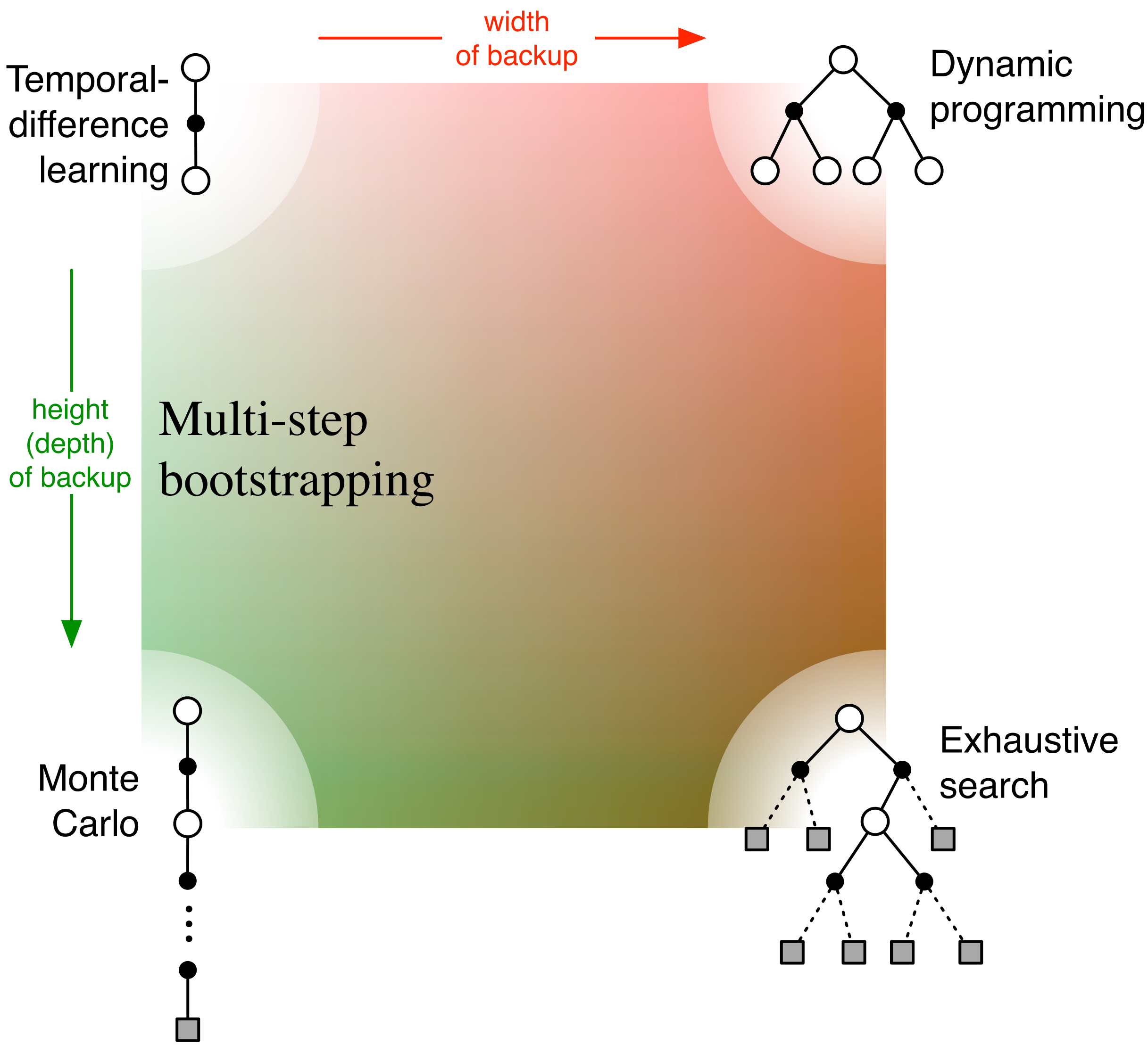
Eligibility Traces

Chapter 12

Eligibility traces are

- Another way of interpolating between MC and TD methods
- A way of implementing *compound λ-return* targets
- A basic mechanistic idea — a short-term, fading memory
- A new style of algorithm development/analysis
  - the forward-view $\Leftrightarrow$ backward-view transformation
  - Forward view:
    conceptually simple — good for theory, intuition
  - Backward view:
    computationally congenial implementation of the f. view

Unified View



Recall *n*-step targets

- For example, in the episodic case, with linear function approximation:
  - 2-step target:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 \mathbf{w}_{t+1}^\top \mathbf{x}_{t+2}$$

  - *n*-step target:

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \mathbf{w}_{t+n-1}^\top \mathbf{x}_{t+n}$$

with $G_{t:t+n} \doteq G_t$ if $t+n \geq T$

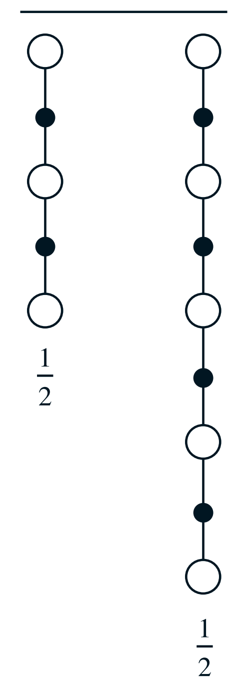
Any set of update targets can be *averaged* to produce new *compound* update targets

- For example, half a 2-step plus half a 4-step

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

- Called a compound backup
  - Draw each component
  - Label with the weights for that component

