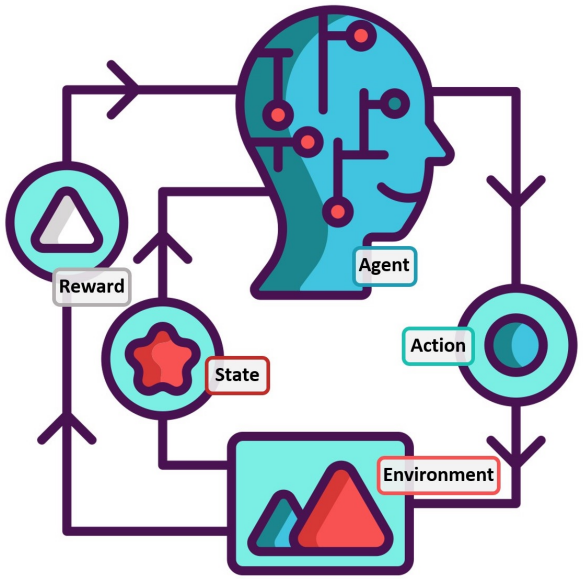
AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #13
TD(λ) & Value Function Approximation

Gian Antonio Susto



Announcements before starting (1/4): 1st partial

- So far, so good…
- Results out probably next weekend
- 2 ‘tricky’ questions

➔ In small grid-world exercise, it was written ‘Reward (R): R(s, a, s') = -1 for every non-terminal transition.’ It was somehow misleading, so both cases where s’ = T (terminal state) R(s, a, T) = -1 or R(s, a, T) = 0 were considered right

➔ The impact of the hyperparameter in Gradient Bandit Algorithm was not discussed before in class…

Gradient Bandit Algorithms

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

- Let's start with all preferences equal to each other: (ie. $H_1(a) = 0 \; \forall \; a$)

- At each step, after selecting action A_t we receive reward R_t and we update our action preferences as follows:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t$$

with:

α the step-size parameter

$\bar{R}_t$ the average of the rewards up to but not including time t; this terms serves as a baseline, if the A_t provides better results than the average then we should prefer this actions over the others!

Gradient Bandit Algorithms

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

- Let's start with all preferences equal to each other: (ie. $H_1(a) = 0 \; \forall \; a$)

- At each step, after selecting action A_t we receive reward R_t and we update our action preferences as follows:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t$$

with:

α the st

$\bar{R}_t$ the a

serves

then we

The higher the (a) the bigger the exploitation! Controls how quickly the action preferences are updated. A high (a) causes faster updates, making the policy more 'deterministic' (more Exploitation); a low (a) leads to slower updates and a smoother, more 'explorative' behavior.

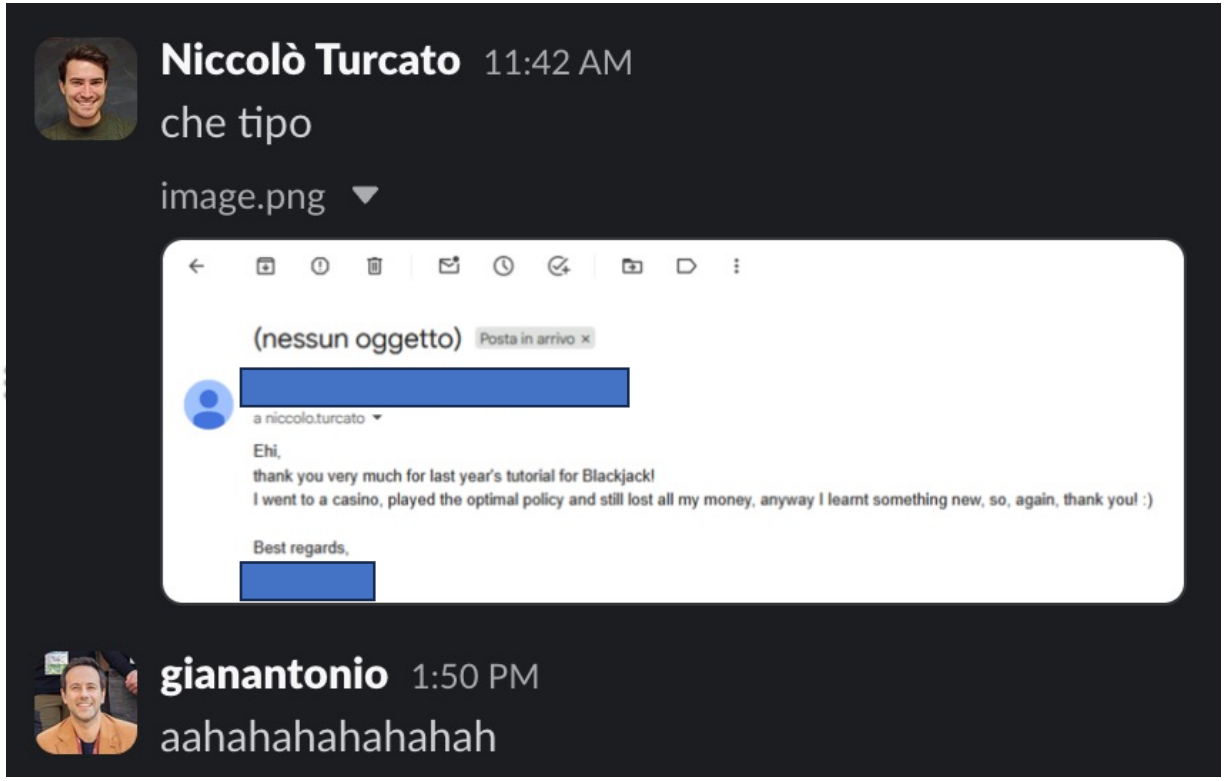
Announcements before starting (2/4): Importance Sampling Algorithm

- I’ve a revised version of the algorithm, however…

Announcements before starting (2/4): Importance Sampling Algorithm

- I've a revised version of the algorithm, however...
- Let's make things more interesting: the one who provide me the 'best' pseudo-code (correct and 'efficient') by this evening at 23:59, will get a bonus point on the 1st partial!

Announcements before starting (3/4): kudos to the TAs



(nessun oggetto) Posta in arrivo ×

[redacted]

a niccolo.turcato

Ehi,
thank you very much for last year's tutorial for Blackjack!
I went to a casino, played the optimal policy and still lost all my money, anyway I learnt something new, so, again, thank you! :)

Best regards,
[redacted]

Announcements before starting (4/4): Some ML Thesis

With Companies
- Breton (Computer Vision, Anomaly Detection)
- Galdi (LLM-based approaches)
- Lfoundry (Manufacturing Applications)
- Statwolf (LLM-based approaches)
- Unox (IoT Applications)
- Zamperla (IoT Applications)
- Zoppas (IoT Applications)

Without Companies: eXplainable AI, Continual Learning, Deep Learning, Unsupervised Learning

If interested, write me an email with CV & exams
We'll talk about RL thesis later in the course

Zoppas Industries
Heating Element Technologies

LFOUNDRY
A SMIC COMPANY

breton

GALDI

ZAMPERLA
THE AMUSEMENT GROUP

STATWOLF

Recap: MC vs. TD(0)

MC: no bias

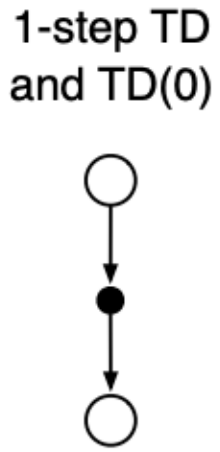
TD(0): low variance

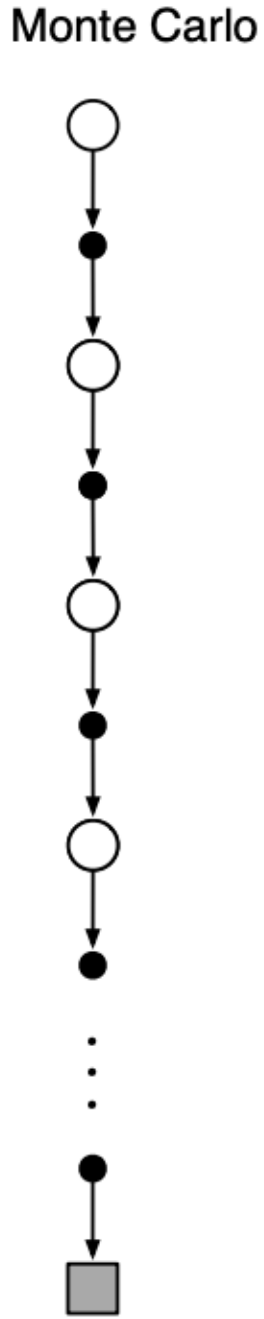
MC uses real return

TD(0) just rewards and bootstrap

MC updates just at the end of the episode

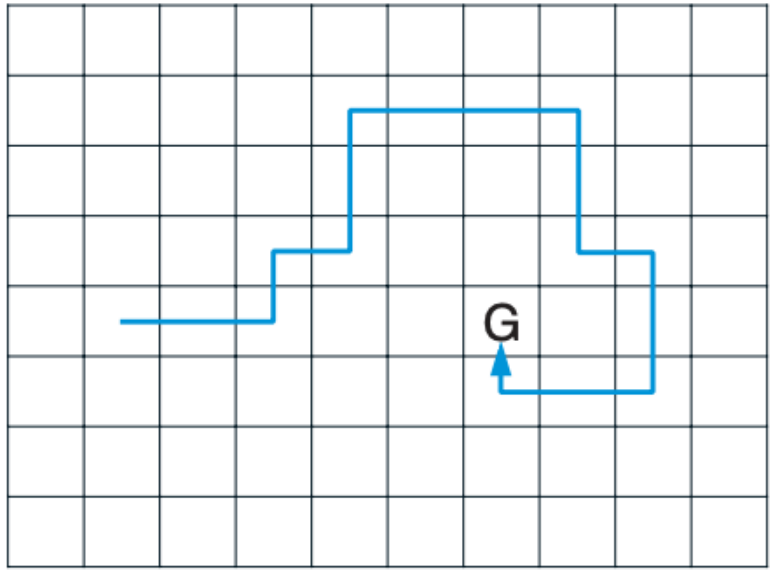
TD(0) updates along the way



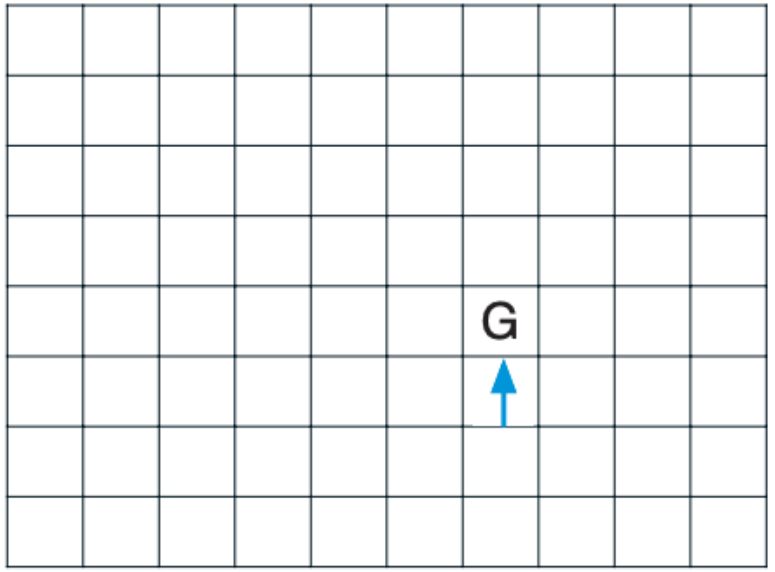


Recap: the credit assignment problem

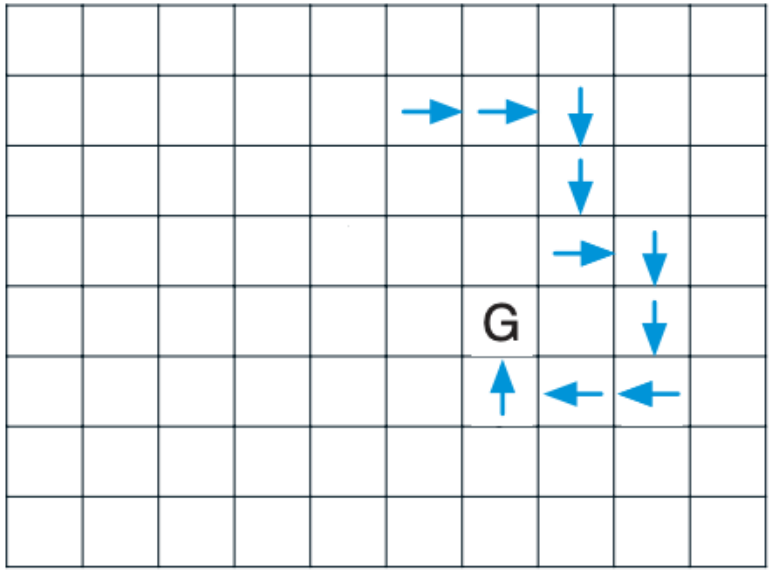
Path taken



Action values increased by one-step Sarsa

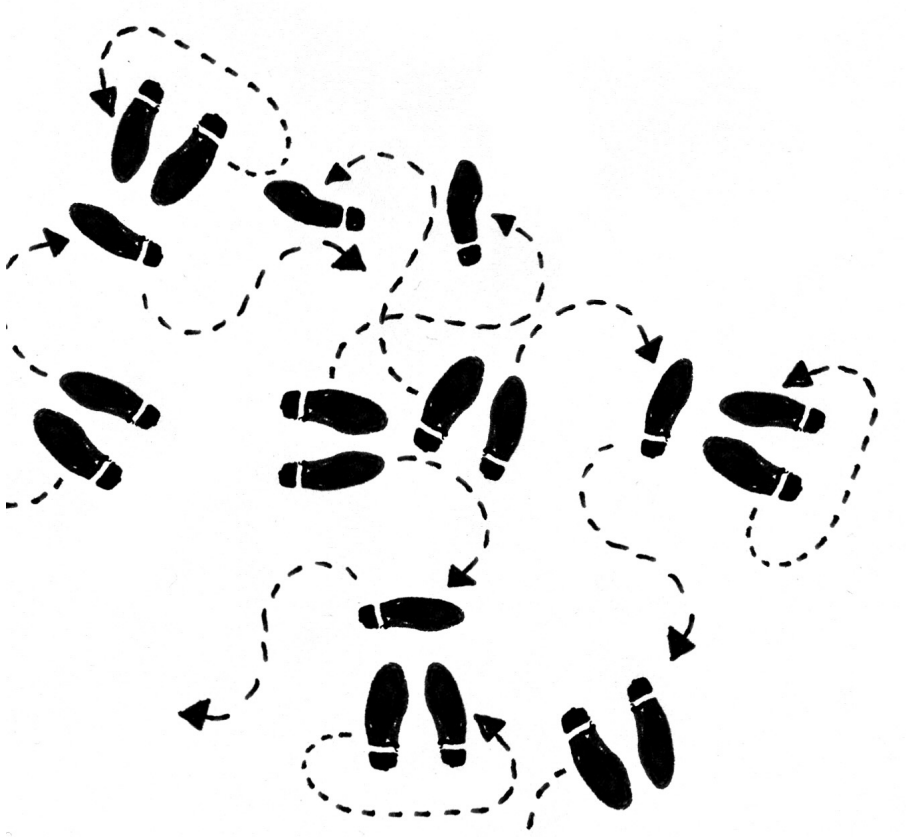


Action values increased by 10-step Sarsa



Recap: Approaches to balance Bias/Variance

n-step bootstrapping - chapter 7



TD(λ) - chapter 12

λ

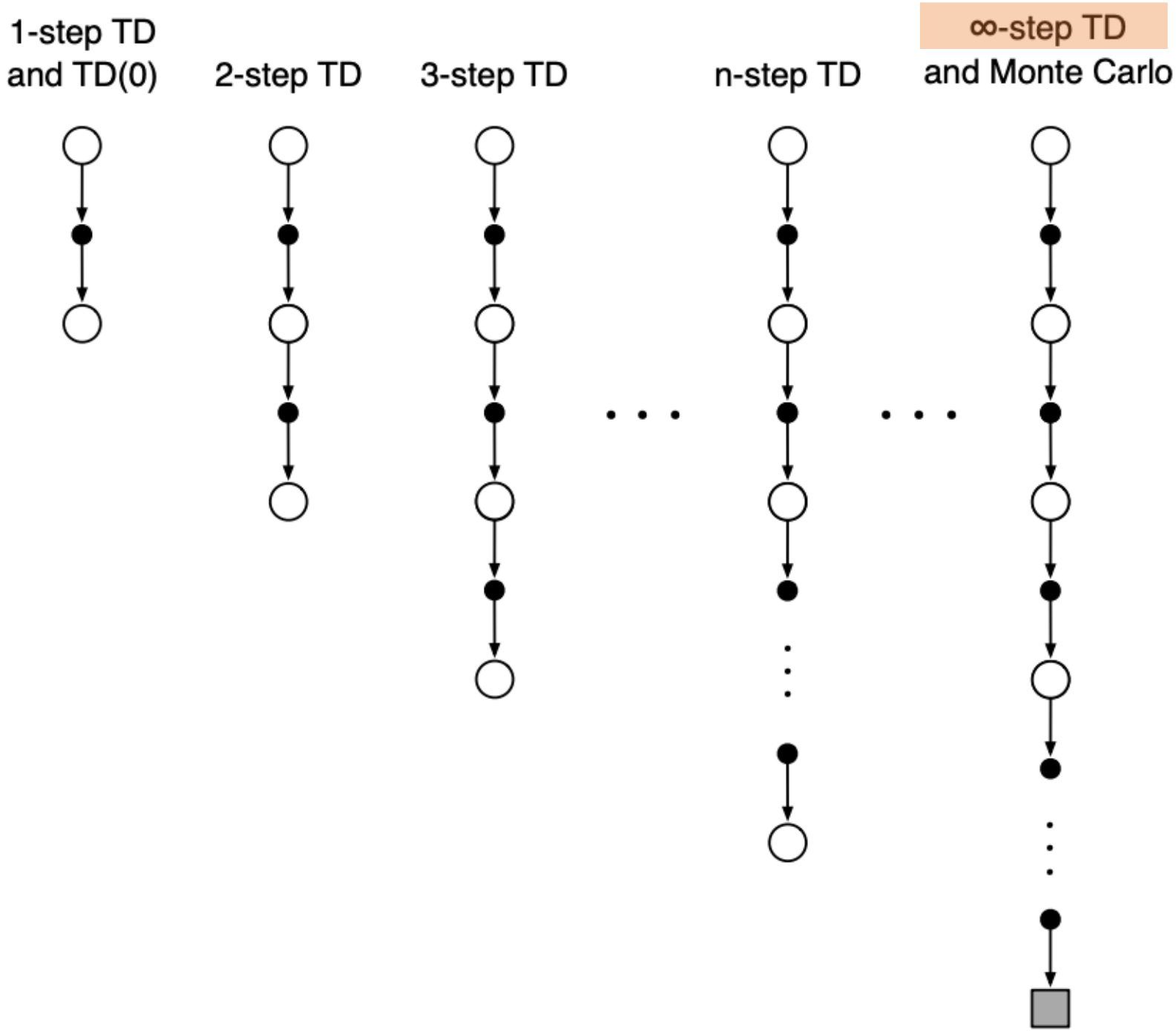
Recap: n-step bootstrapping

MC: low bias

TD(0): low variance

Why don't using a compromise between MC and TD(0)?

