True Online Temporal-Difference Learning

Harm van Seijen
Microsoft Research

joint work with

Rupam Mahmood

Patrick Pilarski

Marlos Machado

Rich Sutton

Outline

- Part 1: true online temporal-difference learning
- Part 2: effective multi-step learning for non-linear FA

Outline

Markov Decision Processes

A *Markov decision process* (MDP) can be described by 5-tuple: $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, with

- $\mathcal{S}$: the set of all states
- $\mathcal{A}$: the set of all actions
- $p(s'|s,a)$: the transition probability function
- $r(s,a,s')$: the reward function
- γ: the discount factor

Markov Decision Processes

A *Markov decision process* (MDP) can be described by 5-tuple: $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, with

- $\mathcal{S}$: the set of all states
- $\mathcal{A}$: the set of all actions
- $p(s'|s,a)$: the transition probability function
- $r(s,a,s')$: the reward function
- γ: the discount factor

policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, function giving the selection probability for each action conditioned on the state

Markov Decision Processes

A *Markov decision process* (MDP) can be described by 5-tuple: $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, with

- $\mathcal{S}$: the set of all states
- $\mathcal{A}$: the set of all actions
- $p(s'|s,a)$: the transition probability function
- $r(s,a,s')$: the reward function
- γ: the discount factor

policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, function giving the selection probability for each action conditioned on the state

The *return* at time t: $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$$

Markov Decision Processes

A *Markov decision process* (MDP) can be described by 5-tuple: $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, with

- $\mathcal{S}$: the set of all states
- $\mathcal{A}$: the set of all actions
- $p(s'|s,a)$: the transition probability function
- $r(s,a,s')$: the reward function
- γ: the discount factor

policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$, function giving the selection probability for each action conditioned on the state

The *return* at time t: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$

state-value function: $v^{\pi}(s) = \mathbb{E}\{G_t \mid S_t = s, \pi\}$

action-value function: $q^{\pi}(s,a) = \mathbb{E}\{G_t \mid S_t = s, A_t, = a, \pi\}$

Estimating the value function

Find a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\hat{V}(s|\boldsymbol{\theta})$ accurately approximates $v_\pi(s)$ for relevant states s.

Estimating the value function

Find a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\hat{V}(s|\boldsymbol{\theta})$ accurately approximates $v_\pi(s)$ for relevant states s.

Error function: $$E(\boldsymbol{\theta}) := \frac{1}{2}\sum_i d_\pi(s_i)\left[v_\pi(s_i) - \hat{V}(s_i|\boldsymbol{\theta})\right]^2$$

where d_π is the stationary distribution induced by π.

Estimating the value function

Find a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\hat{V}(s|\boldsymbol{\theta})$ accurately approximates $v_\pi(s)$ for relevant states s.

Error function: $$E(\boldsymbol{\theta}) := \frac{1}{2}\sum_i d_\pi(s_i)\big[v_\pi(s_i) - \hat{V}(s_i|\boldsymbol{\theta})\big]^2$$

where d_π is the stationary distribution induced by π.

Stochastic gradient descent (sampling from the stationary distribution):

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha\frac{1}{2}\nabla_\theta\big[v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\big]^2 \\
&= \boldsymbol{\theta}_t + \alpha\Big(v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t)
\end{aligned}$$

Estimating the value function

Find a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\hat{V}(s|\boldsymbol{\theta})$ accurately approximates $v_\pi(s)$ for relevant states s.

Error function: $$E(\boldsymbol{\theta}) := \frac{1}{2}\sum_i d_\pi(s_i)\big[v_\pi(s_i) - \hat{V}(s_i|\boldsymbol{\theta})\big]^2$$

where d_π is the stationary distribution induced by π.

Stochastic gradient descent (sampling from the stationary distribution):

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha\frac{1}{2}\nabla_\theta\big[v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\big]^2 \\ &= \boldsymbol{\theta}_t + \alpha\Big(v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t)\end{aligned}$$

Because $v_\pi(S_t)$ is unknown:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\Big(U_t - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t).$$

where U_t is an estimate of $v_\pi(S_t)$ that we will call the update target.

Estimating the value function

Find a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\hat{V}(s|\boldsymbol{\theta})$ accurately approximates $v_\pi(s)$ for relevant states s.

Error function: $$E(\boldsymbol{\theta}) := \frac{1}{2}\sum_i d_\pi(s_i)\left[v_\pi(s_i) - \hat{V}(s_i|\boldsymbol{\theta})\right]^2$$

where d_π is the stationary distribution induced by π.

Stochastic gradient descent (sampling from the stationary distribution):

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha\frac{1}{2}\nabla_\theta\left[v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\right]^2 \\ &= \boldsymbol{\theta}_t + \alpha\Big(v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t) \end{aligned}$$

Because $v_\pi(S_t)$ is unknown:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\Big(U_t - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t).$$

where U_t is an estimate of $v_\pi(S_t)$ that we will call the update target.

With linear function approximation: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\big(U_t - \boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t\big)\boldsymbol{\phi}_t$

Estimating the value function

Find a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $\hat{V}(s|\boldsymbol{\theta})$ accurately approximates $v_\pi(s)$ for relevant states s.

Error function: $$E(\boldsymbol{\theta}) := \frac{1}{2}\sum_i d_\pi(s_i)\left[v_\pi(s_i) - \hat{V}(s_i|\boldsymbol{\theta})\right]^2$$

where d_π is the stationary distribution induced by π.

Stochastic gradient descent (sampling from the stationary distribution):

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha\frac{1}{2}\nabla_\theta\left[v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\right]^2 \\ &= \boldsymbol{\theta}_t + \alpha\Big(v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t) \end{aligned}$$

Because $v_\pi(S_t)$ is unknown:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\Big(U_t - \hat{V}(S_t|\boldsymbol{\theta}_t)\Big)\nabla_\theta\hat{V}(S_t|\boldsymbol{\theta}_t).$$

where U_t is an estimate of $v_\pi(S_t)$ that we will call the update target.