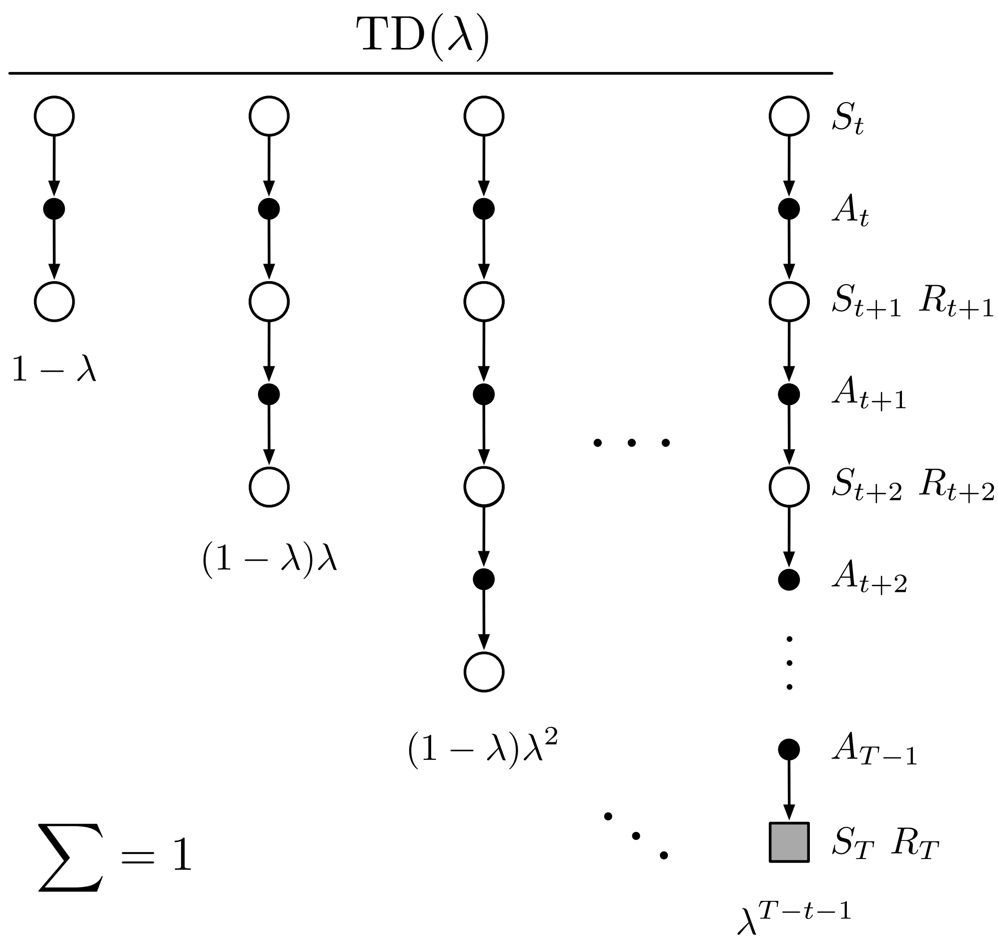
A *compound* backup



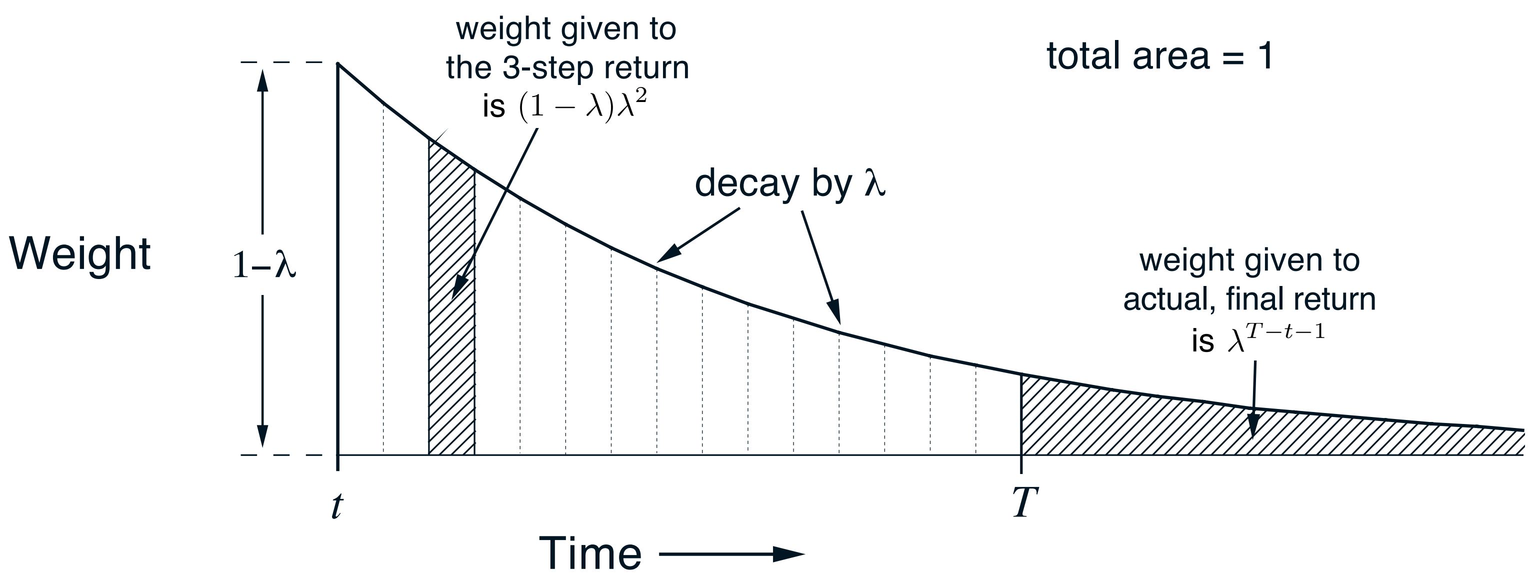
The λ-return is a compound update target

- The λ-return a target that averages all *n*-step targets
    - each weighted by λ^{n-1}

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



λ-return Weighting Function



$$G_t^\lambda = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n}}_{\text{Until termination}} + \underbrace{\lambda^{T-t-1}G_t}_{\text{After termination}}$$

Relation to TD(0) and MC

- The λ-return can be rewritten as:

$$G_t^\lambda = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n}}_{\text{Until termination}} + \underbrace{\lambda^{T-t-1}G_t}_{\text{After termination}}$$

- If $\lambda = 1$, you get the MC target:

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

- If $\lambda = 0$, you get the TD(0) target:

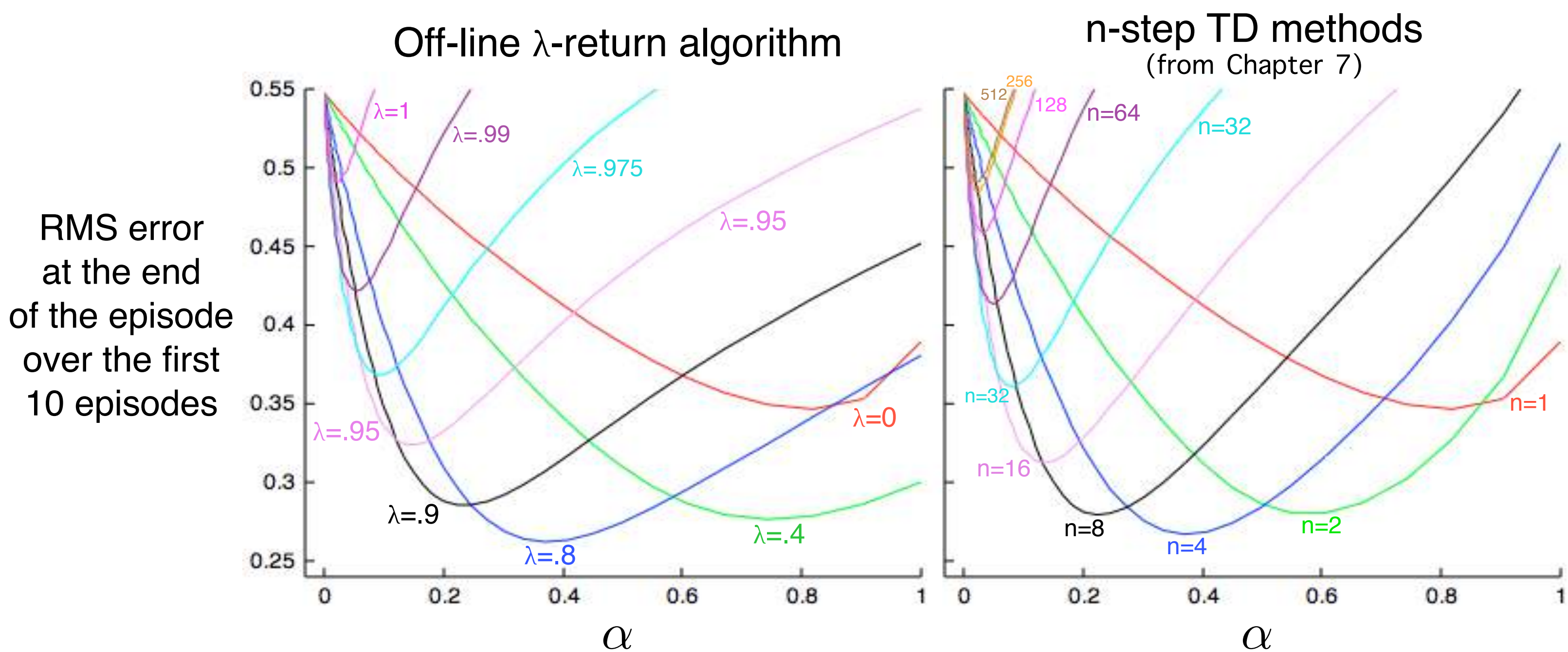
$$G_t^\lambda = (1-0)\sum_{n=1}^{T-t-1}0^{n-1}G_{t:t+n} + 0^{T-t-1}G_t = G_{t:t+1}$$

The off-line λ-return "algorithm"

- Wait until the end of the episode (offline)
- Then go back over the time steps, updating

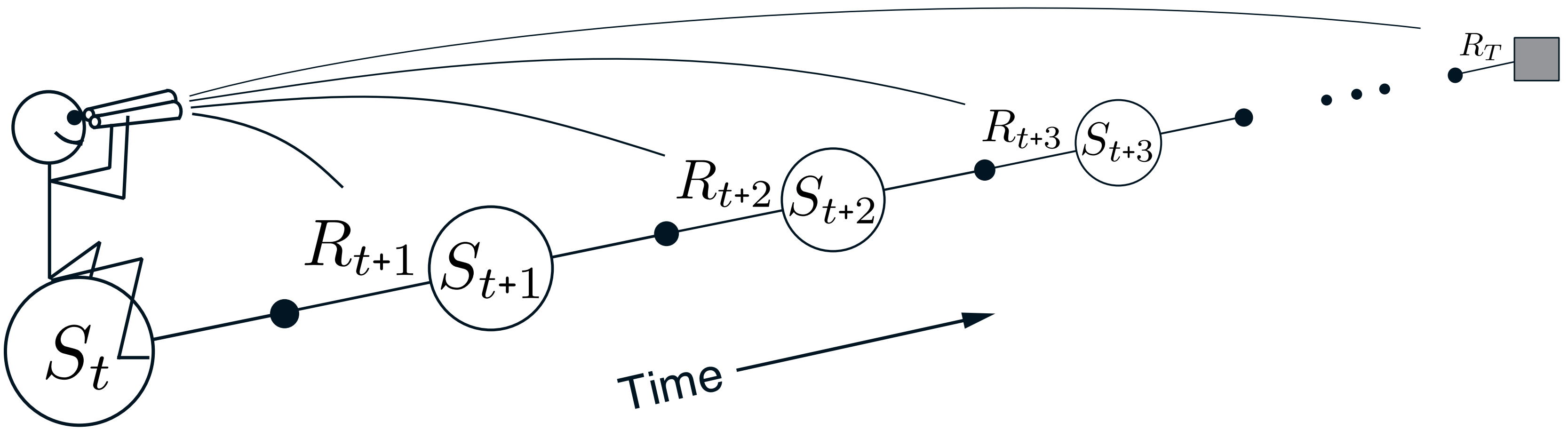
$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[G_t^\lambda - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t), \quad t = 0, \ldots, T-1$$