We can generalize by considering n-step TD!



Recap: *n*-step return

Let's define the n-step returns:

$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}) \\
n = 2 \quad & G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

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

A terminal state

Recap: *n*-step return

Where this is not defined? When there are less then *n* step before the terminal state!

Let's define the n-step returns:

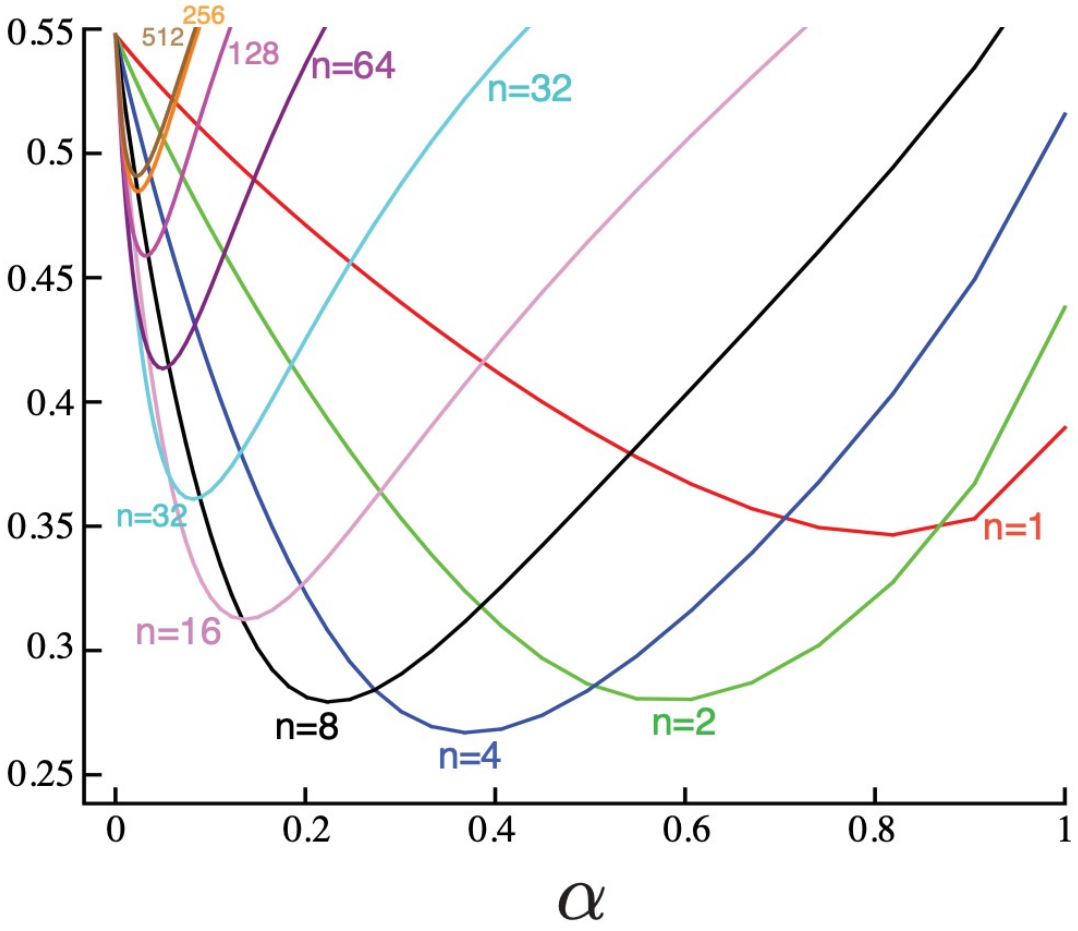
$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}) \\
n = 2 \quad & G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

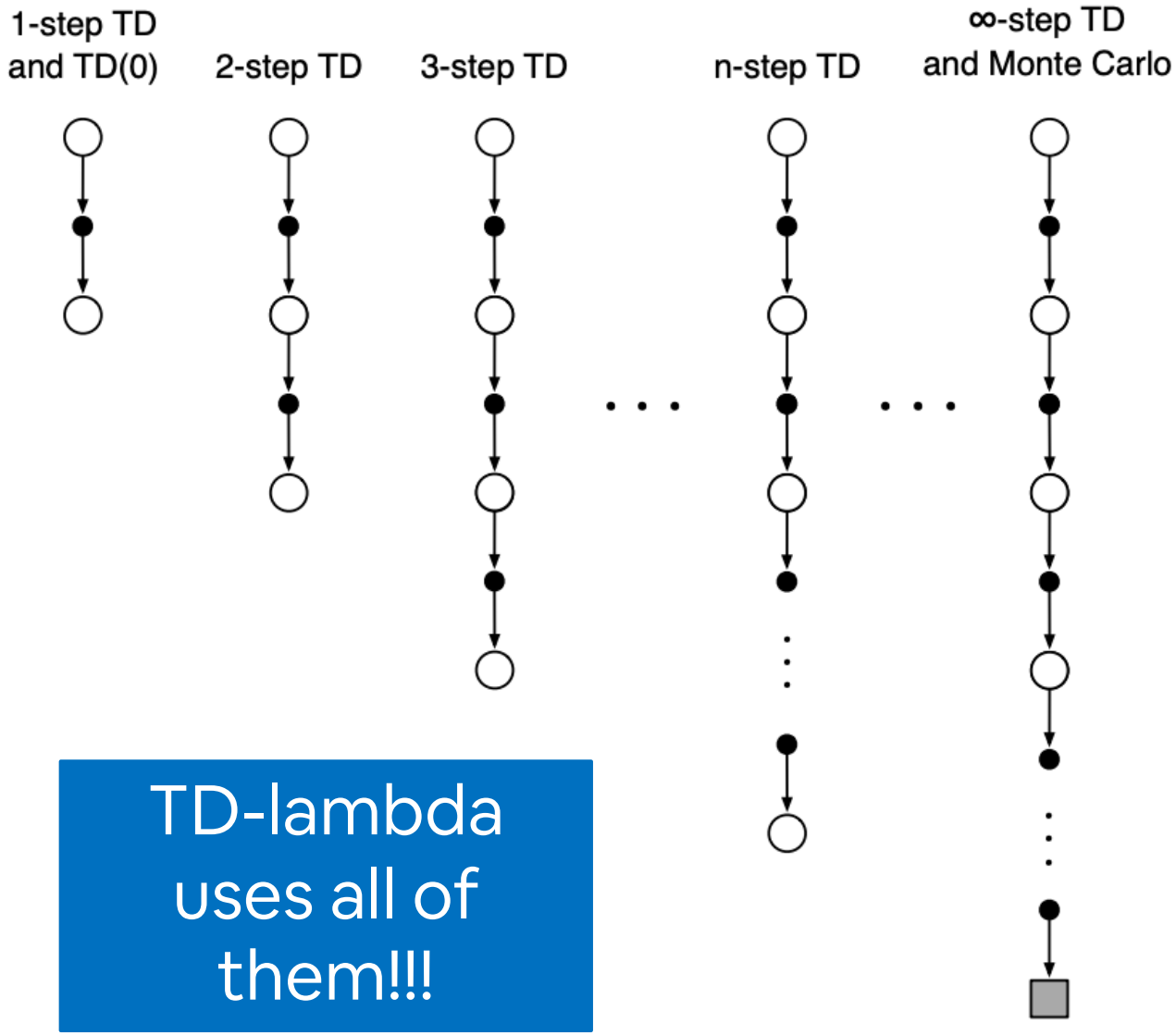
$$
G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})
$$

n-step TD learning will consider the following updates:

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

Recap: TD-lambda

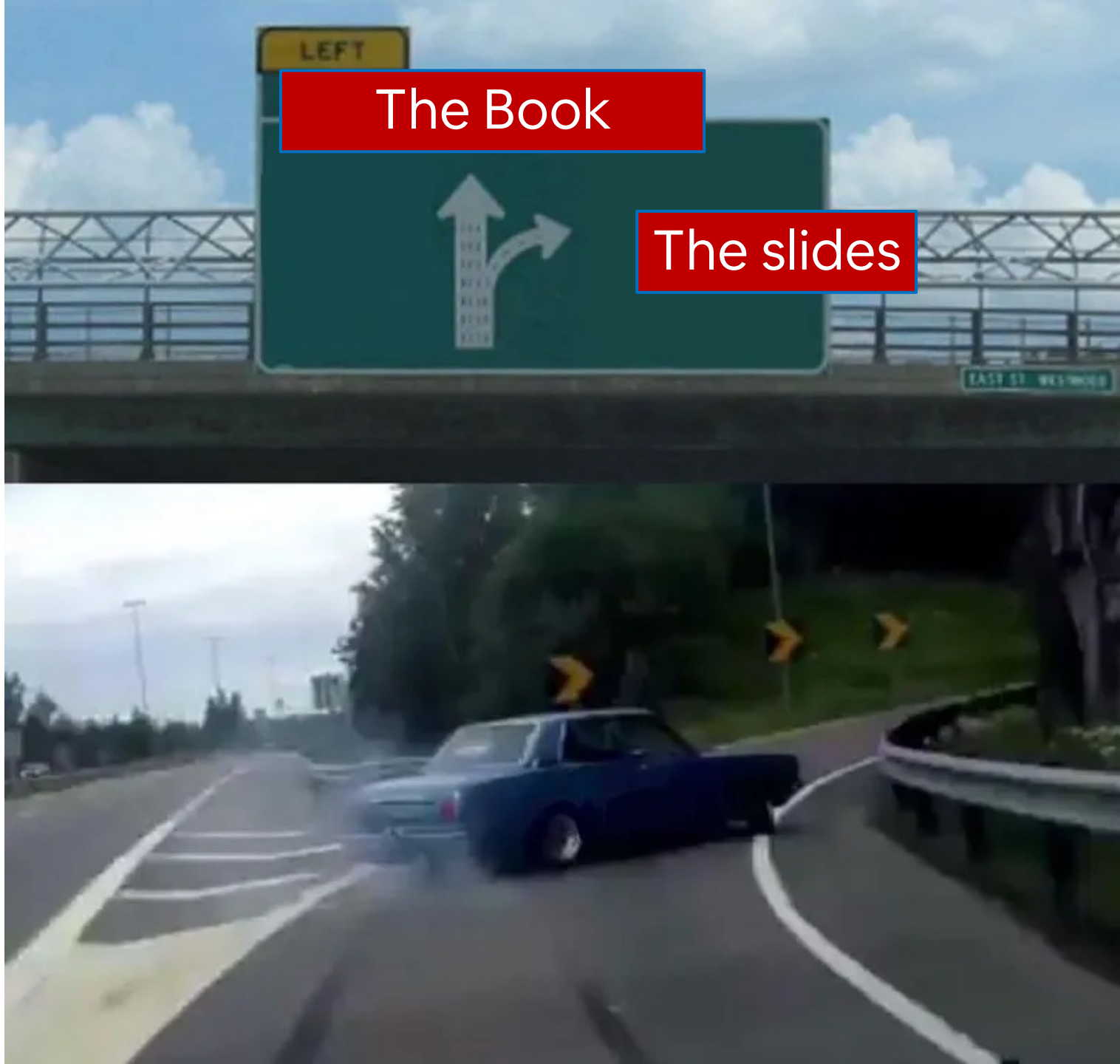




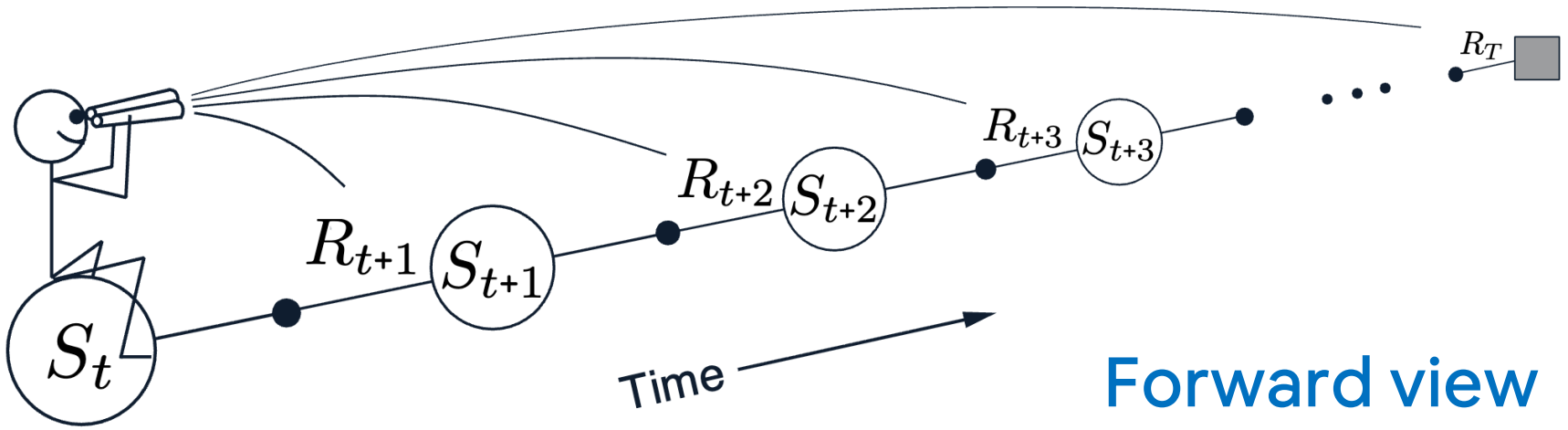
TD-lambda uses all of them!!!

Recap: Chapter 12

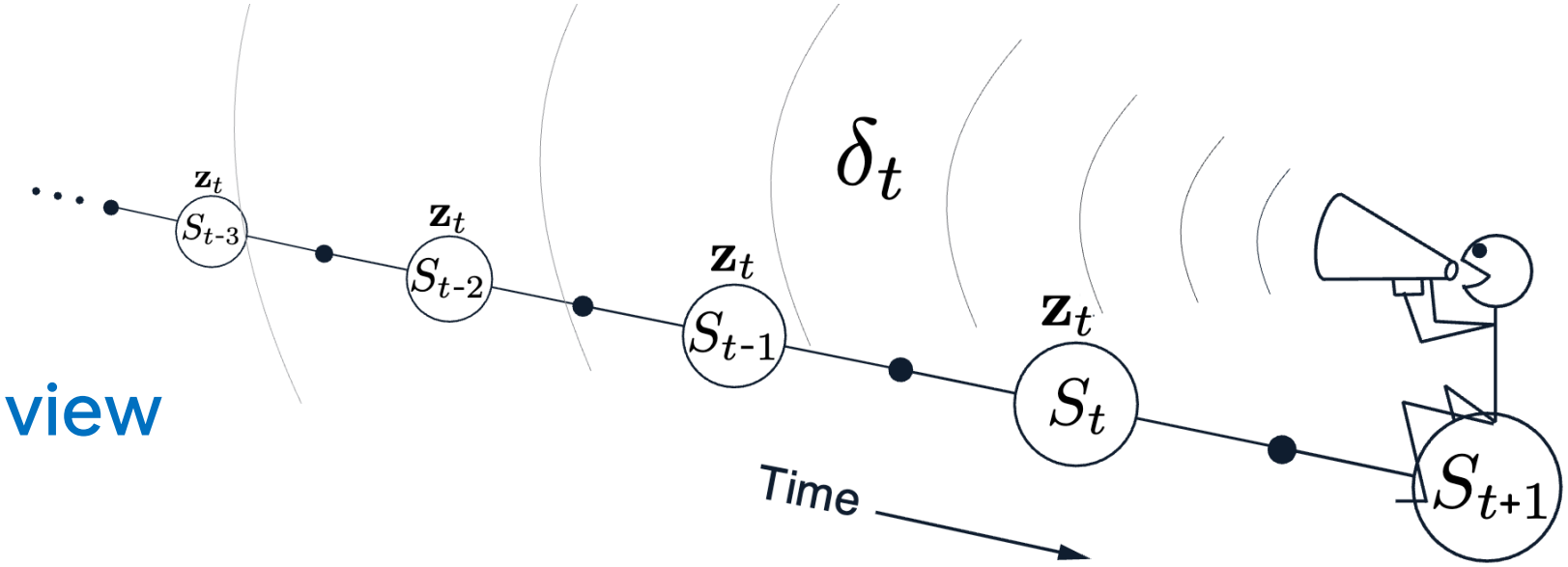
- The last version of the book described TD-lambda and *eligibility traces* (one of the main 'tool' in TD-lambda) in Chapter 12...
- ... but it is mixed with value function approximation VFA algorithms (in my opinion it is a bit confusing)
- For the moment, slides are the reference, after we do VFA (starting next week) the TD-lambda algorithms reported in the book will be clear



