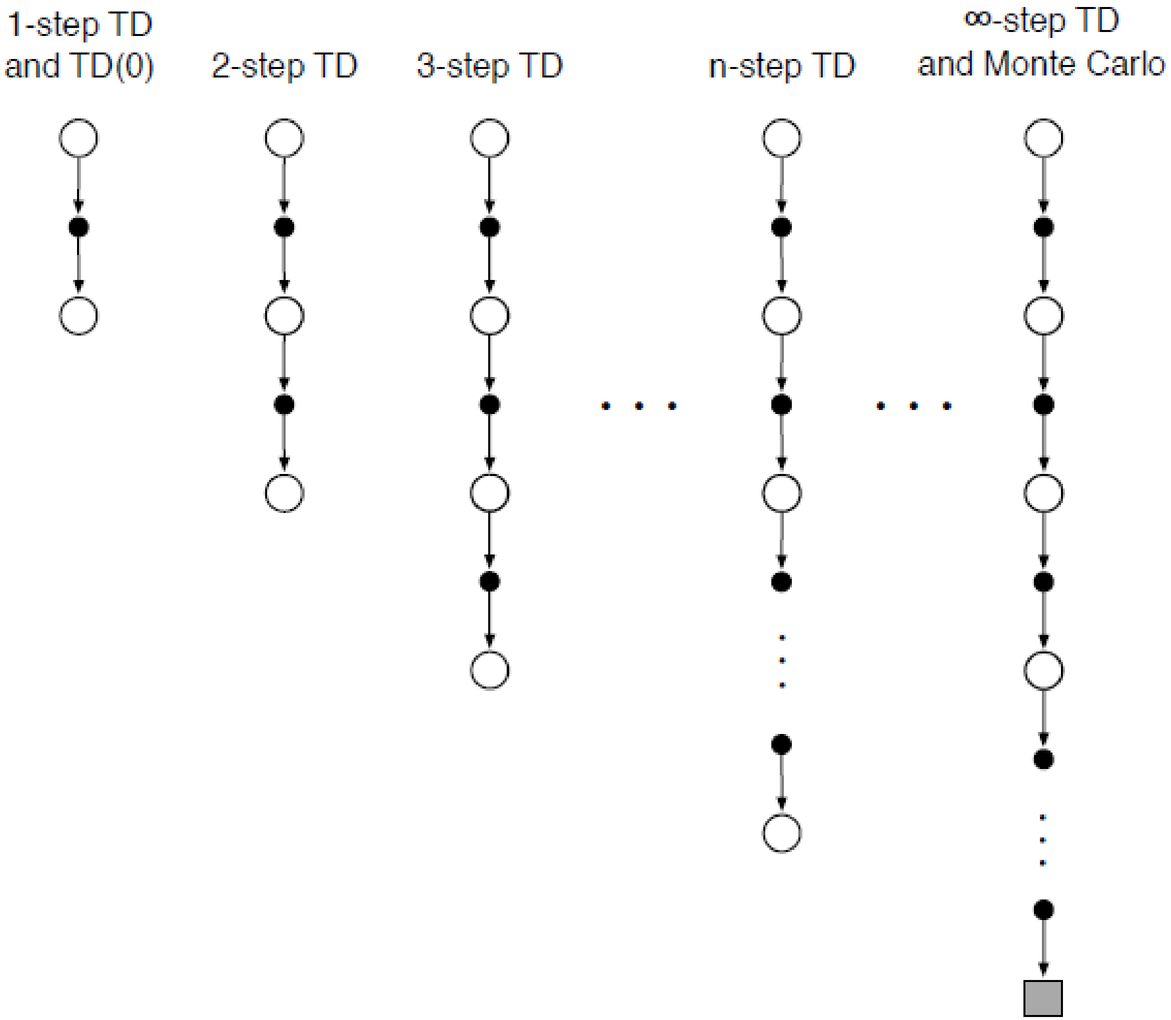
ELIGIBILITY TRACES

Deep Reinforcement Learning
Balázs Nagy, PhD

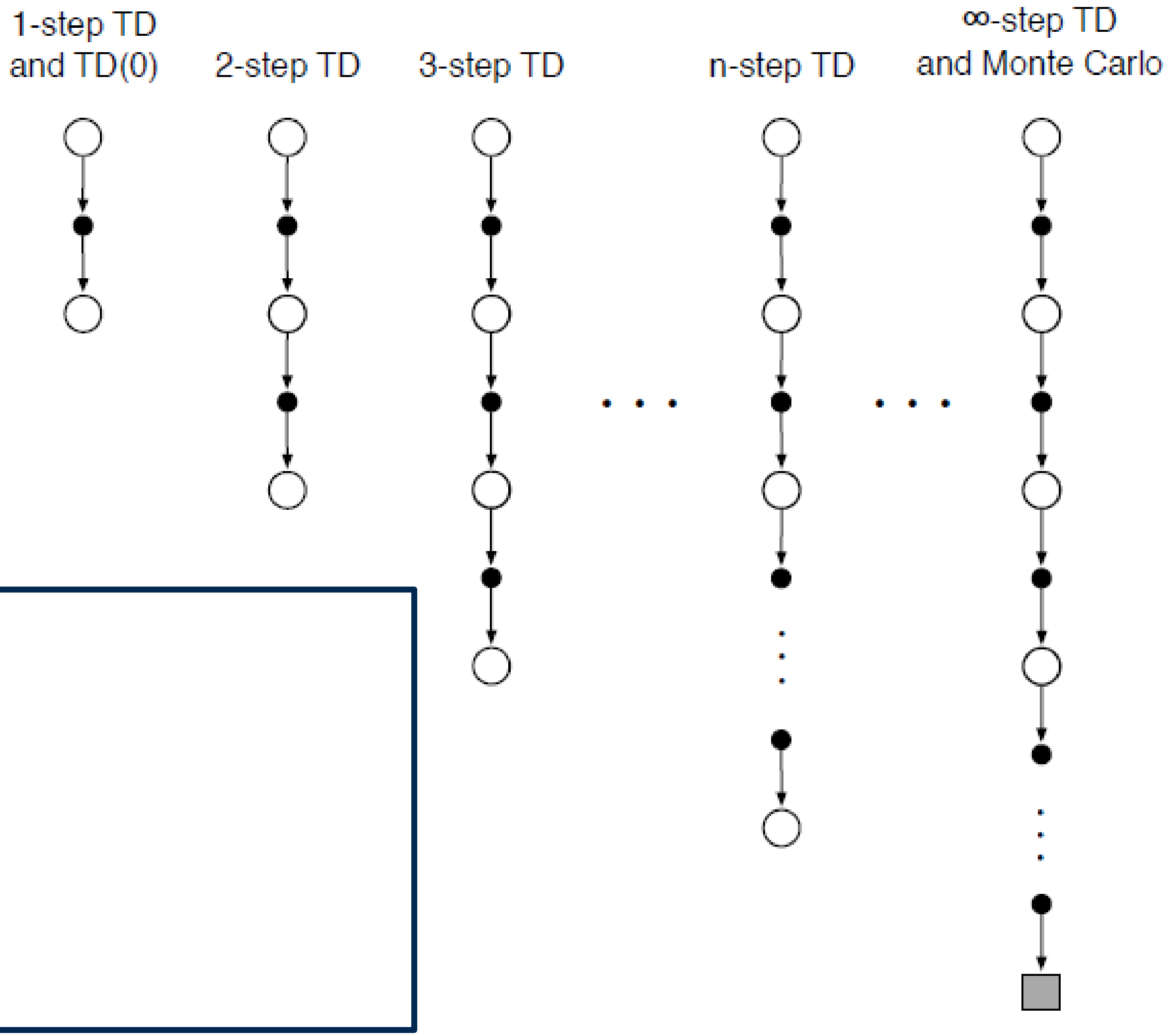
ELTE | IK

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

N-step TD

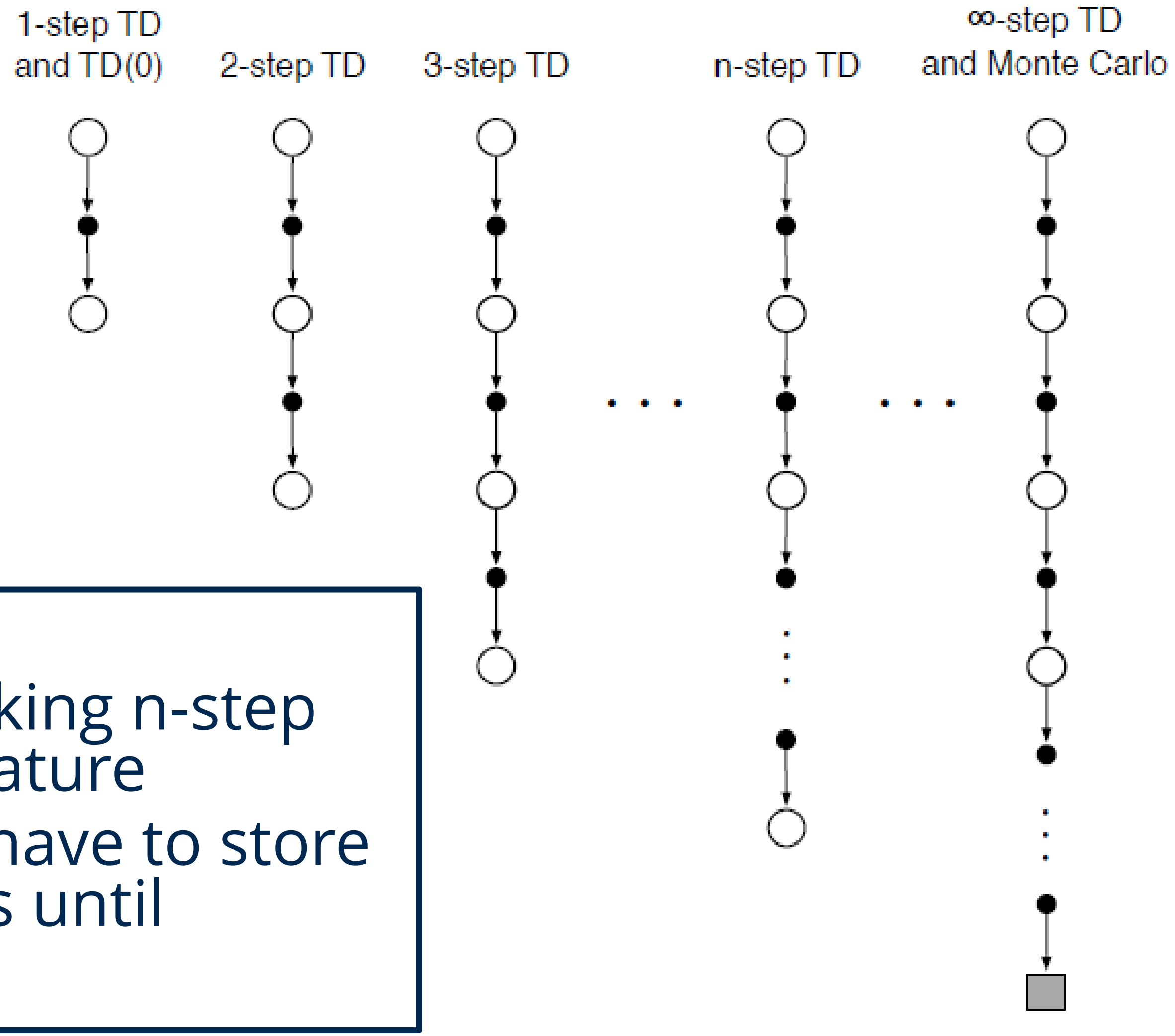

1-step TD
and TD(0)
2-step TD
3-step TD
n-step TD
∞-step TD
and Monte Carlo

N-step TD



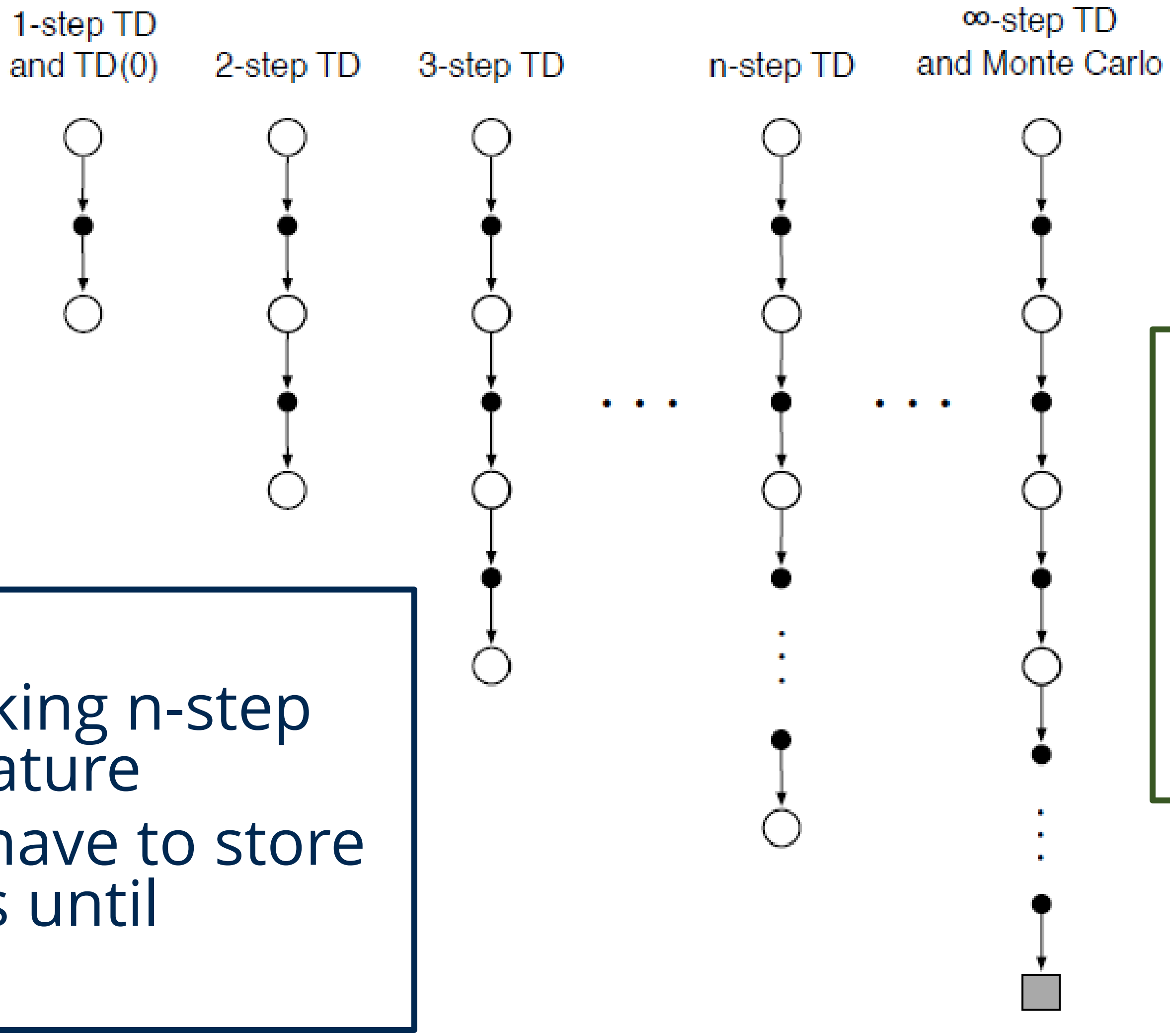
- Drawbacks:

N-step TD



- Drawbacks:
  - **Delay**: looking n-step into the feature
  - **Memory**: have to store *n* transions until calculation

N-step TD



- <u>Drawbacks:</u>
  - **Delay**: looking n-step into the feature
  - **Memory**: have to store *n* transions until calculation

Goal:

Find an algorithm with the same flexibility without the disadvantages

N-step reward review

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

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

N-step reward review

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

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

- Valid update target for
  - a tabular learning update
  - an approximate SGD learning update