The λ-return alg performs similarly to n-step algs on the 19-state random walk (Tabular)

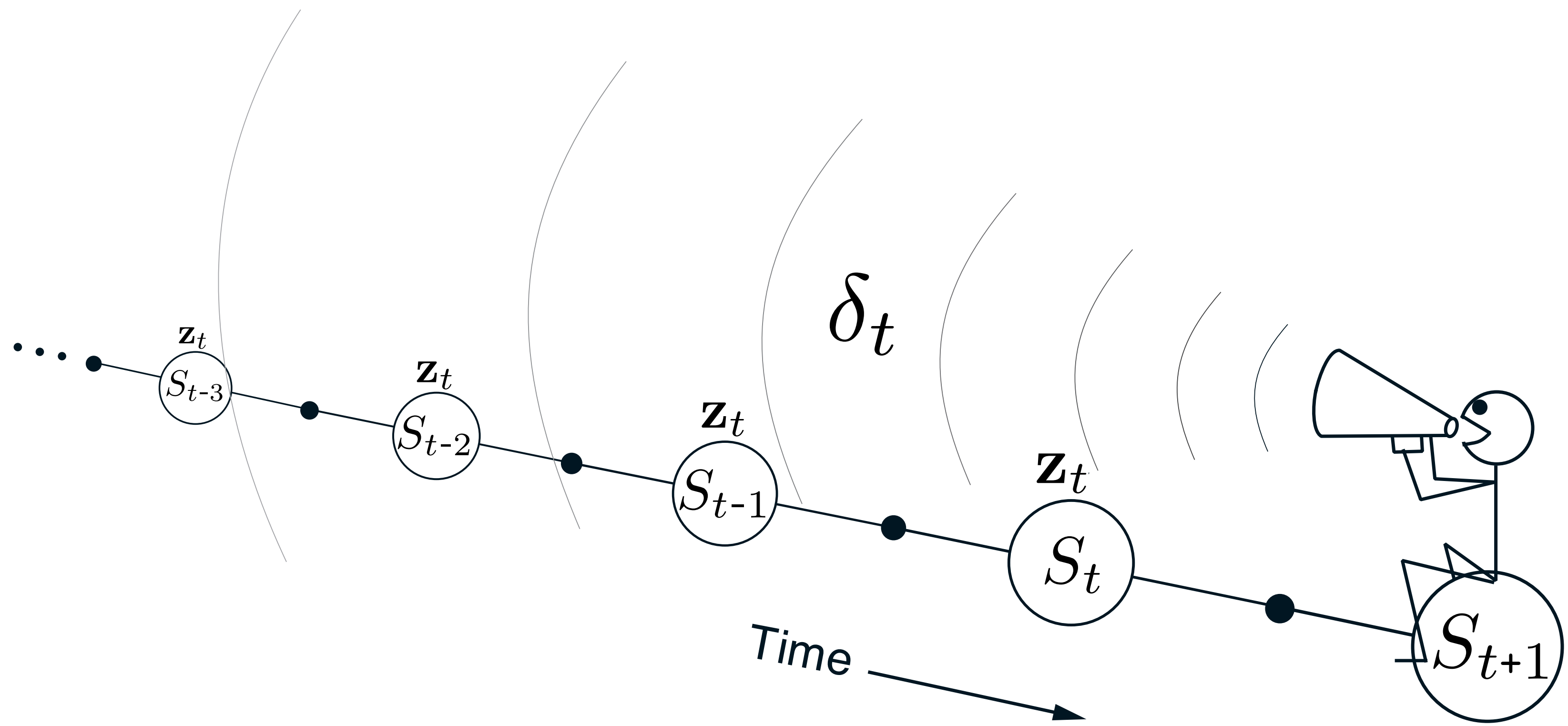


Intermediate λ is best (just like intermediate n is best)
λ-return slightly better than n-step

The forward view looks forward from the state being updated to future states and rewards



The backward view looks back to the recently visited states (marked by eligibility traces)



- Shout the TD error backwards
- The traces fade with temporal distance by $\gamma\lambda$

The Semi-gradient TD(λ) algorithm

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

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

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

New error bound:

$$\overline{\mathrm{VE}}(\mathbf{w}_\infty) \;\leq\; \frac{1-\gamma\lambda}{1-\gamma} \min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w})$$

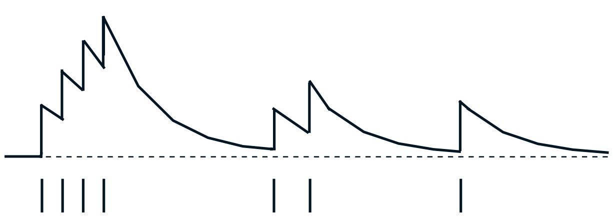
Eligibility traces (mechanism)

- The forward view was for theory
- The backward view is for *mechanism*

same shape as $\mathbf{w}$

- New memory vector called *eligibility trace* $\mathbf{z}_t \in \mathbb{R}^d$
    - On each step, decay each component by γλ and increment the trace for the current state by 1
    - *Accumulating trace*

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



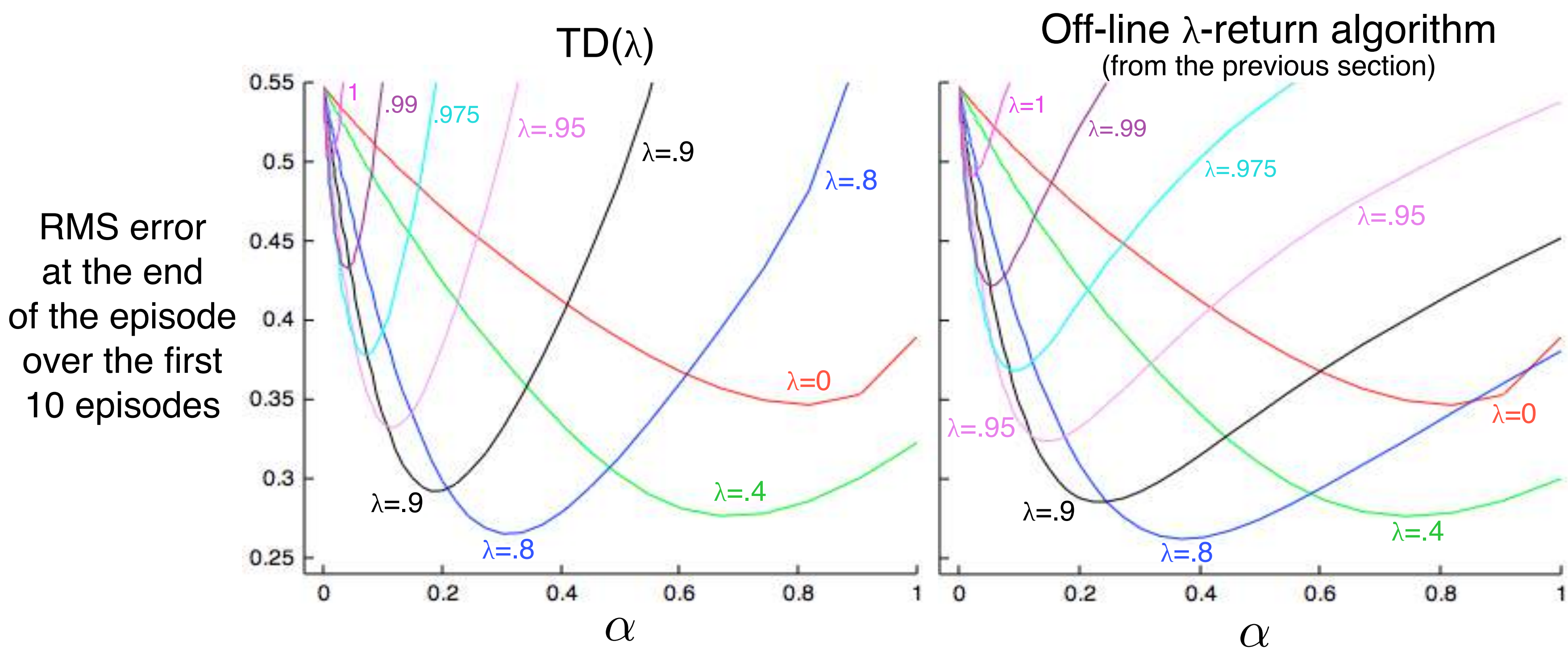
Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
    Initialize S
    $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
    Repeat (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

TD(λ) performs similarly to offline λ-return alg. but slightly worse, particularly at high α

Tabular 19-state random walk task



Can we do better? Can we update online?