Recap: TD(λ) Forward vs Backward

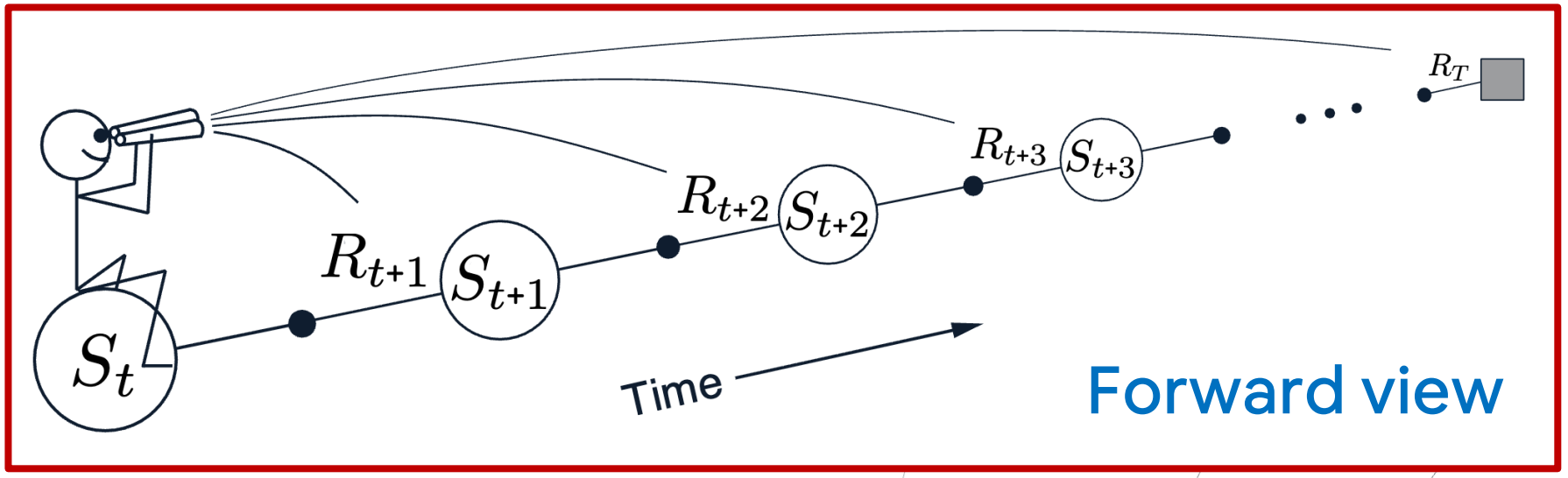


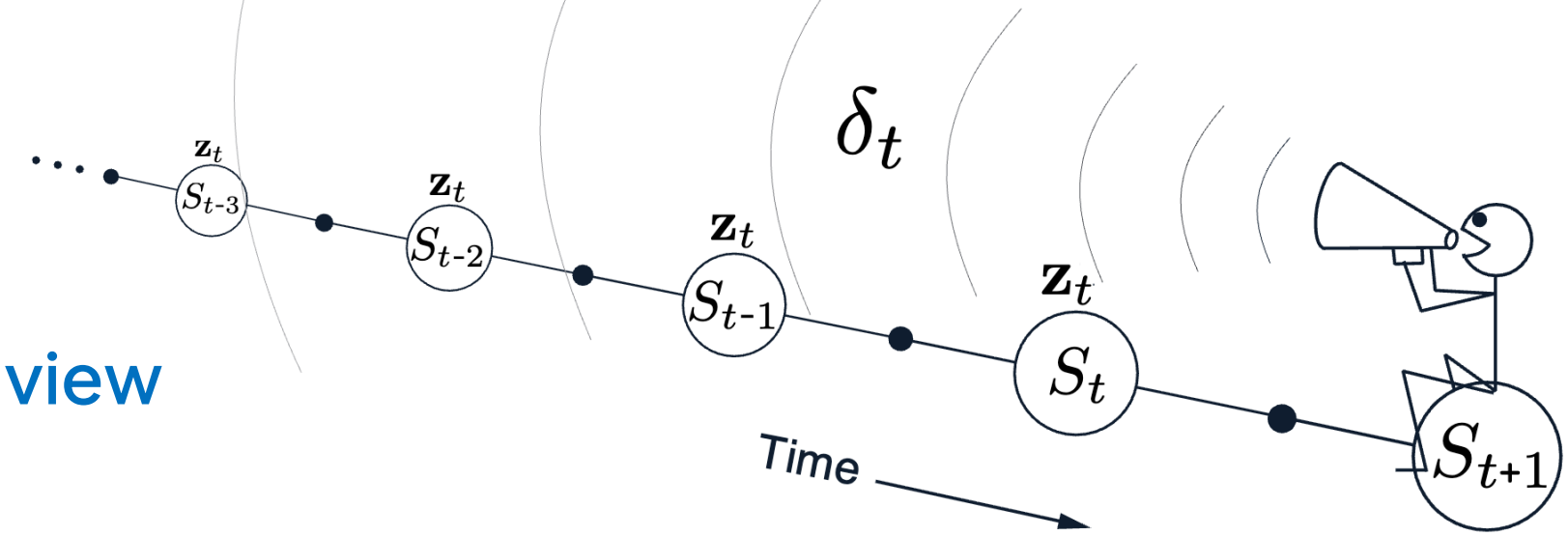
Forward view



Backward view

Recap: TD(λ) Forward vs Backward





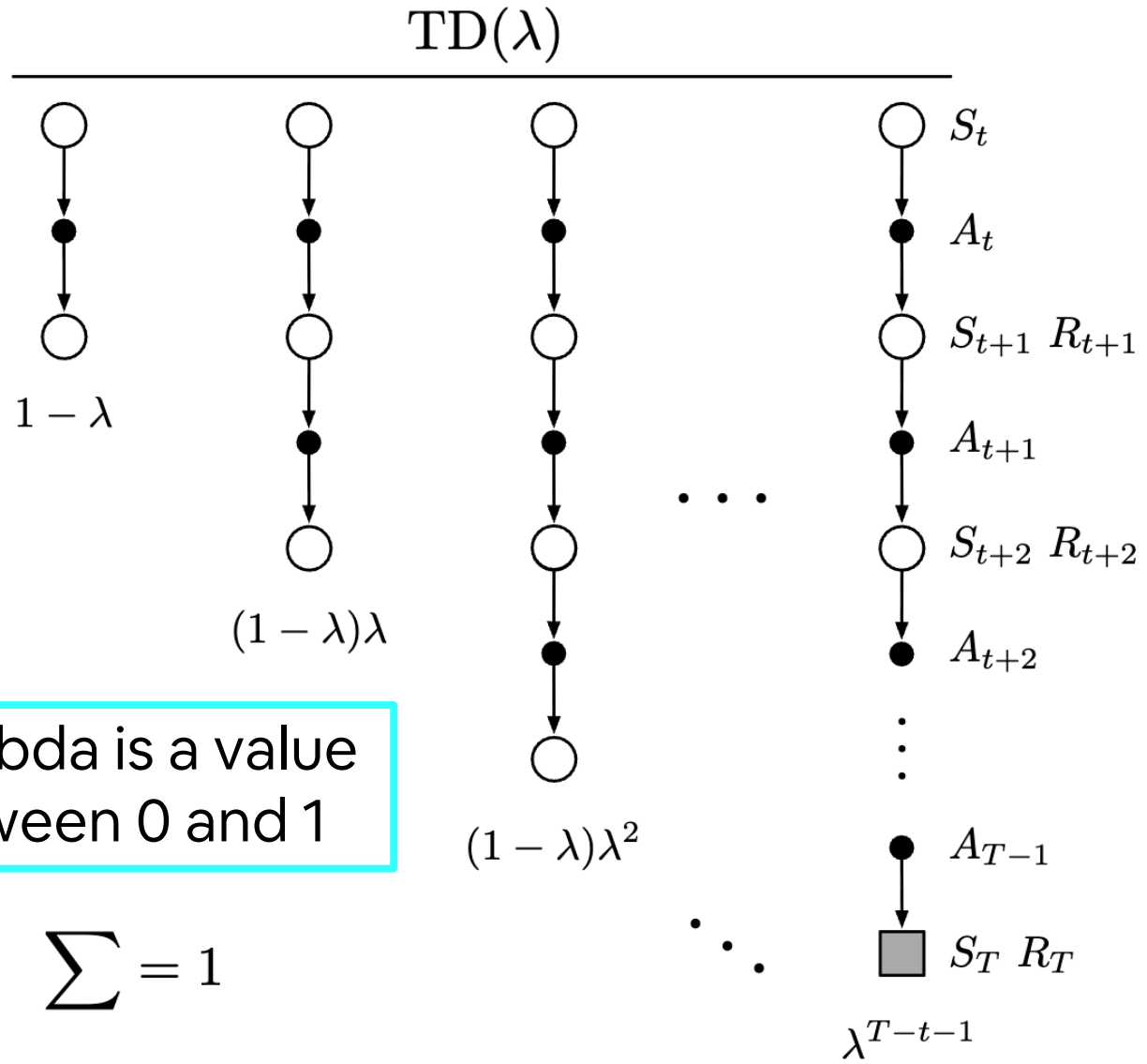
Recap: λ – return

- We consider the λ – return G_t^λ that combines all n-step returns
- It is a weighted sum of all the returns

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

- Instead of the 'usual' return we update

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$



Lambda is a value between 0 and 1

$$\sum = 1$$

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1-\lambda}$$

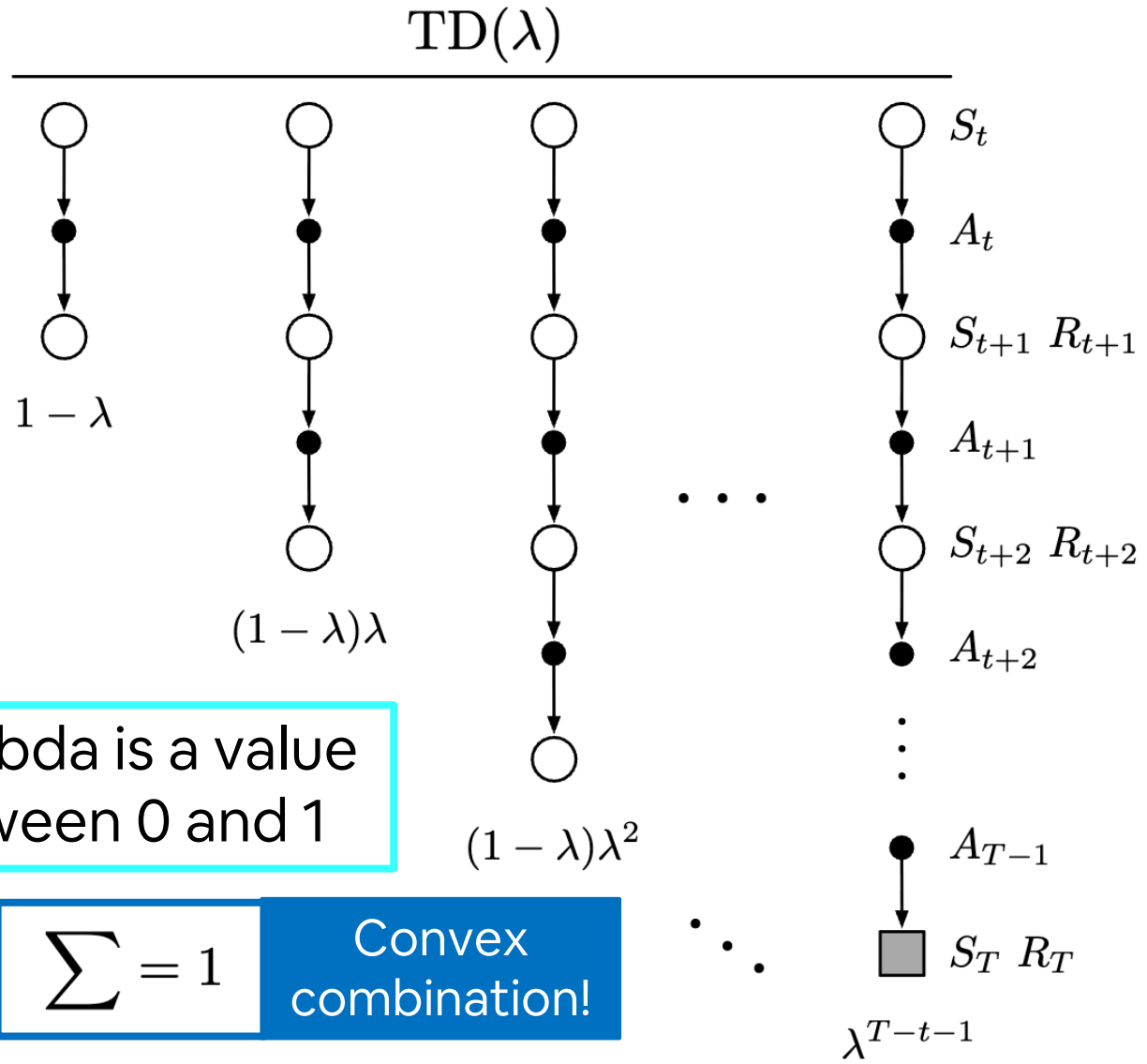
Recap: λ – return

- We consider the λ – return G_t^λ that combines all n-step returns
- It is a weighted sum of all the returns

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

- Instead of the 'usual' return we update

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$



Lambda is a value between 0 and 1

$\sum = 1$ Convex combination!

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1-\lambda}$$

Recap: Why TD(λ) Weighting Function?

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

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

<u>Why we weight the steps?</u>

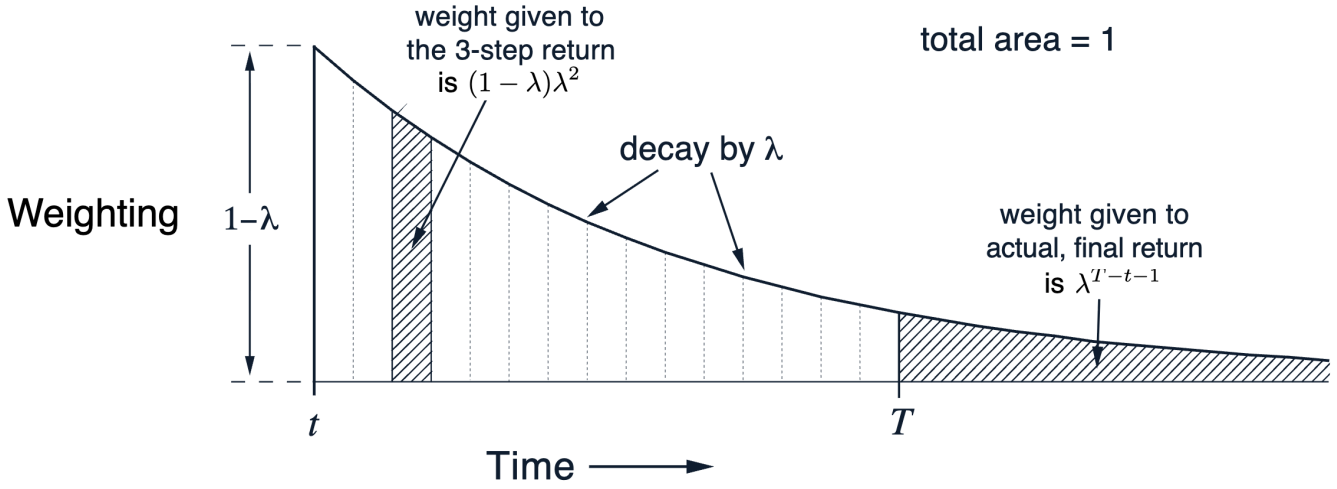
The weights form a geometric decay.

That means:

- 1-step return gets the highest weight

- 2-step, 3-step returns get smaller and smaller weights

This is intentional — it expresses how far into the future we trust our rollouts.