- Update can be done toward any n-step return

N-step reward review

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

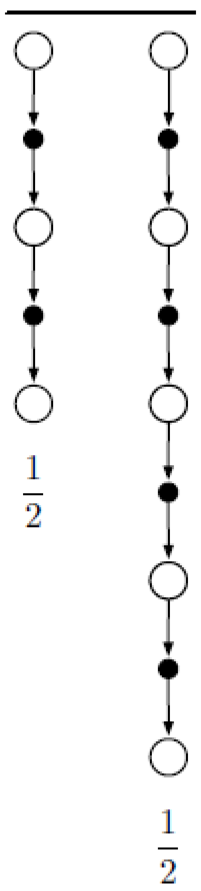
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad 0 \leq t \leq T - n$$

- Valid update target for
  - a tabular learning update
  - an approximate SGD learning update
- Update can be done toward any n-step return
- What if update toward any **average** of n-step returns?

Compound update

- An update can be done toward a target that is half of a two-step return and half of a four-step return

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

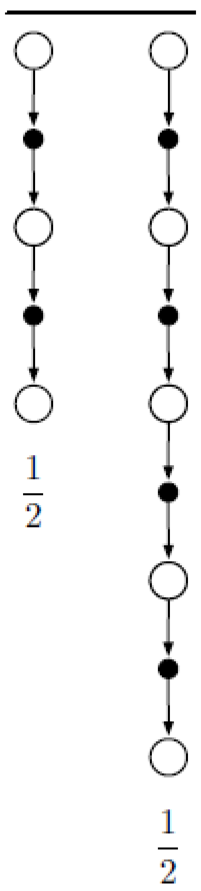


Compound update

- An update can be done toward a target that is half of a two-step return and half of a four-step return

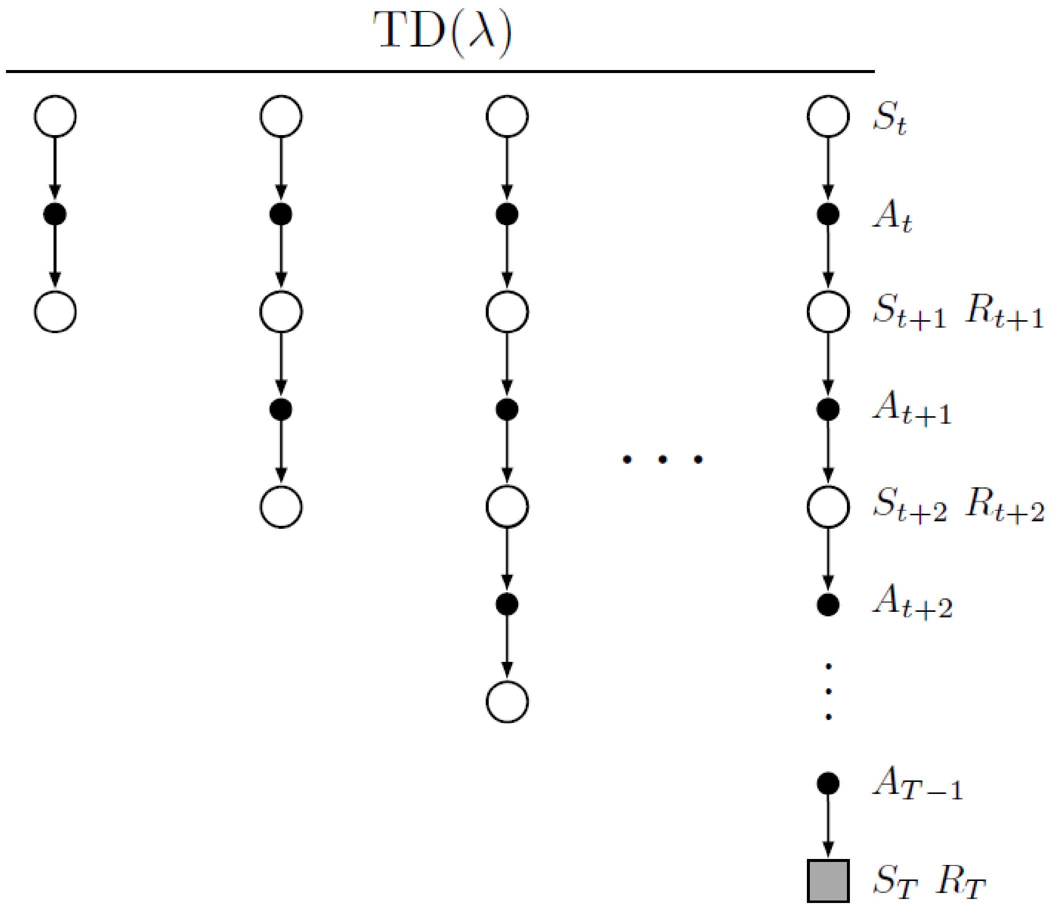
$$\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$$

- Any set of n-step returns can be averaged (even an infinite set, if the weights on the component returns are positive and sum to 1)
- Update that averages simpler component updates is called a **compound update**