The λ-return can also be truncated at some horizon h

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

n-step-truncated λ-return method:

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

The λ-return can also be truncated at some horizon h

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

n-step-truncated λ-return method:

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\left[G^{\lambda}_{t:t+n} - \hat{v}(S_t,\mathbf{w}_{t+n-1})\right]\nabla\hat{v}(S_t,\mathbf{w}_{t+n-1}), \qquad 0 \le t < T$$

For a reasonable n, this may do better than TD(λ), at the cost of the n-step delay of updates

The λ-return can also be truncated at some horizon h

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

n-step-truncated λ-return method:

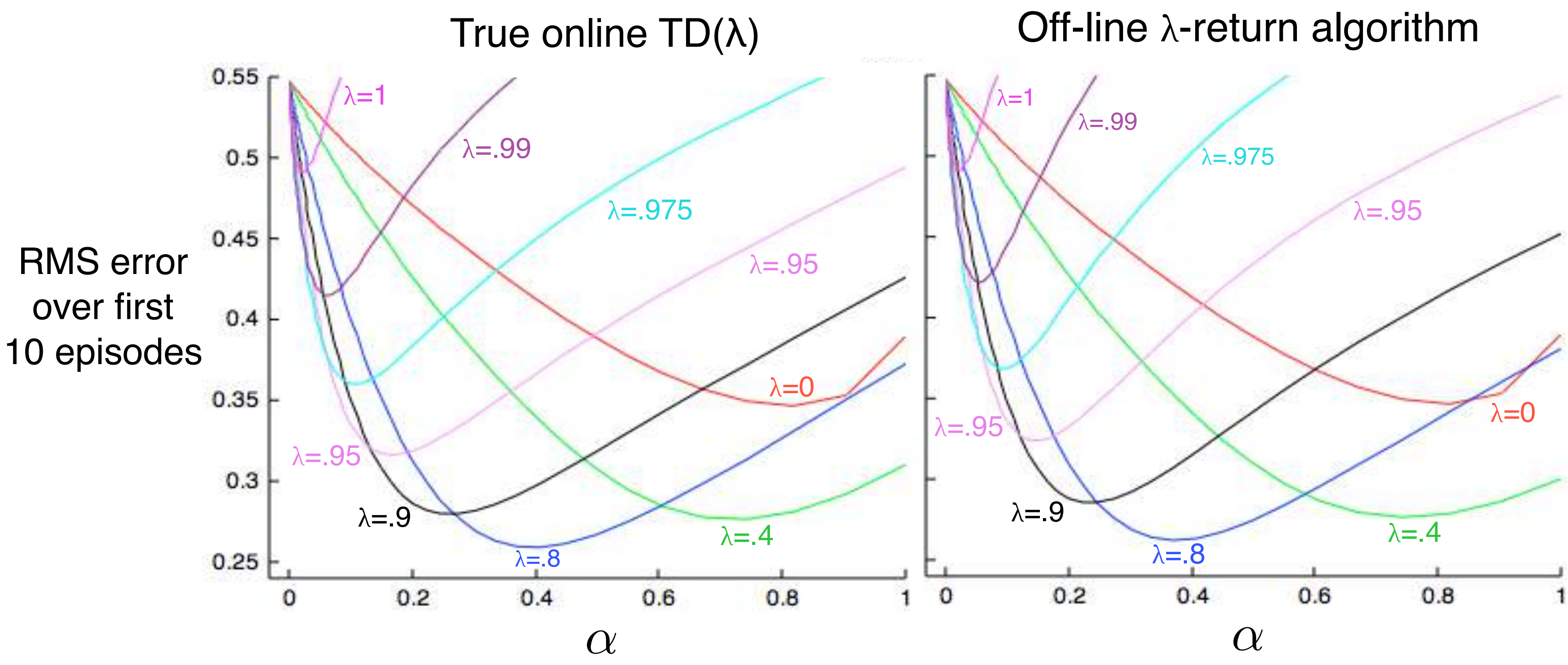
$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha \left[G^{\lambda}_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \qquad 0 \leq t < T$$

For a reasonable n, this may do better than TD(λ), at the cost of the n-step delay of updates

But even better is true online TD(λ)

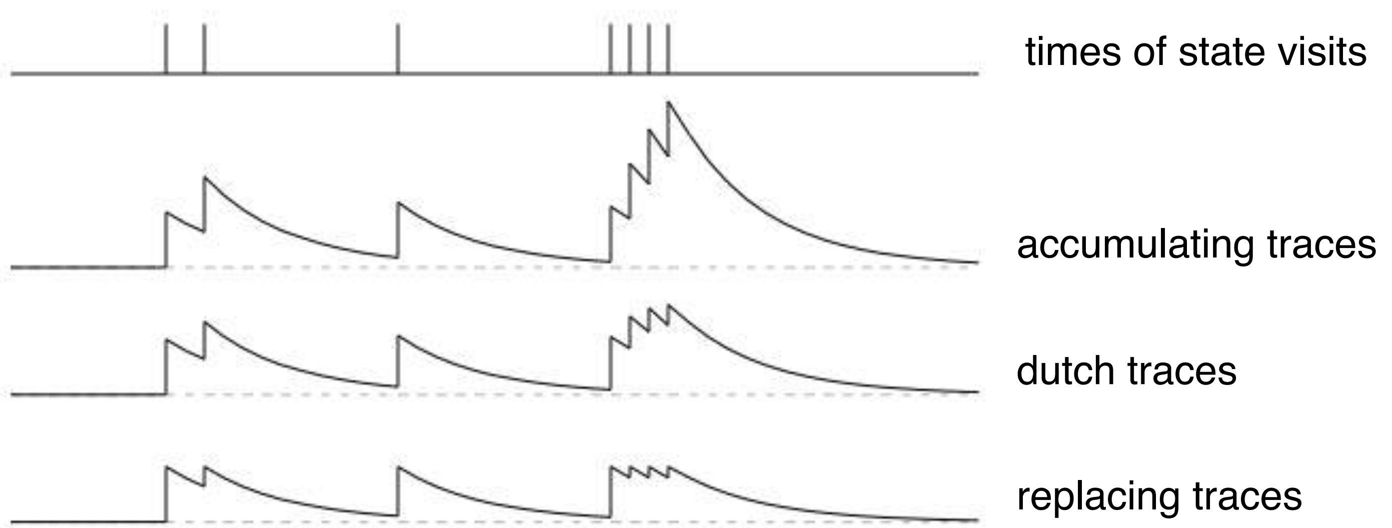
True online TD(λ) performs best of all

Tabular 19-state random walk task



Accumulating, Dutch, and Replacing Traces

- All traces fade the same:
- But increment differently!



True online TD(λ)

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

$$\mathbf{z}_t \doteq \gamma \lambda \mathbf{z}_{t-1} + \left(1 - \alpha \gamma \lambda \mathbf{z}_{t-1}^\top \mathbf{x}_t \right) \mathbf{x}_t$$ *dutch trace*

True online TD(λ)

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

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

dutch trace

True online TD(λ)

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

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

dutch trace

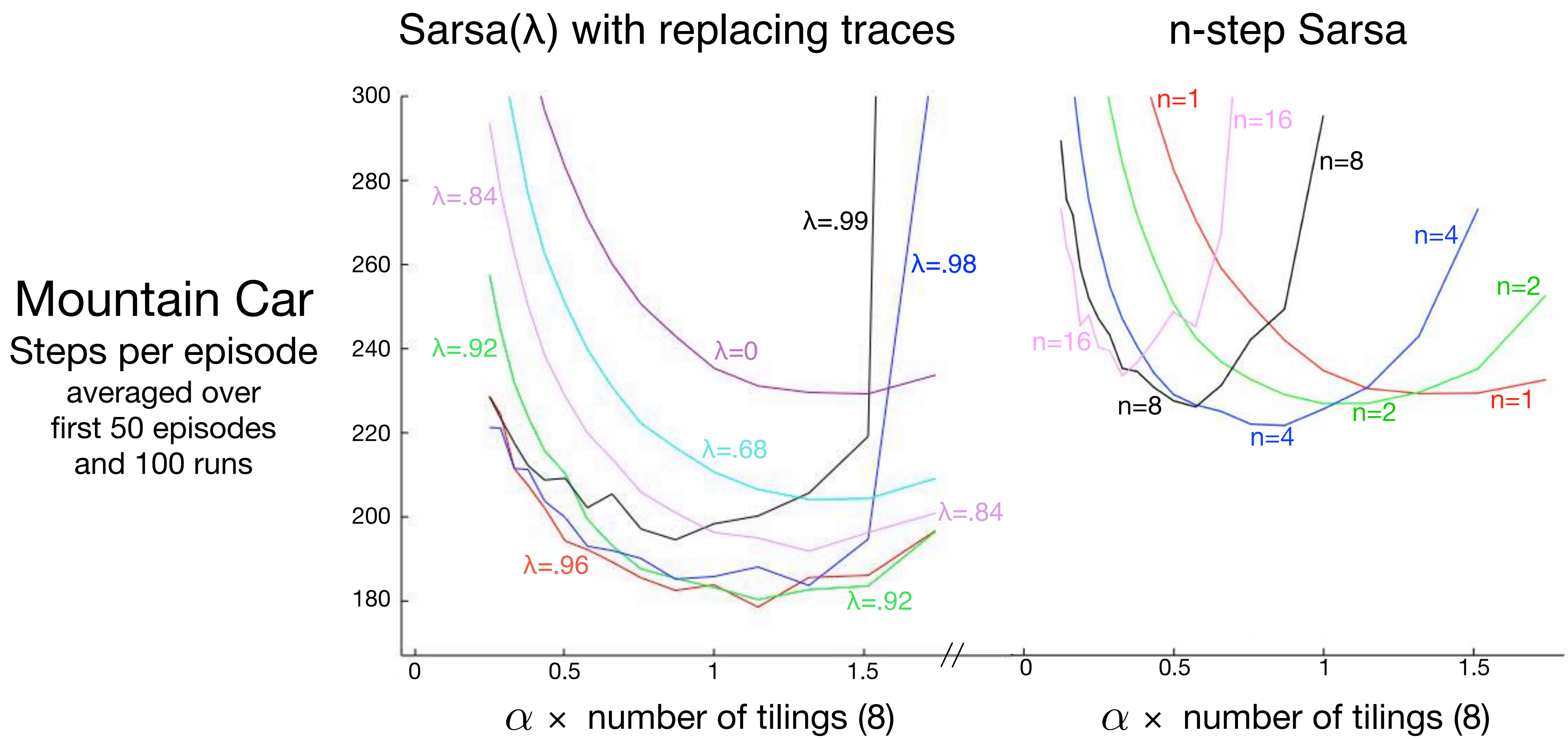
Traces for control — Sarsa(λ) (on-policy)