Recap: λ – return

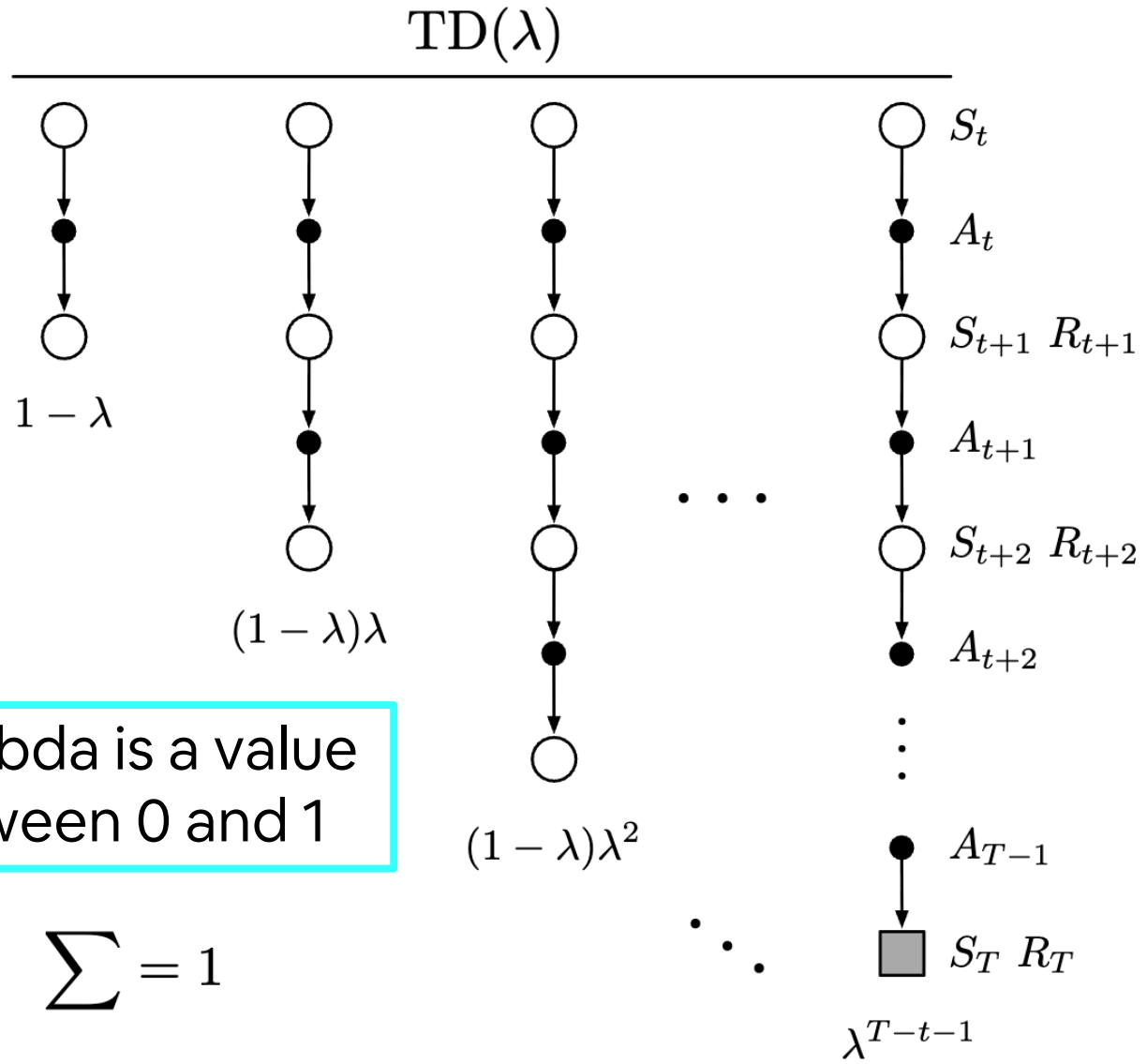
- We consider the λ – return G_t^λ that combines all n-step returns
- It is a weighted
the returns

For the sake of the argument, we were considering the continuous case

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

- Instead of the 'usual' return we update

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$



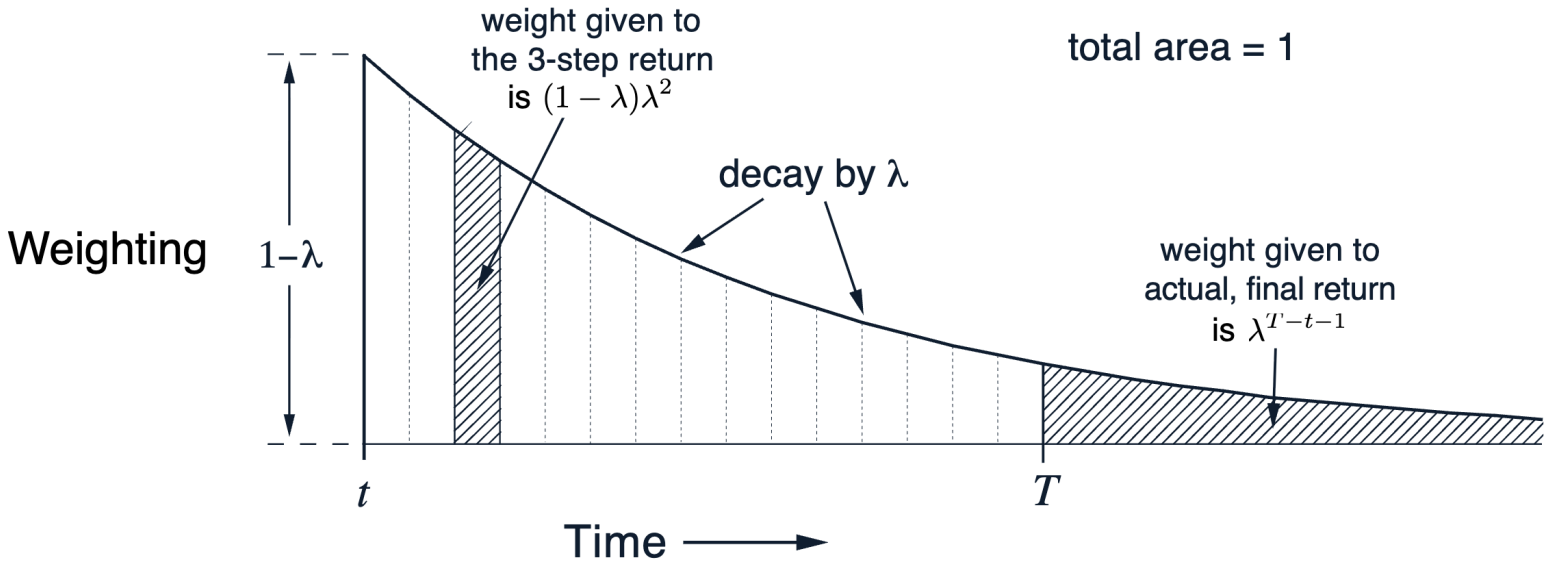
Lambda is a value between 0 and 1

$$\sum = 1$$

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1-\lambda}$$

TD(λ) Weighting Function

What if an episode terminates at T?

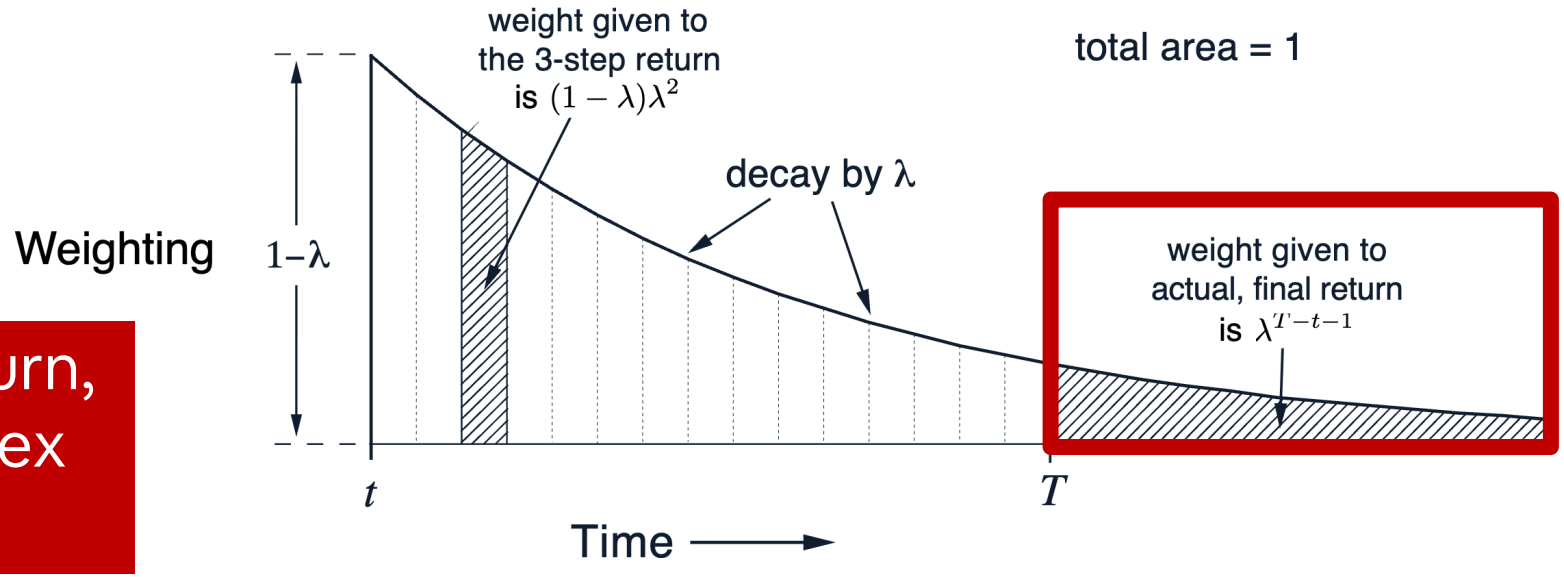


$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + (1-\lambda)\sum_{n=T-t}^{\infty}\lambda^{n-1}G_{t:t+n}$$

TD(λ) Weighting Function

What if an episode terminates at T?

We assign the actual return, and we still keep a convex combination!



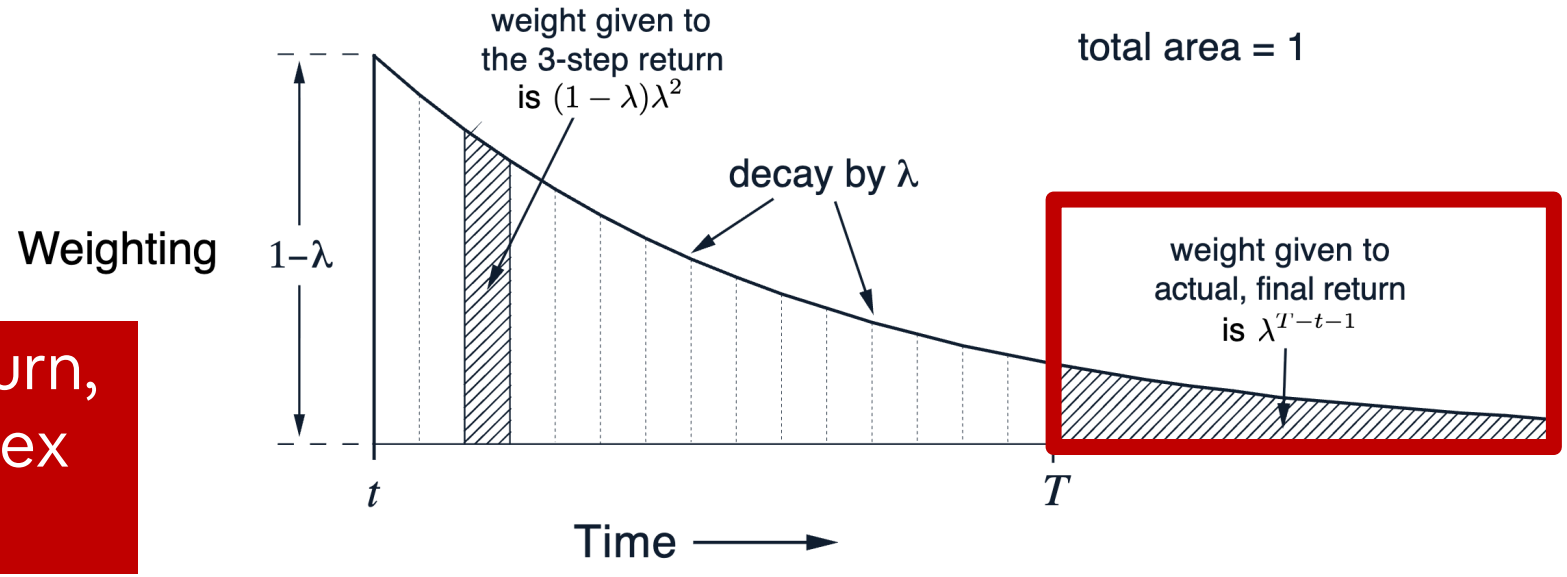
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + (1-\lambda)\sum_{n=T-t}^{\infty}\lambda^{n-1}G_{t:t+n}$$

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

TD(λ) Weighting Function

What if an episode terminates at T?

We assign the actual return, and we still keep a convex combination!



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

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

$$\lambda^{T-t-1}$$

Sum of geometric series starting at T-t

TD(λ) Weighting Function

What if an episode terminates at T?

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

We assign the actual return, and we still keep a convex combination!

Lambda return with non-continuous task

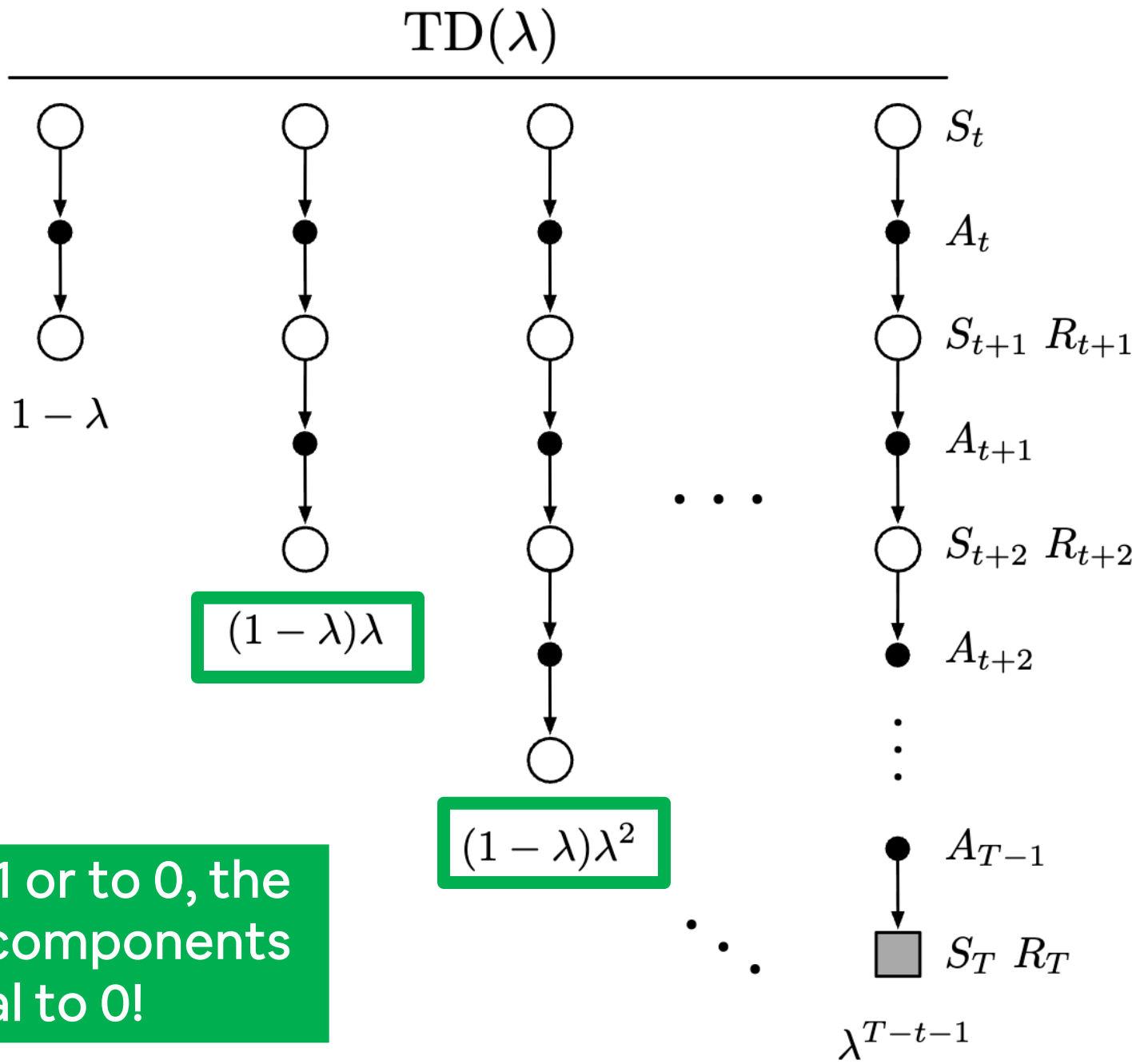
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + (1-\lambda)\sum_{n=T-t}^{\infty}\lambda^{n-1}G_{t:t+n}$$

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

λ^{T-t-1}

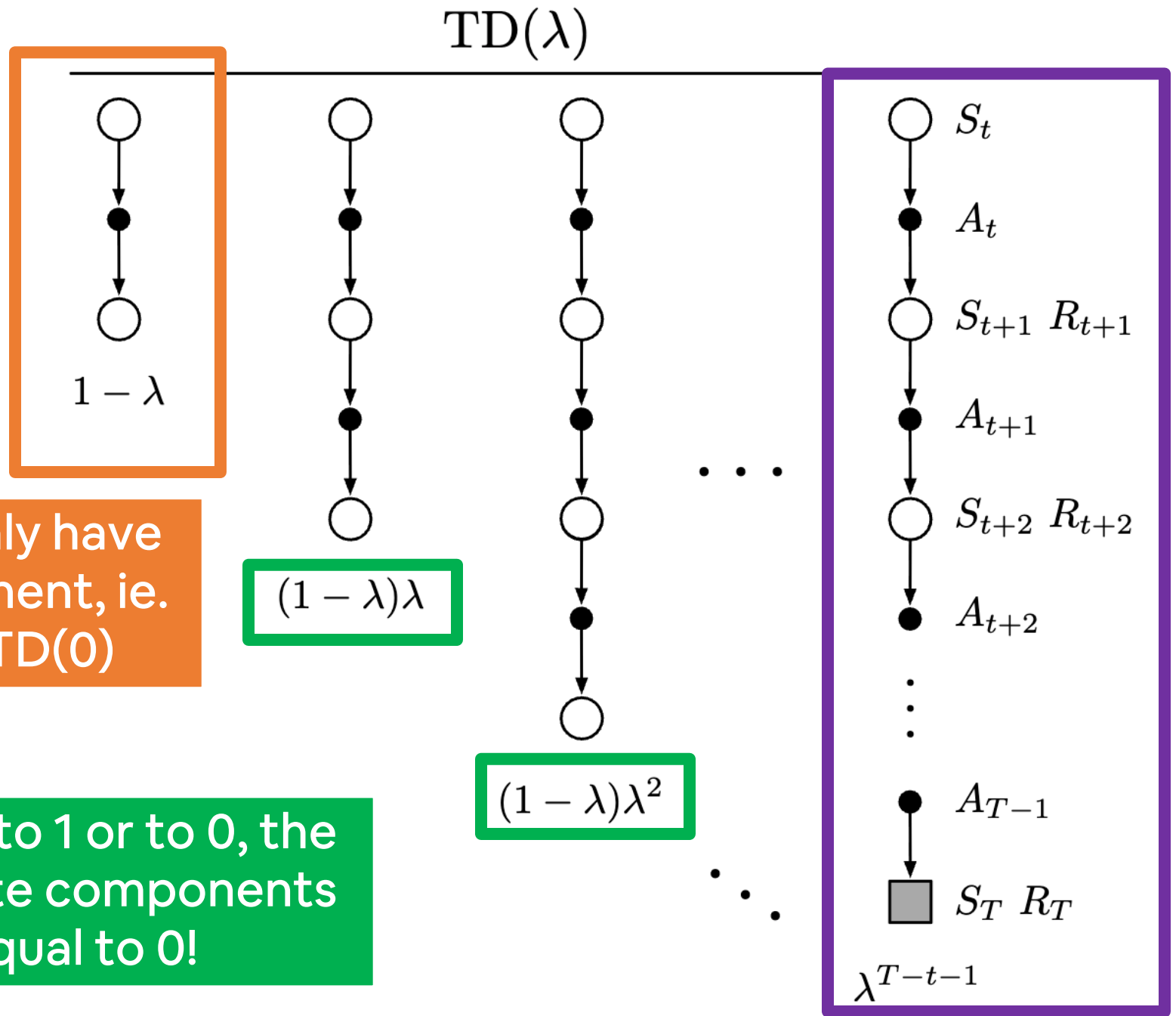
Sum of geometric series starting at T-t

TD(λ)



If λ is equal to 1 or to 0, the intermediate components are equal to 0!