λ return

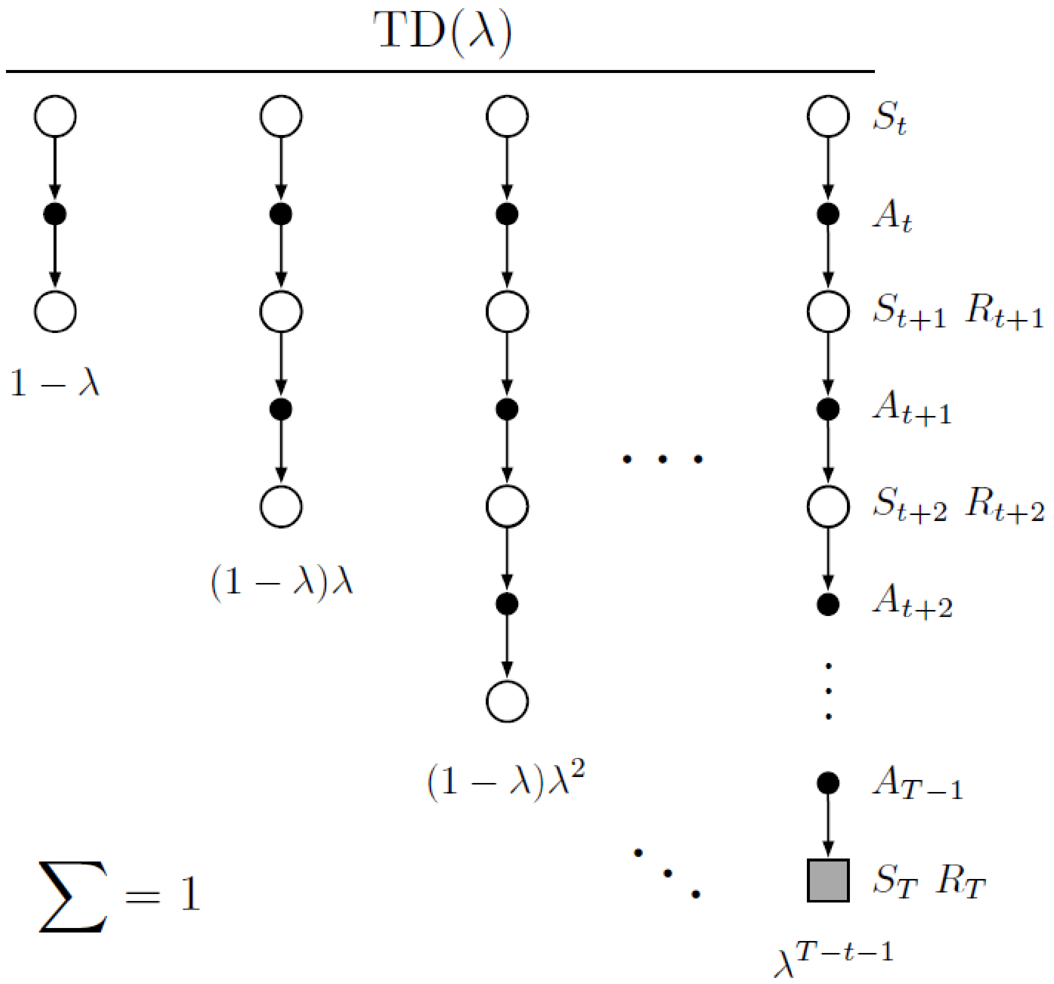
- average contains all the n-step updates



λ return

- average contains all the n-step updates
- λ return:

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

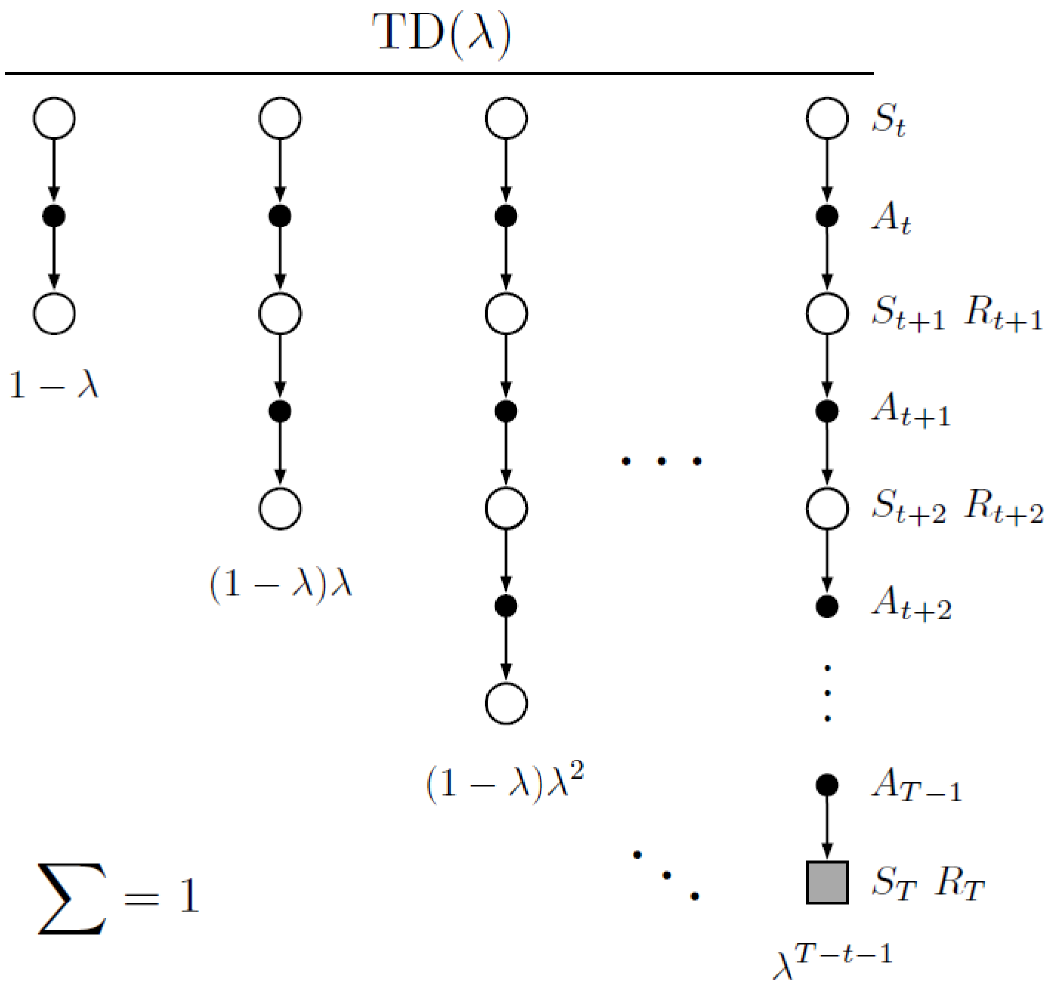


λ return

- average contains all the n-step updates
- λ return:

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

What if:
λ = 0
λ = 1

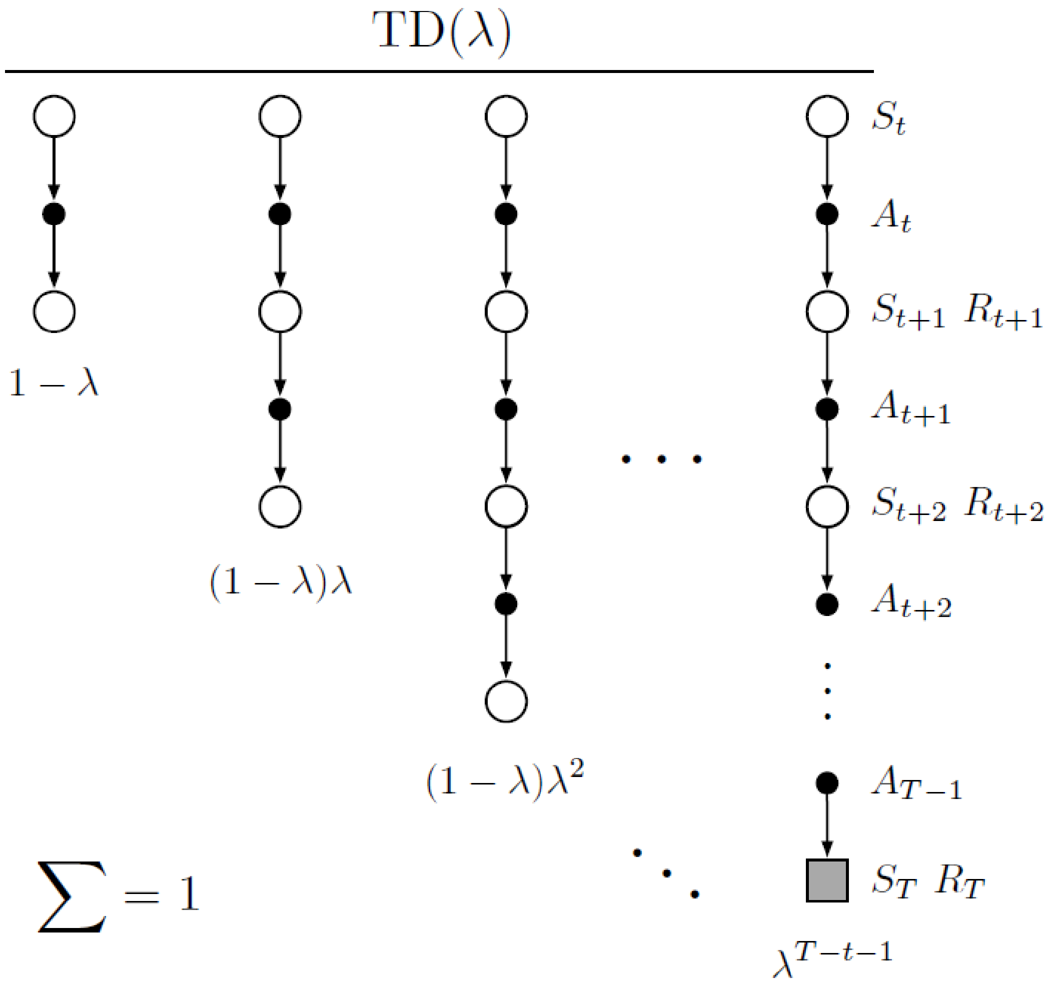


λ return

- average contains all the n-step updates
- λ return:

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

What if:
λ = 0, then TD(0)
λ = 1, then Monte Carlo

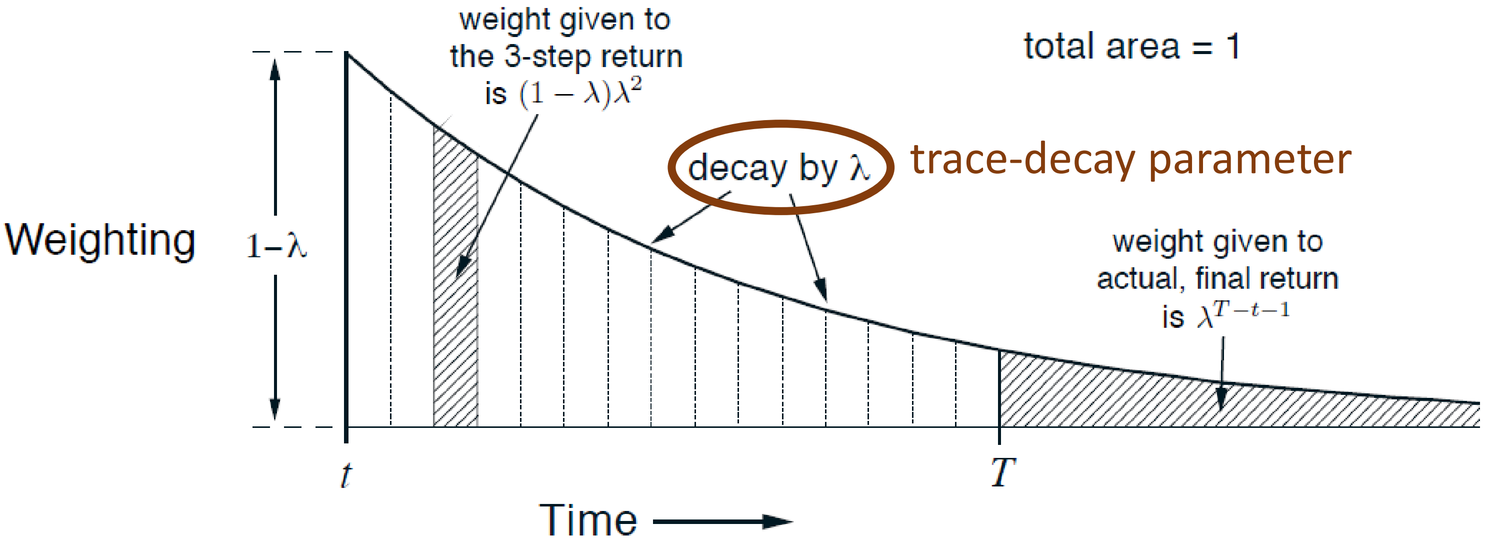


λ return

- λ return general form
- λ return rewritten with separate post-termination term

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

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



weight given to the 3-step return is $(1-\lambda)\lambda^2$
total area = 1
decay by λ
Weighting
$1-\lambda$
weight given to actual, final return is λ^{T-t-1}
t
T
Time