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

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

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

Sarsa(λ) is better than n-step Sarsa on Mountain Car with tile-coding linear function approximation

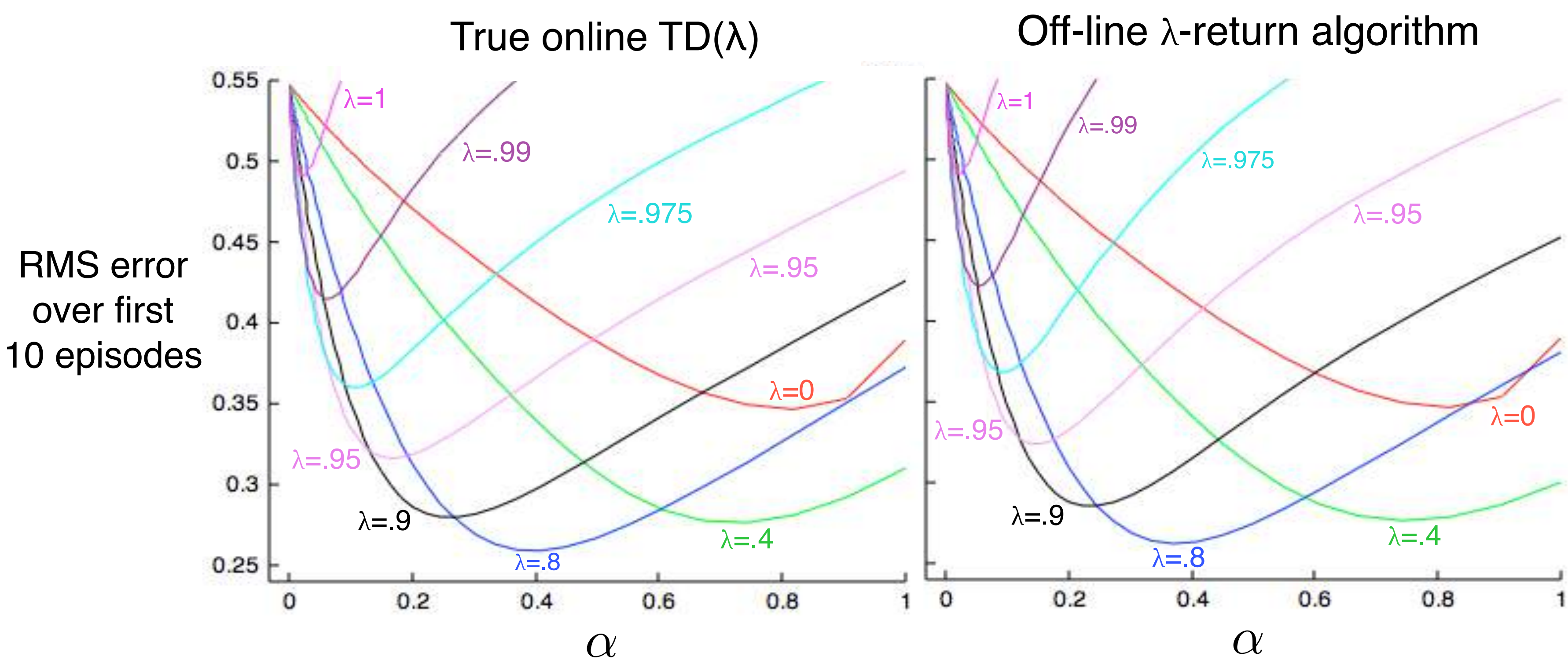


Conclusions regarding Eligibility Traces

- Provide an efficient, incremental way to combine MC and TD
  - Includes advantages of MC (better when non-Markov)
  - Includes advantages of TD (faster, comp. congenial)
- True online TD(λ) is new and best
  - Is exactly equivalent to online λ-return algorithm
- Three varieties of traces: accumulating, dutch, (replacing)
- Traces for prediction and on-policy control
- Trace methods often perform better than *n*-step methods
- Traces do have a small cost in computation (≈x2)

True online TD(λ) performs best of all

Tabular 19-state random walk task



True online TD(λ)

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

$$\mathbf{z}_t \doteq \gamma\lambda \mathbf{z}_{t-1} + \left(1 - \alpha\gamma\lambda \mathbf{z}_{t-1}^\top \mathbf{x}_t \right) \mathbf{x}_t$$ *dutch trace*

True online TD(λ)

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

$$\mathbf{z}_t \doteq \gamma \lambda \mathbf{z}_{t-1} + \left(1 - \alpha \gamma \lambda \mathbf{z}_{t-1}^\top \mathbf{x}_t \right) \mathbf{x}_t$$ *dutch trace*

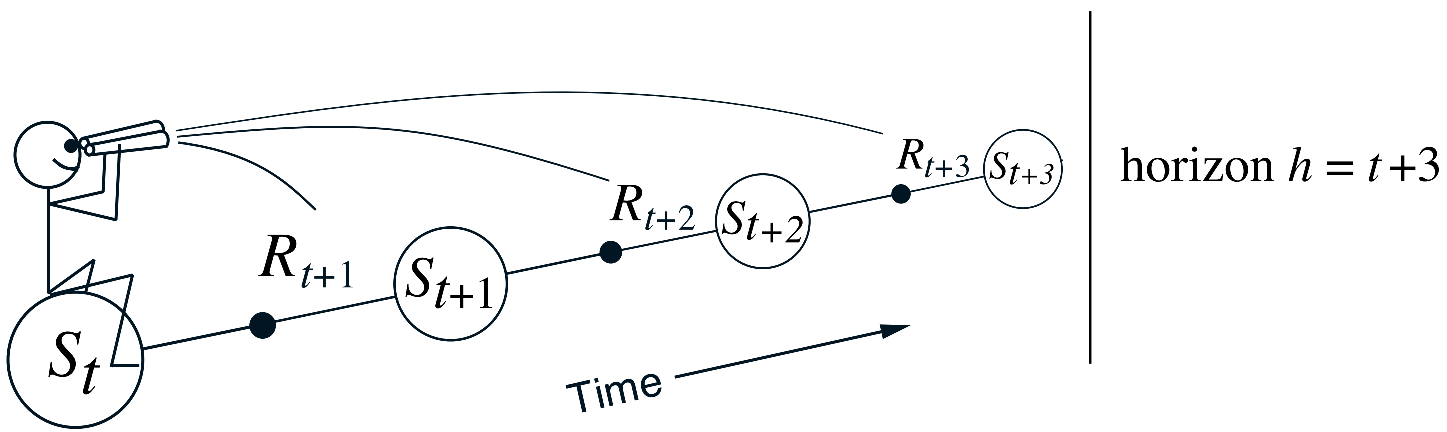
True online TD(λ)

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

$$\mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + \left(1 - \alpha\gamma\lambda\mathbf{z}_{t-1}^\top\mathbf{x}_t\right)\mathbf{x}_t$$ *dutch trace*

The online λ-return alg uses a *truncated λ-return* as its target

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



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

There is a separate $\mathbf{w}$ sequence for each h!

The online λ-return algorithm

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

There is a separate w sequence for each h!

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

The online λ-return algorithm

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

There is a separate w sequence for each h!

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

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

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

$$\vdots$$

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

The online λ-return algorithm

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

There is a separate w sequence for each h!