With linear function approximation: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\left(U_t - \boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t\right)\boldsymbol{\phi}_t$

unbiased estimate of $v_\pi(S_t)$: $U_t = G_t$

Temporal-difference Learning

- Temporal-Difference (TD) learning exploits knowledge about structure of v_π .

Temporal-difference Learning

- Temporal-Difference (TD) learning exploits knowledge about structure of v_π .
- Bellman Equation:

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_\pi(s')\right] \qquad \text{for all } s$$

Temporal-difference Learning

- Temporal-Difference (TD) learning exploits knowledge about structure of v_π .
- Bellman Equation:

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_\pi(s')\right] \qquad \text{for all } s$$

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t \sim \pi(S_t, \cdot)\right]$$

Temporal-difference Learning

- Temporal-Difference (TD) learning exploits knowledge about structure of v_π .
- Bellman Equation:

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\big[r(s,a,s') + \gamma v_\pi(s')\big] \qquad \text{for all } s$$

$$v_\pi(s) = \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t \sim \pi(S_t,\cdot)\big]$$

- TD(0) update target (1-step update target):

$$U_t = R_{t+1} + \gamma\, \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+1}$$

Temporal-difference Learning

- Temporal-Difference (TD) learning exploits knowledge about structure of v_π .
- Bellman Equation:

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_\pi(s')\right] \qquad \text{for all } s$$

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t \sim \pi(S_t, \cdot)\right]$$

- TD(0) update target (1-step update target):

$$U_t = R_{t+1} + \gamma\, \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+1}$$

- 3-step update target:

$$U_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+3}$$

TD(λ)

- update equations for linear function approximation:

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \,, \\
\boldsymbol{e}_t &= \gamma \lambda \boldsymbol{e}_{t-1} + \boldsymbol{\phi}_t \,, \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \delta_t \, \boldsymbol{e}_t \,,
\end{aligned}
$$

TD(λ)

- update equations for linear function approximation:

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \,, \\
\mathbf{e}_t &= \gamma \lambda \mathbf{e}_{t-1} + \boldsymbol{\phi}_t \,, \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \delta_t \, \mathbf{e}_t \,,
\end{aligned}
$$

- TD(λ) is a multi-step method, even though the update target looks like a 1-step update target.

TD(λ)

- update equations for linear function approximation:

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \,, \\
\boldsymbol{e}_t &= \gamma \lambda \boldsymbol{e}_{t-1} + \boldsymbol{\phi}_t \,, \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \delta_t \, \boldsymbol{e}_t \,,
\end{aligned}
$$

- TD(λ) is a multi-step method, even though the update target looks like a 1-step update target.

- This update is different from the general TD update rule.

the traditional forward view of TD(λ)

- the λ-return algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\big(G_t^\lambda - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\big)\boldsymbol{\phi}_t$$

where G_t^λ is the λ-return, defined as:

$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$

with $G_t^{(n)} = \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+n}$

the traditional forward view of TD(λ)

- the λ-return algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\big(G_t^\lambda - \boldsymbol{\theta}_t^\top \phi_t\big)\phi_t$$

where G_t^λ is the λ-return, defined as:

$$G_t^\lambda = (1-\lambda)\textstyle\sum_{n=1}^\infty \lambda^{n-1} G_t^{(n)}$$

with $G_t^{(n)} = \sum_{k=1}^n \gamma^{k-1} R_{t+k} + \gamma^n \boldsymbol{\theta}^\top \phi_{t+n}$

note:

$\lambda = 0 \quad : \quad G_t^\lambda = G_t^{(1)}$

$\lambda = 1 \quad : \quad G_t^\lambda = G_t$

the traditional forward view of TD(λ)

- the λ-return algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\big(G_t^\lambda - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\big)\boldsymbol{\phi}_t$$

where G_t^λ is the λ-return, defined as:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

with $G_t^{(n)} = \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+n}$

note:

$\lambda = 0 \;:\; G_t^\lambda = G_t^{(1)}$

$\lambda = 1 \;:\; G_t^\lambda = G_t$

- **theorem*:** for small step-sizes, $\boldsymbol{\theta}_T$ computed by TD(λ) is approximately the same as $\boldsymbol{\theta}_T$ computed by the λ-return algorithm.

*see: Bertsekas, D. P. and Tsitsiklis, J. N. (1996). Neuro-Dynamic Programming.

How to set λ?

- λ controls a trade-off between variance and bias of the update target, in general the best value of λ will differ from domain to domain.

How to set λ?

- λ controls a trade-off between variance and bias of the update target, in general the best value of λ will differ from domain to domain.
- λ not only influences the speed of convergence, but in case of function approximation it also influences the asymptotic performance.

How to set λ?

- λ controls a trade-off between variance and bias of the update target, in general the best value of λ will differ from domain to domain.

- λ not only influences the speed of convergence, but in case of function approximation it also influences the asymptotic performance.

- theoretical results for TD(λ) (Peter Dayan, 1992):
  - for $\lambda = 1$: convergence to LMS solution
  - for $\lambda < 1$: convergence to a different fixed point

online vs offline methods

- **online method:** the value of each visited state is updated at the time step immediately after the visit.
- **offline method:** the value of each visited state is updated at the end of an episode.

online vs offline methods

- **online method:** the value of each visited state is updated at the time step immediately after the visit.
- **offline method:** the value of each visited state is updated at the end of an episode.

- TD(λ) is an online method; the traditional λ-return algorithm is an offline method.

online vs offline methods

- **online method:** the value of each visited state is updated at the time step immediately after the visit.
- **offline method:** the value of each visited state is updated at the end of an episode.

- TD(λ) is an online method; the traditional λ-return algorithm is an offline method.