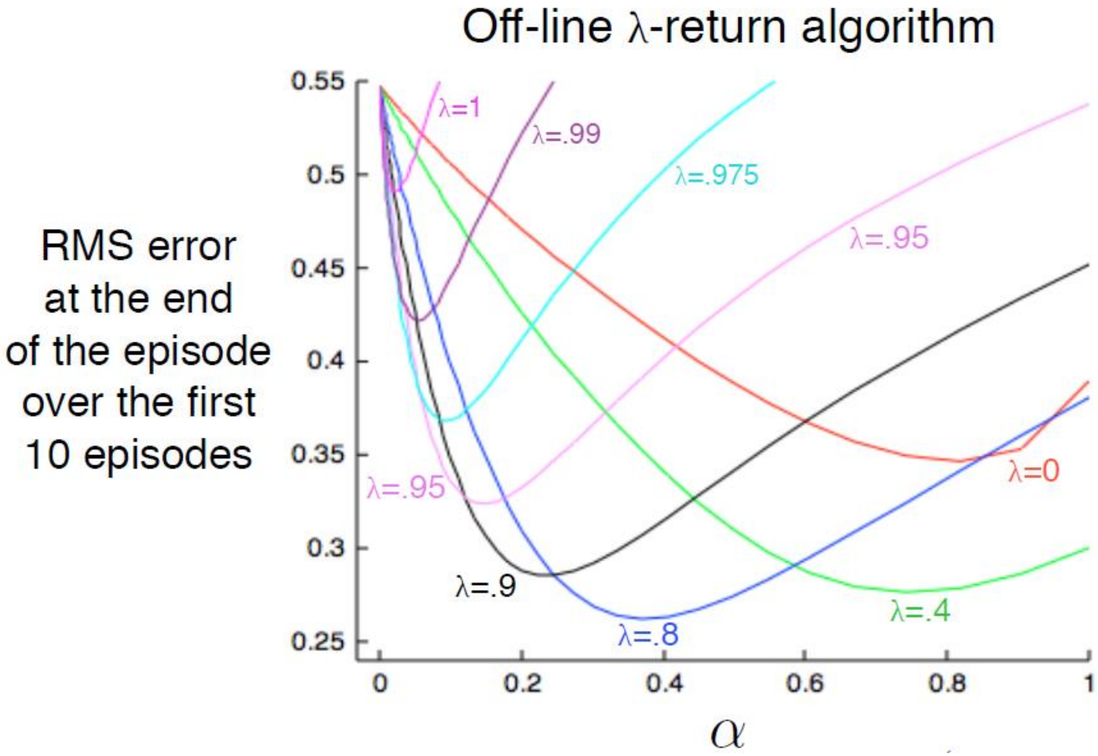
trace-decay parameter

Off-line λ-return algorithm

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

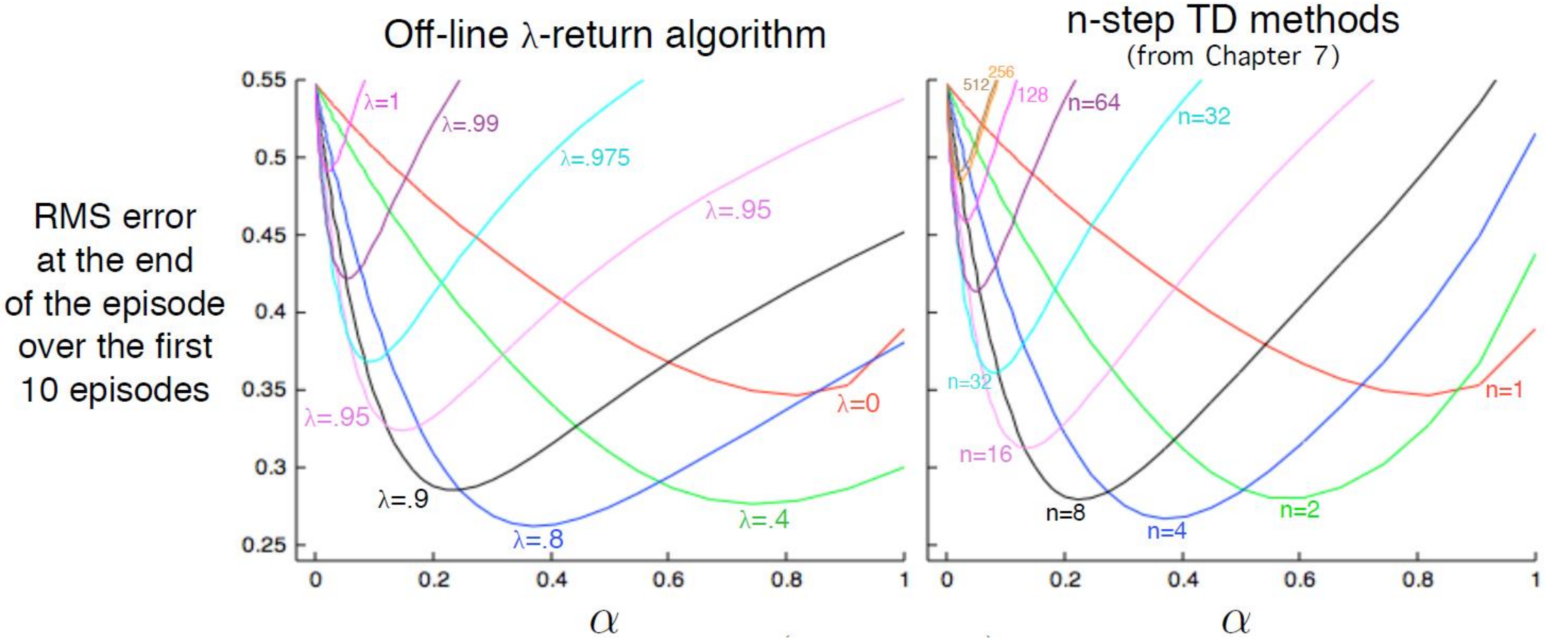
Off-line λ-return algorithm

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



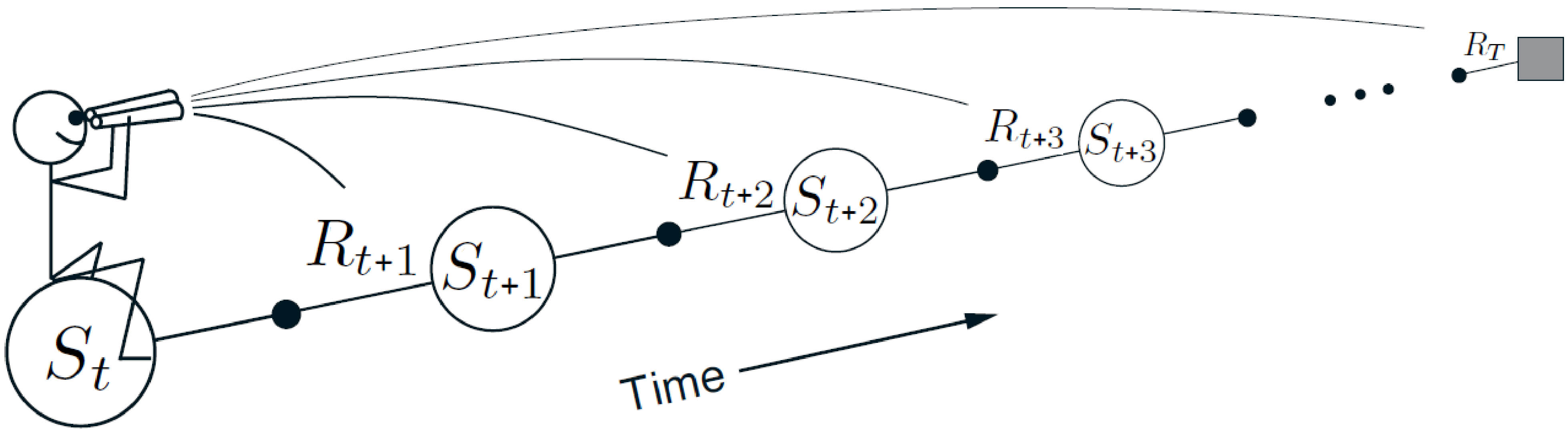
Off-line λ-return algorithm

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



Forward view

- 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



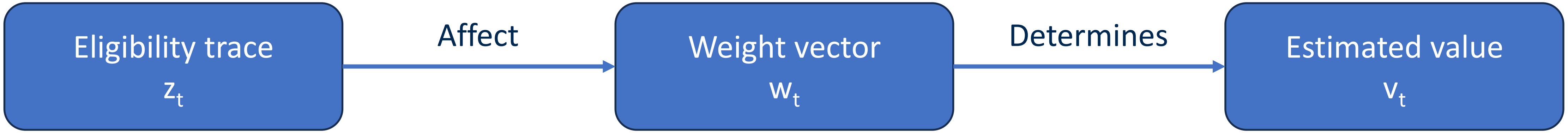
Semi-gradient TD(λ) with function approximation

Improvements over the off-line λ-return algorithm:

- Updates the weight vector on every step of an episode rather than only at the end
- Computations are equally distributed in time rather that all at the end of the episode
- It can be applied to continuing problems rather than just episodic problems

Eligibility trace

- Eligibility trace is a vector z_t with the same number of components as the weight vector w_t
- 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 trace z_t

Affect

Weight vector w_t

Determines

Estimated value v_t

Eligibility trace

- The eligibility trace vector is initialized to zero at the beginning of the episode
- Incremented on each time step by the value gradient, and then fades away

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

In linear function approximation: $\mathbf{x}_t$

TD update

- The trace indicates the eligibility of each component of the weight vector for learning when a reinforcing event occur