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

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

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

$$h = 3: \quad \mathbf{w}_1^3 \doteq \mathbf{w}_0^3 + \alpha \left[G_{0:3}^\lambda - \hat{v}(S_0,\mathbf{w}_0^3) \right] \nabla \hat{v}(S_0,\mathbf{w}_0^3)$$

$$\mathbf{w}_2^3 \doteq \mathbf{w}_1^3 + \alpha \left[G_{1:3}^\lambda - \hat{v}(S_1,\mathbf{w}_1^3) \right] \nabla \hat{v}(S_1,\mathbf{w}_1^3)$$

$$\mathbf{w}_3^3 \doteq \mathbf{w}_2^3 + \alpha \left[G_{2:3}^\lambda - \hat{v}(S_2,\mathbf{w}_2^3) \right] \nabla \hat{v}(S_2,\mathbf{w}_2^3)$$

$$\vdots$$

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

True online TD(λ) computes just the diagonal, cheaply (for linear FA)

The simplest example of deriving a backward view from a forward view

- Monte Carlo learning of a final target
- Will derive dutch traces
- Showing the dutch traces really are not about TD
- They are about efficiently implementing online algs

The Problem:
Predict final target G with linear function approximation

	episode						next episode		
Time	*0*	*1*	*2*	. . .	*T-1*	*T*	*0*	*1*	*2*
Data	$\mathbf{x}_0$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\dots$	$\mathbf{x}_{T-1}$	G			
Weights	$\mathbf{w}_0$	$\mathbf{w}_0$	$\mathbf{w}_0$	$\dots$	$\mathbf{w}_0$	$\mathbf{w}_T$	$\mathbf{w}_T$	$\mathbf{w}_T$	$\mathbf{w}_T$
Predictions $\approx G$	$\mathbf{w}_0^\top \mathbf{x}_0$	$\mathbf{w}_0^\top \mathbf{x}_1$	$\mathbf{w}_0^\top \mathbf{x}_2$	$\dots$	$\mathbf{w}_0^\top \mathbf{x}_{T-1}$				

MC: $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha_t (G - \mathbf{w}_t^\top \mathbf{x}_t)\mathbf{x}_t$

step size

all done at time T

Computational goals

Computation per step (including memory) must be

1. *Constant*. (non-increasing with number of episodes)

2. *Proportionate*. (proportional to number of weights, or O(n))

3. *Independent of span*. (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC: $$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G - \mathbf{w}_t^\top \mathbf{x}_t \right] \mathbf{x}_t, \qquad t = 0, \ldots, T-1$$

step size

all done at time T

Computational goals

Computation per step (including memory) must be

1. *Constant*. (non-increasing with number of episodes)

2. *Proportionate*. (proportional to number of weights, or O(n))

3. *Independent of span*. (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC: $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G - \mathbf{w}_t^\top \mathbf{x}_t\right] \mathbf{x}_t, \qquad t = 0, \ldots, T-1$

step size

all done at time T

What is the span?

Computational goals

Computation per step (including memory) must be

1. *Constant*. (non-increasing with number of episodes)

2. *Proportionate*. (proportional to number of weights, or O(n))

3. *Independent of span*. (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC: $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G - \mathbf{w}_t^\top \mathbf{x}_t \right] \mathbf{x}_t, \qquad t = 0, \ldots, T-1$

step size

all done at time T

What is the span? T

Computational goals

Computation per step (including memory) must be

1. *Constant*. (non-increasing with number of episodes)

2. *Proportionate*. (proportional to number of weights, or O(n))

3. *Independent of span*. (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC: $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G - \mathbf{w}_t^\top \mathbf{x}_t \right] \mathbf{x}_t, \qquad t = 0, \ldots, T-1$

step size

all done at time T

What is the span? T

Is MC indep of span?

Computational goals

Computation per step (including memory) must be

1. *Constant*. (non-increasing with number of episodes)

2. *Proportionate*. (proportional to number of weights, or O(n))

3. *Independent of span*. (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC: $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G - \mathbf{w}_t^\top \mathbf{x}_t \right] \mathbf{x}_t, \qquad t = 0, \ldots, T-1$

step size

all done at time T

What is the span? T

Is MC indep of span? No

Computational goals

Computation per step (including memory) must be

1. *Constant*. (non-increasing with number of episodes)

2. *Proportionate*. (proportional to number of weights, or O(n))

3. *Independent of span*. (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC: $$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[G - \mathbf{w}_t^\top \mathbf{x}_t \right] \mathbf{x}_t, \qquad t = 0, \ldots, T-1$$

step size

$\boxed{\text{all done at time } T}$

Computation and memory needed at step T increases with $T \Rightarrow$ not IoS

Final Result

Given:

$$\mathbf{w}_0 \quad \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T-1} \quad G$$

MC algorithm:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(G - \mathbf{w}_t^\top \mathbf{x}_t)\mathbf{x}_t, \quad t = 0, \ldots, T-1$$

Equivalent independent-of-span algorithm:

$$\begin{aligned}
\mathbf{w}_T &\doteq \mathbf{a}_{T-1} + G\mathbf{z}_{T-1}, && \mathbf{a}_t \in \mathbb{R}^d,\ \mathbf{z}_t \in \mathbb{R}^d \\
\mathbf{a}_0 &\doteq \mathbf{w}_0,\ \text{then } \mathbf{a}_t \doteq \mathbf{a}_{t-1} - \alpha_t \mathbf{x}_t \mathbf{x}_t^\top \mathbf{a}_{t-1}, && t = 1, \ldots, T-1 \\
\mathbf{z}_0 &\doteq \alpha_0 \mathbf{x}_0,\ \text{then } \mathbf{z}_t \doteq \mathbf{z}_{t-1} - \alpha_t \mathbf{x}_t \mathbf{x}_t^\top \mathbf{z}_{t-1} + \alpha_t \mathbf{x}_t, && t = 1, \ldots, T-1
\end{aligned}$$

Proved:

$$\mathbf{w}_T = \mathbf{w}_T \qquad \text{(the final weights of both algorithms are the same)}$$

MC: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(G - \mathbf{w}_t^\top \mathbf{x}_t)\mathbf{x}_t, \quad t = 0, \ldots, T-1$

$$\begin{aligned}
\mathbf{w}_T &= \mathbf{w}_{T-1} + \alpha \left(G - \mathbf{w}_{T-1}^\top \mathbf{x}_{T-1}\right) \mathbf{x}_{T-1} \\
&= \mathbf{w}_{T-1} + \alpha \mathbf{x}_{T-1} \left(-\mathbf{x}_{T-1}^\top \mathbf{w}_{T-1}\right) + \alpha G \mathbf{x}_{T-1} \\
&= \left(\mathbf{I} - \alpha \mathbf{x}_{T-1} \mathbf{x}_{T-1}^\top\right) \mathbf{w}_{T-1} + \alpha G \mathbf{x}_{T-1} \\
&= \mathbf{F}_{T-1} \mathbf{w}_{T-1} + \alpha G \mathbf{x}_{T-1}
\end{aligned}$$

where $\mathbf{F}_t \doteq \mathbf{I} - \alpha \mathbf{x}_t \mathbf{x}_t^\top$ is a *forgetting*, or *fading*, matrix. Now, recursing,

$$\begin{aligned}
&= \mathbf{F}_{T-1} \left(\mathbf{F}_{T-2} \mathbf{w}_{T-2} + \alpha G \mathbf{x}_{T-2}\right) + \alpha G \mathbf{x}_{T-1} \\
&= \mathbf{F}_{T-1} \mathbf{F}_{T-2} \mathbf{w}_{T-2} + \alpha G \left(\mathbf{F}_{T-1} \mathbf{x}_{T-2} + \mathbf{x}_{T-1}\right) \\
&= \mathbf{F}_{T-1} \mathbf{F}_{T-2} \left(\mathbf{F}_{T-3} \mathbf{w}_{T-3} + \alpha G \mathbf{x}_{T-3}\right) + \alpha G \left(\mathbf{F}_{T-1} \mathbf{x}_{T-2} + \mathbf{x}_{T-1}\right) \\
&= \mathbf{F}_{T-1} \mathbf{F}_{T-2} \mathbf{F}_{T-3} \mathbf{w}_{T-3} + \alpha G \left(\mathbf{F}_{T-1} \mathbf{F}_{T-2} \mathbf{x}_{T-3} + \mathbf{F}_{T-1} \mathbf{x}_{T-2} + \mathbf{x}_{T-1}\right) \\
&\vdots \\
&= \underbrace{\mathbf{F}_{T-1} \mathbf{F}_{T-2} \cdots \mathbf{F}_0 \mathbf{w}_0}_{\mathbf{a}_{T-1}} \quad + \quad \alpha G \underbrace{\sum_{k=0}^{T-1} \mathbf{F}_{T-1} \mathbf{F}_{T-2} \cdots \mathbf{F}_{k+1} \mathbf{x}_k}_{\mathbf{z}_{T-1}} \\
&= \mathbf{a}_{T-1} + \alpha G \mathbf{z}_{T-1},
\end{aligned}$$