*Is it possible to construct an online version of the λ-return algorithm that approximates TD(λ) at **all** time steps?*

the challenge of an online forward view

- To compute $\boldsymbol{\theta}_t$, no data beyond time t should be used.
- At the same time, we want to have multi-step update targets that look many time steps ahead.

the challenge of an online forward view

- To compute $\boldsymbol{\theta}_t$, no data beyond time t should be used.
- At the same time, we want to have multi-step update targets that look many time steps ahead.

the trick:

Use update targets that grow with the data-horizon.

interim update target

- normal update targets:

$$\phi_t \quad \rightarrow \quad U_t$$

- interim update targets:

$$\phi_t \quad \rightarrow \quad U_t^1,\ U_t^2,\ U_t^3, \ldots,\ U_t^h$$

interim update target

- normal update targets:

$$\phi_t \quad \rightarrow \quad U_t$$

- interim update targets:

$$\phi_t \quad \rightarrow \quad U_t^1,\ U_t^2,\ U_t^3, \ldots,\ U_t^h$$

data-horizon: time step up to which data is observed

interim λ-return

- λ-return: $$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- interim λ-return: replace all *n*-step returns with $n > h\text{-}t$ with the $(h\text{-}t)$-step return

$$\begin{aligned}
G_t^{\lambda|h} &= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + (1-\lambda)\sum_{n=h-t}^{\infty}\lambda^{n-1}G_t^{(h-t)} \\
&= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + G_t^{(h-t)}\cdot\left[(1-\lambda)\sum_{n=h-t}^{\infty}\lambda^{n-1}\right] \\
&= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + G_t^{(h-t)}\cdot\left[\lambda^{h-t-1}(1-\lambda)\sum_{k=0}^{\infty}\lambda^{k}\right] \\
&= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{h-t-1}G_t^{(h-t)}
\end{aligned}$$

update sequences

$$t = 1: \quad \boldsymbol{\theta}_1^1 = \boldsymbol{\theta}_0^1 + \alpha\Big(G_0^{\lambda|1} - (\boldsymbol{\theta}_0^1)^\top \boldsymbol{\phi}_0\Big)\boldsymbol{\phi}_0$$

$$\begin{aligned} t = 2: \quad \boldsymbol{\theta}_1^2 &= \boldsymbol{\theta}_0^2 + \alpha\Big(G_0^{\lambda|2} - (\boldsymbol{\theta}_0^2)^\top \boldsymbol{\phi}_0\Big)\boldsymbol{\phi}_0 \\ \boldsymbol{\theta}_2^2 &= \boldsymbol{\theta}_1^2 + \alpha\Big(G_1^{\lambda|2} - (\boldsymbol{\theta}_1^2)^\top \boldsymbol{\phi}_1\Big)\boldsymbol{\phi}_1 \end{aligned}$$

$$\begin{aligned} t = 3: \quad \boldsymbol{\theta}_1^3 &= \boldsymbol{\theta}_0^3 + \alpha\Big(G_0^{\lambda|3} - (\boldsymbol{\theta}_0^3)^\top \boldsymbol{\phi}_0\Big)\boldsymbol{\phi}_0 \\ \boldsymbol{\theta}_2^3 &= \boldsymbol{\theta}_1^3 + \alpha\Big(G_1^{\lambda|3} - (\boldsymbol{\theta}_1^3)^\top \boldsymbol{\phi}_1\Big)\boldsymbol{\phi}_1 \\ \boldsymbol{\theta}_3^3 &= \boldsymbol{\theta}_2^3 + \alpha\Big(G_2^{\lambda|3} - (\boldsymbol{\theta}_2^3)^\top \boldsymbol{\phi}_2\Big)\boldsymbol{\phi}_2 \end{aligned}$$

with $\boldsymbol{\theta}_0^t := \boldsymbol{\theta}_{init}$

online lambda-return algorithm.

$$\boldsymbol{\theta}_t := \boldsymbol{\theta}_t^t$$

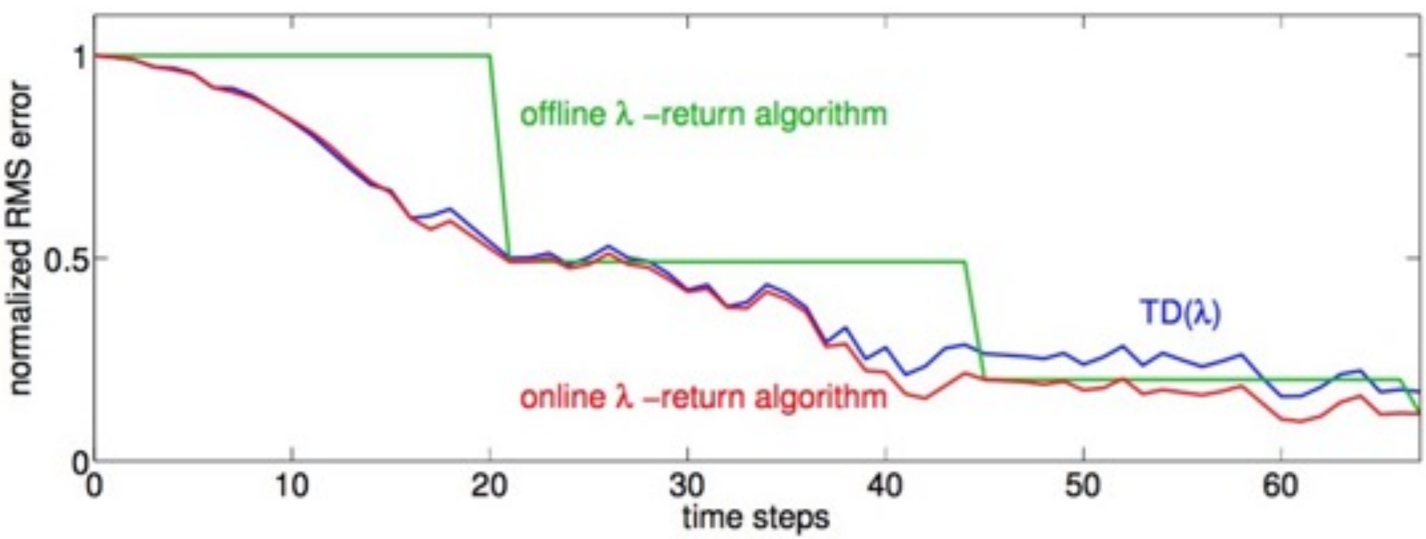
$$\boldsymbol{\theta}_{k+1}^t := \boldsymbol{\theta}_k^t + \alpha\Big(G_k^{\lambda|t} - (\boldsymbol{\theta}_k^t)^\top \boldsymbol{\phi}_k\Big)\boldsymbol{\phi}_k\,, \qquad \text{for } 0 \le k < t$$

with

$$G_k^{\lambda|t} := (1-\lambda)\sum_{n=1}^{t-k-1} \lambda^{n-1} G_k^{(n)} + \lambda^{t-k-1} G_k^{(t-k)}$$

online vs. offline λ-return algorithm

- performance on a 10-state random walk task for the first 3 episodes ($\lambda = 1$, $\alpha = 0.2$)