TD(λ)

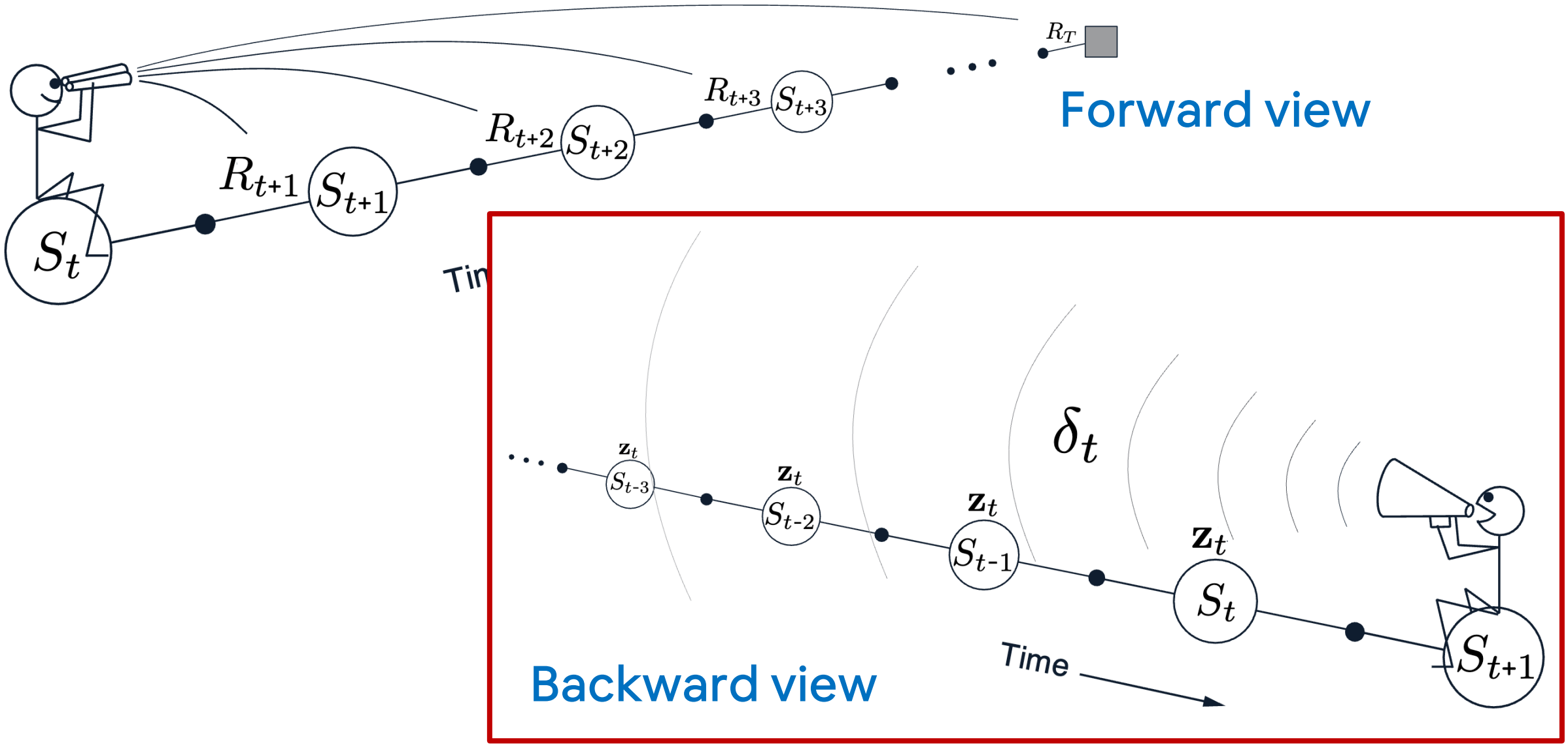


λ = 0, we only have this component, ie. we have TD(0)

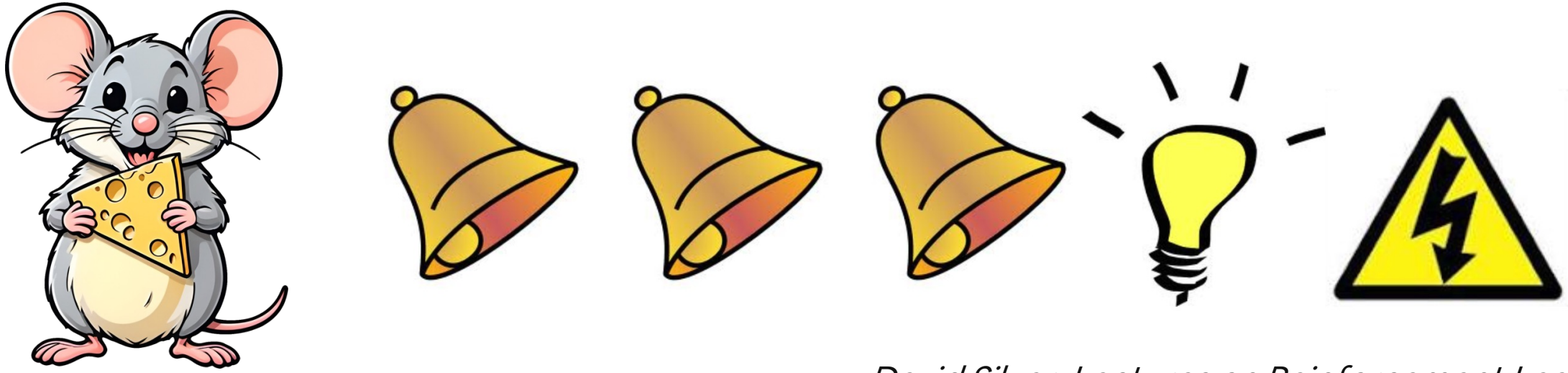
λ = 1, we only have this component, ie. we have MC

If λ is equal to 1 or to 0, the intermediate components are equal to 0!

Recap: TD(λ) Forward vs Backward



Recap: Eligibility traces



David Silver, Lectures on Reinforcement Learning

- Credit assignment problem: did bell or light cause shock?
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states

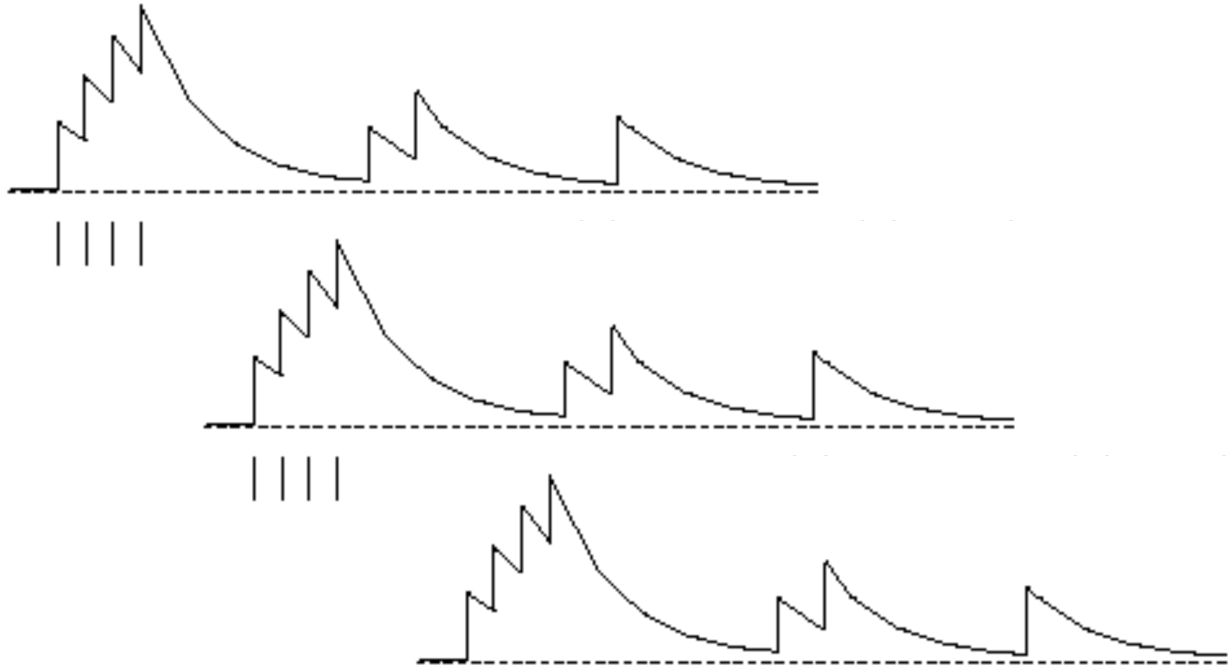
Recap: Eligibility traces (prediction)

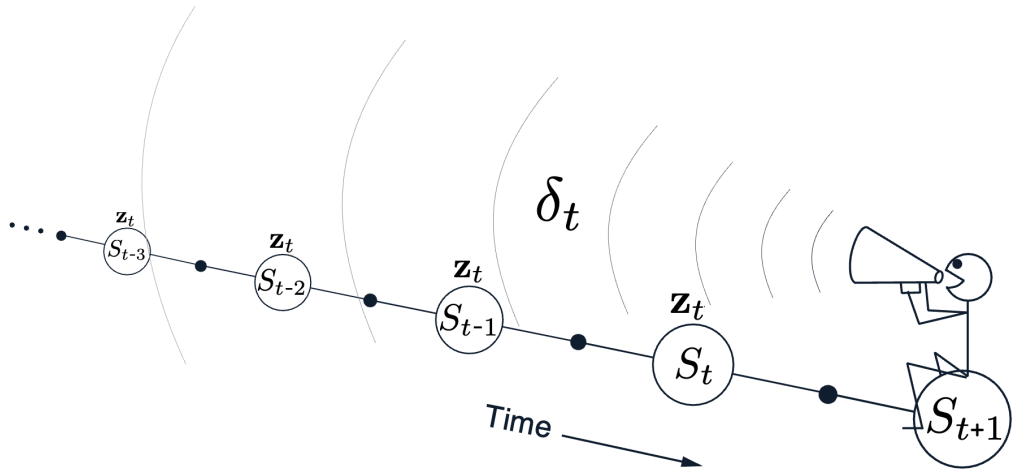
$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$

We have an update for all s (!) at each step (like in TD-0) that depends on the TD-error and the eligibility traces

$$\delta_t \;=\; R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$$

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$





Initialize $V(s)$ arbitrarily (but set to 0 if s is terminal)

Repeat (for each episode):

We are considering the prediction problem

  Initialize $E(s) = 0$, for all $s \in \mathcal{S}$

  Initialize S

  Repeat (for each step of episode):

    $A \leftarrow$ action given by π for S

    Take action A, observe reward, R, and next state, S'

    $\delta \leftarrow R + \gamma V(S') - V(S)$

    $E(S) \leftarrow E(S) + 1$ (accumulating traces)

    For all $s \in \mathcal{S}$:

      $V(s) \leftarrow V(s) + \alpha \delta E(s)$

      $E(s) \leftarrow \gamma \lambda E(s)$

    $S \leftarrow S'$

  until S is terminal

Initialize $V(s)$ arbitrarily (but set to 0 if s is terminal)

Repeat (for each episode):

Initialize $E(s) = 0$, for all $s \in \mathcal{S}$

Initialize S

Repeat (for each step of episode):

$A \leftarrow$ action given by π for S

Take action A, observe reward, R, and next state, S'

$\delta \leftarrow R + \gamma V(S') - V(S)$

$E(S) \leftarrow E(S) + 1$ (accumulating traces)

For all $s \in \mathcal{S}$:

$V(s) \leftarrow V(s) + \alpha \delta E(s)$

$E(s) \leftarrow \gamma \lambda E(s)$

$S \leftarrow S'$

until S is terminal

Eligibility traces are reinitialized at every episode

Eligibility traces are 'decayed' at each step, but if one state is visited, we get a +1

Initialize $V(s)$ arbitrarily (but set to 0 if s is terminal)

Repeat (for each episode):

Initialize $E(s) = 0$, for all $s \in \mathcal{S}$

Initialize S

Repeat (for each step of episode):

$A \leftarrow$ action given by π for S

Take action A, observe reward, R, and next state, S'

$\delta \leftarrow R + \gamma V(S') - V(S)$

$E(S) \leftarrow E(S) + 1$ (accumulating traces)

For all $s \in \mathcal{S}$:

$V(s) \leftarrow V(s) + \alpha \delta E(s)$

$E(s) \leftarrow \gamma \lambda E(s)$

$S \leftarrow S'$

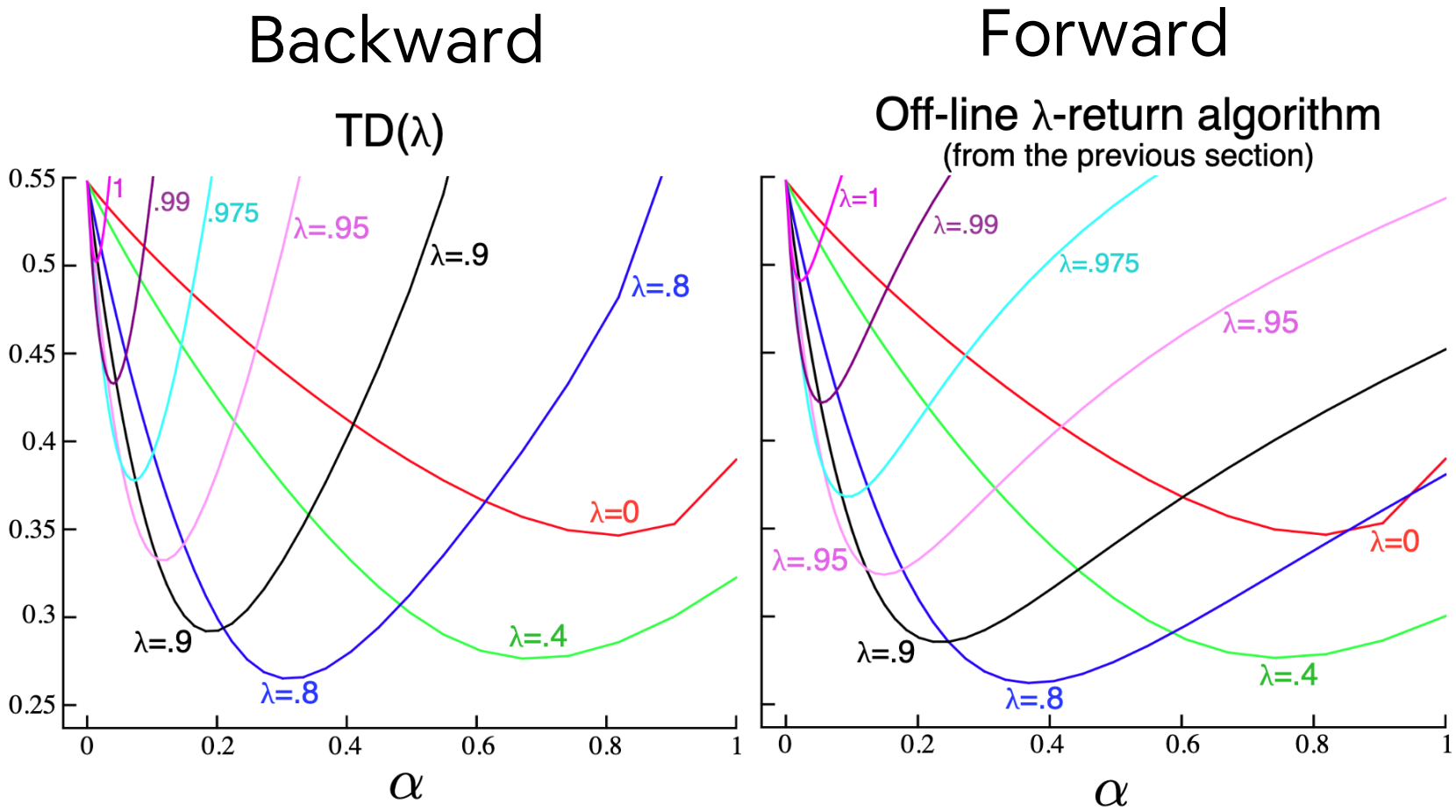
until S is terminal

This is like TD(0), but we need to consider the eligibility traces

This is done for ALL STATES!!!

Forward vs Backward view (prediction) - offline

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



Off-line (after we have completed the episode) we have equivalence of the two approaches... but backward view works without waiting the end of the episode!