auxiliary short-term-memory vectors $\mathbf{a}_t \in \mathbb{R}^d$, $\mathbf{z}_t \in \mathbb{R}^d$

$$
= \underbrace{\mathbf{F}_{T-1}\mathbf{F}_{T-2}\cdots\mathbf{F}_0\mathbf{w}_0}_{\mathbf{a}_{T-1}} \quad + \quad \alpha G \underbrace{\sum_{k=0}^{T-1}\mathbf{F}_{T-1}\mathbf{F}_{T-2}\cdots\mathbf{F}_{k+1}\mathbf{x}_k}_{\mathbf{z}_{T-1}}
$$

$$
= \mathbf{a}_{T-1} + \alpha G \mathbf{z}_{T-1}\,,
$$

$$= \underbrace{\mathbf{F}_{T-1}\mathbf{F}_{T-2}\cdots\mathbf{F}_0\mathbf{w}_0}_{\mathbf{a}_{T-1}} \quad + \quad \alpha G \underbrace{\sum_{k=0}^{T-1} \mathbf{F}_{T-1}\mathbf{F}_{T-2}\cdots\mathbf{F}_{k+1}\mathbf{x}_k}_{\mathbf{z}_{T-1}}$$

$$= \mathbf{a}_{T-1} + \alpha G \mathbf{z}_{T-1},$$

$$\begin{aligned}
\mathbf{z}_t &\doteq \sum_{k=0}^{t} \mathbf{F}_t\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\mathbf{x}_k, \qquad\qquad 1 \le t < T \\
&= \sum_{k=0}^{t-1} \mathbf{F}_t\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\mathbf{x}_k + \mathbf{x}_t \\
&= \mathbf{F}_t \sum_{k=0}^{t-1} \mathbf{F}_{t-1}\mathbf{F}_{t-2}\cdots\mathbf{F}_{k+1}\mathbf{x}_k + \mathbf{x}_t \\
&= \mathbf{F}_t\mathbf{z}_{t-1} + \mathbf{x}_t \\
&= \left(\mathbf{I} - \alpha\mathbf{x}_t\mathbf{x}_t^\top\right)\mathbf{z}_{t-1} + \mathbf{x}_t \\
&= \mathbf{z}_{t-1} - \alpha\mathbf{x}_t\mathbf{x}_t^\top\mathbf{z}_{t-1} + \mathbf{x}_t \\
&= \mathbf{z}_{t-1} - \alpha\left(\mathbf{z}_{t-1}^\top\mathbf{x}_t\right)\mathbf{x}_t + \mathbf{x}_t \\
&= \mathbf{z}_{t-1} + \left(1 - \alpha\mathbf{z}_{t-1}^\top\mathbf{x}_t\right)\mathbf{x}_t,
\end{aligned}$$

$$\mathbf{a}_t \ \doteq \ \mathbf{F}_t\mathbf{F}_{t-1}\cdots\mathbf{F}_0\mathbf{w}_0 \ = \ \mathbf{F}_t\mathbf{a}_{t-1} \ = \ \mathbf{a}_{t-1} - \alpha\mathbf{x}_t\mathbf{x}_t^\top\mathbf{a}_{t-1}, \quad 1 \le t < T$$

Final Result

Given:

$$\mathbf{w}_0 \quad \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T-1} \quad G$$

MC algorithm:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(G - \mathbf{w}_t^\top \mathbf{x}_t)\mathbf{x}_t, \quad t = 0, \ldots, T-1$$

Equivalent independent-of-span algorithm:

$$\begin{aligned}
&\mathbf{w}_T \doteq \mathbf{a}_{T-1} + G\mathbf{z}_{T-1}, && \mathbf{a}_t \in \mathbb{R}^d,\ \mathbf{z}_t \in \mathbb{R}^d \\
&\mathbf{a}_0 \doteq \mathbf{w}_0,\ \text{then } \mathbf{a}_t \doteq \mathbf{a}_{t-1} - \alpha_t \mathbf{x}_t \mathbf{x}_t^\top \mathbf{a}_{t-1}, && t = 1, \ldots, T-1 \\
&\mathbf{z}_0 \doteq \alpha_0 \mathbf{x}_0,\ \text{then } \mathbf{z}_t \doteq \mathbf{z}_{t-1} - \alpha_t \mathbf{x}_t \mathbf{x}_t^\top \mathbf{z}_{t-1} + \alpha_t \mathbf{x}_t, && t = 1, \ldots, T-1
\end{aligned}$$

Proved:

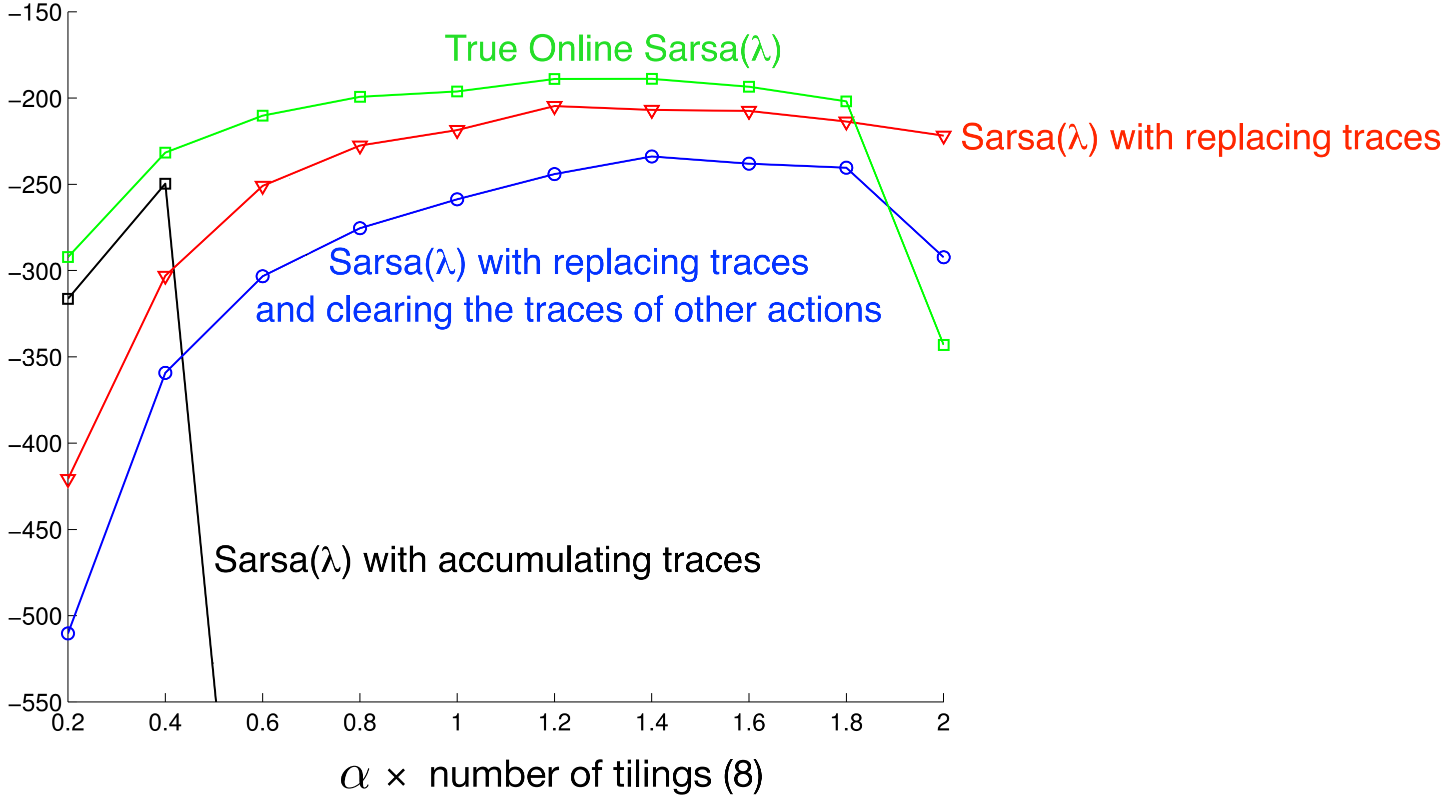
$$\mathbf{w}_T = \mathbf{w}_T \quad \text{(the final weights of both algorithms are the same)}$$

Conclusions from the forward-backward derivation

- We have derived dutch eligibility traces from an MC update, without any TD learning
- Dutch traces, and in fact all eligibility traces, are not about TD; they are about *efficient multi-step* learning
- We can derive new non-obvious algorithms that are equivalent to obvious algorithms but have better computational properties
- This is a different type of machine-learning result, an *algorithm equivalence*