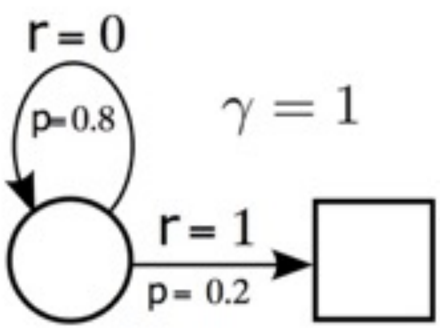


Theorem*

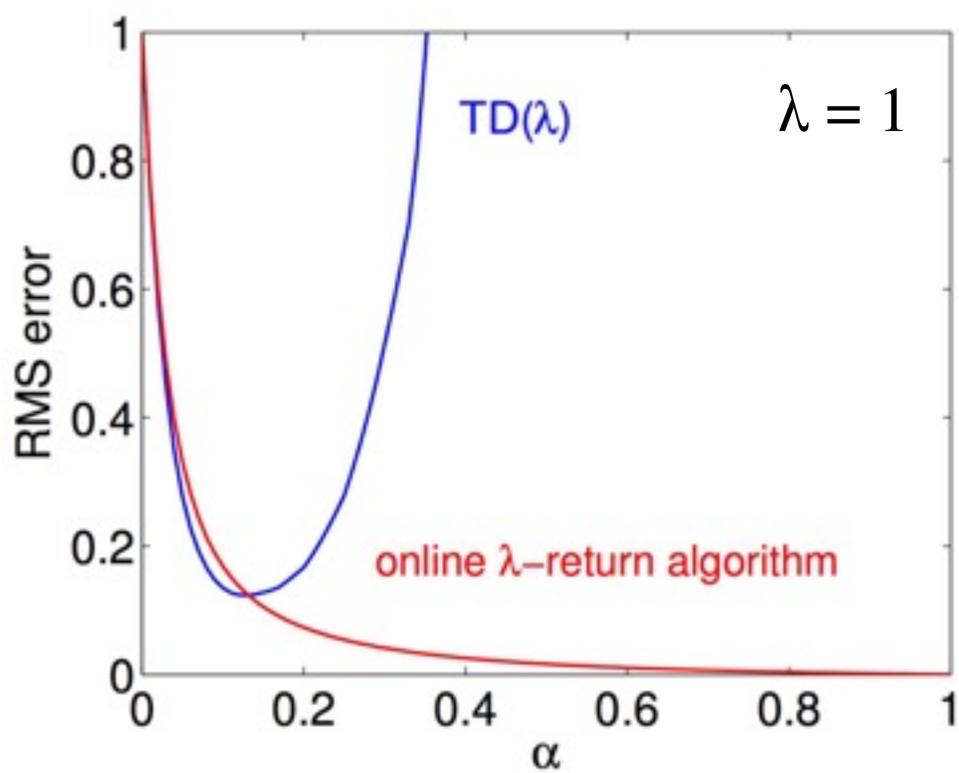
> "For small step-size, the online λ-return algorithm behaves like TD(λ) at all time steps"

*see Theorem 1: *van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., and Sutton, R. S. True online temporal-difference learning. Journal of Machine Learning Research, 17(145):1–40, 2016.*