Forward & Backward TD(λ) – Online vs Offline updates

Online updates:

TD(λ) updates are applied online at each step within episode

$$V(S) \leftarrow V(S) + \Delta V_t(S) \qquad \Delta V_t(S) = \alpha \delta_t E_t(s)$$

Offline updates:

- Updates are accumulated within episode...
- ... but applied 'in batch' at the end of episode

$$V(S) \leftarrow V(S) + \sum_{t=1}^{T} \Delta V_t(S)$$

Forward & Backward TD(λ) – Offline equivalence (optional)

Theorem

The sum of offline updates is identical for forward-view and and backward-view TD(λ)

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t^{\lambda} - V(S_t) \right) \mathbf{1}(S_t = s)$$

Forward & Backward TD(λ) – Offline equivalence (optional)

Theorem

The sum of offline updates is identical for forward-view and and backward-view TD(λ)

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t^{\lambda} - V(S_t) \right) \mathbf{1}(S_t = s)$$

Proof: slides of Prof. Carli, RL course 2024-25

Forward & Backward TD(λ) – Online equivalence (optional)

- Forward and backward-view TD(λ) are slightly different
- TD(λ) eligibility trace discounts time since visit

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

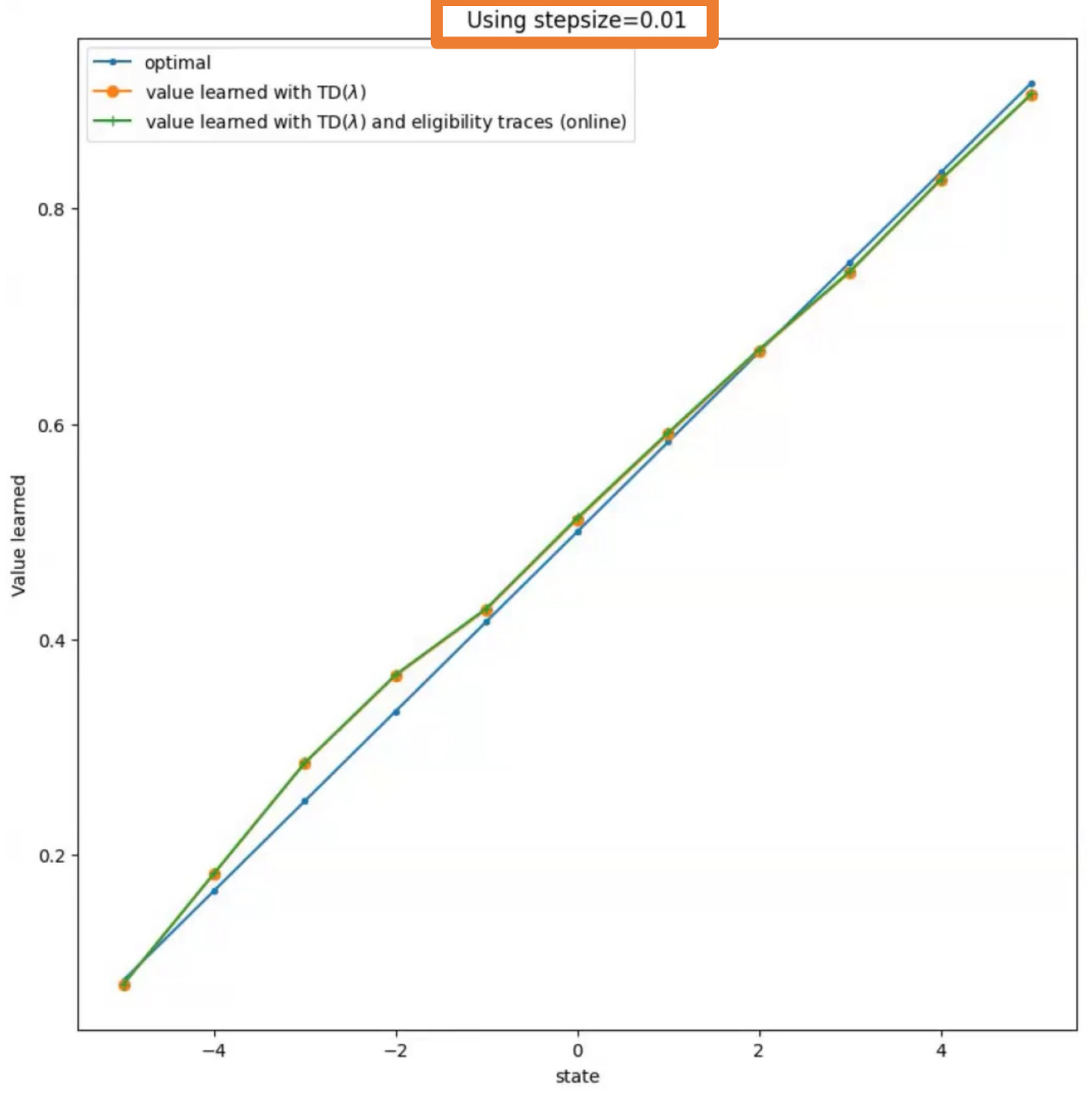
- Backward TD(λ) updates accumulate 'error' online

$$\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \alpha \sum_{t=k}^{T} (\gamma\lambda)^{t-k} \delta_t = \alpha \left(G_k^\lambda - V(S_k) \right)$$

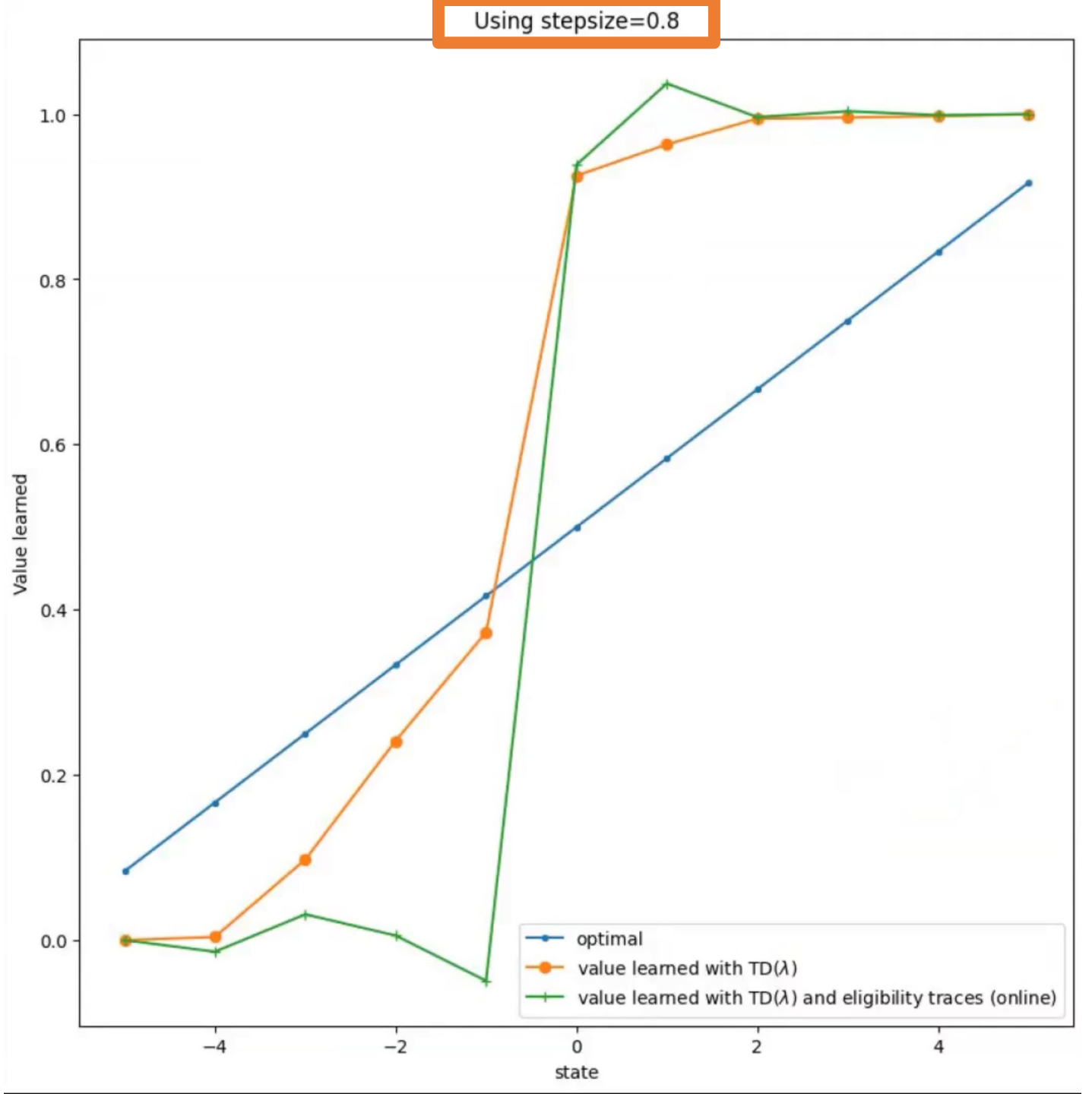
- The smaller α, the closer the algorithms

Importante questo dettaglio

Example: 11-states random walk



Example: 11-states random walk



Forward & Backward TD(λ) – Summary

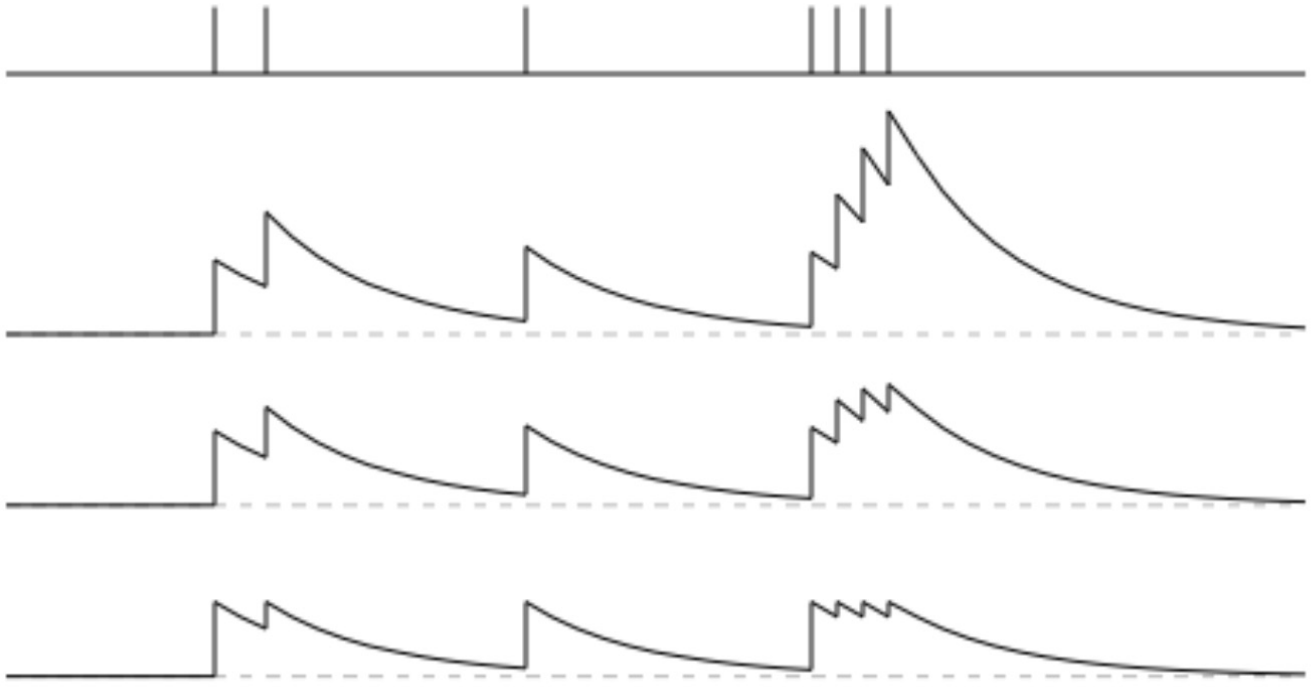
There is a version (we will not treat it in this course), called Exact Online TD(λ) that achieves perfect equivalence between Forward and Backward

Offline updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\Vert$	$\Vert$	$\Vert$
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
skip	skip	skip	skip
	$\Vert$	$\Vert$	$\Vert$
Forward view	TD(0)	Forward TD(λ)	MC
	$\Vert$	$\Vert$	$\Vert$
Exact Online	TD(0)	Exact Online TD(λ)	Exact Online TD(1)

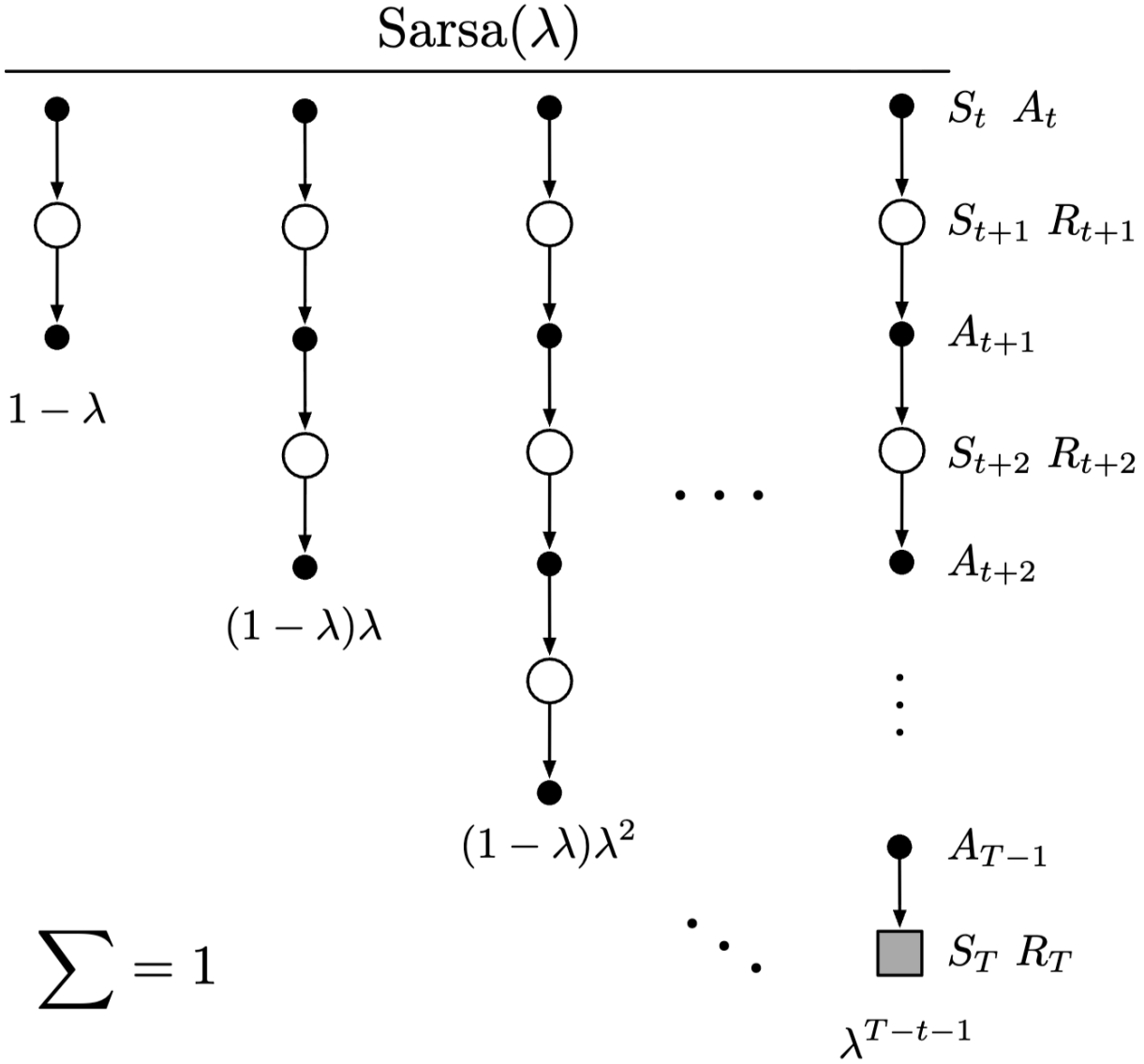
Other Eligibility Traces

The eligibility traces we have seen so far are also called ‘Accumulating traces’

$$E(S) \leftarrow E(S) + 1 \quad \text{(accumulating traces)}$$
$$\text{or } E(S) \leftarrow (1 - \alpha)E(S) + 1 \quad \text{(dutch traces)}$$
$$\text{or } E(S) \leftarrow 1 \quad \text{(replacing traces)}$$



Control – Forward view SARSA(λ)



- As usual in control, we need to combine all n-step Q-returns $G_{t:t+n}$
- The q^λ return combines all n-step Q-returns
- We need to consider the weights

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

- Forward-view SARSA (λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(q_t^\lambda - Q(S_t, A_t)\right)$$

Control – Backward view SARSA(λ)

- Just like TD(λ) for prediction, we use eligibility traces in an online algorithm
- SARSA(λ) has one eligibility trace for each state-action pair

$$E_0(s,a) = 0$$

$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = s, A_t = a)$$

- *Q(s,a)* is updated for every state *s* and action *a* in proportion to TD-error and eligibility trach

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Repeat (for each episode):

$E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize S, A

Repeat (for each step of episode):

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ε-greedy)

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$E(S, A) \leftarrow E(S, A) + 1$

For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

$E(s, a) \leftarrow \gamma \lambda E(s, a)$

$S \leftarrow S'$; $A \leftarrow A'$

until S is terminal

We are considering the control problem

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Repeat (for each episode):

$E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize S, A

Repeat (for each step of episode):

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ε-greedy)

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$E(S, A) \leftarrow E(S, A) + 1$

For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

$E(s, a) \leftarrow \gamma \lambda E(s, a)$

$S \leftarrow S'$; $A \leftarrow A'$

until S is terminal

Eligibility traces are reinitialized at every episode

Eligibility traces are 'decayed' at each step, for each state and each action!