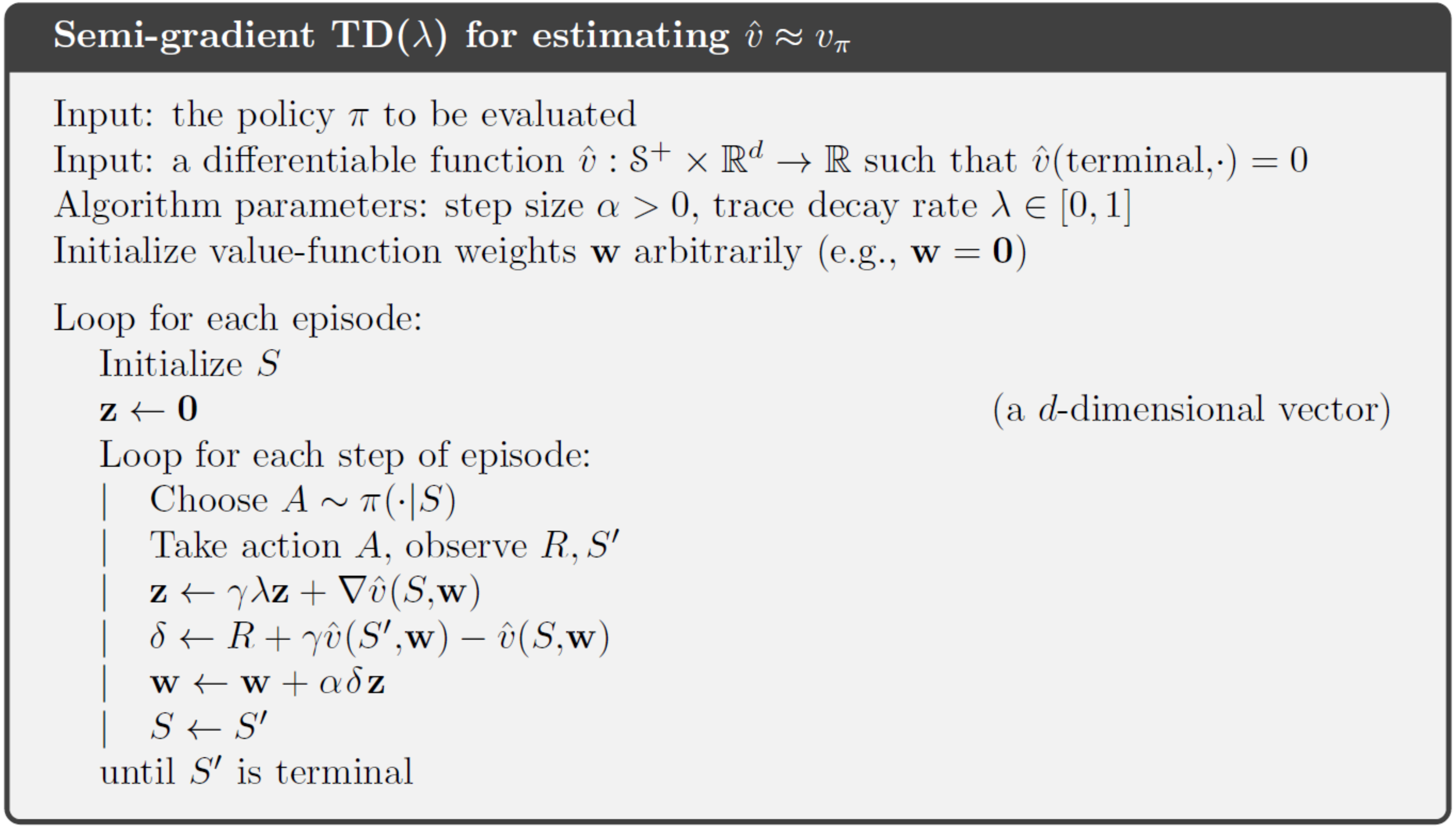
- Possible reinforcing events:
  TD error for state-value prediction

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

- The weight vector is updated on each step proportional to the TD error and the vector eligibility trace

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

Pseudocode



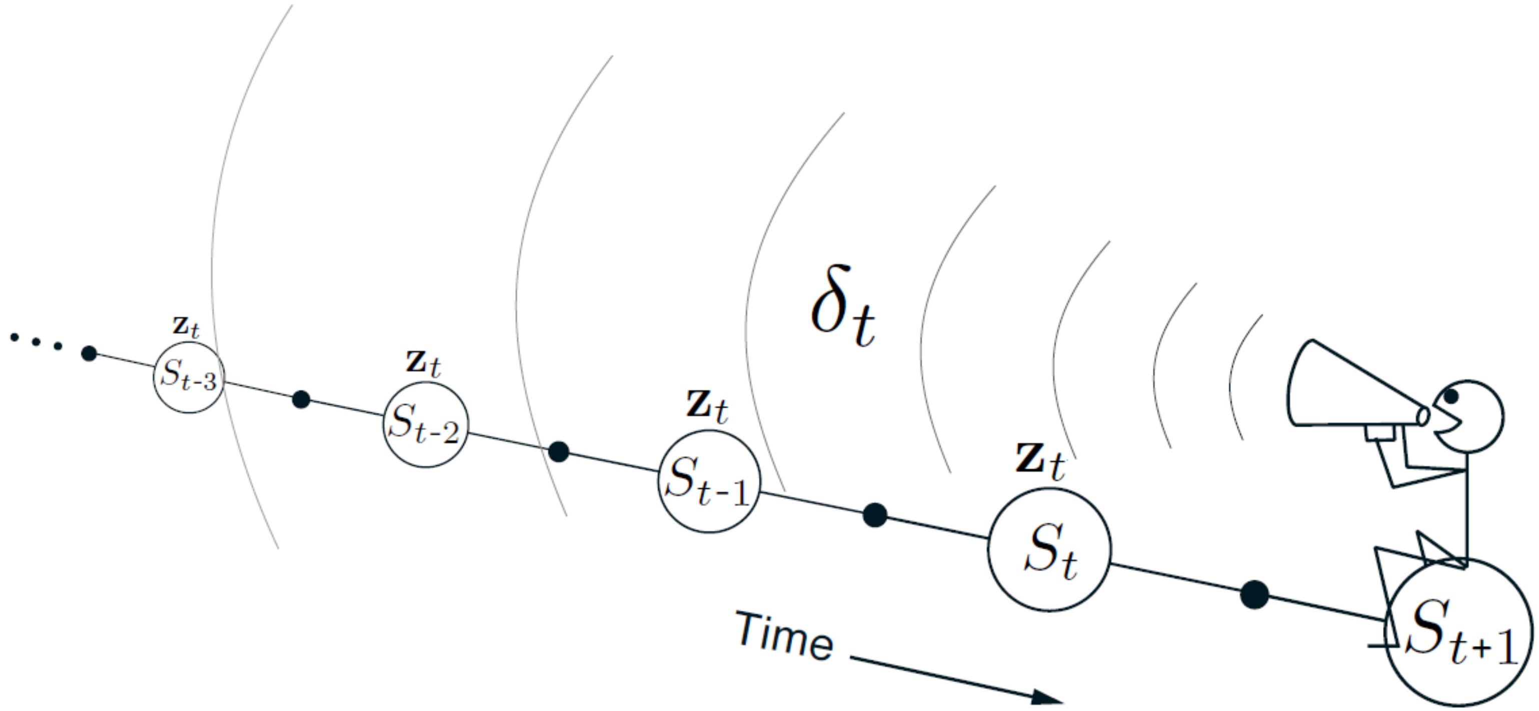
Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

```
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
```

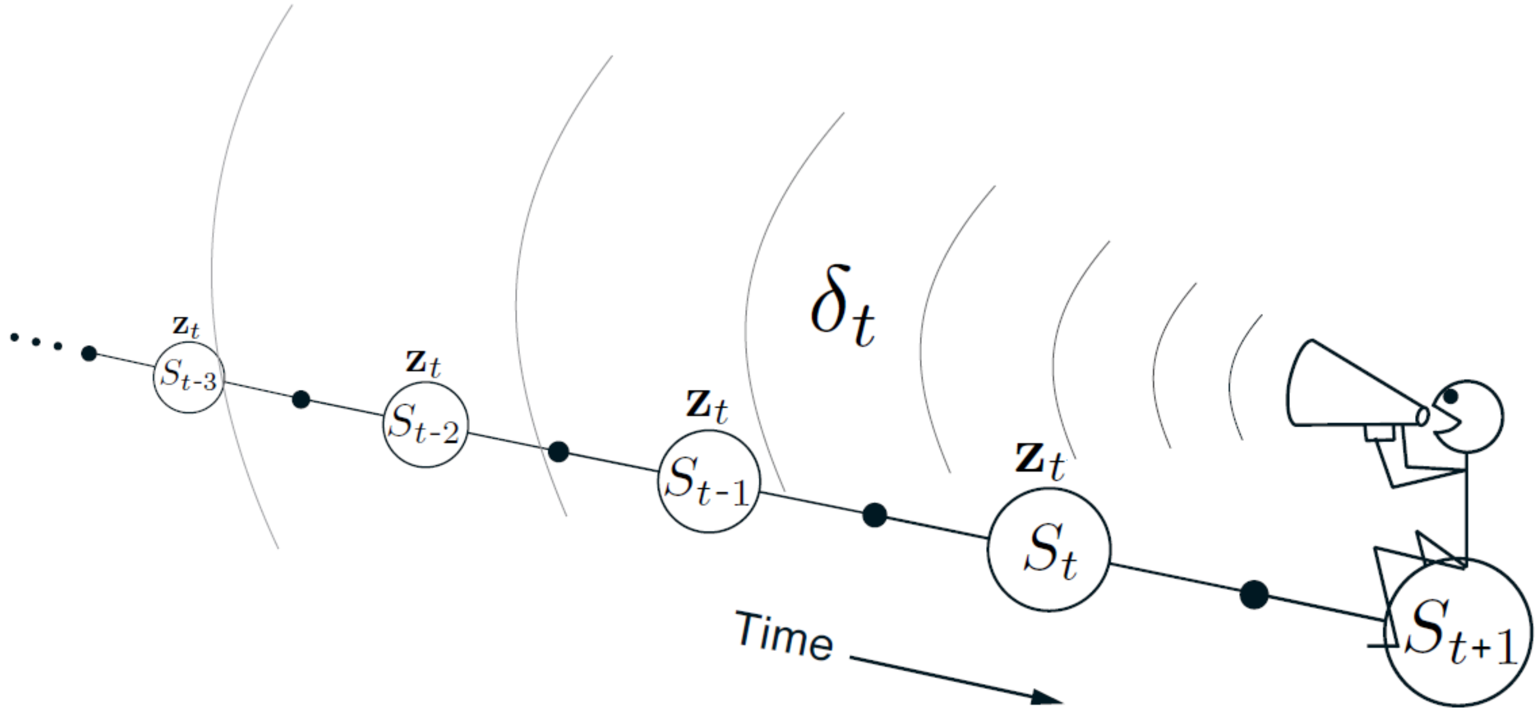
Backward view

- The backward or mechanistic view of TD(λ)
- Each update depends on the current TD error combined with the current eligibility traces of past events



Backward view

- The backward or mechanistic view of TD(λ)
- Each update depends on the current TD error combined with the current eligibility traces of past events