Sensitivity of TD(λ) to Divergence

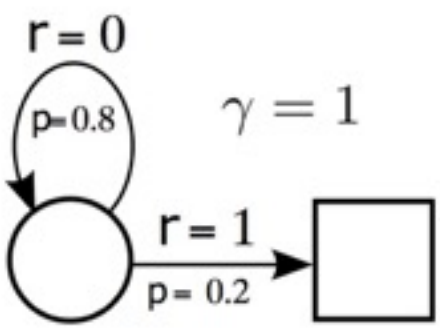


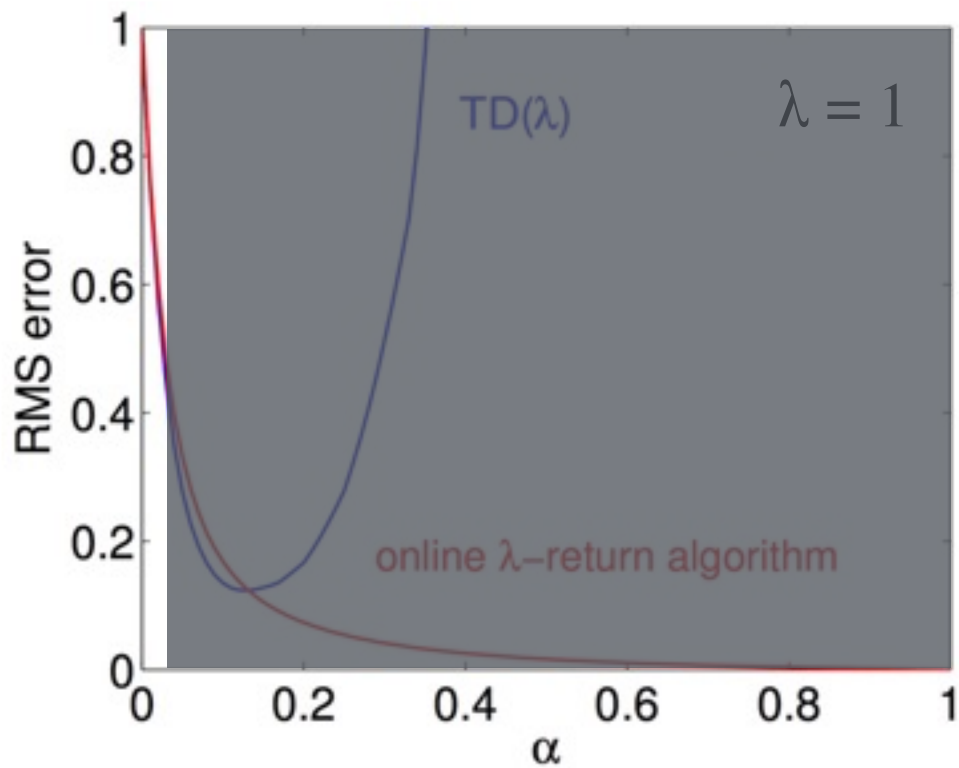
RMS error during early learning



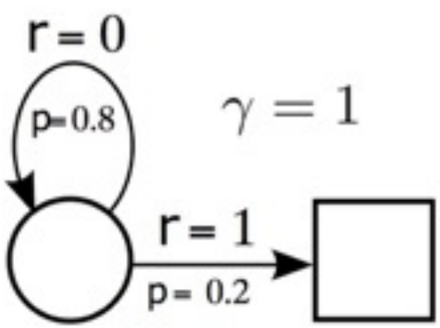
Sensitivity of TD(λ) to Divergence

RMS error during early learning

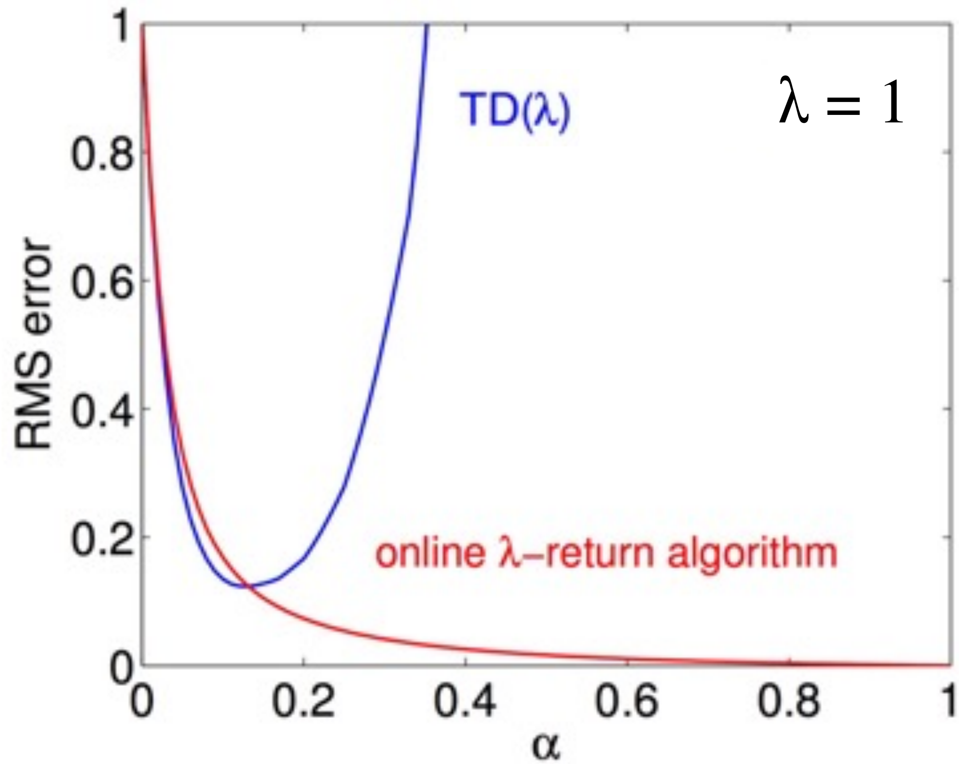




Sensitivity of TD(λ) to Divergence



RMS error during early learning



Computational Complexity

$$h = 1: \quad \boldsymbol{\theta}_1^1 = \boldsymbol{\theta}_0^1 + \alpha\left[G_0^{\lambda|1} - (\boldsymbol{\theta}_0^1)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$h = 2: \quad \boldsymbol{\theta}_1^2 = \boldsymbol{\theta}_0^2 + \alpha\left[G_0^{\lambda|2} - (\boldsymbol{\theta}_0^2)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$\boldsymbol{\theta}_2^2 = \boldsymbol{\theta}_1^2 + \alpha\left[G_1^{\lambda|2} - (\boldsymbol{\theta}_1^2)^\top \boldsymbol{\phi}_1\right]\boldsymbol{\phi}_1$$

$$h = 3: \quad \boldsymbol{\theta}_1^3 = \boldsymbol{\theta}_0^3 + \alpha\left[G_0^{\lambda|3} - (\boldsymbol{\theta}_0^3)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$\boldsymbol{\theta}_2^3 = \boldsymbol{\theta}_1^3 + \alpha\left[G_1^{\lambda|3} - (\boldsymbol{\theta}_1^3)^\top \boldsymbol{\phi}_1\right]\boldsymbol{\phi}_1$$

$$\boldsymbol{\theta}_3^3 = \boldsymbol{\theta}_2^3 + \alpha\left[G_2^{\lambda|3} - (\boldsymbol{\theta}_2^3)^\top \boldsymbol{\phi}_2\right]\boldsymbol{\phi}_2$$