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Repeat (for each episode):

$E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize S, A

Repeat (for each step of episode):

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ε-greedy)

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$E(S, A) \leftarrow E(S, A) + 1$

For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

$E(s, a) \leftarrow \gamma \lambda E(s, a)$

$S \leftarrow S'$; $A \leftarrow A'$

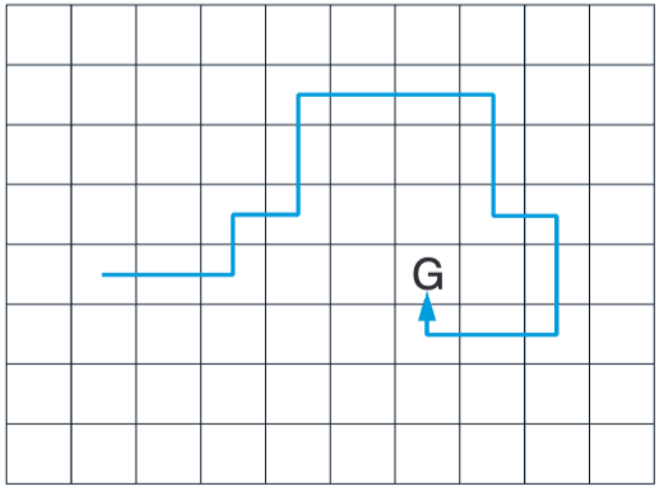
until S is terminal

On-policy

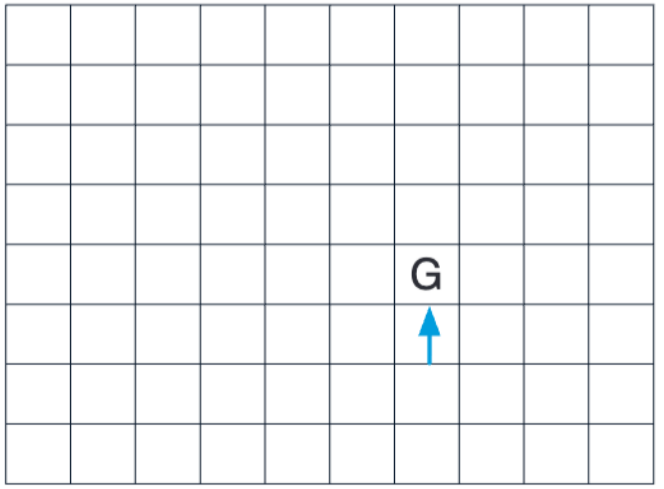
SARSA(λ)

- The one-step method (classic SARSA) strengthens only the last action of the sequence
- *n* -step method the last *n* actions of the sequence
- Trace method strengthens many actions: the degree of strengthening fall off, according to $\lambda\gamma$ with steps from the rewards

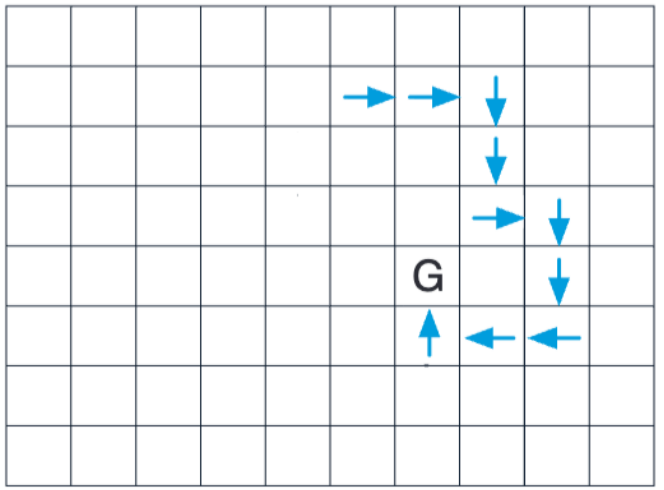
Path taken



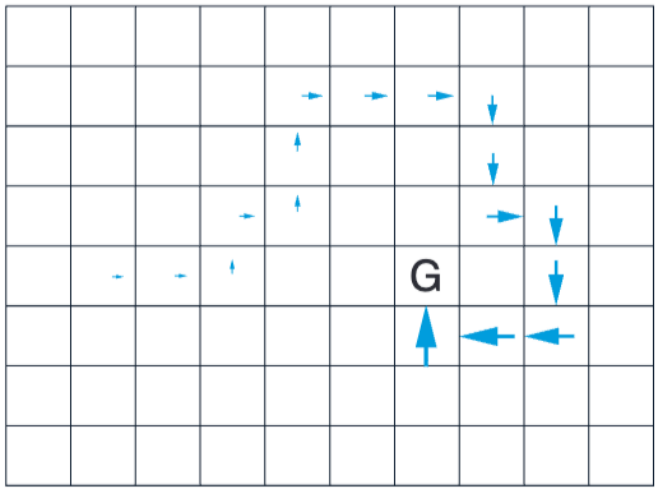
Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa



Action values increased by Sarsa(λ) with λ=0.9



TD(λ): Exam

- Section 12.1, 12.2, 12.3, 12.7 and 12.8 are exam material
- Section 12.4, 12.5 and 12.6 are optional
- No 12.9, 12.10, 12.11 and 12.12
- Keep in mind that you need Value Function Approximation to completely understand notation in Chapter 12

TD(λ): Exam

- Section 12.1, 12.2, 12.3, 12.7 and 12.8 are exam material
- Section 12.4, 12.5 and 12.6 are optional
- No 12.9, 12.10, 12.11 and 12.12
- Keep in mind that you need Value Function Approximation to completely understand notation in Chapter 12

Value Function Approximation (Chapter 9)

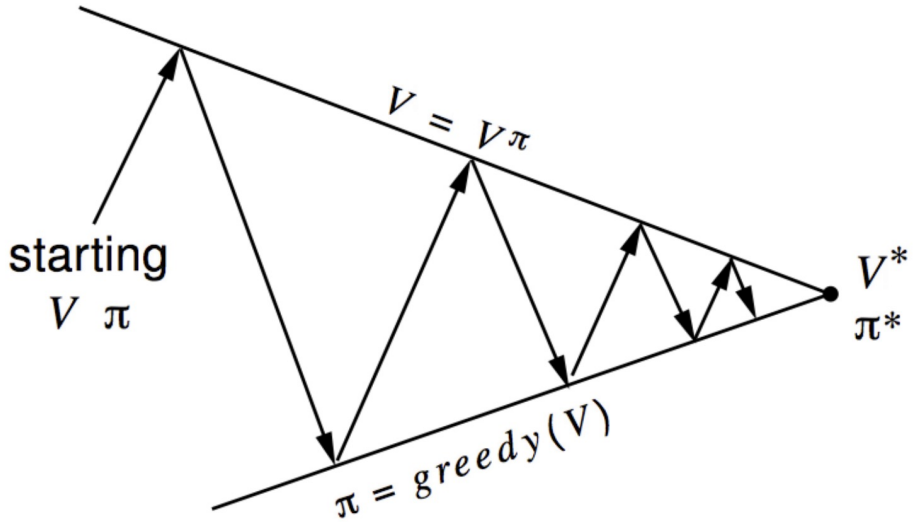
Tabular RL

- Up until now, all our approaches for prediction and control are based on the availability of good estimations of v and q
- So far we have represented value function by a lookup table

-> Every state s has an entry V (s)

-> Or every state-action pair s, a has an entry Q(s, a)

- We are therefore dealing with the so-called tabular RL: we have look-up tables that we reply on



Limits of Tabular RL

Unfortunately, in the naïve version, the tabular RL algorithms seen so far have some strong limitations: if the scale of problem become bigger, tables can become enormous

-> Backgammon: 10^{20} states

-> Go: 10^{170} states

-> Autonomous driving (continuous problem): continuous state space

Limits of Tabular RL

Unfortunately, in the naïve version, the tabular RL algorithms seen so far have some strong limitations: if the scale of problem become bigger, tables can become enormous

-> Backgammon: 10^{20} states

-> Go: 10^{170} states

-> Autonomous driving (continuous problem): continuous state space

Problems

- Memory: tables impossible to store
- Data/Time: to complete all entries on the tables with enough data to have good estimations, we need huge amount of data

From true Q and V to approximation

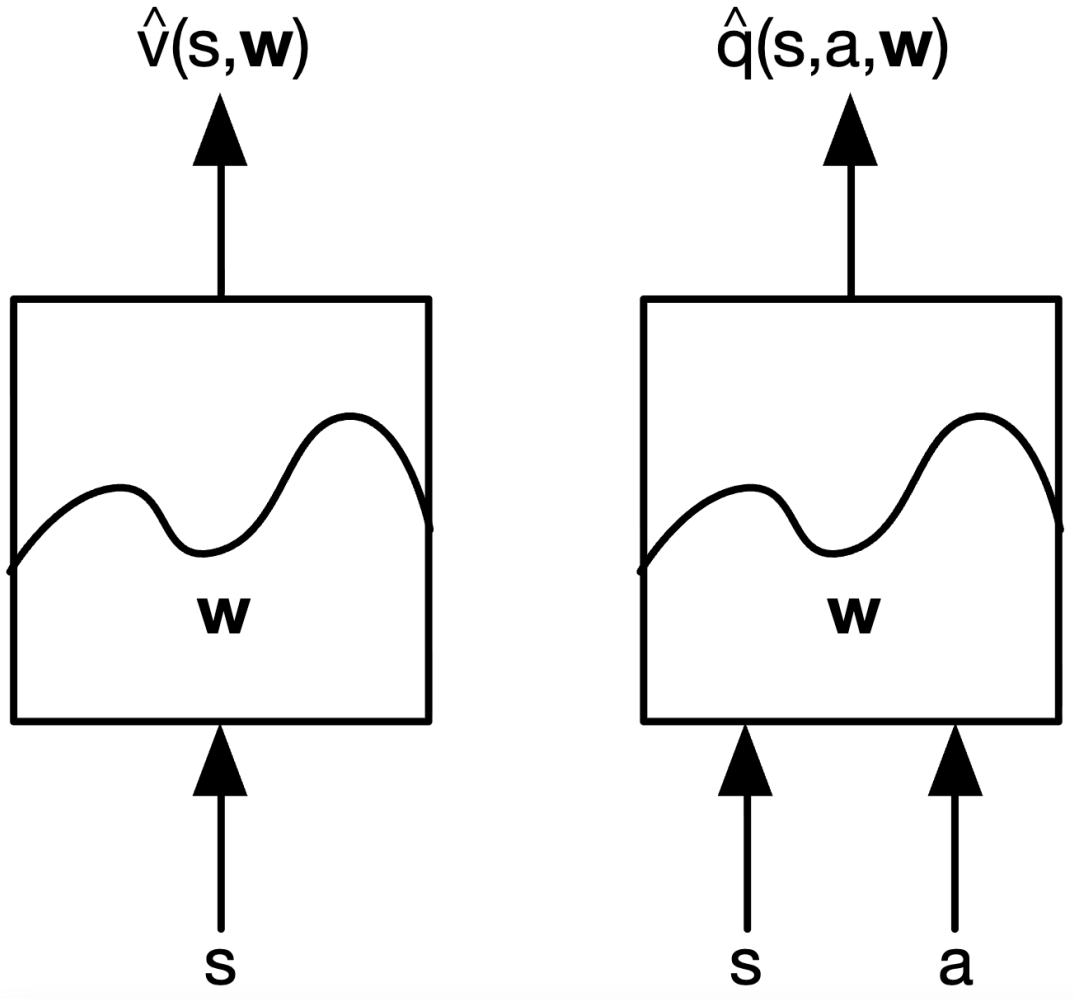
Solution for large MDPs:

- Estimate value function with function approximation

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

- Generalise from seen states to unseen states



From true Q and V to approximation: example

Small grid-world: 16 states

Can we find an alternative, more ‘compact’, description for v?

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

From true Q and V to approximation: example

Small grid-world: 16 states

Can we find an alternative, more ‘compact’, description for v?

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

For example, we can introduce descriptors (features) of the state (x,y) = ([0, …, 3], [0, …, 3]) we can see V is

$$V(x, y) = -(x + y)$$

A function approximator can, for example, have the form:

$$\hat{V}(s; \theta) = \theta_0 + \theta_1 x + \theta_2 y$$

From true Q and V to approximation: example

Small grid-world: 16 states

Can we find an alternative, more ‘compact’, description for v?

For example, we can introduce descriptors (features) of the state (x,y) = ([0, …, 3], [0, …, 3]) we can see V is

$$V(x, y) = -(x + y)$$

A function approximator can, for example, have the form:

$$\hat{V}(s; \theta) = \theta_0 + \theta_1 x + \theta_2 y$$

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

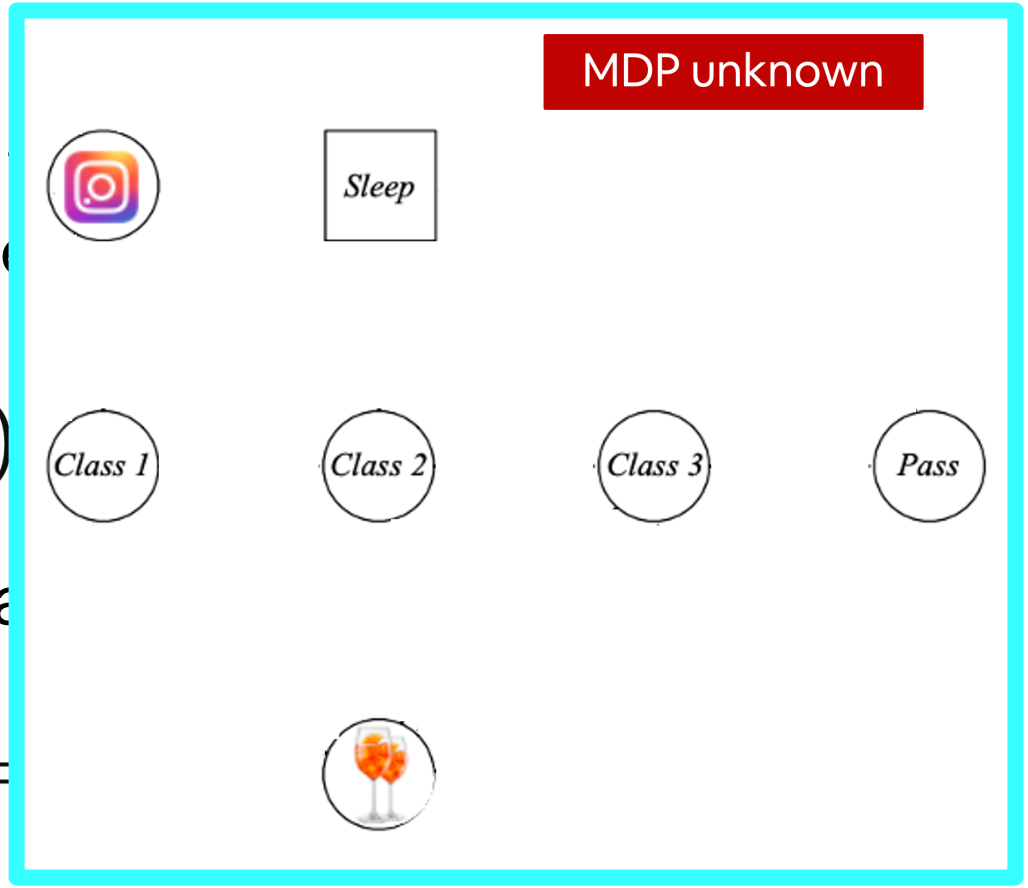
Depending on the problem we can come out with different type of features/compact representations of the problem

From true Q and V to approximation: example

This is a case where we have knowledge of a sort of relationship between states: it is not always the case! Feature definition can be complex!

states

native, more 'compact',



0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

$$V(x, y)$$

A function approxima
the form:

$$\hat{V}(s; \theta) =$$

From true Q and V to approximation: example

Small grid-world: 16 states

Can we find an alternative, more ‘compact’, description for v?

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

For example, we can introduce descriptors (features) of the state (x,y) = ([0, …, 3], [0, …, 3]) we can see V is

$$V(x, y) = -(x + y)$$

A function approximator can, for example, have the form:

$$\hat{V}(s; \theta) = \theta_0 + \theta_1 x + \theta_2 y$$

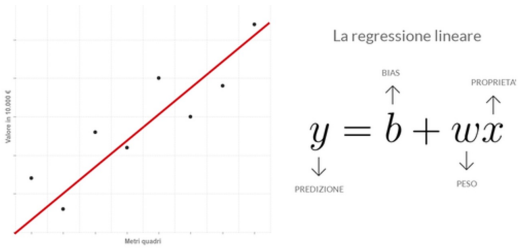
Linear function approximation, other ideas?

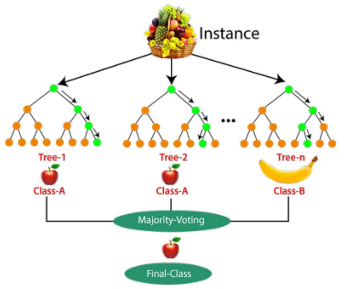
Which function approximator?

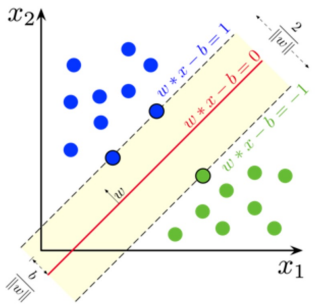
<u>In principle all supervised learning approaches are fine!</u>

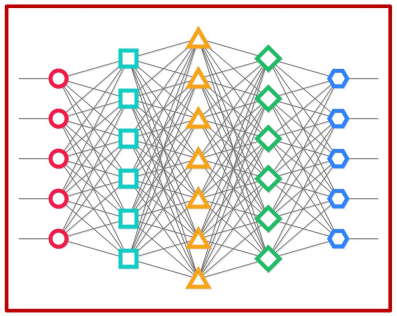
We can use:

- Linear regression
- Neural networks / Deep Learning
- SVM
- Decision trees
- Ensemble approaches (Random Forest, XGBoost, ...)









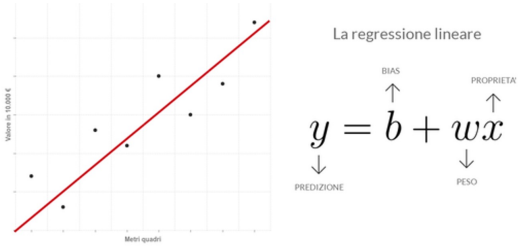
Neural Networks (Deep Learning)

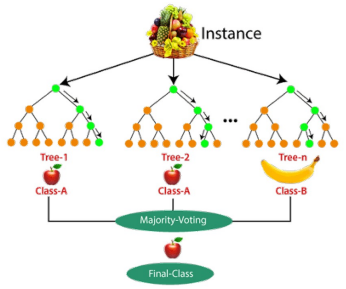
Which function approximator?