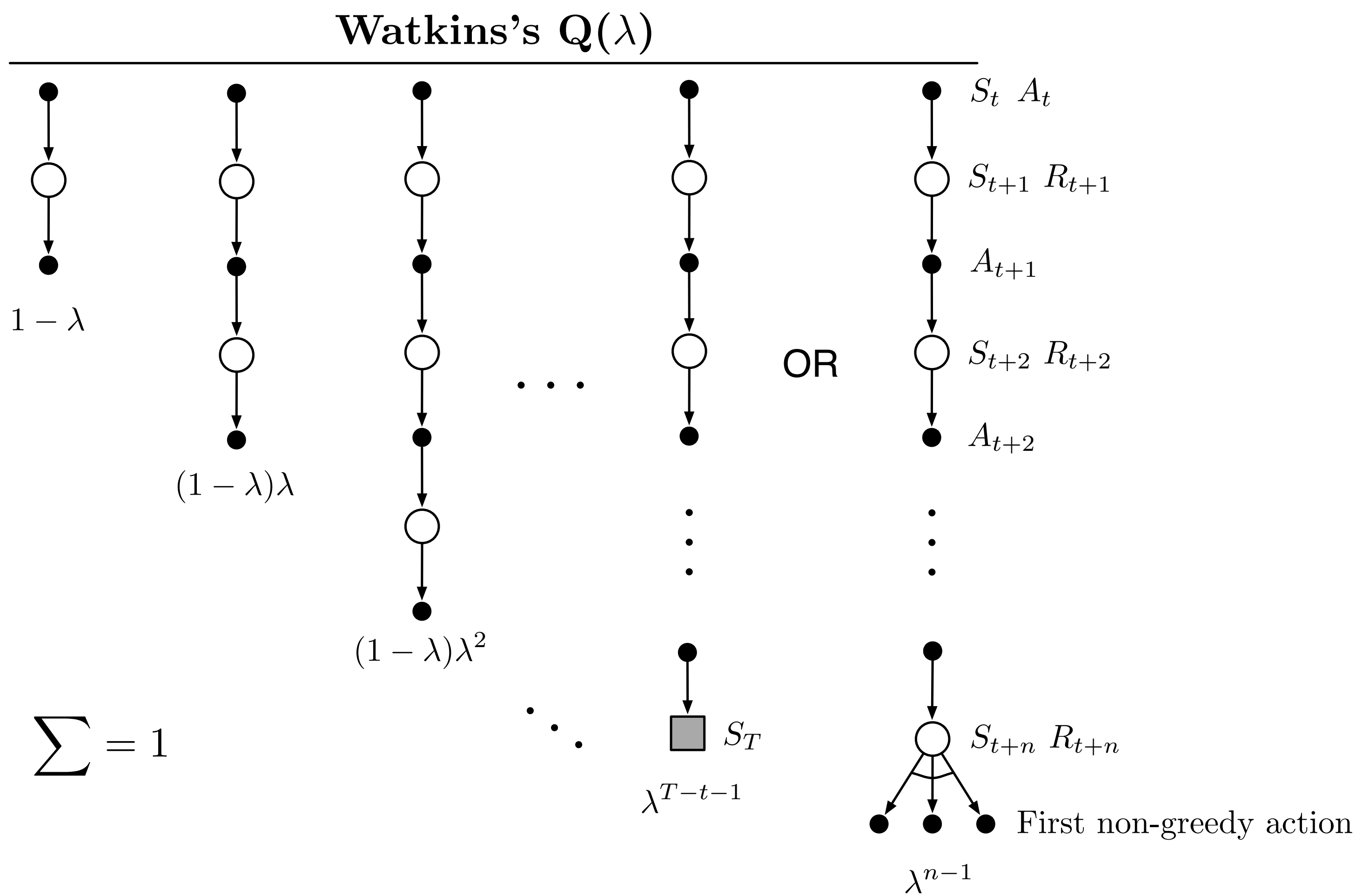
True online Sarsa(λ) results on Mountain Car

Mountain Car
Reward per episode
averaged over
first 20 episodes
and 100 runs

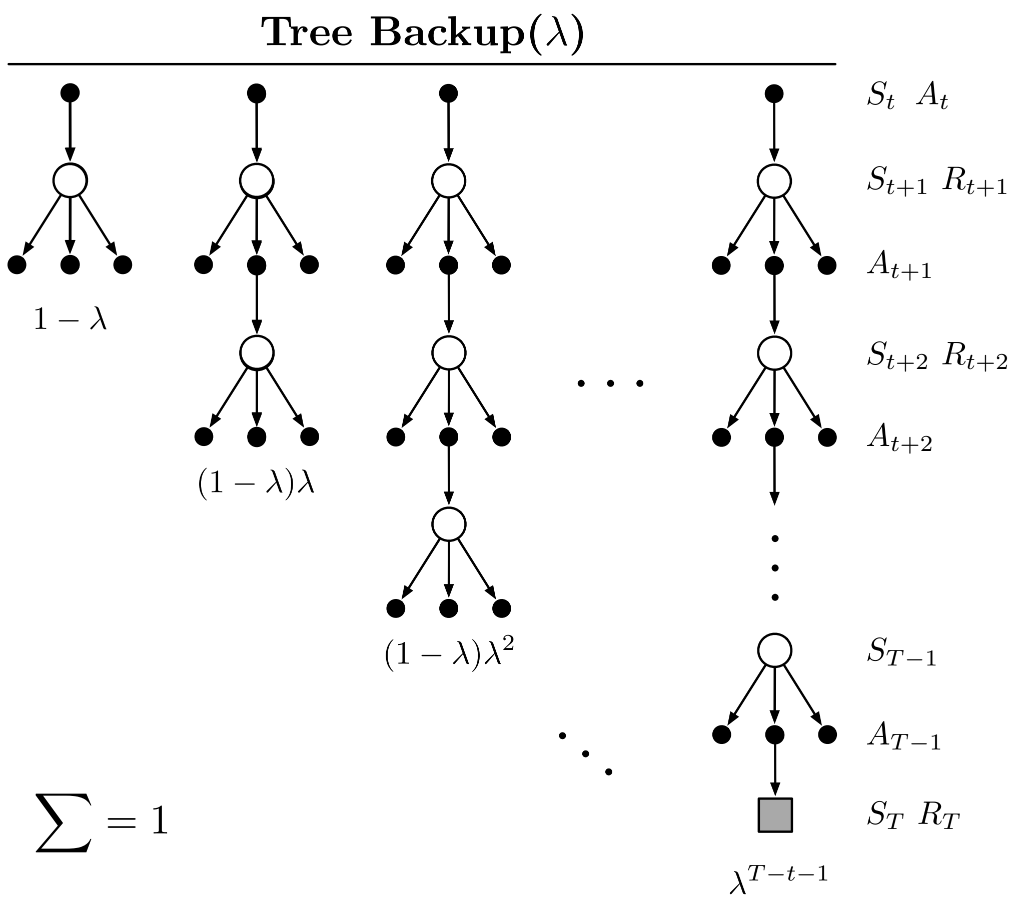


Adapted from van Seijen and Sutton (2014)

Other traces for Q: Original Watkins Q(λ)



Other traces for Q: Tree-Backup(λ)



Other traces for Q: Tree-Backup(λ)

Update Rules

$$\mathbf{z}_t \doteq \gamma_t \lambda_t \pi(A_t|S_t)\mathbf{z}_{t-1} + \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

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

- No importance sampling
- No guarantees of stability when used off-policy with powerful function approximation

Off-policy Traces with importance sampling

- Learning about an arbitrary policy typically requires the use of importance sampling ratios between the behavior policy and the target policy. $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
- We define state based returns, and a forward view update.

$$G_t^{\lambda s} \doteq \rho_t\Big(R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\hat{v}(S_{t+1},\mathbf{w}_t) + \lambda_{t+1}G_{t+1}^{\lambda s}\big)\Big) + (1-\rho_t)\hat{v}(S_t,\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left(G_t^{\lambda s} - \hat{v}(S_t,\mathbf{w}_t)\right)\nabla\hat{v}(S_t,\mathbf{w}_t)$$

- After some work (Section 12.9), we get another trace.

$$\mathbf{z}_t \doteq \rho_t\big(\gamma_t\lambda_t\mathbf{z}_{t-1} + \nabla\hat{v}(S_t,\mathbf{w}_t)\big)$$

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

- This is not guaranteed to be stable with strong function approximation, and importance sampling can introduce substantial variance. Can still work in practice.

Conclusions regarding Eligibility Traces

- Provide an efficient, incremental way to combine MC and TD
  - Includes advantages of MC (better when non-Markov)
  - Includes advantages of TD (faster, comp. congenial)
- True online TD(λ) is new and best
  - Is exactly equivalent to online λ-return algorithm
- There is a true online Sarsa(λ)
- Three varieties of traces: accumulating, dutch, (replacing)
- Traces for prediction and on-policy control
- Traces for off-policy control and prediction
- Trace methods often perform better than *n*-step methods
- Traces do have a small cost in computation (≈x2)