Computational Complexity

$$h = 1: \quad \boldsymbol{\theta}_1^1 = \boldsymbol{\theta}_0^1 + \alpha\left[G_0^{\lambda|1} - (\boldsymbol{\theta}_0^1)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$h = 2: \quad \boldsymbol{\theta}_1^2 = \boldsymbol{\theta}_0^2 + \alpha\left[G_0^{\lambda|2} - (\boldsymbol{\theta}_0^2)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$\boldsymbol{\theta}_2^2 = \boldsymbol{\theta}_1^2 + \alpha\left[G_1^{\lambda|2} - (\boldsymbol{\theta}_1^2)^\top \boldsymbol{\phi}_1\right]\boldsymbol{\phi}_1$$

$$h = 3: \quad \boldsymbol{\theta}_1^3 = \boldsymbol{\theta}_0^3 + \alpha\left[G_0^{\lambda|3} - (\boldsymbol{\theta}_0^3)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$\boldsymbol{\theta}_2^3 = \boldsymbol{\theta}_1^3 + \alpha\left[G_1^{\lambda|3} - (\boldsymbol{\theta}_1^3)^\top \boldsymbol{\phi}_1\right]\boldsymbol{\phi}_1$$

$$\boldsymbol{\theta}_3^3 = \boldsymbol{\theta}_2^3 + \alpha\left[G_2^{\lambda|3} - (\boldsymbol{\theta}_2^3)^\top \boldsymbol{\phi}_2\right]\boldsymbol{\phi}_2$$

Computational Complexity

$$h = 1: \quad \boldsymbol{\theta}_1^1 = \boldsymbol{\theta}_0^1 + \alpha\left[G_0^{\lambda|1} - (\boldsymbol{\theta}_0^1)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$h = 2: \quad \boldsymbol{\theta}_1^2 = \boldsymbol{\theta}_0^2 + \alpha\left[G_0^{\lambda|2} - (\boldsymbol{\theta}_0^2)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$\boldsymbol{\theta}_2^2 = \boldsymbol{\theta}_1^2 + \alpha\left[G_1^{\lambda|2} - (\boldsymbol{\theta}_1^2)^\top \boldsymbol{\phi}_1\right]\boldsymbol{\phi}_1$$

$$h = 3: \quad \boldsymbol{\theta}_1^3 = \boldsymbol{\theta}_0^3 + \alpha\left[G_0^{\lambda|3} - (\boldsymbol{\theta}_0^3)^\top \boldsymbol{\phi}_0\right]\boldsymbol{\phi}_0$$

$$\boldsymbol{\theta}_2^3 = \boldsymbol{\theta}_1^3 + \alpha\left[G_1^{\lambda|3} - (\boldsymbol{\theta}_1^3)^\top \boldsymbol{\phi}_1\right]\boldsymbol{\phi}_1$$

$$\boldsymbol{\theta}_3^3 = \boldsymbol{\theta}_2^3 + \alpha\left[G_2^{\lambda|3} - (\boldsymbol{\theta}_2^3)^\top \boldsymbol{\phi}_2\right]\boldsymbol{\phi}_2$$

True online TD(λ)

- true online TD(λ) is an efficient implementation of the online λ-return algorithm

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \\
\mathbf{e}_t &= \gamma\lambda \mathbf{e}_{t-1} + \boldsymbol{\phi}_t - \alpha\gamma\lambda[\mathbf{e}_{t-1}^\top \boldsymbol{\phi}_t]\,\boldsymbol{\phi}_t \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\delta_t\,\mathbf{e}_t + \alpha[\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t][\mathbf{e}_t - \boldsymbol{\phi}_t]
\end{aligned}
$$

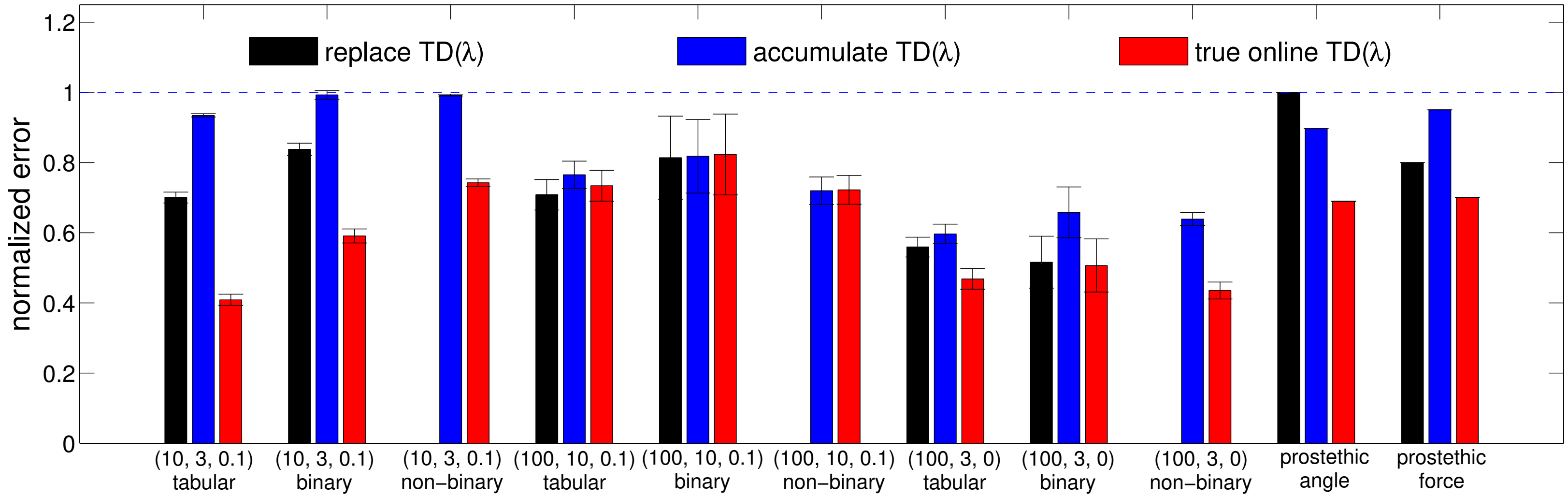
True online TD(λ)

- true online TD(λ) is an efficient implementation of the online λ-return algorithm

dutch trace

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \\
\boldsymbol{e}_t &= \gamma\lambda \boldsymbol{e}_{t-1} + \boldsymbol{\phi}_t - \alpha\gamma\lambda[\boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_t]\, \boldsymbol{\phi}_t \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\delta_t\, \boldsymbol{e}_t + \alpha[\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t][\boldsymbol{e}_t - \boldsymbol{\phi}_t]
\end{aligned}$$

Empirical Comparison



- in all domains, true online TD(λ) performs at least as good as replace/accumulate TD(λ)

Outline

- Part 1:   true online temporal-difference learning
- **Part 2:   effective multi-step learning for non-linear FA**

Computational Cost

- Implementing the online forward view is computationally very expensive.
  - Memory as well as computation time per time step grows over time.