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

n-step truncated λ-return

- The off-line λ-return algorithm is an important ideal, but it is limited because it uses the λ-return, which is not known until the end of the episode.
- In the continuing case, the λ-return is technically never known, as it depends on n-step returns
- Dependence gets weaker for long-delayed rewards
- A natural approximation:
  truncate the sequence after some number of steps

n-step truncated λ-return

- Previously: λ-return for time *t*

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

n-step truncated λ-return

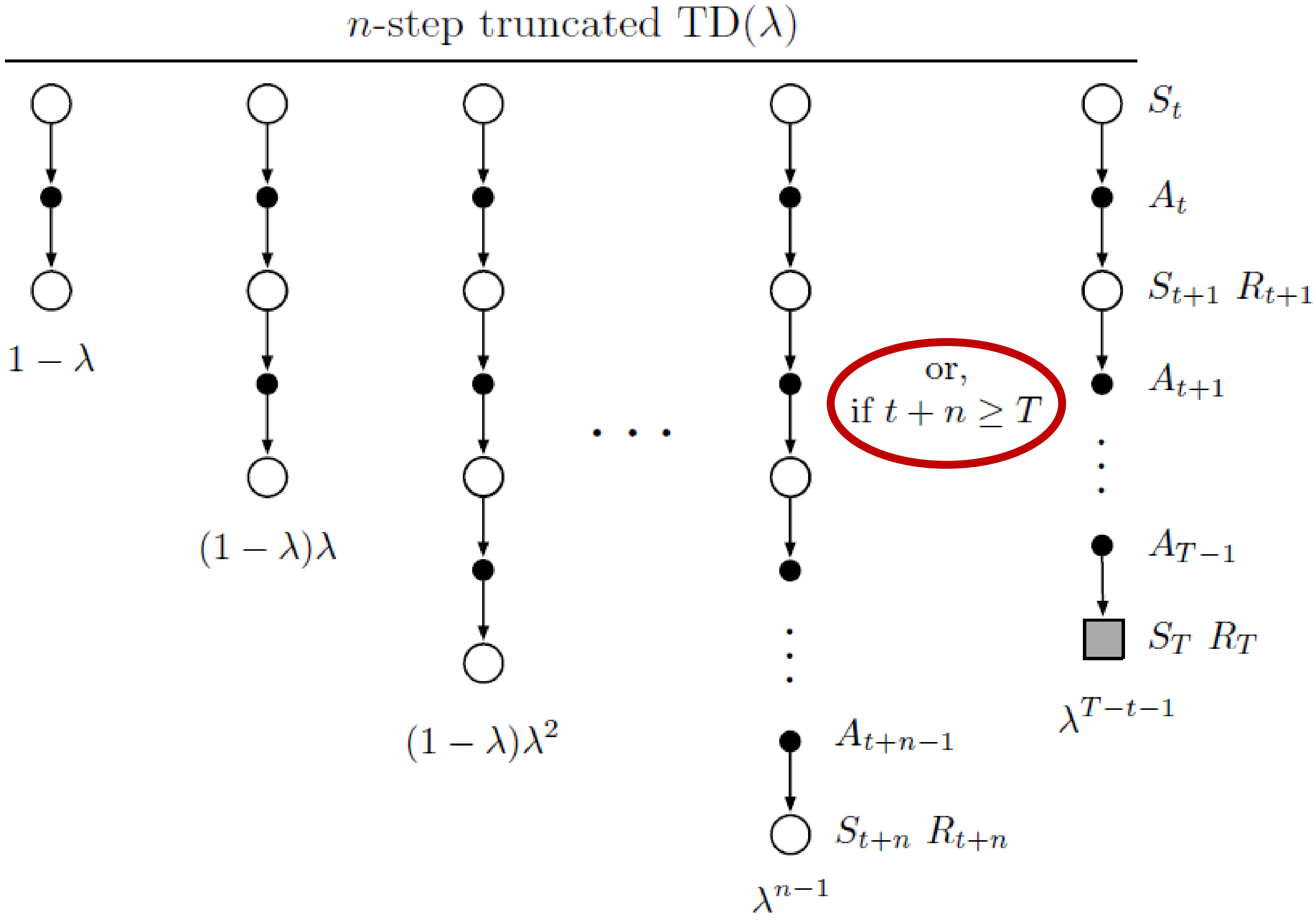
- Previously: λ-return for time *t*

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

- Truncated λ-return for time *t*
- Given data only up to some horizon, *h*

$$G_{t:h}^\lambda \;\doteq\; (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_{t:t+n} \;+\; \lambda^{h-t-1}G_{t:h}, \qquad 0 \le t < h \le T$$

Backup diagram for TTD(λ)



Online λ–return

- Tradeoff for the truncation parameter n
  - Large *n*: closely approximates the off-line λ-return algorithm
  - Small *n*: the updates can be made sooner and can in influence behavior sooner
- Can we get the best of both?

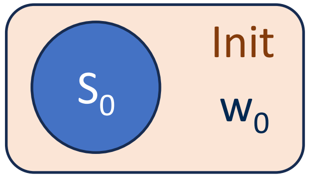
Online λ–return

- Tradeoff for the truncation parameter n
  - Large *n*: closely approximates the off-line λ-return algorithm
  - Small *n*: the updates can be made sooner and can in influence behavior sooner
- Can we get the best of both?
- YES! How:

Online λ–return

- Tradeoff for the truncation parameter n
  - Large *n*: closely approximates the off-line λ-return algorithm
  - Small *n*: the updates can be made sooner and can in influence behavior sooner
- Can we get the best of both?
- YES! How:
  - On each time step a new increment of data is gathered
  - Redo all the updates since the beginning of the current episode
  - The new updates will be better than the ones previously made because now they can take into account the time step's new data

Online λ–return algorithm



Init

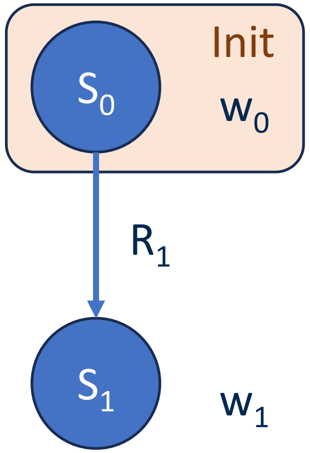
S_0

w_0

$$\boxed{\mathbf{w}_t^h}$$

Online λ–return algorithm

$$h = 1: \quad \mathbf{w}_1^1 \doteq \mathbf{w}_0^1 + \alpha \left[G_{0:1}^{\lambda} - \hat{v}(S_0, \mathbf{w}_0^1) \right] \nabla \hat{v}(S_0, \mathbf{w}_0^1)$$



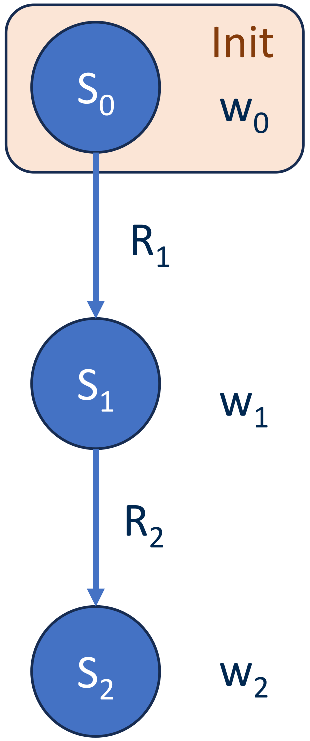
$$\boxed{\mathbf{w}_t^h}$$

Online λ–return algorithm

$$h = 1: \quad \mathbf{w}_1^1 \doteq \mathbf{w}_0^1 + \alpha \left[G_{0:1}^{\lambda} - \hat{v}(S_0, \mathbf{w}_0^1) \right] \nabla \hat{v}(S_0, \mathbf{w}_0^1)$$

$$h = 2: \quad \mathbf{w}_1^2 \doteq \mathbf{w}_0^2 + \alpha \left[G_{0:2}^{\lambda} - \hat{v}(S_0, \mathbf{w}_0^2) \right] \nabla \hat{v}(S_0, \mathbf{w}_0^2)$$

$$\mathbf{w}_2^2 \doteq \mathbf{w}_1^2 + \alpha \left[G_{1:2}^{\lambda} - \hat{v}(S_1, \mathbf{w}_1^2) \right] \nabla \hat{v}(S_1, \mathbf{w}_1^2)$$



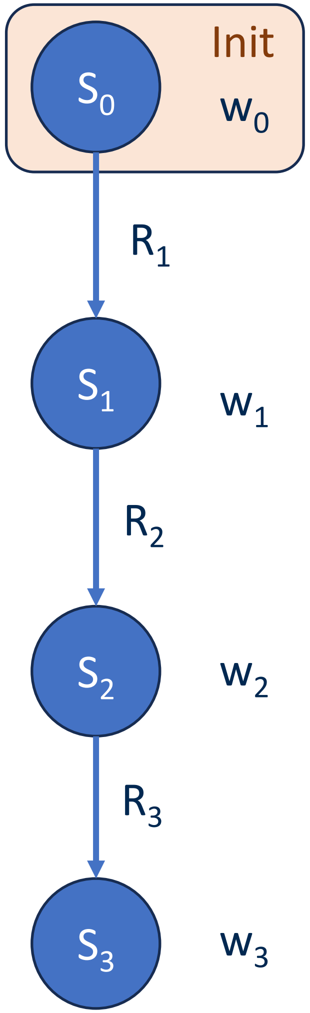
$$\boxed{\mathbf{w}_t^h}$$

Online λ–return algorithm

$$h=1: \quad \mathbf{w}_1^1 \doteq \mathbf{w}_0^1 + \alpha\left[G_{0:1}^{\lambda} - \hat{v}(S_0,\mathbf{w}_0^1)\right]\nabla\hat{v}(S_0,\mathbf{w}_0^1)$$

$$\begin{aligned} h=2: \quad &\mathbf{w}_1^2 \doteq \mathbf{w}_0^2 + \alpha\left[G_{0:2}^{\lambda} - \hat{v}(S_0,\mathbf{w}_0^2)\right]\nabla\hat{v}(S_0,\mathbf{w}_0^2) \\ &\mathbf{w}_2^2 \doteq \mathbf{w}_1^2 + \alpha\left[G_{1:2}^{\lambda} - \hat{v}(S_1,\mathbf{w}_1^2)\right]\nabla\hat{v}(S_1,\mathbf{w}_1^2) \end{aligned}$$

$$\begin{aligned} h=3: \quad &\mathbf{w}_1^3 \doteq \mathbf{w}_0^3 + \alpha\left[G_{0:3}^{\lambda} - \hat{v}(S_0,\mathbf{w}_0^3)\right]\nabla\hat{v}(S_0,\mathbf{w}_0^3) \\ &\mathbf{w}_2^3 \doteq \mathbf{w}_1^3 + \alpha\left[G_{1:3}^{\lambda} - \hat{v}(S_1,\mathbf{w}_1^3)\right]\nabla\hat{v}(S_1,\mathbf{w}_1^3) \\ &\mathbf{w}_3^3 \doteq \mathbf{w}_2^3 + \alpha\left[G_{2:3}^{\lambda} - \hat{v}(S_2,\mathbf{w}_2^3)\right]\nabla\hat{v}(S_2,\mathbf{w}_2^3) \end{aligned}$$



$$\boxed{\mathbf{w}_t^h}$$

Online λ–return algorithm

- General form for the update:

$$\mathbf{w}_{t+1}^{h} \doteq \mathbf{w}_{t}^{h} + \alpha \left[G_{t:h}^{\lambda} - \hat{v}(S_t, \mathbf{w}_t^h) \right] \nabla \hat{v}(S_t, \mathbf{w}_t^h), \quad 0 \leq t < h \leq T$$

$$\mathbf{w}_t \doteq \mathbf{w}_t^t$$

Online λ–return algorithm

- General form for the update:

$$\mathbf{w}_{t+1}^{h} \doteq \mathbf{w}_{t}^{h} + \alpha \left[G_{t:h}^{\lambda} - \hat{v}(S_t,\mathbf{w}_t^h)\right] \nabla \hat{v}(S_t,\mathbf{w}_t^h), \quad 0 \leq t < h \leq T$$

$$\mathbf{w}_t \doteq \mathbf{w}_t^t$$

- Fully online
  determining a new weight vector w_t at each step *t* during an episode
- Using only information available at time *t*
- Forward-view algorithm
- Drawback: computationally complex

True online TD(λ)

- Is there a way to invert the forward-view algorithm to produce an efficient backward-view algorithm using eligibility traces?