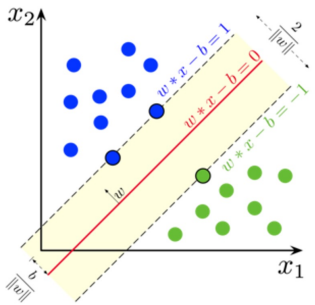
<u>In principle all supervised learning approaches are fine!</u>

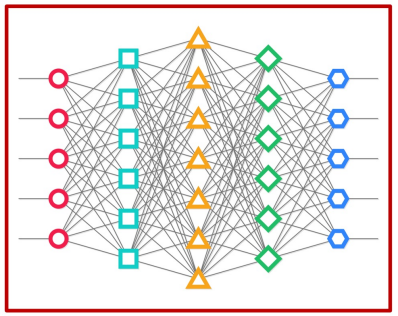
We can use:

- Linear regression
- Neural networks / Deep Learning
- SVM
- Decision trees
- Ensemble approaches (Random Forest, XGBoost, …)









Neural Networks (Deep Learning)

However, we have a strong preference for differentiable approaches: some approach that I can learn along the way (ie. while I'm collecting experience interacting with the real world)

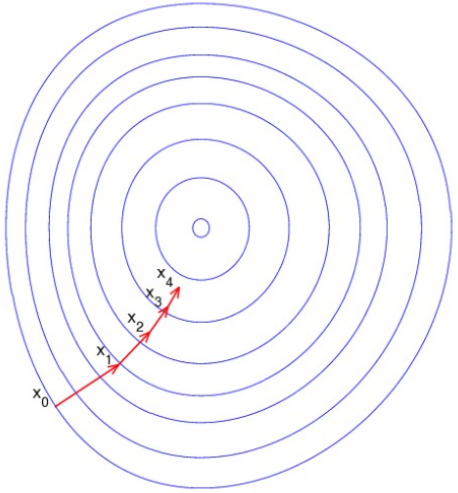
Differentiable Supervised Learning approaches

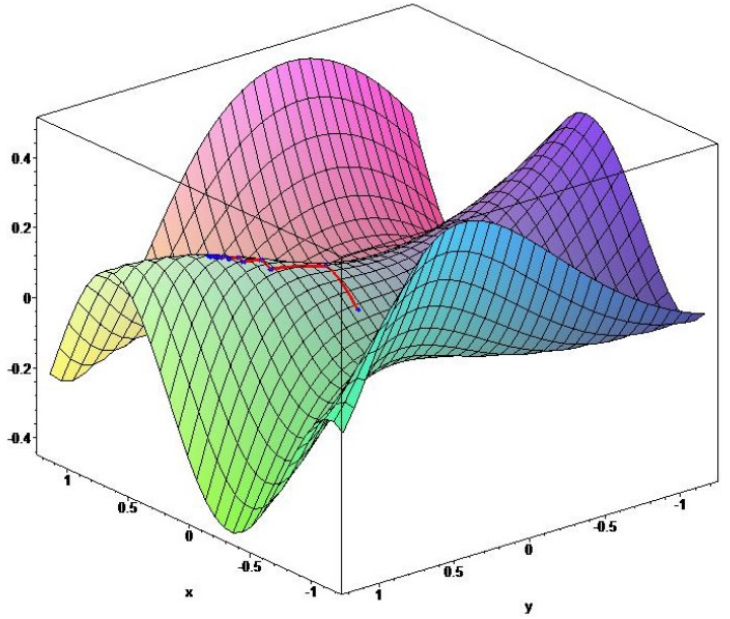
With:

- Linear approaches
- Neural networks / Deep Learning

we can use stochastics gradient descent and update (for example with new data)

The procedure is incremental: we can see this as ‘learning along the way’



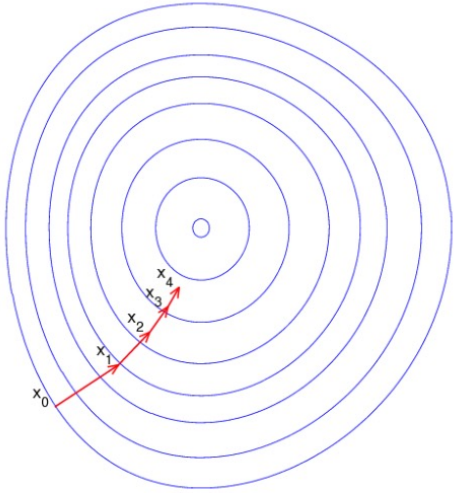


Differentiable Supervised Learning approaches

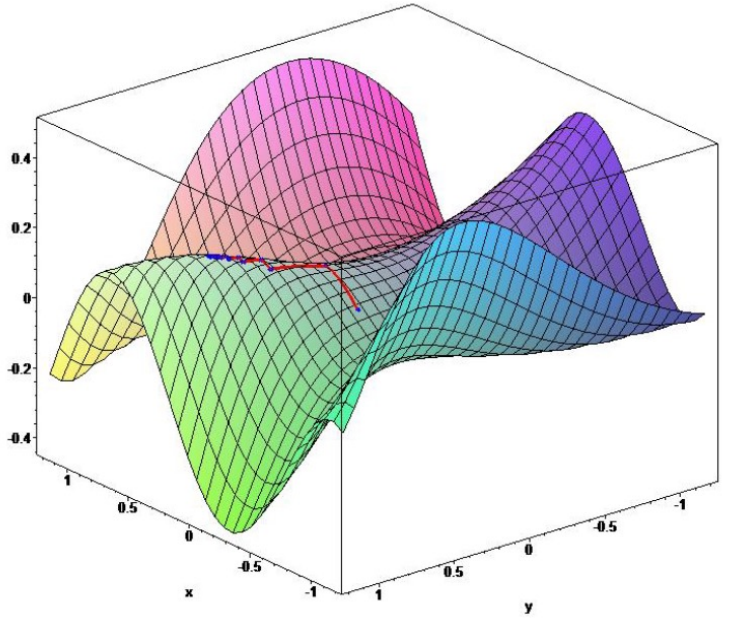
With:

- Linear approaches
- Neural networks / Deep Learning

we can use stochastics gradient descent and update (for example with new data)



The procedure is incremental: we can see this as '<u>learning along the way</u>'



Who has seen:

1. SGD
2. Neural Networks?

Next week lectures on NNs!

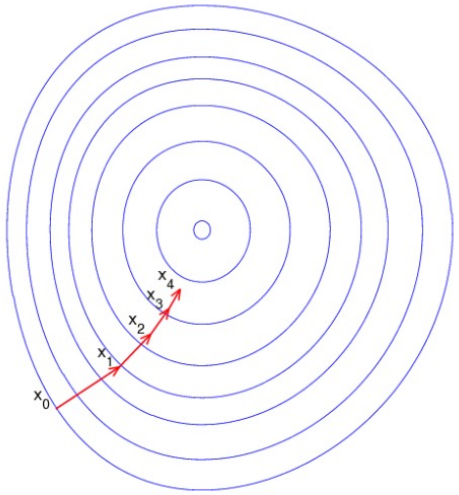
Differentiable Supervised Learning approaches

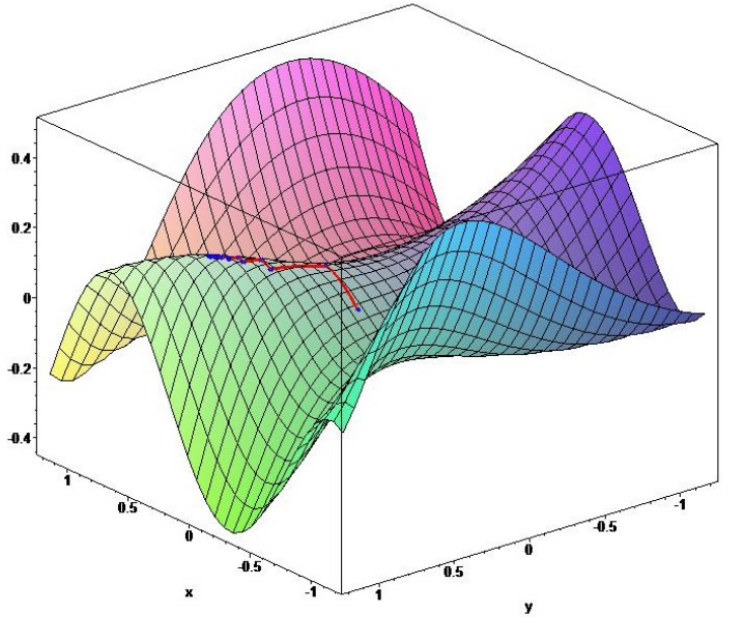
With:

- Linear approaches
- Neural networks / Deep Learning

we can use stochastics gradient descent and update (for example with new data)

The procedure is incremental: we can see this as '<u>learning along the way</u>'





Who has seen:

1. SGD
2. Neural Networks?

Next week lectures on NNs!

Recap on Stochastic Gradient Descent (SGD)

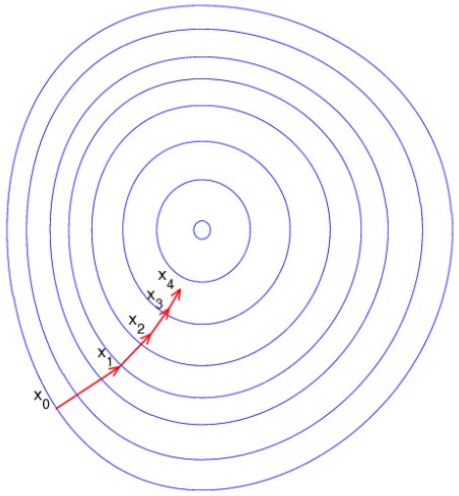
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

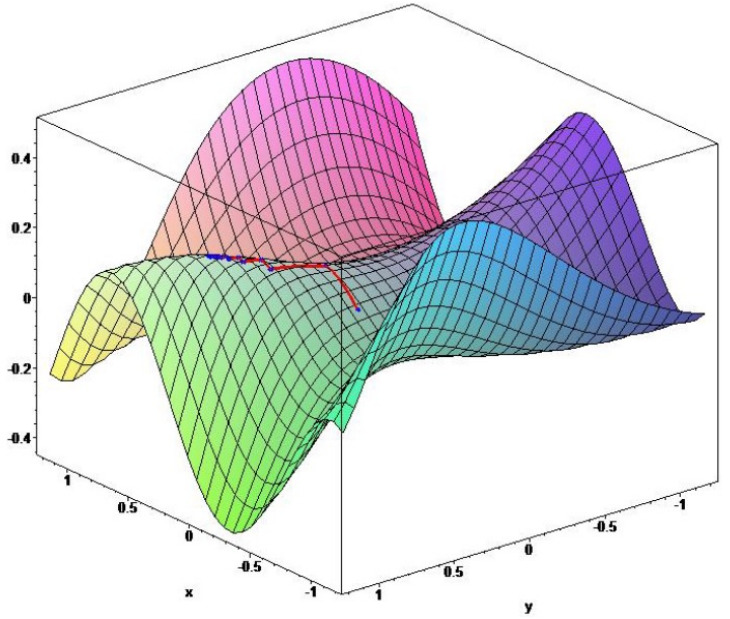
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction of -ve gradient

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

where α is a step-size parameter





Recap on Stochastic Gradient Descent (SGD)

- Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

So, we can use Supervised Learning now... can we?

So, we can use Supervised Learning now… can we?

- Supervised learning approaches requires tagged examples! We will need a dataset of examples

$$(s_1, v(s_1)), (s_2, v(s_2)), \dots, (s_D, v(s_D))$$

$$(s_1, a_1, q(s_1, a_1)), (s_2, a_2, q(s_2, a_2)), \dots, (s_D, a_D, q(s_D, a_D))$$

So, we can use Supervised Learning now… can we?

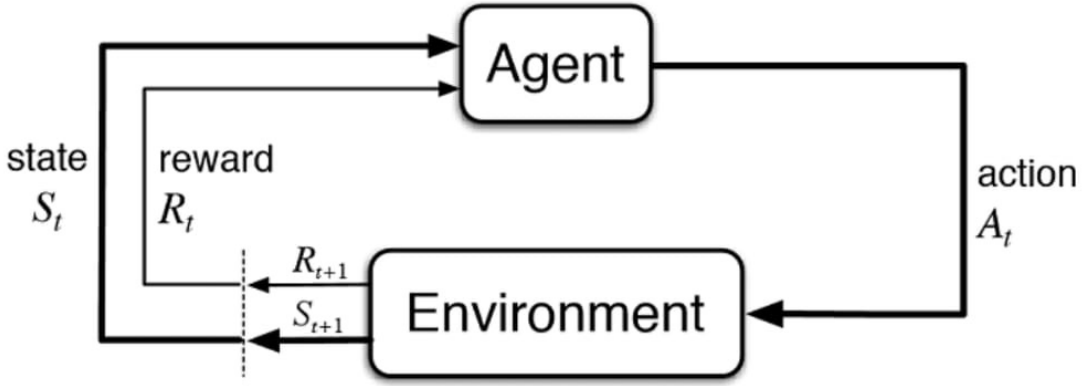
- Supervised learning approaches requires tagged examples! We will need a dataset of examples

$$(s_1, v(s_1)), (s_2, v(s_2)), \dots, (s_D, v(s_D))$$

$$(s_1, a_1, q(s_1, a_1)), (s_2, a_2, q(s_2, a_2)), \dots, (s_D, a_D, q(s_D, a_D))$$

- But no one is telling us in real world problem what v and q are! In RL we have no supervision, only rewards!

- Any ideas?



Core idea of Value Function Approximation

To create a supervised setting, instead of the true v and q, we use a target

- Monte Carlo target: the return

$$(s_1, G_t^1), (s_2, G_t^2), \dots, (s_D, G_t^D)$$

- TD(0) target: the TD target

$$(s_1, R_{t+1}^1 + \gamma \hat{v}(S_{t+1}^1, \boldsymbol{w})), (s_2, R_{t+1}^2 + \gamma \hat{v}(S_{t+1}^2, \boldsymbol{w})), \dots, (s_D, R_{t+1}^D + \gamma \hat{v}(S_{t+1}^D, \boldsymbol{w}))$$

- TD(λ) target: the λ-return

$$(s_1, G_1^\lambda), (s_2, G_2^\lambda), \dots, (s_D, G_D^\lambda)$$

Value Function Approximation: Monte Carlo

- Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, ..., \langle S_T, G_T \rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t) \end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

Value Function Approximation: TD(0)

- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w}) \rangle, ..., \langle S_{T-1}, R_T \rangle$$

- For example, using *linear TD(0)*

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha \delta \mathbf{x}(S) \end{aligned}$$

- Linear TD(0) converges (close) to global optimum

Value Function Approximation: TD(0)

- The λ-return G_t^λ is also a biased sample of true value $v_\pi(s)$
- Can again apply supervised learning to "training data":

$$\left\langle S_1, G_1^\lambda \right\rangle, \left\langle S_2, G_2^\lambda \right\rangle, ..., \left\langle S_{T-1}, G_{T-1}^\lambda \right\rangle$$

- Forward view linear TD(λ)

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

Value Function Approximation: TD(0)

- The λ-return G_t^λ is also a biased sample of true value $v_\pi(s)$
- Can again apply supervised learning to "training data":

$$\left\langle S_1, G_1^\lambda \right\rangle, \left\langle S_2, G_2^\lambda \right\rangle, ..., \left\langle S_{T-1}, G_{T-1}^\lambda \right\rangle$$

Each 'couple' has info for many states

- Forward view linear TD(λ)

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

Value Function Approximation: TD(0)

- The λ-return G_t^λ is also a biased sample of true value $v_\pi(s)$
- Can again apply supervised learning to "training data":

$$\left\langle S_1, G_1^\lambda \right\rangle, \left\langle S_2, G_2^\lambda \right\rangle, ..., \left\langle S_{T-1}, G_{T-1}^\lambda \right\rangle$$

Each 'couple' has info for many states

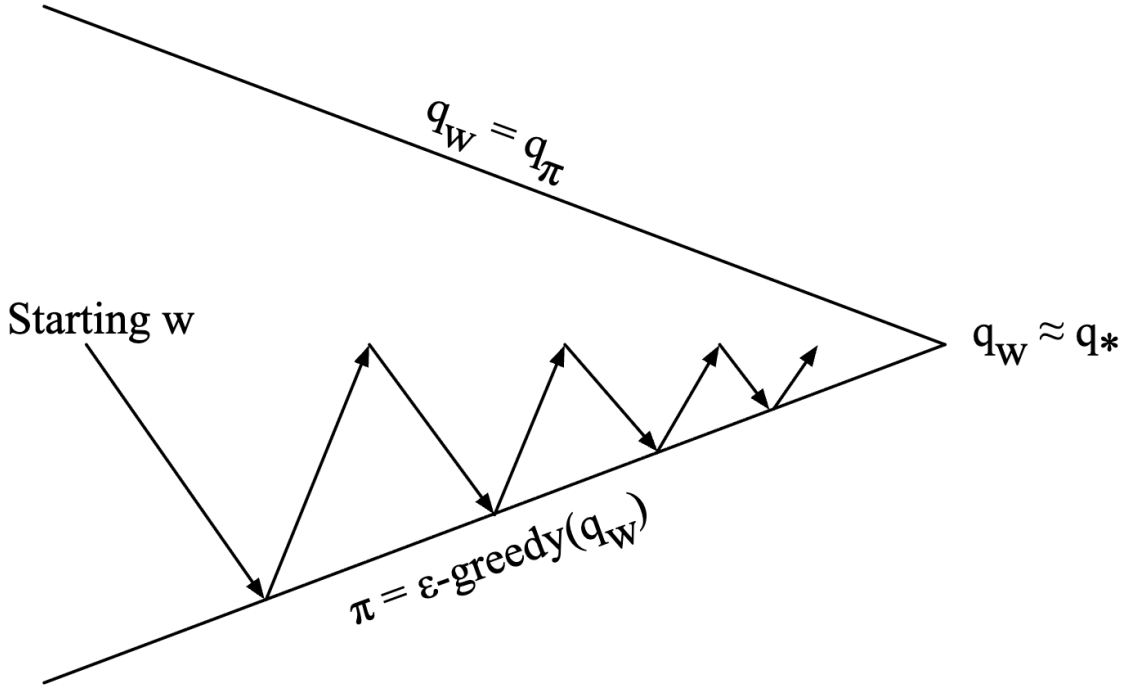
- Forward view linear TD(λ)

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

Even if we are dealing with a new 'framework', we leverage on the algorithms we have already seen!



Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Credits

- David Silver ‘Lecture on RL’, UCL 2015
- Image of the course is taken from C. Mahoney ‘Reinforcement Learning’ https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6

AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning 2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Thank you!
Questions?

Lecture #13: TD(λ) & Value Function Approximation

Gian Antonio Susto