- In the case of linear FA there is an efficient backward view with exact equivalence: true online TD(λ).
  - Computational cost is span-independent and linear in the number of features.

- In the case of non-linear FA such an efficient backward view does not appear to exist.

New Research Question

Is it possible to construct a different online forward view, with a performance close to that of the online λ-return algorithm, that can be implemented efficiently?

New Research Question

Is it possible to construct a different online forward view, with a performance close to that of the online λ-return algorithm, that can be implemented efficiently?

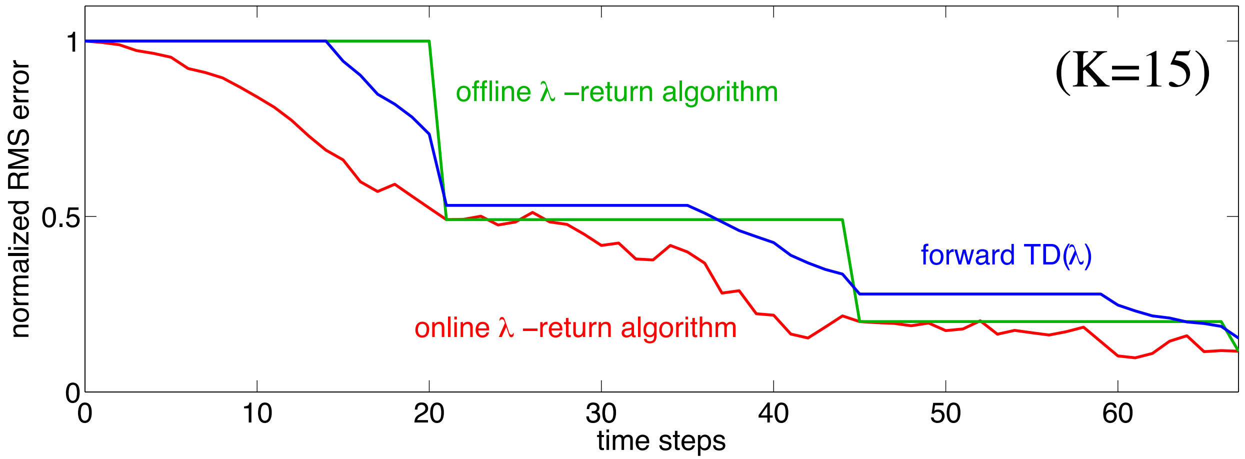
Answer: Yes

forward TD(λ)

- Uses online λ-return with fixed horizon, K steps ahead: $G_t^{\lambda|t+K}$
- As a consequence, updates occur with a delay of K time steps.
- Computational cost is span-independent and efficient (computational complexity equal to TD(0)).

forward TD(λ)

- Uses online λ-return with fixed horizon, K steps ahead: $G_t^{\lambda|t+K}$
- As a consequence, updates occur with a delay of K time steps.
- Computational cost is span-independent and efficient (computational complexity equal to TD(0)).



How to set K?

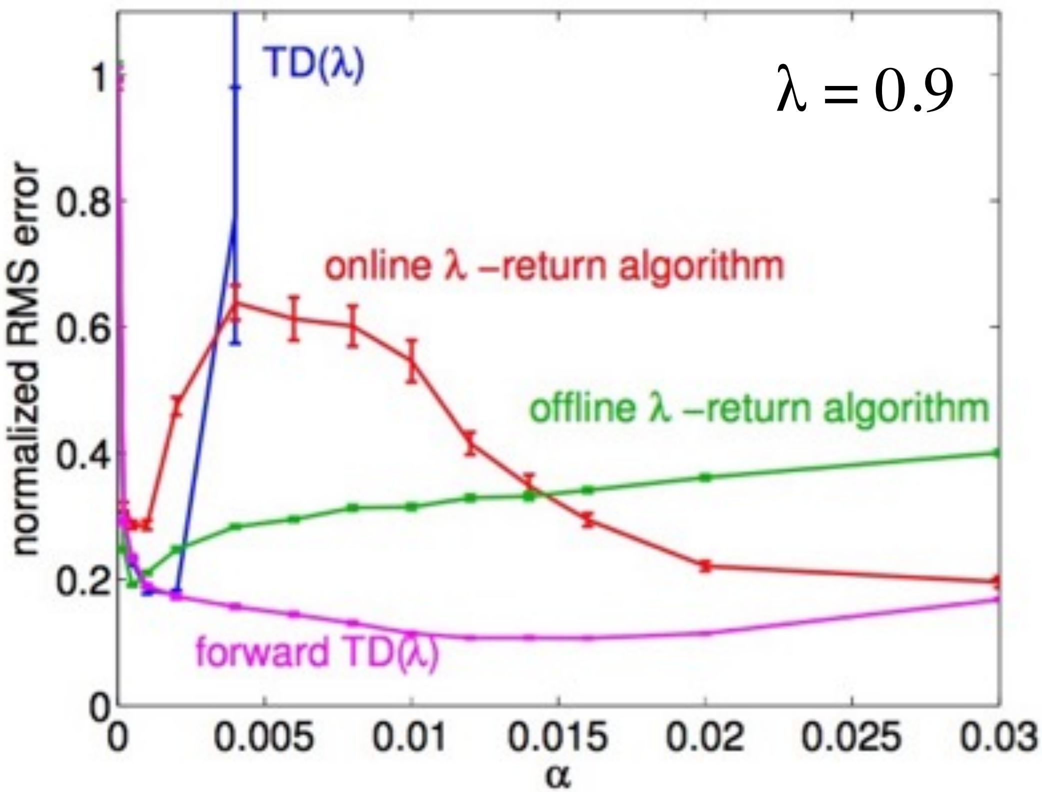
- Setting K involves a trade-off:
  - small K : less delay in updates
  - large K : better approximation of the λ-return
- How well $G_t^{\lambda|t+K}$ approximates G_t^{λ} depends on K, but also on $\gamma\lambda$.
- Whereas the weight of R_{t+1} in G_t^{λ} is 1, the weight of R_{t+n} is only $\gamma\lambda^{n-1}$.
  - Example: $\gamma\lambda = 0.5$ and $n = 20$, then $\gamma\lambda^{n-1}$ is about 10^{-6}.
- Strategy: set K such that $\gamma\lambda^{K-1}$ is just below η, with η some tiny number like 0.01