True online TD(λ)

- Is there a way to invert the forward-view algorithm to produce an efficient backward-view algorithm using eligibility traces?

$$\begin{array}{llllll}
\mathbf{w}_0^0 & & & & & \\
\mathbf{w}_0^1 & \mathbf{w}_1^1 & & & & \\
\mathbf{w}_0^2 & \mathbf{w}_1^2 & \mathbf{w}_2^2 & & & \\
\mathbf{w}_0^3 & \mathbf{w}_1^3 & \mathbf{w}_2^3 & \mathbf{w}_3^3 & & \\
\vdots & \vdots & \vdots & \vdots & \ddots & \\
\mathbf{w}_0^T & \mathbf{w}_1^T & \mathbf{w}_2^T & \mathbf{w}_3^T & \cdots & \mathbf{w}_T^T
\end{array}$$

$$\mathbf{w}_{t+1}^h \doteq \mathbf{w}_t^h + \alpha \left[G_{t:h}^\lambda - \hat{v}(S_t,\mathbf{w}_t^h) \right] \nabla \hat{v}(S_t,\mathbf{w}_t^h)$$

$$\mathbf{w}_t \doteq \mathbf{w}_t^t$$

True online TD(λ)

- Is there a way to invert the forward-view algorithm to produce an efficient backward-view algorithm using eligibility traces?

$$\begin{array}{cccccc}
\boxed{\mathbf{w}_0^0} & & & & & \\
\mathbf{w}_0^1 & \boxed{\mathbf{w}_1^1} & & & & \\
\mathbf{w}_0^2 & \mathbf{w}_1^2 & \boxed{\mathbf{w}_2^2} & & & \\
\mathbf{w}_0^3 & \mathbf{w}_1^3 & \mathbf{w}_2^3 & \boxed{\mathbf{w}_3^3} & & \\
\vdots & \vdots & \vdots & \vdots & \ddots & \\
\mathbf{w}_0^T & \mathbf{w}_1^T & \mathbf{w}_2^T & \mathbf{w}_3^T & \cdots & \boxed{\mathbf{w}_T^T}
\end{array}$$

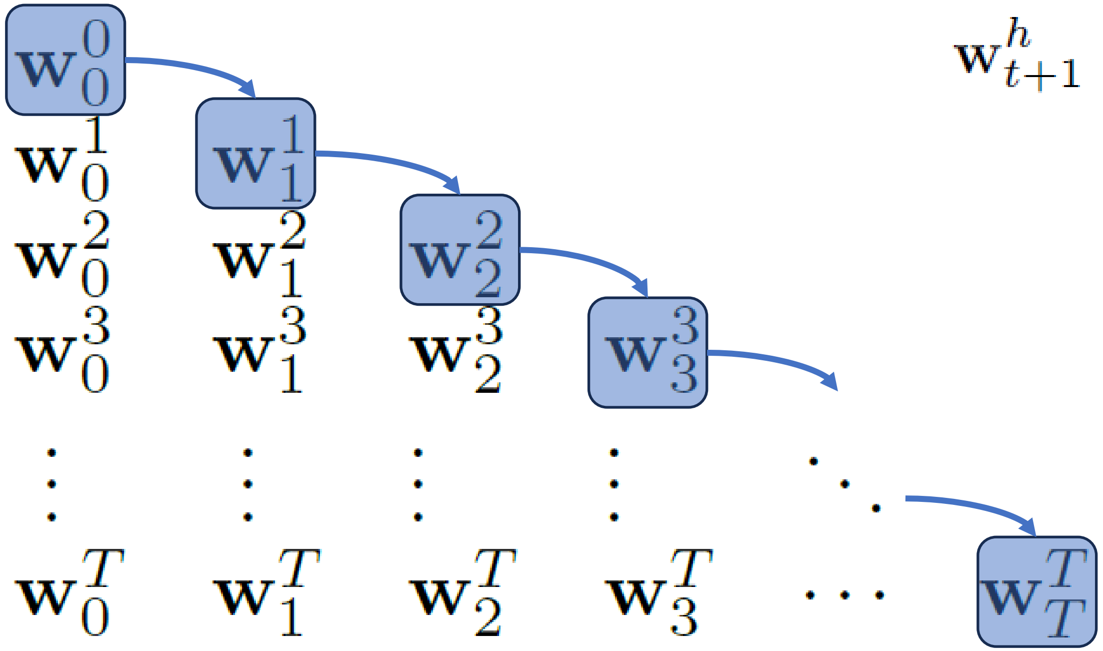
$$\mathbf{w}_{t+1}^h \doteq \mathbf{w}_t^h + \alpha \left[G_{t:h}^\lambda - \hat{v}(S_t,\mathbf{w}_t^h) \right] \nabla \hat{v}(S_t,\mathbf{w}_t^h)$$

$$\mathbf{w}_t \doteq \mathbf{w}_t^t$$

Only the weight vectors on the diagonal are really needed

True online TD(λ)

- Is there a way to invert the forward-view algorithm to produce an efficient backward-view algorithm using eligibility traces?


$\mathbf{w}_0^0$
$\mathbf{w}_0^1$ $\mathbf{w}_1^1$
$\mathbf{w}_0^2$ $\mathbf{w}_1^2$ $\mathbf{w}_2^2$
$\mathbf{w}_0^3$ $\mathbf{w}_1^3$ $\mathbf{w}_2^3$ $\mathbf{w}_3^3$
$\vdots$ $\vdots$ $\vdots$ $\vdots$ $\ddots$
$\mathbf{w}_0^T$ $\mathbf{w}_1^T$ $\mathbf{w}_2^T$ $\mathbf{w}_3^T$ $\cdots$ $\mathbf{w}_T^T$

$$\mathbf{w}_{t+1}^h \doteq \mathbf{w}_t^h + \alpha \left[G_{t:h}^\lambda - \hat{v}(S_t,\mathbf{w}_t^h) \right] \nabla \hat{v}(S_t,\mathbf{w}_t^h)$$

$$\mathbf{w}_t \doteq \mathbf{w}_t^t$$

Only the weight vectors on the diagonal are really needed

Find a compact, efficient way of computing each w_t from the one before

True online TD(λ)

- In a linear case:

$$\hat{v}(s,\mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$$

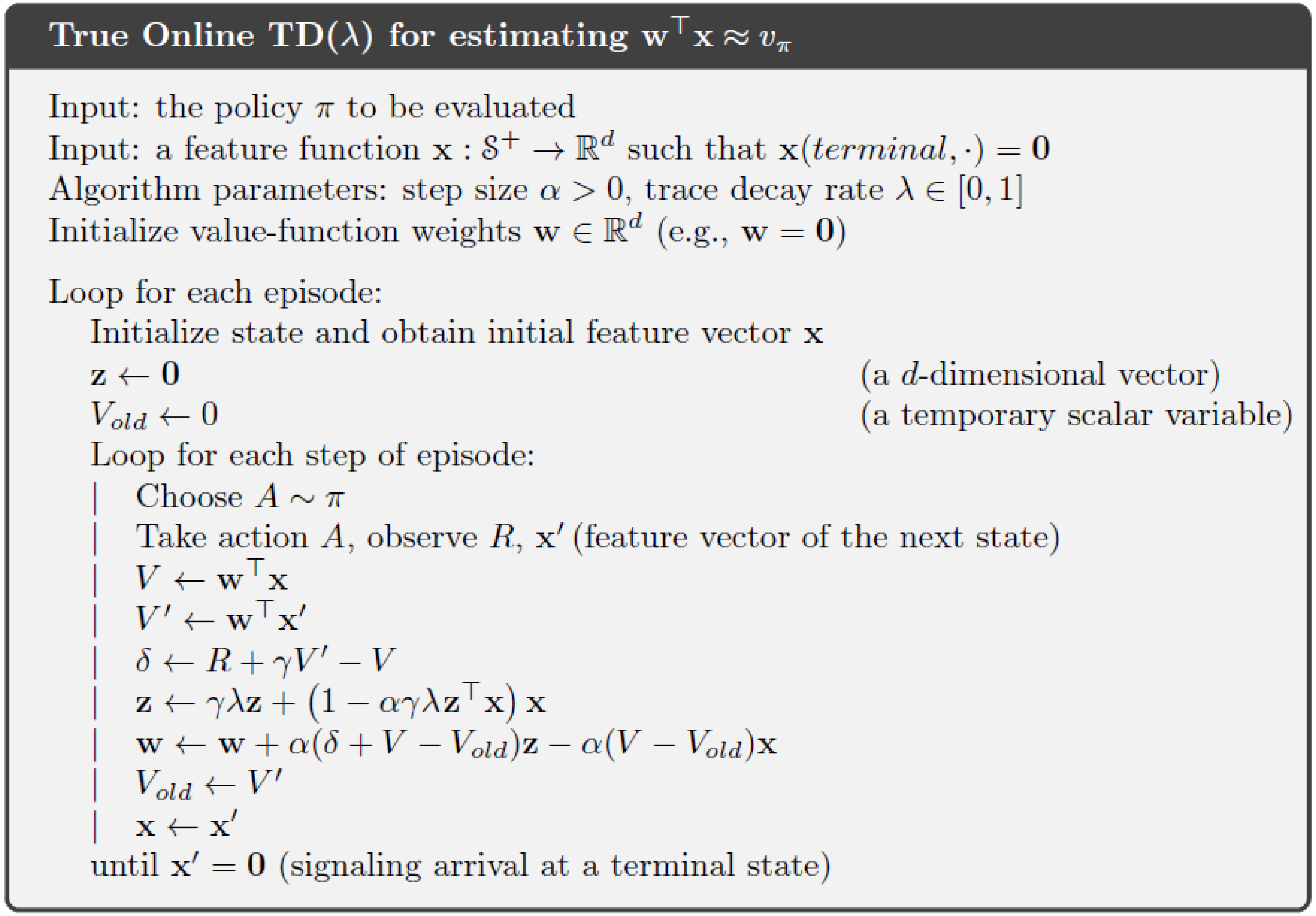
- Weight update for true online TD(λ):

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t + \alpha \left(\mathbf{w}_t^\top \mathbf{x}_t - \mathbf{w}_{t-1}^\top \mathbf{x}_t \right) \left(\mathbf{z}_t - \mathbf{x}_t \right)$$

- Eligibility trace is defined by:

$$\mathbf{z}_t \doteq \gamma\lambda \mathbf{z}_{t-1} + \left(1 - \alpha\gamma\lambda \mathbf{z}_{t-1}^\top \mathbf{x}_t \right) \mathbf{x}_t \longleftarrow \boxed{\text{Called the } \textit{Dutch trace}}$$

Pseudocode

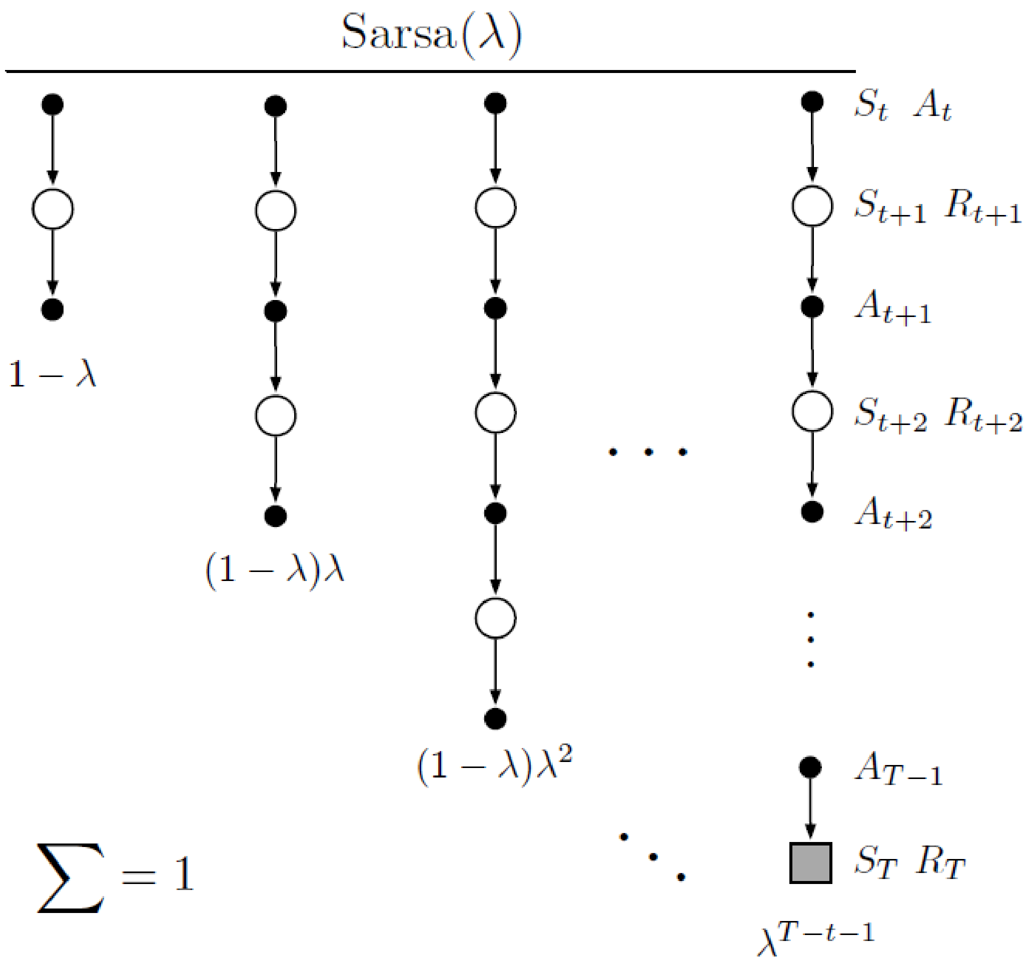


```
True Online TD($\lambda$) for estimating $\mathbf{w}^\top \mathbf{x} \approx v_\pi$

Input: the policy $\pi$ to be evaluated
Input: a feature function $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ 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)
```

Sarsa(λ)


Sarsa(λ)
$S_t \ A_t$
$S_{t+1} \ R_{t+1}$
A_{t+1}
$S_{t+2} \ R_{t+2}$
A_{t+2}
A_{T-1}
$S_T \ R_T$
$1-\lambda$
$(1-\lambda)\lambda$
$(1-\lambda)\lambda^2$
λ^{T-t-1}
$\sum = 1$

Sarsa(λ)

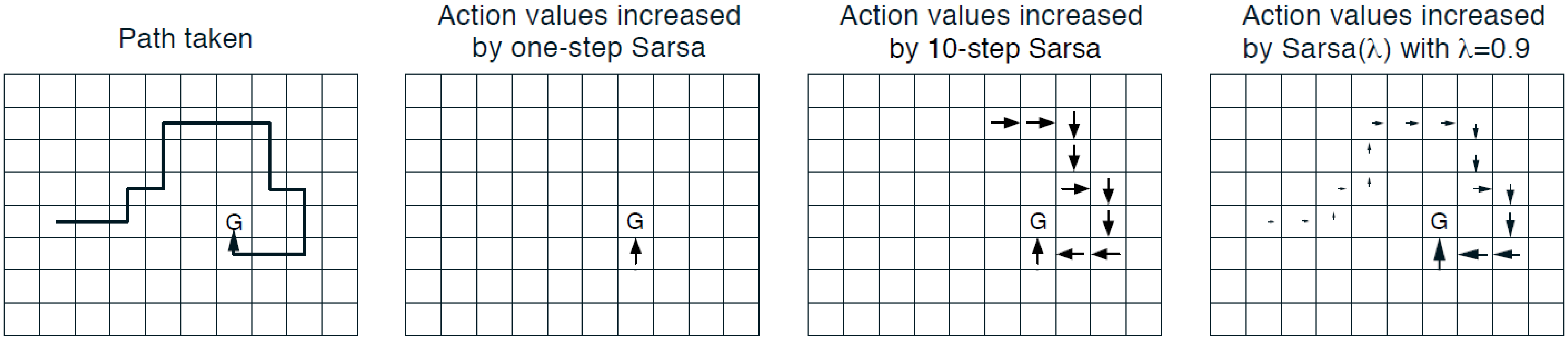
- Reward: $$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \underline{\hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1})}$$

- Weight update: $$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[G_t^\lambda - \underline{\hat{q}(S_t, A_t \mathbf{w}_t)} \Big] \underline{\nabla \hat{q}(S_t, A_t, \mathbf{w}_t)}$$
$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \, \delta_t \, \mathbf{z}_t$$

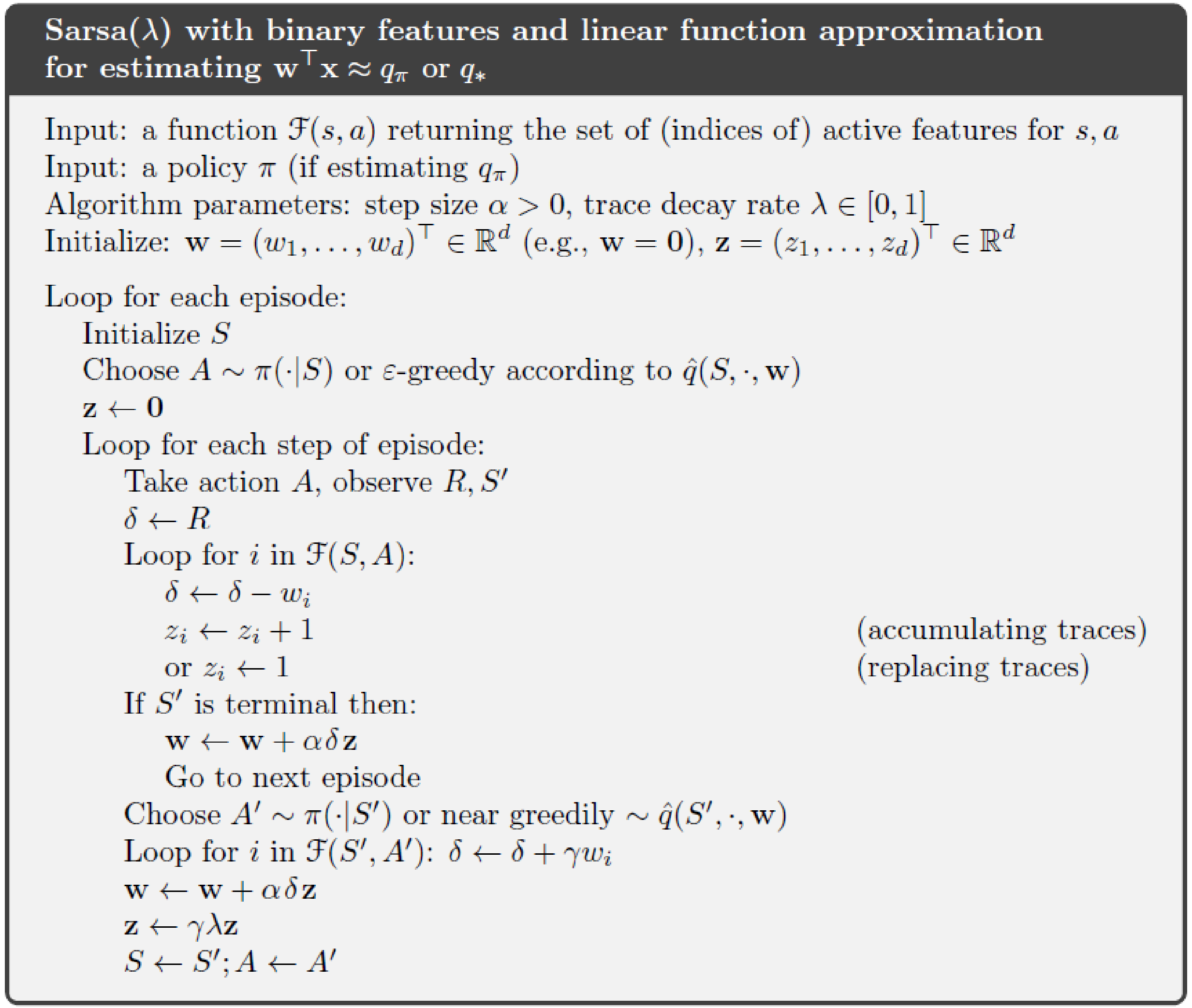
- Error: $$\delta_t \doteq R_{t+1} + \gamma \underline{\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)} - \underline{\hat{q}(S_t, A_t, \mathbf{w}_t)}$$

- Eligibility trace: $$\mathbf{z}_{-1} \doteq \mathbf{0},$$
$$\mathbf{z}_t \doteq \gamma \lambda \mathbf{z}_{t-1} + \underline{\nabla \hat{q}(S_t, A_t, \mathbf{w}_t)}$$

Traces in a Gridworld


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

Pseudocode



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 $\pi$ (if estimating $q_\pi$)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize: $\mathbf{w} = (w_1,\dots,w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1,\dots,z_d)^\top \in \mathbb{R}^d$

Loop for each episode:
   Initialize $S$
   Choose $A \sim \pi(\cdot|S)$ or $\varepsilon$-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 near greedily $\sim \hat{q}(S',\cdot,\mathbf{w})$
      Loop for $i$ in $\mathcal{F}(S',A')$: $\delta \leftarrow \delta + \gamma w_i$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
      $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
      $S \leftarrow S'; A \leftarrow A'$
```

ELTE | FACULTY OF INFORMATICS

Thank you for your attention!