How to set K?

- Setting K involves a trade-off:
  - small K : less delay in updates
  - large K : better approximation of the λ-return
- How well $G_t^{\lambda|t+K}$ approximates G_t^{λ} depends on K, but also on $\gamma\lambda$.
- Whereas the weight of R_{t+1} in G_t^{λ} is 1, the weight of R_{t+n} is only $\gamma\lambda^{n-1}$.
  - Example: $\gamma\lambda = 0.5$ and $n = 20$, then $\gamma\lambda^{n-1}$ is about 10^{-6}.
- Strategy: set K such that $\gamma\lambda^{K-1}$ is just below η, with η some tiny number like 0.01

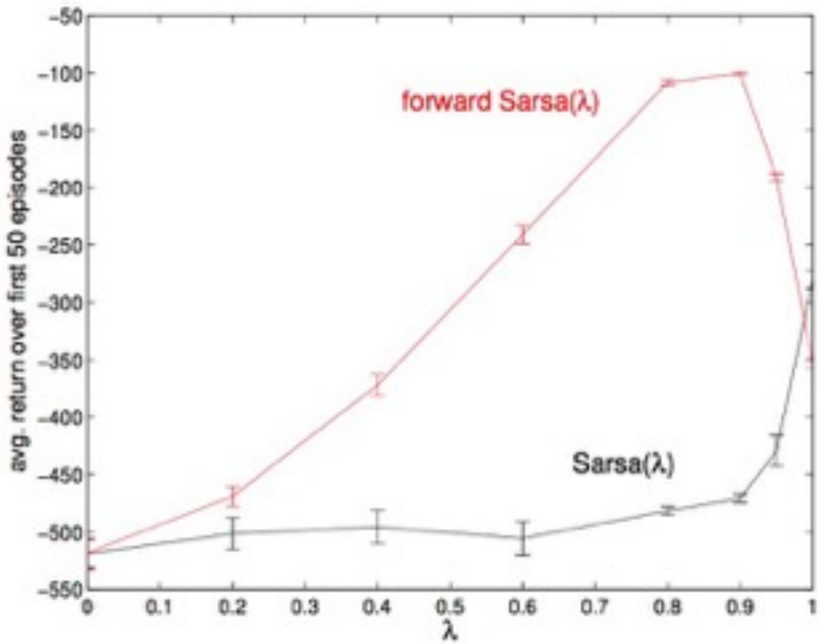
Results on Prediction Task

mountain-car task with non-linear FA

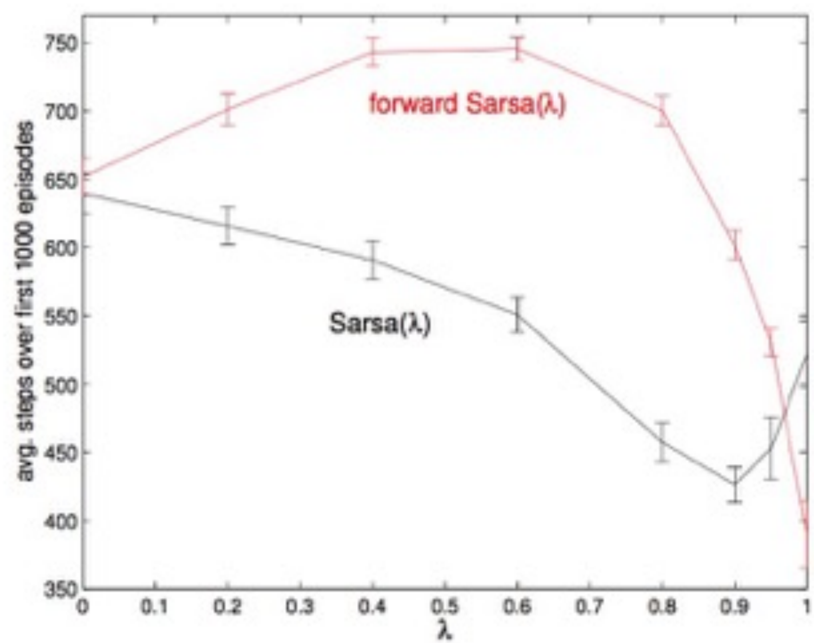


Results on 2 Control Tasks

mountain-car task



cart-pole task



Summary

- The online λ-return algorithm outperforms TD(λ), but is computationally very expensive.
- For linear FA, an efficient backward view exists with exact equivalence: true online TD(λ).
- For non-linear FA, such an efficient backward view does not appear to exist.
- Forward TD(λ) approximates the online λ-return algorithm and can be implemented efficiently for non-linear FA.
- The price that forward TD(λ) pays is a delay in the updates.
- Empirically, forward TD(λ) outperforms TD(λ)

Thank you!

References:
1) *van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., and Sutton, R. S.*
True online temporal-difference learning. Journal of Machine Learning Research, 17(145):1–40, 2016.
2) *van Seijen, H. Effective multi-step temporal-difference learning for non-linear function approximation.*
arXiv:1608.05151, 2016.